4-1-1997

# Evolution of solutions to real-time problems

Greg Semeraro

# Evolution of Solutions to

# Real-Time Problems

by

**Greg P. Semeraro**

A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Computer Engineering

Approved by:

___________________________________

Dr. Tony H. Chang, Professor

___________________________________

Dr. Roy Czernikowski, Professor and Department Head

Department of Computer Engineering
College of Engineering
Rochester Institute of Technology
Rochester, New York
April, 1997

# THESIS RELEASE PERMISSION FORM

*ROCHESTER INSTITUTE OF TECHNOLOGY*

*COLLEGE OF ENGINEERING*

Title: The Evolution of Solutions to Real-Time Problems

I, Greg P. Semeraro, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or part.

_______________________________

Greg P. Semeraro

29 April 1997

Date

# Evolution of Solutions to Real-Time Problems

by

**Greg P. Semeraro**

# Abstract

This thesis develops the theory and tools necessary for the determination of a near optimal Real-Time Operating System (RTOS) scheduling policy for an arbitrary multitasking problem specification. The solution is determined using a Genetic Algorithm (GA).

All real-time operating systems provide some means of 'tuning' the characteristics of the scheduling policy to accurately meet the application requirements. This thesis shows the applicability of using a GA to determine these parameters for an arbitrary application. In addition, the RTOS parameters considered are broad enough to allow the results to be used for specifying and/or choosing an RTOS for the actual implementation of a real-time system.

The domain of real-time applications which is of particular interest is that of embedded systems. In the embedded systems domain, real-time multitasking problems are specified by a series of timing constraints, time deadlines and practical available resources. These constraints guide the analysis of the results.

A PC-based RTOS/GA tool set is the end result of this thesis and can be used for the analysis of arbitrary real-time applications.

# Table of Contents

# Table of Figures

# Table of Tables

# Table of Equations

# Glossary

Note: All terms appearing in this glossary will appear for the first time as underlined text in the main body of the thesis.

| Term | Page | Description |
|---|---|---|
| Allele | 21 | Any of a group of possible mutational forms of a gene. |
| Epistatic Interaction | 7 | A genetic interaction which is nonrecipricating and is between nonalternative forms of genes in which one gene suppresses the expression of another affecting the same part of an organism. |
| Expected Allele Coverage | 27 | Given the following: A binary string encoding of length $L$, (with are $2L$ possible alleles) and a random population of $N$ encodings. The expected allele coverage is the expected proportion of all possible alleles that occur in the population and is $1-(1/2)^N$ [Ref. 19, pg. 32]. |
| GA | 1 | A Genetic Algorithm (GA) is the general term applied to computer algorithms which search for a solution to a given problem by |

| **Term** | **Page** | **Description** |
| --- | --- | --- |
| | | imitating the 'natural selection' which steers the evolution of living organisms. |
| Genotype | 21 | The genetic make-up of an organism. |
| NP-Complete | 1 | A problem is said to be NP-Complete if computing an exact solution to the problem requires nondeterministic polynomial time and it has been proven to be Complete. |
| Nyquist Theorem | 11 | Nyquist Theorem [Ref. 16, pg. 519]: Let $x(t)$ be a bandlimited signal with $X(\omega) = 0$ for $\|\omega\| > \omega_M$. Then $x(t)$ is uniquely determined by its samples $x(nT)$, $n = 0, \pm1, \pm2, \ldots$ if $\omega_S > 2\omega_M$ where $\omega_S = 2\pi/T$. |
| Phenotype | 21 | The environmentally and genetically determined observable appearance of an organism, especially as considered with respect to all possible genetically influenced expressions of one specific character. |
| RTOS | 1 | A Real-Time Operating System (RTOS) is a computer operating |

| Term | Page | Description |
| --- | --- | --- |
| | | systems which provide the necessary constructs to allow for the creation of multitasking applications capable of communicating with each other and meeting critical timing constraints. |
| TSR | 59 | A Terminate and Stay Resident (TSR) program is an MS-DOS program that remains in memory after the program has finished executing. TSRs typically hook into interrupt vectors and appear to execute in the background under MS-DOS. Since their execution can be sporadic, their impact on system performance is not defined. |

# 1. Introduction

The objective of this thesis is to develop a tool to assist the user in determining a near optimal Real-Time Operating System (RTOS) scheduling policy for an arbitrary multitasking problem specification. The solution is determined using a Genetic Algorithm (GA). The basic idea is that determining the 'best' RTOS scheduling policy is a very complicated task. This determination is usually done by an individual, or small group of individuals, with significant knowledge about the problem domain of the application. With multitasking software becoming increasingly complex, knowing that the RTOS scheduling policy that is being used is the most effective for the specific application is no longer something that can be determined merely by understanding the problem domain. What is needed is detailed knowledge about the interaction of the tasks that make up the multitasking application. Even with this knowledge, the problem of determining an optimal RTOS configuration is NP-Complete.

> "The problem of determining that a given schedule is optimal has been proven to be an NP-complete problem in general. ... The most notable exception from NP-completeness is determining a schedule for a monoprocessor system, in which all tasks are initially released simultaneously. If a problem is NP-complete, it is not likely that an algorithm exists that performs better than an algorithm based on enumerative methods."
>
> [Ref. 2, pg. 33]

For many optimization problem classes, it has been shown that GAs are uniquely qualified to search complex problem spaces for a global optimum, especially when that search space contains numerous local optima.

> "... The developed evolution program displayed some qualities not always present in the other (gradient-based) systems:
>
> - The optimization function for the evolution program need not be continuous. At the same time some optimization packages will not accept such functions at all.
> - Some optimization packages are all-or-nothing propositions: the user has to wait until the program completes. Sometimes it is not possible to get partial (or approximate) results at some early stages. Evolution programs give the users additional flexibility, since the user can monitor the 'state of the search' during the run time and make appropriate decisions. In particular, the user can specify the computation time (s)he is willing to pay for (longer time provides better precision in the answer).
> - The computational complexity of evolution programs grows at a linear rate; most of other search methods are very sensitive to the length of the optimization horizon. As usual, we can easily improve the performance of the system using parallel implementations; often this is difficult for other optimization methods"
>
> [Ref. 12, pg. 117].

It is for this reason that a GA will be used to determine the 'best' RTOS scheduling policy for the given problem specification.

All real-time operating systems provide some means of 'tuning' the characteristics of the scheduling policy to accurately meet the application requirements. The main objective of this thesis is to show the applicability of using a GA to determine these parameters for an arbitrary application. In so doing, a tool set will be developed which completely automates this tuning process. In addition, the RTOS parameters that are considered are broad enough to allow the results to be used for specifying and/or choosing an RTOS for the actual implementation of the real-time system.

The domain of real-time applications which is of particular interest is that of embedded systems. In the embedded systems domain, real-time multitasking problems are specified by the usual real-time properties, such as a series of timing constraints and time deadlines. In addition, embedded systems are further confined to using processing resources that are both practical and available in the particular application environment. These additional constraints are probably the most significant attributes of embedded real-time systems which must be considered during the design of the system. These constraints arise from the very nature of embedded systems and take the form of size, weight, power and cost restrictions.

Therefore, in addition to evaluating the ability of various RTOS scheduling policies to meet the real-time timing constraints and time deadlines (both hard and soft), the effectiveness of the scheduling policy is evaluated with regard to

CPU utilization. One of the most common means of reducing the size, weight, power and cost of an embedded system is to reduce the required CPU processing power. As a result of reducing the CPU processing power, the clock frequency and memory speed can often be reduced as well. This is the area of the system design where size, weight, power and cost savings are achieved. Therefore, the 'best' RTOS scheduling policy (which is application-specific) is the one which maximizes the following criteria:

- Percentage of timing constraints met.
- Weighted percentage of deadlines met (recognizing that for applications with hard deadlines, anything less than 100% is unacceptable).
- Percentage of CPU processing bandwidth available (i.e., surplus processing power).

Real-time operating systems typically provide relatively few parameters which can be 'tuned' to a particular application. This thesis evaluates the effectiveness of the following RTOS parameters (this list is intended to include all configuration parameters likely to be found in most available RTOSs):

- Tasking Model - Both cooperative and preemptive multitasking models are used.
- Timeslice - The processing timeslice ranges from 50 μSec to 65 mSec (only applicable for preemptive multitasking).

- Priority Inheritance - Scheduling policies with and without priority inheritance are used (enabling priority inheritance guarantees that unbounded priority inversion cannot occur).
- Priority Allocation - During run-time, the task priorities are either fixed at the initial assigned value (static priority allocation) or changed through the use of a rotation scheme (dynamic priority allocation).
- Initial Priority Assignment - The priority of each task is initially assigned using one of the following algorithms:
  - ◊ Uniform (constant) Assignment - All tasks are assigned the same priority regardless of execution profile.
  - ◊ Ad-hoc (random) Assignment - Tasks are assigned random priorities, uniformly distributed within the allowable range, without regard to execution profile.
  - ◊ Rate Monotonic Assignment - Tasks are assigned priorities based on execution period. Tasks with high execution rate (low period) are assigned high priorities; lower execution rates (higher periods) are assigned proportionally lower priorities. Execution period is defined as the amount of time which elapses between consecutive executions of a task (periodic execution is assumed). See *Figure 1 - Task Execution Profile*, $T_R$.
  - ◊ Deadline Monotonic Assignment - Tasks are assigned priorities based on execution deadlines. Tasks with short deadlines are

assigned high priorities; longer deadlines are assigned lower priorities. Execution deadlines are defined as the amount of time which can elapse between when a task begins executing and when it must reach a critical point in its execution. See *Figure 1 - Task Execution Profile*, $T_D$.

◊ Workload Monotonic (greedy) Assignment - Tasks are assigned priorities based on the amount of work that the task must perform. Tasks with high workloads are assigned high priorities; lower workloads are assigned proportionally lower priorities. Task workload is defined as the amount of time which elapses between when a task begins executing and when it completes executing. See *Figure 1 - Task Execution Profile*, $T_W$.

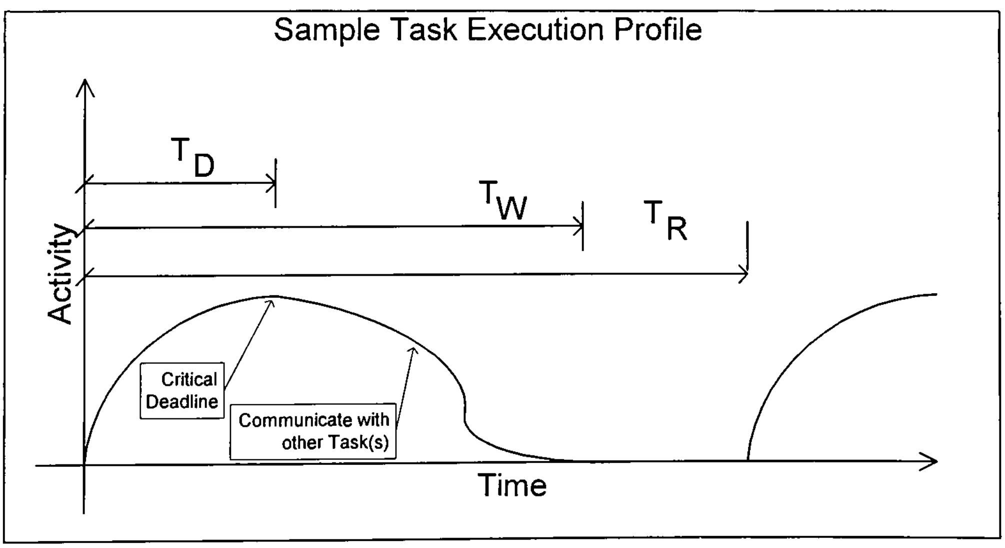



*Figure 1 - Task Execution Profile*

The Genetic Algorithm used to perform this RTOS evaluation is a generalized algorithm with application-specific genetic operators (population crossover and mutation), see *Figure 2 - Genetic Algorithm.* It has been shown that genetic operators are most capable of performing the required search when they are designed with prior knowledge of the problem domain.

> "... although the gains are not dramatic, $RAR_2$ out-performs uniform crossover with a simple penalty function on a family of problems in which the solution is a fixed-size set, using fitness functions which vary from highly epistatic to non-epistatic. This therefore provides some evidence that forma [sic] analysis is capable of leading to effective genetic search in problem domains not well suited to conventional binary representations."
>
> [Ref. 17, pg. 27]

Therefore, the approach taken is to fully define the problem domain (i.e., RTOS configuration parameters) and then design the genetic operators so as to properly evolve a solution to this specific problem domain.

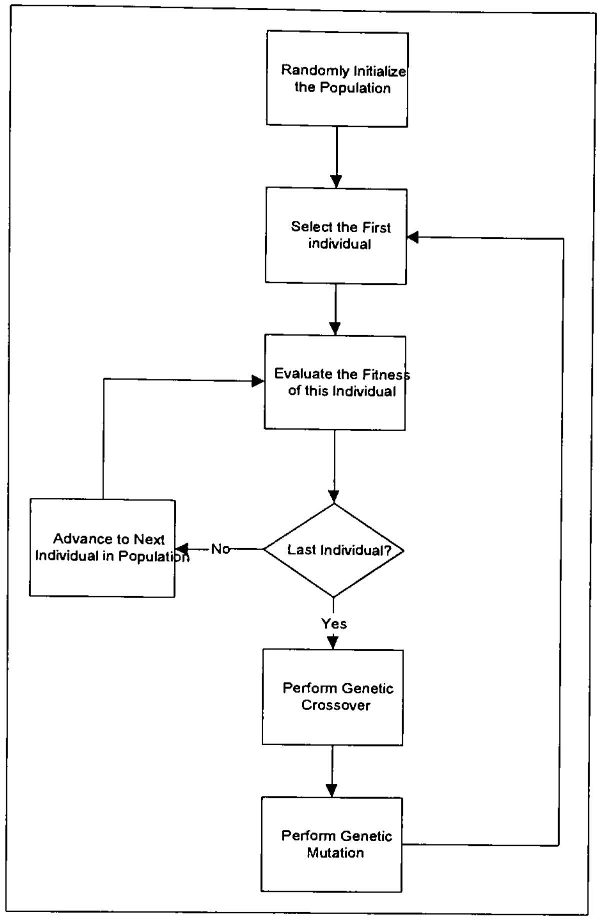



*Figure 2 - Genetic Algorithm*

It is obvious from *Figure 2* above that the GA itself is extremely simple. The power of the algorithm comes from the fact that as it progresses, it does two distinct things: 1) it continuously improves and, 2) it explores solutions which may provide additional improvements. Both of these operations are encompassed in the genetic operators of population crossover and mutation. These operators manipulate the genes of the individuals to produce the continuously improving and experimentation properties of the GA. Since the genes of the individuals are used to determine how it behaves (i.e., how well it solves the problem), and the genetic operators manipulate those genes, the genetic operators must be intricately tied to the representation of the genes. This results in genetic operators that are specific to the problem domain. Significant research has been done, attempting to determine 'universal genetic operators' based on 'universal gene representations'. Unfortunately, as the following quote summarizes, these attempts have not been very successful:

> "The EC (Evolutionary Computation) community differs widely on opinions and strategies for selecting appropriate representations, ranging from universal binary encodings to problem-specific encodings for TSP (Traveling Salesman Problem) problems and real-valued parameter optimization problems. The tradeoffs are fairly obvious in that universal encodings have a much broader range of applicability, but are frequently outperformed by problem-specific representations which require extra effort to implement but exploit additional knowledge about a particular problem class."
>
> [Ref. 7, pg. 619]

It is for these reasons that the GA developed for this thesis will use application specific genetic operators.

# 2. Background

In order to provide a firm foundation from which to establish this thesis, the concepts of real-time applications and genetic algorithms must be fully defined. This section will provide a generalized, detailed description of these very important concepts by using two simple, but concrete, examples.

## *2.1 Real-Time Applications*

It is important to first discuss the characteristics of real-time problems in general, after which, the problems associated with determining the appropriate configuration parameters of an RTOS will become apparent. The following example will be used to illustrate the general characteristics of real-time applications.

This example will consider the problem of a Reaction Rate Controller (RRC) of a chemical processing plant [Ref. 1]. The RRC is responsible for managing the reaction rate for a single chemical process. For this simplified example, the reaction rate is a function of the relative proportions of only two reagents and the temperature and pressure within the reaction chamber.

The user interface to the RRC is an operator console; the RRC must accept the desired reaction rate from the operator and provide reaction status to the operator. This operator interaction is relatively slow due to the fact that human operators respond slowly and require some time to assimilate information. It is assumed that the operator cannot request reaction rate changes more frequently than once every two seconds and that the operator cannot recognize status changes more frequently than four times per second. In order to ensure that the operator is always informed about the status of the reaction, the RRC must update the operator display no less than once per second.

The RRC must also accommodate the fact that the reaction rate within the reaction chamber can change within 100 milliseconds. In order to maintain a safe reaction environment, the RRC must respond to that change by re-positioning the reagent valves within an additional 100 milliseconds. This means that in the worst case, the reagent valves may be re-positioned as frequently as every 200 milliseconds. The RRC must also ensure that the reaction pressure and temperature are sampled at a sufficiently high rate to ensure that all changes are detected. Again, since the reaction rate within the reaction chamber can change within 100 milliseconds, the Nyquist Theorem states that the sampling period for the pressure and temperature sensors must be no more than 50 milliseconds.

The system software architecture for the RRC is shown in *Figure 3 - Reaction Rate Controller (RRC)*, which shows data and control flow within the system, timing parameters and software tasks.

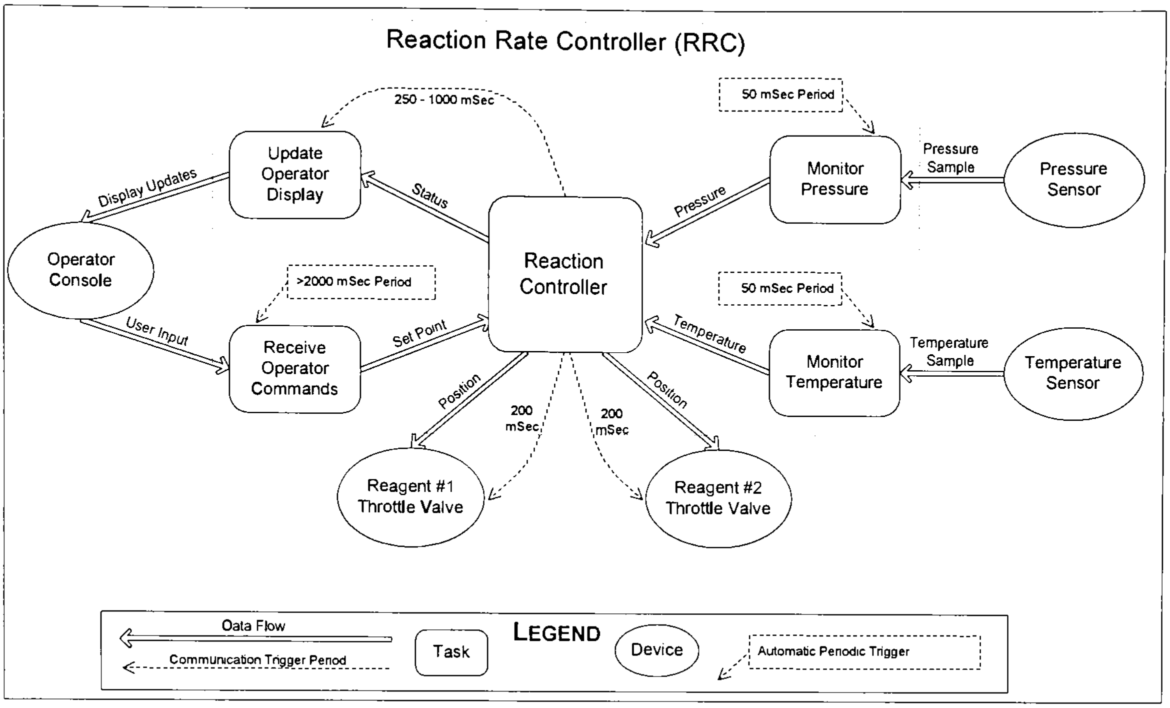



*Figure 3 - Reaction Rate Controller (RRC)*

The physical input/output devices connected to the RRC, the information flow to/from the RRC and a description of the timing requirements imposed on the RRC are summarized in *Table 1* below.

| ID # | I/O Device | Data Flow | Direction | Timing Requirements |
|---|---|---|---|---|
| 1 | Operator Console | Desired Reaction Rate | To RRC | Non-Periodic: >2000 mSec |
| 2 | Operator Console | Actual Reaction Rate & Reaction Status | From RRC | Periodic: 250 mSec (Design Goal), <1000 mSec (Requirement) |
| 3 | Reagent #1 Valve | Valve Position | From RRC | <100 mSec after Reaction Change |
| 4 | Reagent #2 Valve | Valve Position | From RRC | <100 mSec after Reaction Change |
| 5 | Pressure Sensor | Pressure Sample | To RRC | Periodic: 50 mSec |
| 6 | Temperature Sensor | Temperature Sample | To RRC | Periodic: 50 mSec |

*Table 1 - RRC Requirements*

From this example it is readily apparent that real-time applications are most notably characterized by deadlines. That is, real-time applications are specified by a series of deadlines that the software (in this example, the RRC) must meet. These deadlines typically arise from the very nature of the application and the processing that the application must perform. Careful analysis of the problem can expose all of the deadlines that a system must meet. In addition to identifying the deadlines, it is also important to characterize the nature of each deadline. For example, some deadlines are more 'important' to the proper operation of the system than others. In the RRC example above, it is obvious from *Table 1 - RRC Requirements* that requirement #2 (Display Update) is 'least' important because a range of allowable performance values is specified. Intuitively that makes sense as well because failure to update the operator display as frequently as four times per second will result in a graceful performance degradation. On the other hand, failure to properly control the reagent valves may result in an unsafe reaction; therefore the requirements associated with actually controlling the reaction are 'more' important to the correctness of the system.

The different relative importance of real-time deadlines is expressed as to whether a given deadline is 'hard' or 'soft'. Hard deadlines are those which must always be met in order for the system to be said to operate correctly. Soft deadlines are those which can be missed (occasionally) without severely impacting the correctness of the system. In other words, if a system fails to meet a hard deadline, a catastrophic failure usually results. Failure to meet a soft deadline usually results in graceful degradation of some aspect of system performance. In the above example, failure to meet a hard deadline associated with controlling the reaction rate may result in an unsafe reaction which could result in loss of life or property. Failure to meet the soft deadline associated with updating the operator display will result in the inability of the operator to adjust the reaction rate to the desired level (if the display is not updated, the operator does not know that the reaction rate is something other than the desired rate). Therefore, in the above example, all deadlines are hard except for update of the operator display. In essence, the system can be said to be operating properly even if the operator display is updated less frequently than four timer per second.

In large, complex systems, the concept of strictly hard or soft deadlines is usually too restrictive to adequately describe all of the deadlines present within the system. Obviously, some deadlines will still be strictly hard or soft, but some soft deadlines may be more important than other soft deadlines. It is typically necessary to extend hard and soft deadlines to include a 'degree of hardness' so that all of the subtle aspects of required system performance can be captured. This allows the system to be evaluated based on how well the hard deadlines are met ('Is the system operating correctly/safely?'), as well as how well the varying degrees of hardness deadlines are met ('Is the performance of the system adequate?'). Another method of specifying the relative importance of a soft deadline is to use the benefit accrual model of timeliness [Ref. 2, pg. 30]. In this model, all deadlines are modeled as a function of time, $f(t)$, which corresponds to the benefit achieved to the system by completing the deadline at time $t$. In this way the complete spectrum of hard and soft deadlines can be modeled consistently and thoroughly.

The problem associated with determining the appropriate configuration parameters of an RTOS for a given real-time application is extremely difficult.

One reason for this difficulty is that there is often very little to guide a designer in choosing the 'best' RTOS parameters. Even though it may be possible to fully specify the real-time application in terms of its deadlines, that alone in no way guarantees that the deadlines will actually be met. The RTOS configuration will affect the system performance which will in turn determine to what extent the system performs correctly (hard deadlines) or to what degree the system performance is degraded from the design goals (soft deadlines).

### 2.1.1 Formal Methods for Real-Time Analysis

Recently, there has been significant research into defining formal methods for analyzing the correctness of real-time systems [Ref. 9]. The appeal of this approach is that by using some of these formal methods, the correctness of a real-time system design can often be proven analytically. Unfortunately, for large complex systems, the applicability of these methods can be limited.

The techniques that have been developed for analyzing real-time systems generally fall into two distinct categories: model-based reasoning and proof-based reasoning [Ref. 9, pg. 14]. Each of these has associated advantages and disadvantages, and each has been used successfully to model real-time systems. The remainder of this section outlines the major accomplishments in defining formal methods for the analysis of real-time systems.

#### *2.1.1.1 Model-Based Analysis*

For model-based analysis, the real-time system is typically specified using a graphical model tool set. The main advantage to this approach is that the model can be created and maintained by individuals with no formal mathematical background, and the analysis is automatically performed by the tool set. In addition, the graphical model provides a description of the system that can be used to document the real-time design. One disadvantage of this approach is that it does not scale well to large complex systems. The problem of creating and maintaining a single large model that fully describes the entire system can easily result in errors and areas of model inadequacy. Another problem is that as the model grows, the analysis process becomes increasingly more time consuming. This is due to the fact that the complexity of the analysis grows as $O(2^N)$ where $N$ is the number of states in the real-time system.

One promising method of model-based analysis that has been introduced is based on the Modechart specification language [Ref. 14]. This specific method provides a set of tools which is used to create the system specification, simulate the model to exhibit individual behaviors of the system, and verify global properties of the system. This method is promising because it provide significant automation of the analysis using the tool set provided. Unfortunately this, like all state-based methods suffers from the state explosion problem. That is, as the real-time system being modeled more closely parallels a real-world problem, the number of states required to accurately represent the system becomes unmanageably large. Ultimately what often happens with such a system is that, over time, the model loses synchronization with the actual system being designed and becomes obsolete. Another potential problem with state-based models is that the computations that must be performed to verify the design grow of order $2^N$. Again, the number of states creates an often unacceptable computation time which is not proportional to the increase in design complexity. Even a marginal increase in complexity of the system can result in a tremendous increase in the complexity of the computations necessary to 'prove' the validity of the design. For small-scale real-time systems this method may be feasible but it is unlikely to be applicable for large, complex systems.

#### *2.1.1.2 Proof-Based Analysis*

For proof-based analysis, standard logical deduction methods are used to formally prove that one specification is equivalent to another specification. With the system design and system requirements being used as the starting and ending points of the proof, this method can prove that a specific real-time design specification is equivalent to the corresponding system requirements specification. The main advantage to this approach is that a formal proof of the design meeting the requirements is achieved. The disadvantages are that producing the proof requires formal mathematical training, and since the proof must be done by hand, changes in the design and/or requirements can necessitate the complete recalculation of the proof.

One promising method of proof-based analysis that has been introduced is based on a timed process algebra which supports time consuming actions, instantaneous events, and the concept of prioritized interactions [Ref. 11]. In order to utilize this method, the real-time system and the requirements must be specified in terms of the ACSR specification language [Ref. 11, pg. 169-179]. This may not seem like a severe limitation but it often is. First, expertise in the ACSR specification language is necessary, although this is not significant it is knowledge that is not often present within the real-time development community. Secondly, and more importantly, using this analysis method requires that not only the system being designed but also the system requirements be specified in the ACSR specification language. System requirements are very infrequently defined to a degree that would allow a formal specification to be developed. More often than not the system requirements are being developed at the same time that the system design is being developed. Again, this is not to say that this method is not without merit, it is just that for real-world problems where the requirements of the system are dynamic this method may not be feasible.

## 2.2 Genetic Algorithms

It is important to first discuss the characteristics of genetic algorithms (GAs) in general, after which, the particular application of a GA to determining appropriate configuration parameters of an RTOS can be investigated. One way to understand GAs is to view them as algorithms which search the space of possible solutions to a problem for the optimal solution. In this regard, GAs are very similar to other parametric search algorithms (hill climbing, simulated annealing, etc.). The most important difference between GAs and other search methods is that GAs attempt to model the natural world laws of 'random perturbations' and 'survival of the fittest'. This property is unique to GAs.

The basic architecture of a GA is one of a collection of individuals, where each individual is a candidate solution to the problem being solved. In turn, the basic architecture of each individual is one of a collection of 'genes', where each gene represents one component of the parameterized solution to the problem.

The GA uses this 'population' of individuals and 'gene pool' of genes together with the natural laws of 'random perturbations' and 'survival of the fittest' to allow the 'best' solution to evolve over many generations, see *Figure 4 - General Genetic Algorithm.* From this discussion it is apparent that the GA is parameterized as follows:

- Population Size, $N$ - The number of individuals that will be used to comprise the population. This value is chosen such that statistical significance within the population is maintained and such that the initial (and potential) gene pool is sufficiently diverse.
- Mutation Rate, $P_m$ - The rate at which 'random perturbations' will be introduced into the gene pool. This value is chosen in such a way as to ensure that any useful genetic material which is lost due to crossover, is re-introduced into the gene pool and to allow the GA to explore areas of the search space that have not yet been explored.
- Crossover Rate, $P_c$ - The rate at which individuals participate in genetic recombination, i.e., the rate at which genes are recombined from individuals of the present generation to form individuals of the next generation. This is the means that 'fit' genetic material 'survives' from generation to generation. The value is chosen in a manner that allows for a reasonable rate of convergence, a trade-off between execution time and probability of false convergence must be made.

It is obvious that this approach does not constitute an exact or formal method. Instead, as the example that follows will illustrate, the GA seems to meander about in its search for the 'best' solution, never really knowing were the search will lead but amazingly enough almost always able to find a reasonably appropriate solution.

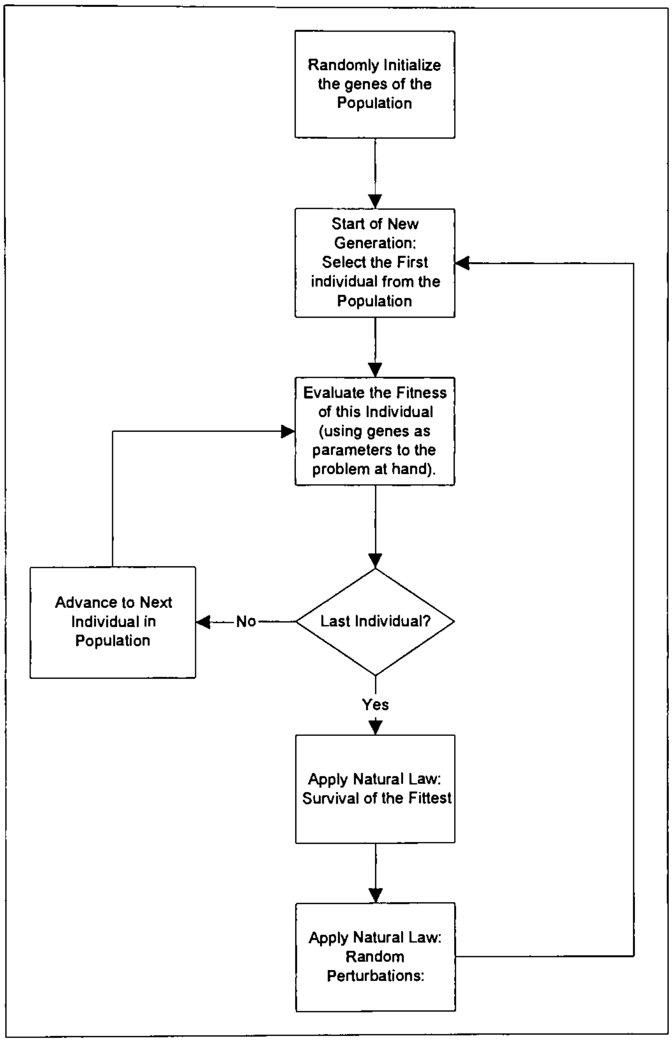



*Figure 4 - General Genetic Algorithm*

Note that the answer found by the GA is not necessarily optimal. This arises from the fact that the GA is never really finished, in theory the solutions would evolve over an infinite number of generations (notice that there is no exit condition specified in the *Figure 4*). This means that in practice, one of the more difficult problems associated with using a GA is determining when to stop the search. This decision is often based on statistical measurements of the population and/or gene pool, as stated below.

> "There are two basic categories of termination conditions, which use the characteristic of the search for making termination decisions. One category is based on the chromosome structure (genotype); the other - on the meaning of a particular chromosome (phenotype). Termination conditions from the first category measure the convergence of the population by checking the number of converged alleles, where an allele is considered converged if some predetermined percentage of the population have the same (or similar - for non-binary representations) value in this allele. If the number of converged alleles exceeds some percentage of total alleles, the search is terminated. Termination conditions from the second category measure the progress made by the algorithm in a predefined number of generations; if such progress is smaller than some epsilon (which is given as a parameter of the method), the search is terminated."
>
> [Ref. 12, pg. 65].

The natural laws of 'random perturbations' and 'survival of the fittest' were first articulated by Charles Darwin in The Origin Of Species. Even though these concepts were focused entirely on biological organisms and their interactions, it is in this rich background that the field of genetic algorithms is based.

As shown below, Darwin uses the term 'individual differences' to express the concept of random perturbations as it relates to biological organisms; the exact same principles are used when applying genetic algorithms to computational problems, see section *3.2 Genetic Algorithm*.

> "The many slight differences which appear in the offspring from the same parents, or which it may be presumed have thus arisen, from being observed in the individuals of the same species inhabiting the same confined locality, may be called individual differences. No one supposes that all the individuals of the same species are cast in the same mould. These individual differences are of the highest importance for us, for they are often inherited, as must be familiar to every one; and they thus afford materials for natural selection to act on and accumulate, in the same manner as man accumulates in any given direction individual differences in his domesticated productions."
>
> [Ref. 5, pg. 59]

As Darwin indicates below, individual biological organisms within a population compete with each other for the right to survive. Together with random perturbations, natural selection creates a continuously improving population. This is true even though, for any given generation, there will exist individuals that are less fit for survival.

> "Can it, then, be though improbable, seeing that variations useful to man have undoubtedly occurred, that other variations useful in some way to each being in the great and complex battle of life, should occur in the course of many successive generations. If such do occur, can we doubt (remembering that many more individuals are born than can possibly survive) that individuals having any

advantage, however slight, over others, would have the best chance of surviving and of procreating their kind? On the other hand, we may feel sure that any variation in the least degree injurious would be rigidly destroyed. This preservation of favourable individual differences and variations, and the destruction of those which are injurious, I have called Natural Selection, or the Survival of the Fittest. Variations neither useful nor injurious would not be affected by natural selection, and would be left either a fluctuating element, as perhaps we see in certain polymorphic species, or would ultimately become fixed, owing to the nature of the organism and the nature of the conditions."

[Ref. 5, pg. 87-88]

The following simple example will be used to illustrate the general characteristics of genetic algorithms by determining the maximal value of a complex trigonometric function. Consider the following function of two real-valued variables:

$$f(x, y) = 21.5 + x \cdot \sin(4 \cdot \pi \cdot x) + y \cdot \sin(10 \cdot \pi \cdot y)$$
$$-3.0 \le x \le 12.1$$
$$4.1 \le y \le 5.8$$

*Equation 1 - Sample Optimization Function*

It is not at all obvious what values of $x$ and $y$ result in the maximal value of this function. The plot below, *Figure 5 - Sample Optimization Function Plot*, shows the function for the given range.

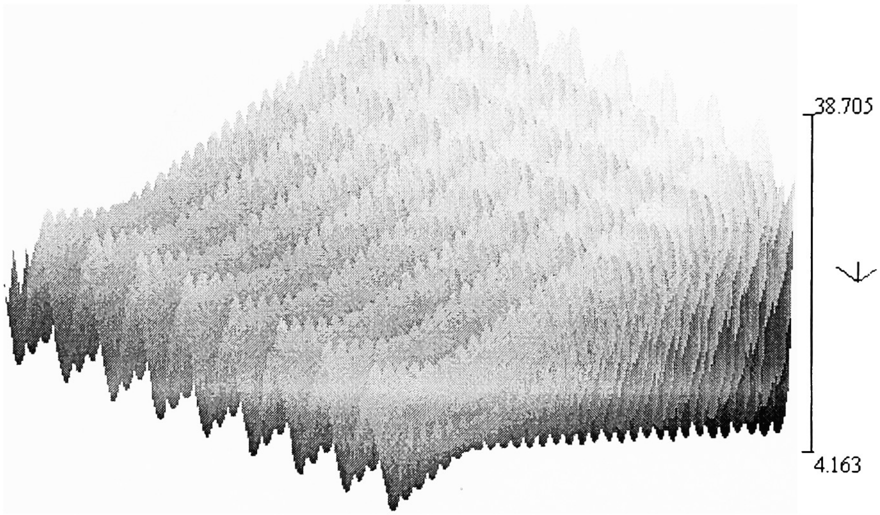



*Figure 5 - Sample Optimization Function Plot*

Because the function has so many local maxima, most optimization algorithms would 'find' a local maximum instead of the global maximum unless an extremely close initial value was used, GAs do not posses such a limitation. After localizing the exhaustive search to the region shown in *Equation 2 - Sample Optimization Function (Localized)*, the maximum value is determined. The graph is shown in *Figure 6 - Sample Optimization Function Plot (Localized)* shows the maximum value of the function is $f(11.626,5.65)=38.775$.

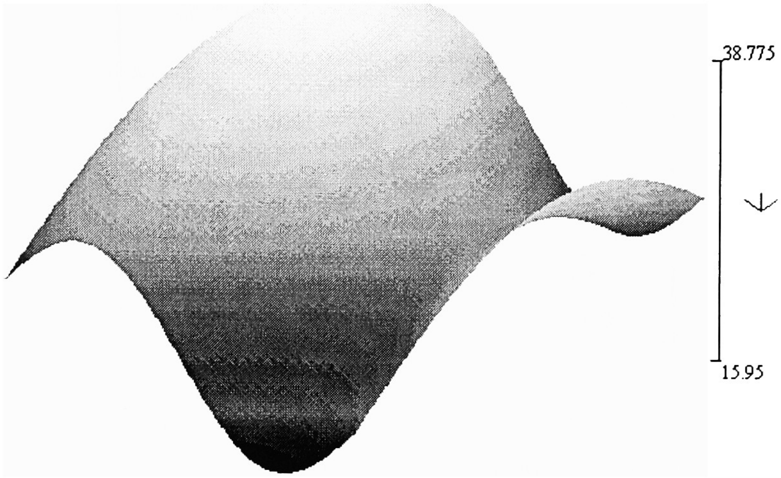



*Figure 6 - Sample Optimization Function Plot (Localized)*

$$f(x,y) = 21.5 + x \cdot \sin(4 \cdot \pi \cdot x) + y \cdot \sin(10 \cdot \pi \cdot y)$$
$$11.5 \leq x \leq 11.7$$
$$5.5 \leq y \leq 5.7$$

*Equation 2 - Sample Optimization Function (Localized)*

The following table shows the results of a single run of `GMATH-GA.EXE`, the results show the percent error after 903 generations. Notice that even without any knowledge of the optimal solution, and searching the entire range (see *Equation 1*), the GA performed remarkably well at determining the maximum value of the function. This function optimization example will be revisited in the remainder of this section as the various aspects of genetic algorithms are detailed.

| *f(x,y)* | *x* | *y* |
|---|---|---|
| 38.668 (0.276%) | 11.615 (0.095%) | 5.650 (0.000%) |
| 38.664 (0.286%) | 11.615 (0.095%) | 5.651 (0.018%) |
| 38.662 (0.291%) | 11.615 (0.095%) | 5.652 (0.354%) |
| 38.657 (0.304%) | 11.615 (0.095%) | 5.648 (0.354%) |
| 38.603 (0.443%) | 11.615 (0.095%) | 5.645 (0.088%) |

*Table 2 - GA Solutions to Sample Function*

### 2.2.1 Genetic Operators

The most important aspect of the genetic algorithm is the internal representation of the genes. There is nothing which is more intrinsic to a specific GA for a problem than the gene representation and the genetic operators which manipulate those genes. There are two genetic operators which essentially comprise the genetic algorithm: mutation(...) and crossover(...). These operators are responsible for the behavior of the GA, its ability to converge to a solution, and the rate at which the convergence occurs.

It is important to note that both of these operators must be designed with complete knowledge of the precise genetic representation which is being used. There are two different representation methods which are typically used to create a GA: pure binary genes and representational genes. There is significant debate within the GA research community as to whether one method is superior to the other, most indications are that pure binary genetic representations may be out performed by representations which more closely match the problem domain [Ref. 17, pg. 27].

The following sections will describe the mutation(...) and crossover(...) operators for both type of genetic representation.

### *2.2.1.1 Mutation*

The mutation(...) operator attempts to model the natural law of 'random perturbations'. Quite simply, mutation is the random altering of genetic material of some members of the population. One of the parameters of a GA is the mutation rate (or probability of mutation). This parameter controls how frequently (or with what probability) genes will be mutated.

Determining the 'correct' mutation rate for a given genetic algorithm optimization problem is itself an optimization problem, this is also true for determining other parameter values for a GA. Since it is not feasible to construct a GA which determines the parameters for another GA, although this has, been done [Ref. 6, pg. 92], typically, general principles are used to select approximate GA parameter values.

One of the most important measurements of the GA parameters is the expected allele coverage of the population. The mutation rate and the population size directly effect the expected allele coverage, 99% expected allele coverage is typically used [Ref. 19]. The way that the expected allele coverage is calculated is different for pure binary genes than for more representational genes. *Equation 3* and *Equation 4* provide guidelines for determining the mutation rate or population size for binary and representational genes respectively [Ref. 19].

For pure binary genes the genetic material is uniform, therefore expected allele coverage is simply a function of the population size and the number of choices for each bit (i.e., 2), see *Equation 3 - Expected Allele Coverage (Binary Representations)* below [Ref. 19, pg. 33].

$$E[ac] = 1-(1/2)^N$$

Where: $N$ is the population size

*Equation 3 - Expected Allele Coverage (Binary Representations)*

For more representational genes the genetic material is not uniform, therefore a summation is required to accumulate the weighted expected allele coverage, see *Equation 4 - Expected Allele Coverage (Non-Binary Representations)* below [Ref. 19, pg. 33].

The probability that exactly $m$ distinct symbols are used for any given gene within the entire population is given by:

$$p_m(N,K) = \frac{1}{K^N}\cdot\binom{K}{m}\cdot\sum_{v=0}^{K-m-1}(-1)^v\cdot\binom{K-m}{v}\cdot(K-m-v)^N$$

Where: $K$ is the number of distinct symbols used in the encoding and $N$ is the population size.

Since all gene positions are disjoint with respect to alleles, the expected allele coverage is given by:

$$E[ac] = \frac{1}{K}\cdot\sum_{m=1}^{K} m\cdot p_m(N,K)$$

*Equation 4 - Expected Allele Coverage (Non-Binary Representations)*

For pure binary representations, carrying out the mutation operation is very straightforward. Since each individual's genotype is represented as a single bit string of length $m$, where individual genes that comprise the genotype are assigned sub-regions of the bit string, and the population is composed of $n$ individuals, the entire gene pool is composed of $n \bullet m$ bits. The mutation operation is performed once every generation where each bit is mutated (i.e., inverted) with probability $P_m$.

For representational genes, carrying out the mutation operation is slightly more complex. Since is it possible that simply mutating single bits will result in illegal (i.e. meaningless) genes, the mutation operation must always produce a gene which is valid within the problem domain. For this reason the mutation operator must be closely tied to the actual problem being solved. Again, the mutation operation is performed once every generation but in this case each gene is mutated (the exact meaning of which is application specific) with probability $P_m$.

In the example above, the genes are two real-valued numbers in the ranges specified in *Equation 1 - Sample Optimization Function*. Since the genes are not binary encodings, mutation for this example is defined as replacing the gene to be mutated with a randomly selected value from the corresponding range. This is typical of a representational genotype. The mutation rate, $P_m$, is determined by choosing a population size, *N*, of 175 individuals, setting the desired expected allele coverage to 97% and evaluating *Equation 4*, above. The resulting mutation rate, for this example, is $P_m = 3\%$, resulting in accuracy of 1/50th of the respective ranges of each variable.

#### *2.2.1.2 Crossover*

The crossover(...) operator attempts to model the natural law of 'survival of the fittest'. Genetic crossover allows individuals to pass their genetic material on to individuals of the next generation. To do this, those individuals that are most fit are chosen to participate in genetic crossover. When crossover(...) is applied to the population, two relatively fit individuals are chosen at random and their genetic material is combined in some way to produce an offspring. The most common type of crossover is 'single-point crossover'. Single-point crossover occurs when a randomly selected gene is used to split the genotypes of the two parents, the offspring inherits one half of the genotype from each parent. There are important consequences of this operation, as described below.

> "One important feature of one-point crossover you should be aware of is that it can produce children that are radically different from their parents. ... Another important feature is that one-point crossover will not introduce differences for a bit in a position where both parents have the same value. ... An extreme instance occurs when both parents are identical. In such cases, crossover can introduce no diversity in the children."
>
> [Ref. 6, pg. 17]

There is no theoretical guideline for determining the proper crossover probability, $P_c$, for a GA, typical values range from 25% to 75%. Values used for GAs in this thesis will be determined empirically but will be from the typical range.

In the example above, single-point crossover with a crossover probability of $P_c$=25% was used.

# 3. Evolution of Solutions to Real-Time Problems

The most promising applications of genetic algorithms arise in the area of engineering design. This is true because many design problems are very complex (often NP-Complete or at least NP-Hard) and once the engineering and business trade-offs have been made, these problems can be expressed as an optimization problem. GAs are uniquely qualified for these exact type of optimization problems where there are far too many system constraints for an engineer (or small group of engineers) to keep in mind during the design phase. If sufficiently large computers are available there is really no limit to the number of constraints that can be managed by a GA. In addition, GAs are capable of solving problems without actually requiring that the specific design problem be solved (or solvable) at all, as the following summarizes:

> "Assume, then, that (for some reason) we have to (or like to) build a new system to solve a nontrivial optimization problem. ... Then we have to make a choice: either we can try to construct an evolution program or we can approach the problem using some traditional (heuristic) methods. It is interesting to note that in a traditional approach it usually takes three steps to solve an optimization problem:
>
> 1. Understand the problem
> 2. Solve the problem
> 3. Implement the algorithm found in the previous step

In the traditional approach, a programmer should solve the problem - only then may a correct program be produced. However, very often an algorithmic solution of a problem is not possible, or at least is very hard. On top of that, for some applications, it is not important to find the optimal solution - any solution with a reasonable margin of error (relative distance from the optimum value) will do. ... An evolution programming approach usually eliminates the second, most difficult step. Just after we understand the problem, we can move to the implementational issues. The major task of a programmer in constructing an evolution program is a selection of appropriate data structures as well as genetic operators to operate on them (the rest is left for the evolution process)."

[Ref. 12, pg. 302]

In addition to the all too often cited example of VLSI circuit design, there are many other examples of systems that have been designed using, at least in part, a genetic algorithm. One such example is the parametric design of aircraft:

"The problem addressed here is parametric design using a design concept typical of modern fighter aircraft. The problem is representative of real-world aircraft conceptual design problems. It is similar to practical problems that are currently being solved using methods described here. The aircraft takes off, accelerates subsonically and then perhaps supersonically, transits to an engagement area, conducts combat operations, returns to base, loiters, and lands. The parameters describing the mission include altitude, duration and velocity of those activities, range, and acceleration and maneuverability requirements for combat.

Representative parameters of the aircraft's geometry and configuration include fuselage width, height and length; wing planform and thickness characteristics; vertical and horizontal tail surface dimensions; and engine size (that is, aircraft thrust-to-weight ratio). Parameters of the design concept include unscaled engine weight and dimensions, and weights of aircraft subsystems and the weapons complement. The total number of parameters for this problem is fifty."

[Ref. 3, pg. 113]

The engineering design problem that is being considered in this thesis is the system software architecture design for an embedded, real-time application. This design problem can be viewed as an optimization problem. The 'function' to be optimized is the applicability of a real-time operating system (RTOS) configuration to a given problem specification. The parameters of the 'function' are the configuration items for the RTOS. It is reasonable to hypothesize that there is no optimal RTOS configuration for all real-time applications, instead the best RTOS configuration for any given application will, in some unknown manner, be a function of the application itself. Therefore, the tools developed must be capable of accepting a user specified real-time application. The tools will then generate a list of the 'best' RTOS configurations for that application. A genetic algorithm will be used to determine these RTOS configurations. In general, a genetic algorithm can be used to find the optimal solution to a given problem but only if given infinite time. Since infinite time is not feasible, the solutions provided by the tools developed in this thesis will produce sub-optimal solutions. The following sections describe the RTOS, genetic algorithm and real-time application specification aspects of the tool set.

### 3.1 Real-Time Operating System (TASKING)

TASKING provides all of the RTOS services necessary for real-time application development. It is important to note that the services provided merely allow for the development of real-time applications -- they do not ensure that a system design will actually perform in real-time. The services provided are easily segregated into the following categories: task management, software events, binary semaphores, counting ("Dijkstra") semaphores, message passing, condition variables and interactive I/O handling.

The task management services primarily allow tasks to be dynamically created and destroyed. In addition, services are provided so that tasks can be suspended, resumed or change their priority. These services are sufficient to allow for the design of a multitasking application; however, additional services are necessary to allow the tasks to communicate and interact. The remaining categories of services provide these necessary capabilities.

Inter-task communication and interaction are achieved by the use of one (or more) of the following constructs: software events, binary semaphores, counting ("Dijkstra") semaphores, message passing or condition variables. These constructs are actually a superset of those that are required for inter-task communication. It has been shown that at the most basic level, all of these constructs can be considered to be a special case of a simple semaphore.

> "In the previous sections we have studied four different interprocess communication primitives. Over the years, each one has accumulated supporters who maintain that their favorite way is the best way. The truth of the matter is that all these methods are essentially semantically equivalent (at least as far as single CPU systems are concerned). Using any of them, you can build the other ones."
>
> [Ref. 18, pg. 52]

Nonetheless, in order to ease application development, they are all provided by `TASKING`. Note that when these constructs are created they are just like any other variable in that their value is undefined. Before the construct is used it should be initialized to a value appropriate for the given construct. In addition, task interaction with the application user is performed by a set of blocked I/O handling routines.

Software events are one of the more simple constructs; events can take on only one of two possible values: signaled or unsignaled. In addition to this restriction, events can retain only a single signal, i.e., once an event has been signaled, all subsequent signals to that event are superfluous (the event remains signaled) but are not considered errors. Tasks may communicate simple information with events but most often events are used for task synchronization. Quite simply one task would wait for an event to be signaled by another task which would indicate that some temporal situation was satisfied (e.g., a data buffer had just become non-empty or non-full, etc.). At that point the tasks would be synchronized.

Whereas software events are typically used to task synchronization, binary semaphores are typically used to ensure mutually exclusive access to a resource which is shared between two (or more) tasks. Binary semaphores, like software events, can only take on two possible values: 0 or 1. In addition, it is logically an error to signal an already signaled binary semaphore; `TASKING` will enforce this restriction by detecting and reporting this as an error.

"Dijkstra", or counting, semaphores are similar to binary semaphores except that they can retain multiple signals because `TASKING` maintains information so that it knows how may signals are stored in the semaphore at all times. Whenever a given semaphore is signaled the count is incremented. Tasks that wait on the semaphore are only suspended when the semaphore contains no signals (i.e., the semaphore value is zero). This allows "Dijkstra" semaphores to provide control over access to a resource that has more than one instance. That is $n$ tasks can access $m$ similar resources, where $n \geq m$.

`TASKING` also supports passing arbitrary, application-specific messages between tasks; this is accomplished when the source task allocates memory for the message and sends a pointer to the message to the destination task. The receiving task is responsible for de-allocating the memory for the message after it has been consumed. There are no restrictions on the size, structure or number of messages that can be used within the application. `TASKING` manages mailboxes for the destination of the messages and all messages are delivered in a non-prioritized, first-in-first-out manner. Extreme care must be taken to ensure that these message queues do not cause unbounded priority inversion; note that this form of priority inversion cannot be detected or prevented by TASKING.

The last construct is condition variables. Condition variables do not have any memory associated with them; if a task is already waiting on a condition variable when it is signaled it will become unblocked, but if a condition variable is signaled and no task is waiting on it, then the signal is lost. Condition variables are typically used in the implementation of Hoare Monitors or for simple forms of task synchronization.

The other service that TASKING can provide is blocked I/O. Both keyboard and MS-Mouse I/O can be performed in a completely blocking manner where the task is blocked (suspended) until input is available. The only restriction is that only one task each can perform blocked keyboard input and blocked MS-Mouse input.

The detailed design of each service and the general operating system philosophy are presented in the remainder of this section.

### 3.1.1 Facilities and Capabilities

TASKING allows for the creation of up to 2047 simultaneous, parallel executing tasks, or threads of execution, within the application. The tasks execute in an environment of 1000 distinct priority levels. The error handling options provided by TASKING are quite flexible; each task, or set of tasks, can have its own specific error handling routines; all tasks can use global application error handlers; or TASKING can handle any/all of the application errors. When default error handlers are used, TASKING will attempt to continue executing in the presence of errors, but it should be noted that under error conditions, task deadlock is likely to occur (under task deadlock conditions TASKING will terminate the application). As regards inter-task constructs, the number of these constructs used by the application is limited only by the amount of available memory in the system. TASKING imposes no limits on these constructs. In addition to these features, TASKING provides performance and operational statistic reports which can be used to determine how well TASKING has been configured to match the needs of the application (reports may also be useful for determining the needs of the application tasks).

The structure of TASKING is given in *Figure 7 - Operating System Facilities/Structure.* Detailed component descriptions are provided in the following section and complete TASKING source code can be found in *Appendix A RTOS Source Code* [*TASKING.INT* and *TASKING.PAS*].

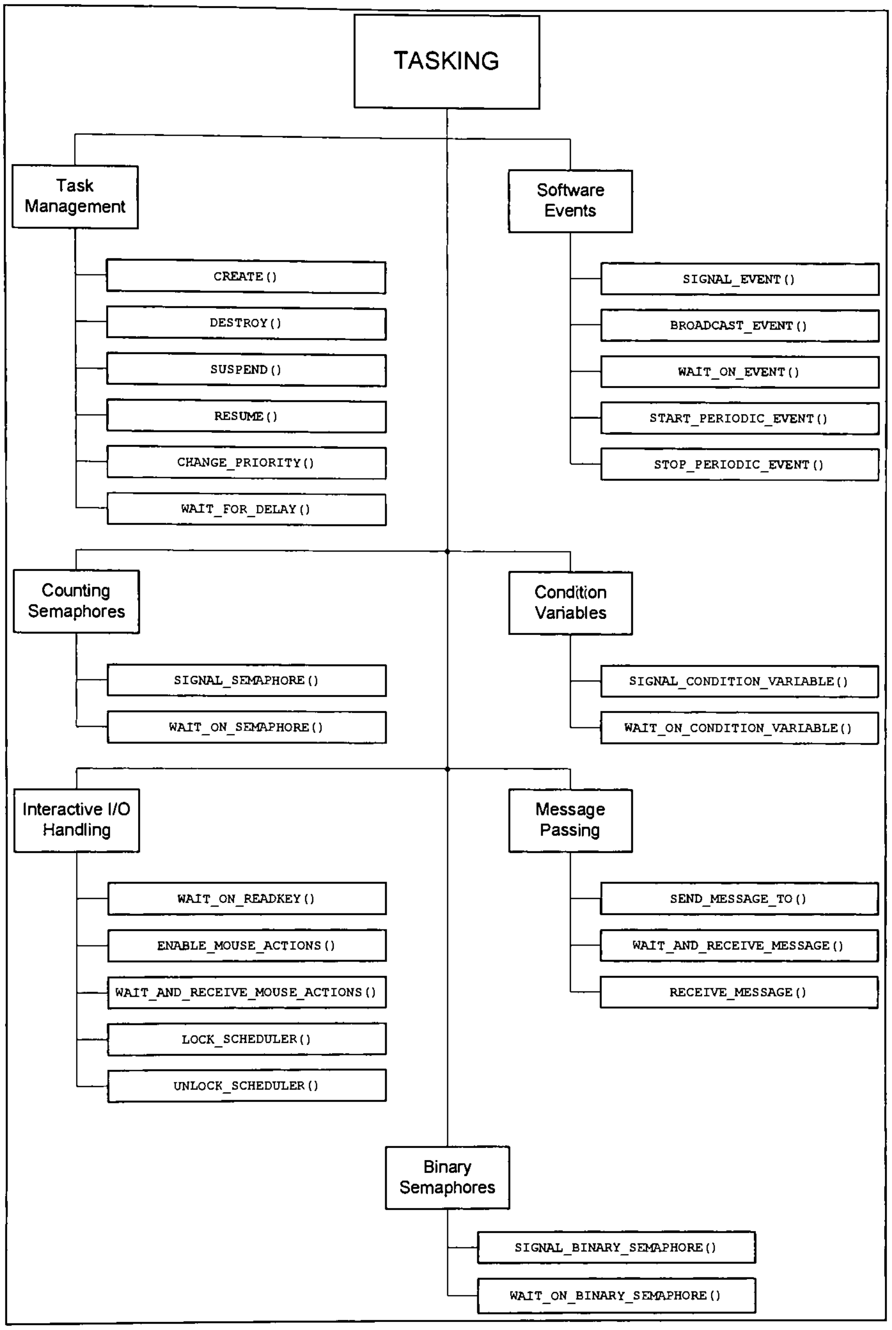



*Figure 7 - Operating System Facilities/Structure*

The basic architecture of a `TASKING` application consists of a program 'main' body, used only to initialize global data and create initial tasks, and the tasks themselves. No tasks that are created by the 'main' body will begin executing until the 'main' body actually terminates. Essentially, the main program is used only to initialize the multitasking application. A task automatically ceases to exist when the body of the task terminates. The program as a whole terminates when all tasks have terminated or when the *halt(...)* system call is made. *Appendix C Support Software Source Code* [*DINING.PAS*] includes a sample `TASKING` program, it is a solution to the classic multitasking problem proposed and solved by Dijkstra in 1965 called *The Dining Philosophers Problem* [Ref. 18, pg. 56].

### *3.1.1.1 CREATE(...)*

This routine creates all system data structures for a parallel executing task. The task will be made ready to run and will be available for execution. Actual execution occurs when the scheduler activates the task (i.e., precise execution time is unknown and depends on the task priority and the relative priority of other 'ready' tasks). In addition, no tasks are allowed to execute until the application 'main' has terminated. In this manner all tasks created in the 'main' body begin executing simultaneously. In addition to creating tasks in the 'main' body, tasks can also create other tasks. There is no concept of task parent-child relationships within `TASKING`; when a task is created it continues to exists until it is destroyed or terminates itself, i.e., the task existence is independent of the existence of the task that created it.

### *3.1.1.2 DESTROY(...)*

This routine forces the task to terminate and releases all of the task's resources so that they can be used by other tasks and will then remove all evidence of the task ever having existed. Tasks may only destroy other tasks, not themselves. Tasks destroy themselves by merely exiting the task procedure.

This operation should only be performed when the full consequences of task destruction are known. This is particularly important in cases when the task that is being destroyed communicates with other tasks. If the destroyed task has a resource locked (with a semaphore, for example), there may be no way to reclaim that resource. Another potential risk is that the destroyed task may be the recipient of a message. Since the mailbox structure for that task will cease to exist after the task is destroyed, any subsequent message sent to that task will cause an 'Illegal Operation' error. It is important to note that except for the most simple multitasking designs, deadlock is likely to result from the task destruction. In most cases `TASKING` will detect system deadlock and terminate the application.

#### *3.1.1.3 SUSPEND(...)*

This routine causes a previously ready task to become blocked unconditionally. Once this has occurred, the only way that the suspended task can execute again is if another task resumes it (see section *3.1.1.4 RESUME(...)*). Note that suspending a task is distinctly different from destroying a task. A suspended task is essentially waiting for the conditions to exist which will cause another task to awaken it. A destroyed task can never be awakened (although it may be re-created).

#### *3.1.1.4 RESUME(...)*

This routine causes a previously suspended task to become ready to execute. After the task has been resumed, it is not necessarily the next task to execute, as is always the case when a task becomes unblocked. The task priorities of all ready tasks and the scheduling policy determine which task will be next to execute. Note that although typically this action is performed on a task that has been suspended, TASKING will allow any task to be resumed, regardless of the reason that the task is blocked. It is important to realize that resuming a task prematurely is likely to cause logical errors in the way that the tasks interact; deadlock may occur.

#### *3.1.1.5 CHANGE_PRIORITY(...)*

This routine allows the currently running task to change its priority. It is not possible for a task to change the priority of another task. After the task's priority has been changed, the normal scheduling arbitration takes place and a new task is chosen to execute. It is important to realize that designing a real-time system where a task changes its priority is extremely difficult. There may be isolated instances where it is useful, but in general, it should be avoided.

#### *3.1.1.6 WAIT_FOR_DELAY(...)*

This routine causes the currently running task to become blocked for a specified length of time. The time specified can be as long as thirty-two days with accuracy to one millisecond. When the time delay expires, the task becomes ready to execute. As is always the case when a task becomes unblocked, the task priorities of all ready tasks and the scheduling policy determine which task will be next to execute. Although it is possible to use this mechanism to create a task which performs some operation periodically, TASKING provides a more accurate means to do this (see sections *3.1.1.10 START_PERIODIC_EVENT(...)* and *3.1.1.11 STOP_PERIODIC_EVENT(...)*).

#### *3.1.1.7 SIGNAL_EVENT(...)*

This routine signals the event specified; if there are tasks waiting on the event, then one task is made ready (based on a non-prioritized, first-in-first-out algorithm). If there are no tasks waiting on the event, then the signal is saved, the next task to wait on the event will consume the signal. Only one signal is maintained no matter how many times the event is signaled. Note that sending multiple signals to the same event is not considered to be an error, but can lead to logical problems on the receiving task's subsequent reactivation.

#### *3.1.1.8 BROADCAST_EVENT(...)*

This routine is functionally identical to section *3.1.1.7 SIGNAL_EVENT(...)* with minor differences. If there are tasks waiting on the event when it is signaled, then they are all made ready to execute. If there are no tasks waiting on the event, then the signal is saved and the next single task to wait on the event will consume it.

#### *3.1.1.9 WAIT_ON_EVENT(...)*

This routine either causes the calling task to be suspended waiting for the event to be signaled or else consumes the signal already stored in the event. If the event has already occurred, then the calling task will not be suspended but instead will immediately become ready to execute. Although technically the task has not become unblocked, `TASKING` treats these two conditions identically. Therefore, the task priorities of all ready tasks and the scheduling policy determine which task will be next to execute.

#### *3.1.1.10 START_PERIODIC_EVENT(...)*

This routine causes `TASKING` to begin the periodic signaling of the event specified at the specified interval. This mechanism is most useful for creating periodic tasks. Since `TASKING` is responsible for signaling the event, the task is guaranteed that the signal will occur at the specified interval. It is important to note that the task priorities of all ready tasks and the scheduling policy will actually determine the exact execution period of the task. Requesting a periodic event is not the same as a real-time design which guarantees that the periodic execution will result.

#### *3.1.1.11 STOP_PERIODIC_EVENT(...)*

This routine causes `TASKING` to stop the periodic signaling of the event specified (see section *3.1.1.10 START_PERIODIC_EVENT(...)*). Note that the event is not destroyed by this operation; it may still be used by restarting the periodic event.

#### *3.1.1.12 SIGNAL_BINARY_SEMAPHORE(...)*

This routine performs an 'Up' operation on the specified binary semaphore. Since the ‘Up’ operation is not defined for a binary semaphore that is already set, `TASKING` will detect this condition as an error. If there are tasks waiting on the semaphore when it is signaled, then one is made ready to execute. If there are none waiting then the semaphore is set.

#### *3.1.1.13 WAIT_ON_BINARY_SEMAPHORE(...)*

This routine performs a 'Down' operation on the specified binary semaphore. If the binary semaphore is already zero, then the running task is suspended until the semaphore is signaled. If it is non-zero, then the semaphore is cleared and execution continues, based on the priorities of all ready tasks and the scheduling policy that is in effect.

#### *3.1.1.14 SIGNAL_SEMAPHORE(...)*

This routine performs an 'Up' operation on the specified "Dijkstra" semaphore. If there are tasks waiting on the semaphore when it is signaled, then one is made ready to execute. If there are no tasks waiting, then the semaphore is incremented. In either case, execution continues, based on the priorities of all ready tasks and the scheduling policy that is in effect.

#### *3.1.1.15 WAIT_ON_SEMAPHORE(...)*

This routine performs a 'Down' operation on the specified "Dijkstra" semaphore. If the "Dijkstra" semaphore is already zero, then the running task is suspended until the semaphore is signaled. If it is non-zero, then the semaphore is decremented and execution continues, based on the priorities of all ready tasks and the scheduling policy that is in effect.

#### *3.1.1.16 SEND_MESSAGE_TO(...)*

This routine sends the message (actually, the pointer to the message) to the specified task by placing it in the task's mailbox (the mailbox will automatically be created if necessary). Although the calling task cannot block when sending a message, `TASKING` re-evaluates which task should be executing, and execution continues based on the priorities of all ready tasks and the scheduling policy that is in effect. Typically, messages are dynamically allocated (using *getmem(...)*)

before being sent, the receiver is responsible for deallocation (using *freemem(...)*) after the message has been used.

### *3.1.1.17 WAIT_AND_RECEIVE_MESSAGE(...)*

This routine retrieves a message (actually, a pointer to a message) from the caller's mailbox (which is created if necessary). If there are no messages in the mailbox then the task is blocked waiting for a message to arrive. Because the caller is blocked until a message is available, the return pointer is guaranteed not to be nil. Typically, the receiving task will dispose of the message after it has been used. Messages are queued in first-in-first-out order into the task's mailbox (extreme caution should be used in the design of the message passing tasks to avoid unbounded priority inversion).

### *3.1.1.18 RECEIVE_MESSAGE(...)*

This routine is functionally identical to section *3.1.1.17 WAIT_AND_RECEIVE_MESSAGE(...)* with minor differences; if there are no messages present in the caller's mailbox at the time of the call, then the return value is nil. Therefore, the task cannot become blocked by calling this routine.

### *3.1.1.19 SIGNAL_CONDITION_VARIABLE(...)*

This routine signals the condition variable specified; if there is a task waiting on the condition variable then it is made ready. Multiple waiting tasks are handled in first-in-first-out order. If there are no tasks waiting on the condition variable when this routine is called, the signal is lost, (recall that condition variables have no storage capabilities).

### *3.1.1.20 WAIT_ON_CONDITION_VARIABLE(...)*

This routine is guaranteed to cause the calling task to become blocked. Because condition variables do not store signals, the task is guaranteed to be waiting on the condition variable as a result of calling this routine. As soon as another task signals the condition variable the waiting task will be made ready.

### *3.1.1.21 WAIT_ON_READKEY(...)*

This routine provides the means for an application task to perform I/O in a 'blocking' manner. This means that while there is no input from the keyboard, the calling task is suspended. As soon as input arrives from the keyboard, the calling task is made ready to execute. In addition, the ASCII code of the key that was pressed is returned to the caller. This in an example of interrupt driven multitasking. The task that calls this function (of which there can be only one) provides the interface between the rest of the application tasks and the outside world (in this case, the user). Typically, the task responsible for user input would send messages to other tasks or signal some type of inter-task communication construct as a result of receiving the user input. Note that `TASKING` cannot detect deadlock if this feature is used within the application. This is due to the very nature of an interrupt driven task. It could be awakened at any time by the keyboard; therefore deadlock is not detectable.

### *3.1.1.22 ENABLE_MOUSE_ACTIONS(...)*

This routine enables the specified MS-Mouse events to be received by the task which is (or will be) waiting to receive mouse actions. Essentially, this allows the task to communicate with `TASKING` the exact MS-Mouse features that it is using. This routine can be called at any time to change the mouse actions that are of interest to the task.

### *3.1.1.23 WAIT_AND_RECEIVE_MOUSE_ACTIONS(...)*

This routine receives MS-Mouse action parameters from the MS-Mouse driver and passes them to the waiting task. If none of the enabled mouse actions has occurred, then the task is blocked, waiting for one (or more) action to occur. This is another means of providing an application task the ability to perform I/O in a 'blocking' manner. Note that `TASKING` cannot detect deadlock if this feature is used within the application. This is due to the very nature of an interrupt driven task. It could be awakened at any time by the MS-Mouse action; therefore deadlock is not detectable.

### *3.1.1.24 LOCK_SCHEDULER(...)*

This routine allows a task to 'lock' the scheduler and prevent all further task rescheduling until it is unlocked (see section *3.1.1.25 UNLOCK_SCHEDULER(...)*).

One use of this routine is when multiple tasks need to write to the display, and it is desired that the output from each task occurs without the output from other tasks intermixing with it. Note that during the time that the scheduler is 'locked' there will be no preemptive or cooperative task rescheduling. The application should avoid locking the scheduler if possible. If that is not feasible or practical for the application, then the time spent with the scheduler locked should be an absolute minimum.

It is imperative that each call to lock the scheduler have a corresponding call to unlock it. Successive (nested) calls to lock the scheduler are allowed, but the application should ensure that each scheduler lock request has a corresponding call to unlock it.

#### *3.1.1.25 UNLOCK_SCHEDULER(...)*

This procedure 'unlocks' the scheduler and allows task rescheduling to continue based on the previously existing configuration. Note that the scheduler must previously have been locked, or an 'Illegal Operation' error will occur (see section *3.1.1.24 LOCK_SCHEDULER(...)*).

### 3.1.2 Configuration Parameters

`TASKING` provides a number of configurable parameters which help to match the RTOS to the real-time application. The following parameters must be initialized by the application before multitasking begins; therefore this initialization must occur in the application 'main' body or from the application INI file, which `TASKING` will read before the 'main' body executes. Once multitasking begins (i.e., when the 'main' body terminates), `TASKING` will ignore all changes to these parameters:

- Scheduling Policy - Task priorities can be either static throughout the execution of the application (default), or they can rotate on every context switch. Static priorities will ensure reliable, predictable system performance whereas rotating priorities will add a degree of uncertainty to the scheduling algorithm which may result in better performance for some applications. For example, an application where ‘fairness’ is an important system feature may benefit from rotating priorities.

- Priority Inheritance - Task priority inheritance will be either enabled or disabled. Priority inheritance is effective in solving the problem of unbounded priority inversion. Priority inversion can occur when low and high priority tasks share a common resource and there exist intermediate priority tasks. If a low priority task 'locks' a resource and a high priority task requests that resource (i.e., attempts to 'lock' the resource), it is possible for an intermediate priority task to indefinitely prevent the low priority task from 'unlocking' the resource. This essentially creates the condition where an intermediate priority task can prevent a high priority task from executing (see *Figure 8 - Unbounded Priority Inversion*).

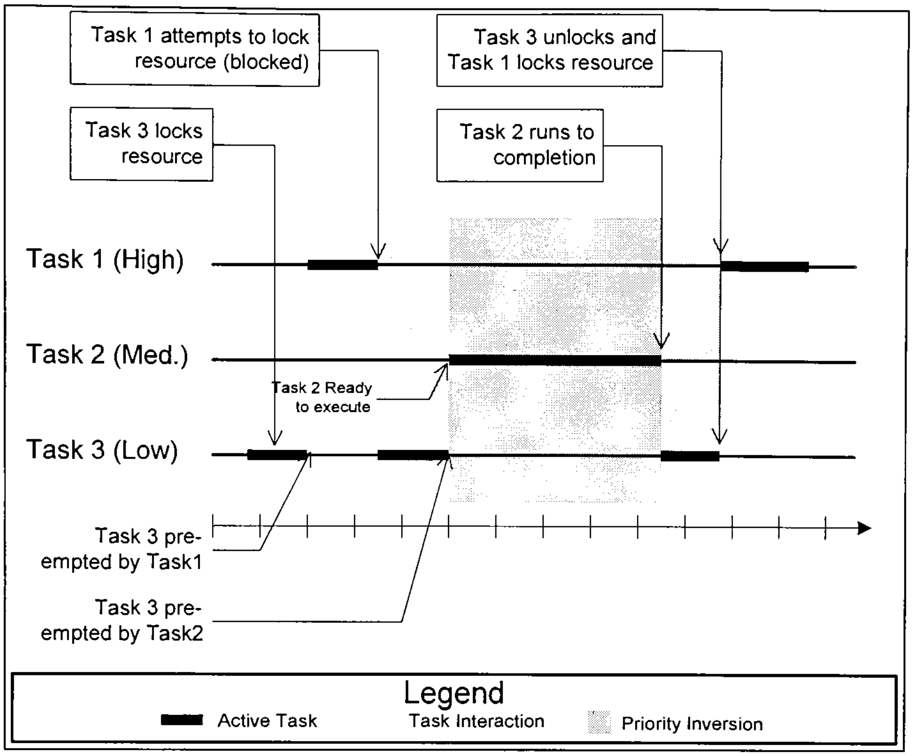



*Figure 8 - Unbounded Priority Inversion*

Priority inheritance prevents this situation from arising by temporarily raising the priority of a task with a locked resource to the level of the highest priority task attempting to lock that same resource for the duration of time that the lower priority task maintains control of the resource (see *Figure 9 - Priority Inheritance Protocol*).

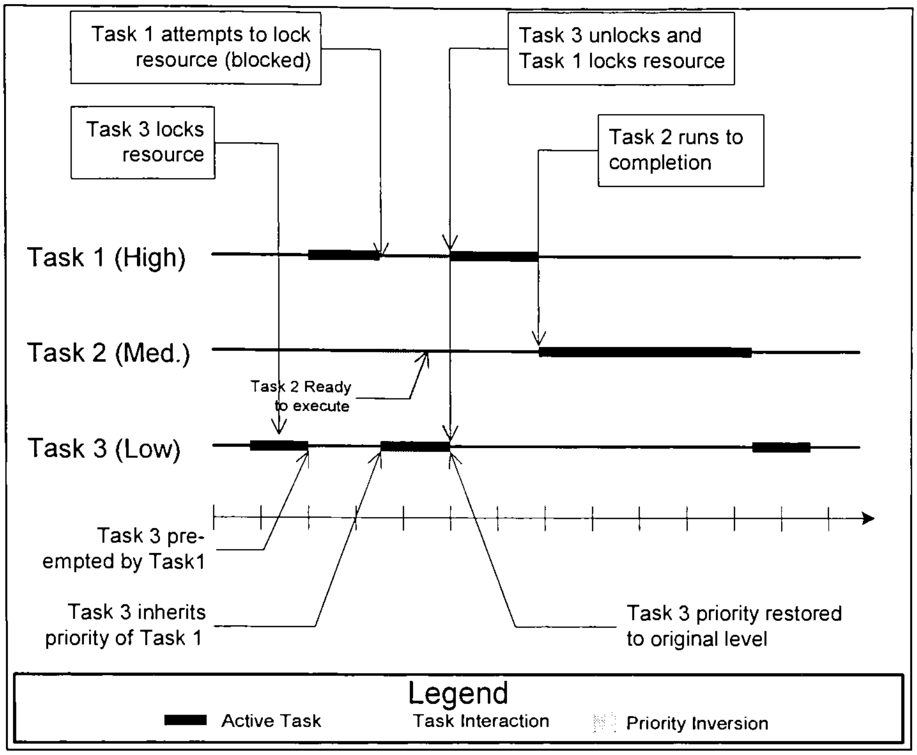



*Figure 9 - Priority Inheritance Protocol*

- Tasking Model - Both cooperative and preemptive multitasking models are supported. Cooperative multitasking only allows context switching when `TASKING` services are called. Cooperative multitasking requires that all tasks 'surrender' the CPU when they have completed their work (i.e., the tasks must cooperate by blocking to allow other tasks to execute, and there can be no continuously executing background tasks). This tasking model is often referred to as 'prioritized, run-to-completion'. In addition to allowing context switches when `TASKING` services are called, preemptive multitasking causes context switches periodically based on the timeslice parameter. The timeslice can range from 50 μSec. to 65 mSec. Preemptive multitasking removes the requirement that the tasks must cooperate in passing the CPU resource from task to task. When preemptive multitasking is selected, `TASKING` will 'give' the CPU to a task for a single timeslice. After that, the CPU will be 'taken' from that task and 'given' to another task which is chosen based on the relative task priorities of all tasks which are ready to execute and the scheduling policy that is in effect. Note that under this multitasking model, continuously executing background tasks are allowed.

### 3.1.3 RTOS Performance

The most important performance parameter of an RTOS is that of overhead imposed by the RTOS on the application. This is the amount of processing that is performed by the RTOS to achieve its goals which in no way contributes to the completion of the application requirements. `TASKING` performance in this area is shown in the following figures. Note that although these measurements were made on a 66 MHz Pentium processor, a different x86-class processor would produce similar results, i.e., a faster processor would show a decrease in overhead proportional to its speed relative to a 66 MHz Pentium.

In order to fully understand the performance merits of `TASKING` there are two things that must be explained: how the measurements were made and why the results turned out as they did.

The measurements were made using two special programs to generate the data and another program to produce a graph, see *Appendix C Support Software Source Code* [*TASKS-A.PAS*, *TASKS-B.PAS* and *TSK-BNCH.PAS*]. First, a baseline measurement was made using a program which performs a set of computations without using `TASKING` in any way [*TASKS-B.PAS*]. This produces a measurement of the best possible performance since any additional processing performed by `TASKING` results in performance which is degraded from this baseline measurement. Next, measurements are made using a program that performs the same set of computations but in this case `TASKING` is used to create multiple tasks each of which performs a subset of the computations [*TASKS-A.PAS*]. For these measurements `TASKING` is configured for preemptive multitasking with priority inheritance disabled (which is not relevant because none of the tasks interacts or shares resources). In addition, all tasks execute at the same priority level with static priorities. The point is to measure the impact of context switching on the time required for the application to perform all of the computations. This RTOS configuration achieves that goal by disabling or not using all other `TASKING` features. The only RTOS parameter that is allowed to vary is the target timeslice. The timeslice is varied from 1 mSec to 250 mSec; this range encompasses most reasonable timeslice values.

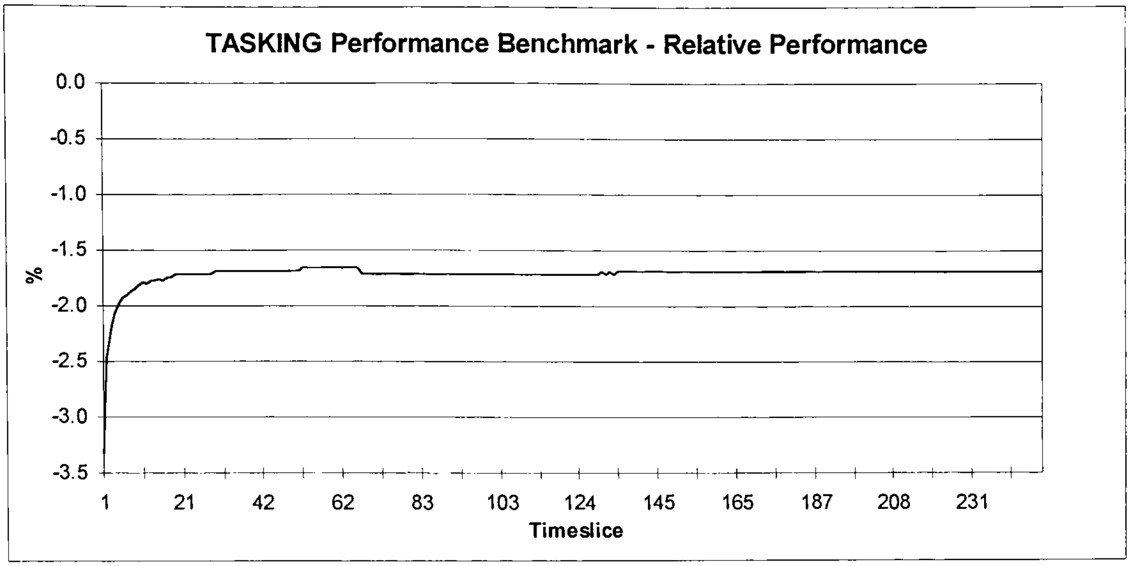



*Figure 10 - TASKING Benchmark (Relative Performance)*

The results from these measurements are shown above in the *Figure 10 - TASKING Benchmark (Relative Performance)*. This graph shows that `TASKING` imposes a minimum of approximately 1.7% performance penalty on the application. What is most interesting is that after the timeslice value exceeded approximately 20 mSec, the performance degradation remained relatively constant. This implies that ~1.7% performance degradation is the best that `TASKING` can achieve. As the graph clearly shows, the overhead increases significantly for small timeslice values. The reason for this is that smaller timeslice values result in more context switches. As the following graphs show (*Figure 11* and *Figure 12*), the context switch overhead is the dominate factor in `TASKING`'s performance only for small timeslice values. For larger timeslice values, the dominate (and constant) factor in `TASKING`'s performance is `TASKING`'s management of the system clock. That is, the handling of the system clock interrupt and the internal delay request checking performed during that interrupt service routine.

*Figure 11 - TASKING Benchmark (Context Switching Overhead)* below shows that context switching is not responsible for the performance degradation. Note that after the timeslice value exceeded approximately 20 mSec, the context switching overhead remained relatively constant at zero. It is only for small timeslice values that the context switch overhead is appreciable.

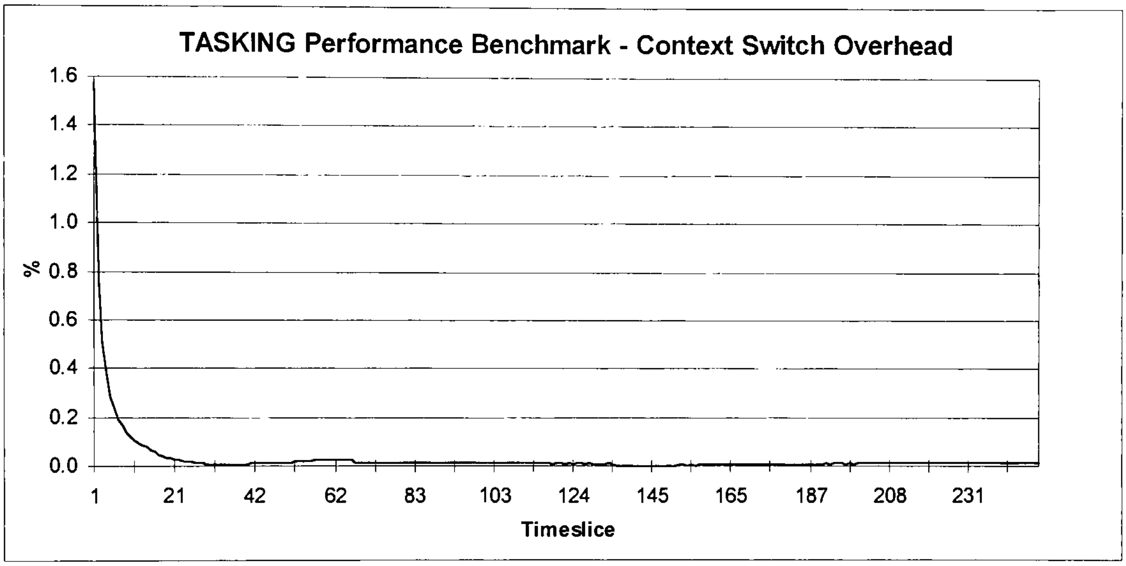



*Figure 11 TASKING Benchmark (Context Switching Overhead)*

As shown below in *Figure 12 - TASKING Benchmark (Context Switching Percentage)*, context switching time itself is negligible when compared to the timeslice value. The context switch time is only significant for small timeslice values. It is the presence of `TASKING` itself that impacts performance. This is a good indication that the performance degradation is relatively constant regardless of the application being used. This is a very important characteristic of a real-time operating system. Note that none of the values of *Figure 12* is zero, rather, some values are merely very small (or more likely beyond the measurement capabilities of the data collected).

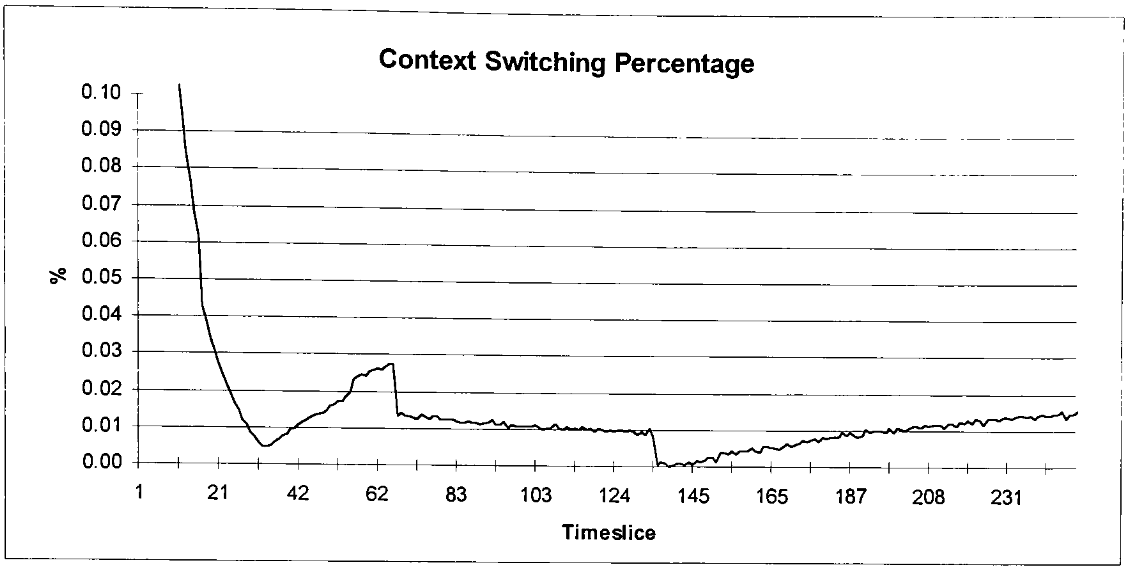



*Figure 12 TASKING Benchmark (Context Switching Percentage)*

Analysis of the context switching time of `TASKING` over the same range of timeslice values as in *Figure 12* yields a value which never exceeds approximately 37 µSec. The context switching time does vary somewhat over this range but never beyond the accuracy of the measurements. That is, to within the accuracy possible with the data collected, the context switching time of `TASKING` remains relatively constant over this range of timeslice values. These two context switching time properties (relatively constant and guaranteed upper bound) are probably the most important RTOS characteristics for real-time applications. A constant context switch time allows accurate, verifiable timing analysis to be performed and an upper limit on the context switch time allows worse-case timing analysis to be performed. No system is actually 'real-time' if the context switch time varies excessively or is unbounded.

The performance degradation which occurs when using `TASKING` is a result of the monitoring of the system clock that `TASKING` must do in order to achieve preemptive multitasking. `TASKING` must change the clock interrupt period of the main system clock hardware of the PC to a minimum of 1.000 kHz (1.000 mSec period) for timeslice values of 1 mSec or more. Because of this and the fact that the PC clock is originally set to 18.20 Hz (54.93 mSec period), TASKING performance is best at multiples of approximately 55 mSec. This is illustrated below, (see *Figure 13 - TASKING Benchmark (System Clock Impact)*). The graph shows the excess processing that TASKING must perform in order to simulate the MS-DOS clock timer. TASKING must determine the proper time that the original MS-DOS clock timer interrupt should be called so that MS-DOS will continue to keep the proper 'clock time'. Since MS-DOS assumes that it's timer interrupt handler will be activated once every 54.93 mSec, in order for MS-DOS to keep proper time, TASKING must ensure that it does. It is this simulated interrupt which tends to add modulated irregularities to the TASKING performance curves. If MS-DOS was not used by the application, TASKING would not need to simulate the interrupt. That would stabilize the performance curves for TASKING but would not allow the application to use any MS-DOS services (such as screen and/or disk I/O). Since it is not practical to include all of those services into TASKING, the only alternative is to understand the performance measurement limitations and factor those limitations into the interpretation of the performance data.

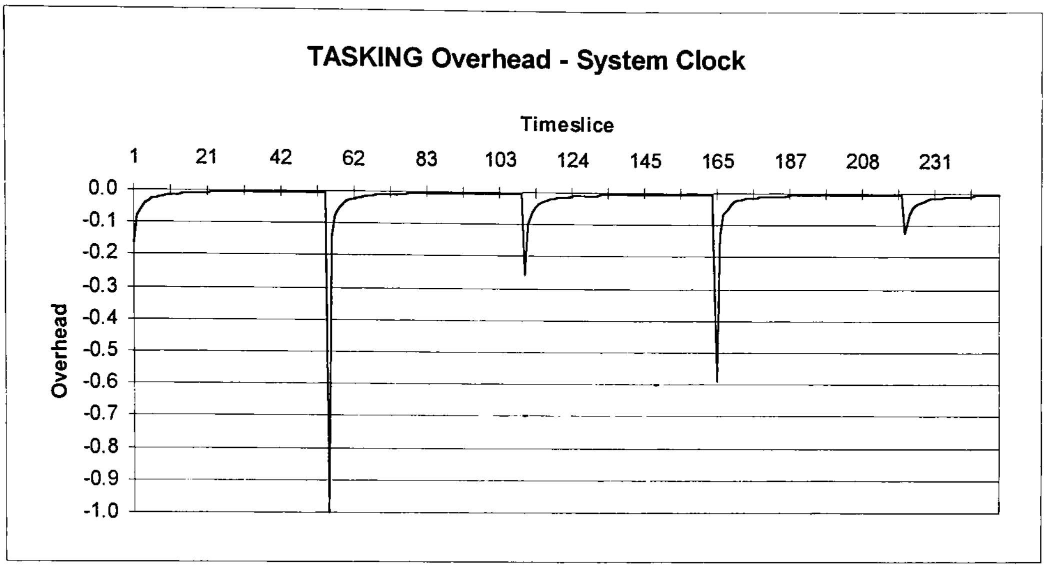



*Figure 13 TASKING Benchmark (System Clock Impact)*

In general, `TASKING` provides reliable, consistent performance. The only concern is that the application is in fact degraded by approximately 1.7%. Although this value is not excessive, it is also not negligible. Given that all of `TASKING` is written in Pascal and that there is no appreciable assembly language, these results are not surprising and certainly not unreasonable.

### *3.2 Genetic Algorithm*

The GA is constructed in the 'standard' genetic algorithm manner (see *Figure 14 - Genetic Algorithm Process Flow*). The only exception to this is that the information within the GA will flow through data files created on the computer hard drive. This is necessary because the GA must be completely suspended in order to run the `TASKING` programs which are used to evaluate the individuals of the population. In order to achieve accurate and reproducible results, the GA must be run from a minimum PC configuration, i.e., no Terminate and Stay Resident (TSR) programs loaded. In addition, the GA must be run in a

pure MS-DOS® environment, i.e., not from within an MS-DOS® shell under MS- - -
Windows®.

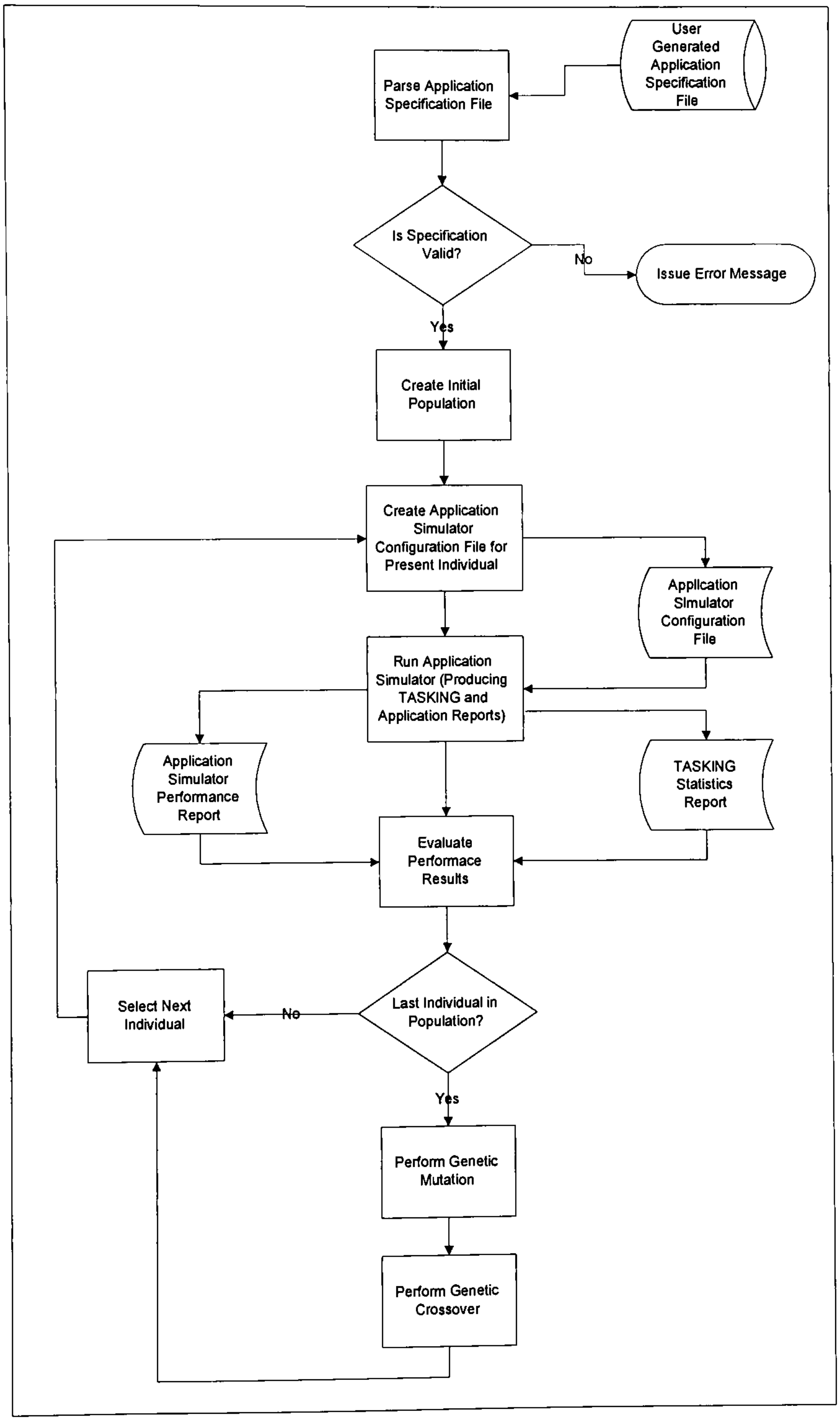



*Figure 14 - Genetic Algorithm Process Flow*

The most important aspect of the GA design is in the choice and/or design of the genetic operators: mutate(...) and crossover(...). These operators will affect the ability of the GA to converge to a solution, the rate of convergence and the probability of false convergence. Most of the research associated with this thesis has involved these operators. There is no shortage of opinions regarding the 'best' universal operators, but there has been nothing published regarding operators specific to this problem domain. A certain amount of experimentation is required in determining viable operators. The following sections detail the appropriate operators for this application and also describe the desirable characteristics of genetic operators.

Before the genetic operators can be designed, the genetic representation that will be used for the problem must be determined. The following section describes the genetic representation used for the GA.

### 3.2.1 Genetic Representation

The genetic representation for the RTOS is representational in nature, that is, it is not a pure binary representation. Instead the representation will closely mirror the configuration parameters of the RTOS, see Section *3.1.2 Configuration Parameters*. The following table lists the genes that comprise the genotype used for the RTOS configuration:

| Gene | Possible Values |
|---|---|
| Tasking Model | Cooperative, Preemptive |
| Target Timeslice | 50 - 65,535 μSec |
| Priority Inheritance Enabled | True, False |
| Priority Allocation | Static, Rotating |
| Initial Priority Assignment | Constant, Random, Rate Monotonic, Deadline Monotonic, Workload Monotonic |

*Table 3 - RTOS Genotype*

Since the GA will not use a binary representation, it is important to note that there are five genes in the genotype above. All five are considered to be equal even though the actual representation length (bit length) of each gene is different. For the purposes of genetic mutation and crossover, they are considered to be of equal length. Therefore, each of the five has equal probabilities whenever a gene is to be operated upon in a random manner. This is considerably different from a binary representation GA where each bit is considered equal, and the length of the group of bits that comprise a gene has a significant impact on the relative probability of that gene becoming involved in mutation and/or crossover.

### 3.2.2 Mutation

The genetic mutation(...) operator for this GA simply mutates (which is a gene specific operation) all genes of the population with equal probability $P_M$. For example, suppose $P_M$=4% (i.e., 0.04), the number of individuals $N$=225, and the number of genes per genotype $M$=5; then for a typical generation there would be $P_M \bullet N \bullet M = 0.04 \bullet 225 \bullet 5 = 45$ genetic mutations, on average.

For the representation used here, the mutation to be performed is a function of exactly which gene is to be mutated. As is the case for all representational genotypes, this genetic algorithm mutation operator randomly selects a value from the allowable range for the gene to be mutated. In this way all genotypes are guaranteed to be made up of valid genes. This ensures that all individuals represent valid solutions to the problem at hand even if they have been involved in a genetic mutation. The allowable values for each gene are defined in *Table 3 - RTOS Genotype* above.

### 3.2.3 Crossover

The genetic crossover(...) operator for this GA is implemented using single-point crossover. First, two parents are selected at random from all above average individuals (i.e., individuals that have a fitness value greater than the average fitness for all individuals of the population). Then a crossover point is randomly selected. The crossover is performed as all genes up to the crossover point are taken from the first parent, and the remaining genes are taken from the second parent. The newly created individual replaces a below-average individual. Since there are five genes in the genotype, the crossover point is from one to four. This ensures that at least one gene is taken from each parent.

The results achieved when using single point crossover are somewhat dependent upon the ordering of the genes. Since the genes that are taken from the parents are contiguous, it is useful to group related genes together. For this application, the only genes that are related are the Tasking Model and Target Timeslice genes. As indicated in *Table 3 - RTOS Genotype* above, these two genes are adjacent within the genotype.

## 3.3 Problem Specification

The real-time application must be specified in the most flexible manner possible in order to support an arbitrary real-time design. In order to do that, the GA reads the design information from a 'standard' data file. This approach allows the real-time application specification to be created independently of the GA. The following sections describe which aspects of the real-time application must be specified and how the specification must be created so that the GA can correctly process (parse) the information.

### 3.3.1 Specification Parameters

In order to completely specify the real-time application, the user will have to provide details about the characteristics of the tasks that make up the application. The only aspects of the application design that are germane to the GA are those that involve the tasks themselves and the task interaction. The details of what operations the tasks perform are irrelevant. The necessary task characteristics are as follows:

- Task Name - This will be a text string that will be used when performance results are reported and when communication is performed with other tasks.
- Period - This will be the amount of time between successively beginning executions of a periodic task. A value of zero is used to indicate that the task is non-periodic. Non-periodic tasks will only execute once. See *Figure 15 - Task Execution Profile*, $T_R$.

- Deadline - This will be the amount of time between when a task begins execution and when it must reach a critical point in its execution. See *Figure 15 - Task Execution Profile*, $T_D$.
- Deadline Hardness - This parameter will indicate the degree to which the previously specified deadline is 'hard'. A linear scale between 1 (soft) and 10 (hard) is used.
- Workload - This will be the amount of time between when a task begins executing and when it completes. A value of zero is not allowed. See *Figure 15 - Task Execution Profile*, $T_W$.

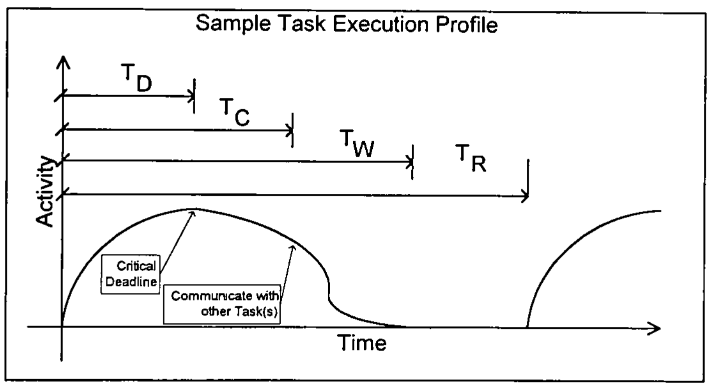



*Figure 15 Task Execution Profile*

In addition, the user must specify the inter-task communication aspects of the real-time system. The following communication characteristics may also be specified (some applications and/or tasks will not require these features):

- Task Name - This will be the name of the task that is to be the recipient of the communication.

- Communication Type - This will be the `TASKING` inter-task communication type that is to be used (Message, Semaphore, Binary Semaphore, Event or Condition Variable).
- Communication Time - This will be the amount of time from when the task begins executing to when the communication should be performed with the other task. See *Figure 15 - Task Execution Profile*, $T_C$.

Up to two pairs of the above communication parameters can be specified. This will allow most complex inter-task communication architectures to be fully described.

### 3.3.2 Specification File Format

Since it is not the goal of this thesis to develop an application for creating the real-time application specification, a standard PC spreadsheet program will be used to create the file used to specify the design of the real-time system. Any spreadsheet can be used as long as the file is saved in the standard 'Comma Separated Value' format (all commercial spreadsheet applications support this format, as well as most commercial database applications). Text fields will be enclosed in double quotes; integer fields will be ASCII text but will not be enclosed in quotes; real numbers are not necessary and will not be supported.

Each line of the file will represent a single task specification, the length of which is limited to 255 characters. The order of the fields within each line of the spreadsheet must as specified in *Table 4 - Real-Time Application Specification* below.

| Task Name | Period | Deadline | Hardness | Workload | Receive Resource #1 | Receive Resource Type #1 | Receive Resource Time #1 | Signal Resource #1 | Signal Resource Type #1 | Signal Resource Time #1 | Receive Resource #2 | Receive Resource Type #2 | Receive Resource Time #2 | Signal Resource #2 | Signal Resource Type #2 | Signal Resource Time #2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Text | mSec | mSec | | mSec | Text | Semaphore<br>Cond_Var<br>Event<br>Message<br>BSemaphore | mSec | Text | Semaphore<br>Cond_Var<br>Event<br>Message<br>BSemaphore | mSec | Text | Semaphore<br>Cond_Var<br>Event<br>Message<br>BSemaphore | mSec | Text | Semaphore<br>Cond_Var<br>Event<br>Message<br>BSemaphore | mSec |
| 25 Char | 0-1000 | 0-1000 | 1..10 | 0-1000 | 25 Char | 10 Char | 0-1000 | 25 Char | 10 Char | 0-1000 | 25 Char | 10 Char | 0-1000 | 25 Char | 10 Char | 0-1000 |

*Table 4 - Real-Time Application Specification*

# 4. Evolution Tool Set

This thesis will result in a set of PC-based tools which will compute near optimal RTOS parameters for any real-time application. The tools will provide a means of creating an application specification and performing the GA based analysis, using a custom RTOS - `TASKING`. Performing the analysis will include real-time graphical display of the progress of the GA 'population' and a report with the recommended near optimum RTOS configuration.

The set of project deliverables is as follows (detailed in the following sections):

- Custom RTOS (`TASKING`) to be used in evaluating RTOS parameters.
- Tool to analyze and evaluate an arbitrary real-time application using a specific set of RTOS parameters.
- Genetic Algorithm based tools to evolve a near optimal RTOS configuration for the real-time application.
- Set of real-time application specifications and analysis to verify proper operation of the tool set (i.e., problems with known solutions).

## *4.1 Custom RTOS (TASKING)*

The RTOS used to evaluate the real-time application has already been described in detail (see section *3.1 Real-Time Operating System (TASKING)*). The aspect of `TASKING` that has not been described is how it is instrumented to provide performance information. `TASKING` is capable of capturing and reporting performance information about three different aspects of the execution of an application. These aspects are:

- Hardware Interrupts - includes information about which interrupts were serviced, how many times they were serviced and how frequently they were serviced (report filename: `IRQ.RPT`).
- Software Interrupts - includes information about which MS-DOS services were used by the application, how many times they were called and how frequently they were called (report filename: `SERVICES.RPT`).
- Task Statistics - includes information about the CPU and stack utilization of each task as well as the percentage of periodic events that failed to be signaled at the proper time (report filename: `TASKING.RPT`).

As it relates to the evaluation of an RTOS configuration for a real-time application, the task statistics information is the most useful. The following figure shows a sample report for the Dining Philosophers Problem (*Figure 16 - Sample TASKING.RPT File*).

```
 1: Preemptive Multitasking Statistical Information
 2:   -Priority Inheritance Enabled
 3:   -Static Priorities
 4:
 5:   Task Activity:
 6:     Context Switches            2002 (~640 per second)
 7:       Cooperative                361 (~115 per second)
 8:       Preemptive                1641 (~524 per second)
 9:     Target Time Slice       =  1500 usec
10:     Achieved Time Slice     = ~1563 usec
11:     Available CPU Bandwidth =   0.000 %
12:     Periodic Event Faults      29.941 %
13:     Tasks at Termination:
14:          Task State     Task              Stack Size  Stack Used  Stack Used  CPU Used
15:       (Delayed Until)    ID    Priority    (words)     (words)       (%)         (%)
16:       ---------------  ----  ---------  ----------  ----------  ----------  --------
17:          Ready            1     9991        4000         110         2.8       58.4
18:          Ready            2     9986        4000         110         2.8       39.5
19:          Ready            3     9966        4000          31         0.8        0.0
20:        Running            4    10000        2000         687        34.3        2.1
21:
22:   Maximum percent of stack used: 34.3%
23:
24:   Heap Information (bytes):
25:     Total Available:               100,000
26:     Used for Application Stacks:    28,000
27:     Available to Application:       72,000
28:
29:   Execution Time:
30:     Effective =    3.070 seconds
31:     Absolute  =    3.13 seconds
```

*Figure 16 - Sample TASKING.RPT File*

The parameters of interest are on lines 11 and 12. The CPU utilization is determined by `TASKING` by recording the task that is executing when system timer interrupts occur. Since `TASKING` maintains a 'null task' which executes only when there are no application tasks which are ready to execute, `TASKING` can very easily (and accurately) determine the percentage of available CPU bandwidth. The percentage of periodic events that are missed by the application is also very easily determined. Since `TASKING` is in control of the system clock, whenever a periodic event is signaled the state of the event is examined. If the event is already signaled then the application will miss that event (i.e., the application was not able to finish its execution during the previous time period). These two parameters represent `TASKING`'s contribution to the RTOS evaluation by providing information usually available only to the operating system to the genetic algorithm.

## 4.2 Real-Time Application (RTOS-APP)

The real-time application produces a report which includes a measure of timeliness for various aspects of the application timing requirements. The real-time application monitors the system clock (provided by `TASKING`) to determine if the task's deadline and periodic execution occurred at the appropriate time. Since the measurements are made by the tasks themselves, the results are very accurate. `TASKING` is capable of signaling a periodic event at the correct time but the RTOS configuration and application design dictate whether the task will actually receive the signal at the correct time or not. The following figure shows a sample real-time application report file:

```
 1: Individual Task Timeliness
 2:     Task Name: TASK #1
 3:       Periodicity: 213/213 (100.00%)
 4:       Deadline:    14/214 (6.54%)
 5:
 6:     Task Name: TASK #2
 7:       Periodicity: 141/141 (100.00%)
 8:       Deadline:    68/141 (48.23%)
 9:
10:     Task Name: TASK #3
11:       Periodicity: 0/0 (0.00%)
12:       Deadline:    0/0 (0.00%)
13:
14:
15:   Average Task Timeliness
16:       Periodicity: 66.67%
17:       Deadline:    13.51%
18:
19:   Application Timeliness = 40.09%
```

*Figure 17 - Sample RTOS-APP.RPT File*

The parameter of interest is shown on line 19, it is this value that is used to contribute to the evaluation of the RTOS configuration for the specific real-time application.

## 4.3 Genetic Algorithm (GRTOS-GA)

The GA performs the operations which result in evolving a near real-time RTOS configuration for the real-time application. The evaluation of the RTOS configuration is performed by parsing the `TASKING.RPT` and `RTOS-APP.RPT` files and combining the results. At the completion of the evolution process, the

GA produces a report (`GA_XX.RPT`, where XX is a sequence number) which indicates what the top five RTOS configurations were. The startup screen for the tools is shown below (*Figure 18 - GRTOS-GA Startup Screen*). The tool accepts the following commands from the user (the underlined letter indicates the key required to activate the command, also note that the tool only responds to user commands at generation boundaries) :

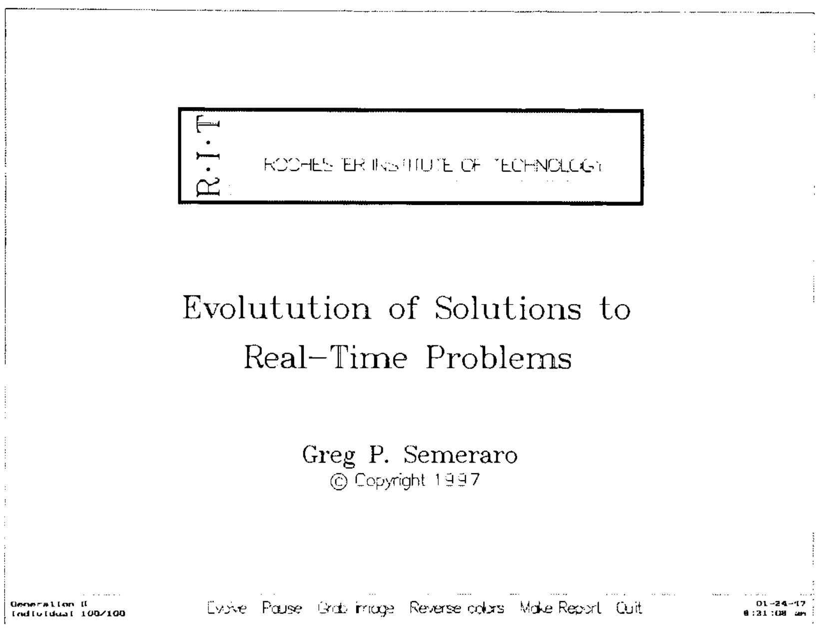



*Figure 18 - GRTOS-GA Startup Screen*

- Evolve - Starts (and stops) the evolution process. After receiving this command the tool will switch between the startup screen (see *Figure 18*) and begin displaying the evolution statistics graphs.

- Pause - Pauses (and restarts) the evolution process at the next generation boundary.

- Grab image - Grabs the currently displayed image (in MS Windows Bit Map format) to the file `IMAGE_XX.BMP`, where XX is a sequence number. The file is saved in the directory specified in the INI file.
- Reverse colors - Reverses the color scheme of the display. Both the normal and reversed color schemes can be specified in the INI file.
- Make report - Makes a report file (`GA_XX.RPT`, where XX is a sequence number) in the same directory as the tool executable file. The report generated represents the current genetic algorithm statistics. Note that a report file is always generated when the program is terminated.
- Quit - Terminates the evolution tool (and produces a report file).

# 5. Test Suite Description

In order to evaluate the tool set it is necessary to use the tools to solve increasingly complex problems. As these problems are solved, confidence in the ability of the tools to provide correct solutions will increase. The real-time applications that were used to evaluate the tool set are described in the following sections and are of increasing complexity. Obviously, the ability of the tool to provide correct solutions to problems with known solutions is the first step in validating its applicability to arbitrary real-time applications.

A partial formal analysis of these problems is also possible and provides insight into the schedulability of the tasks that make up the problem. This analysis does not help in determining the solution but does determine if a solution exists which can satisfy the problem requirements. This schedulability test is defined in *Equation 5* below [Ref. 1, pg. 28]. If the inequality is met, then there is guaranteed to exist an RTOS configuration which will satisfy the problem requirements. If the inequality is not met then it is guaranteed that an RTOS configuration that satisfies the problem requirements does not exists.

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \le N \cdot (2^{\frac{1}{N}} - 1)$$

Where: $N$ is the number of tasks

$C_i$ is the execution time of task $i$

$T_i$ is the period of task $i$

*Equation 5 - Task Schedulability*

## 5.1 Verification of Capabilities

The first step in verifying that the GA is capable of finding a reasonable solution to a real-time application is to use the GA to solve a problem with a known solution. The least complicated class of real-time applications is the class of purely independent, periodic tasks. The following figure (*Figure 19 - Real-Time Problem #1*) completely describes one such real-time application.

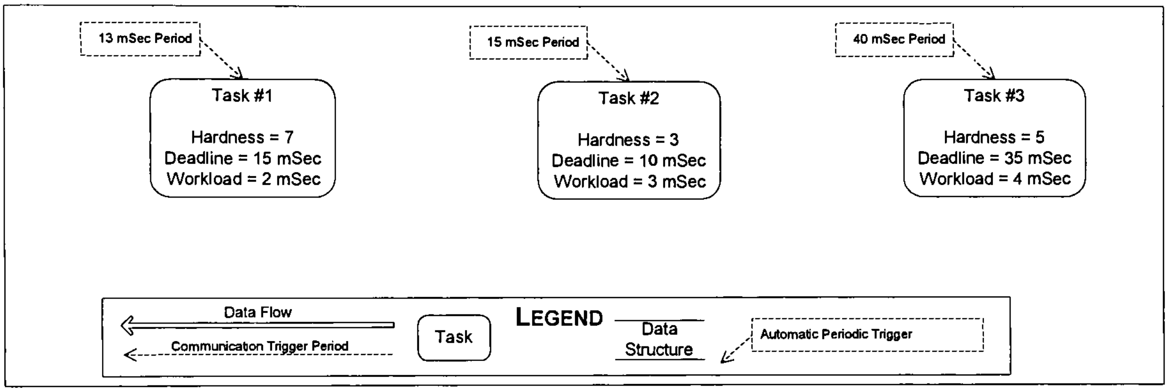



*Figure 19 - Real-Time Problem #1*

Note that none of the tasks interact in any way, this is an example of the simplest class of real-time problems. Applying the schedulability test of *Equation 5* to this problem yields the following results:

$$\frac{2}{13}+\frac{3}{15}+\frac{4}{40} \leq 3\cdot(2^{\frac{1}{3}}-1)$$
$$45.4\% \leq 77.9\% \text{ - Passed}$$

*Equation 6 - Schedulability, Problem #1*

From this analysis it is clear that under all conditions, the tasks are schedulable, i.e., there is guaranteed to exist a task schedule (RTOS configuration) which will allow all of the tasks to meet all requirements at all times. The results from running `GRTOS-GA.EXE` are as follows:

| Rank | Fitness | Tasking Model | Timeslice (µSec) | Priority Inheritance | Priority Allocation | Priority Assignment |
|---|---|---|---|---|---|---|
| 1 | 84.24 | Preemptive | 1273 | Enabled | Rotating | Rate Monotonic |
| 2 | 84.12 | Preemptive | 1227 | Enabled | Static | Deadline Monotonic |
| 3 | 83.74 | Preemptive | 1519 | Disabled | Rotating | Random (674788598) |
| 4 | 82.73 | Preemptive | 2560 | Enabled | Rotating | Rate Monotonic |
| 5 | 52.26 | Cooperative | N/A (34282) | Disabled | Rotating | Rate Monotonic |

*Table 5 -* `GRTOS-GA.EXE` *Results, Problem #1*

These results are not surprising, for purely independent tasks, an initial priority assignment using a rate monotonic algorithm produces an optimal task scheduling policy. Rate monotonic assignment theory does not provide any insight into the proper choice for the other RTOS configuration parameters. The results above (see *Table 5 _ GRTOS-GA.EXE Results, Problem* #1) clearly show that a preemptive multitasking environment using priority inheritance with a timeslice of approximately 1.2 mSec provides the best results. It would be imprudent to read anything more into the results, recall that this tool provides guidelines for configuring the RTOS for the application. For example it appears that rotating priorities may provide some added benefit, although this result may not be conclusive.

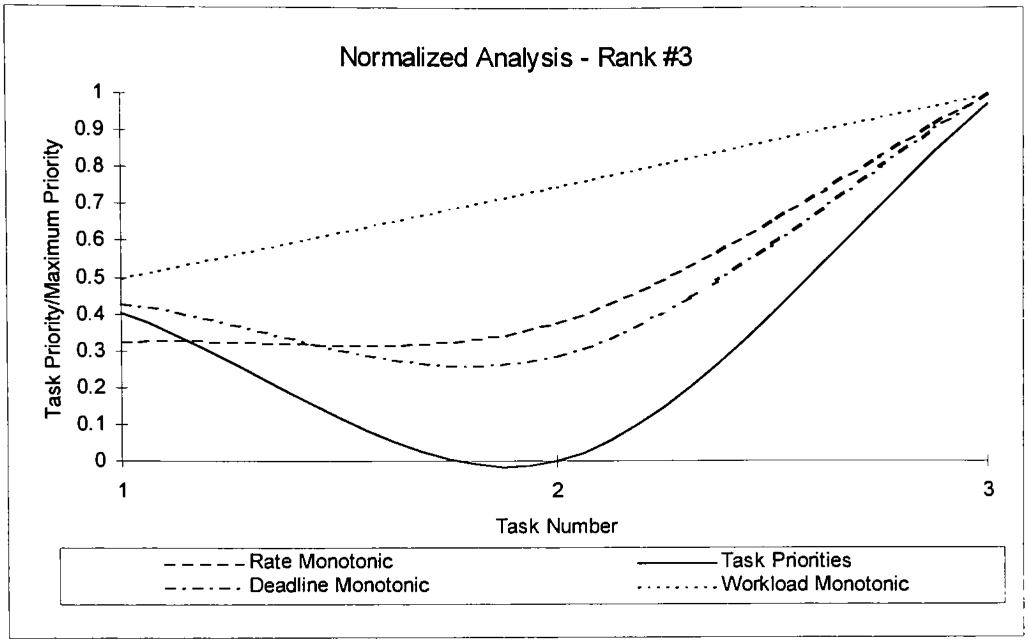



*Figure 20 - Problem #1, 'Random' Assignment Analysis Rank #3*

Analyzing the priorities used in the random assignment above (ranked 3rd) shows that this random seed actually results in a (albeit non-proportional) deadline monotonic assignment (see *Figure 20 Problem #1, 'Random' Assignment Analysis - Rank #3*, above). It is evident from the fact that the 'Task Priorities' and 'Deadline Monotonic' curves possess the same general characteristics. This supports the conclusion that this 'random' task priority assignment produces priority assignments which are similar to a deadline monotonic assignment scheme.

## 5.2 Verification of Performance

The next step in the verification process is to use the GA to find a solution to a real-time application for which a solution can be predicted but not proven. This type of problem was created by adding task dependencies to the basic application. The advantage of using the GA to solve this type of problem is that it is relatively straight forward to see that the GA arrived at a reasonable solution.

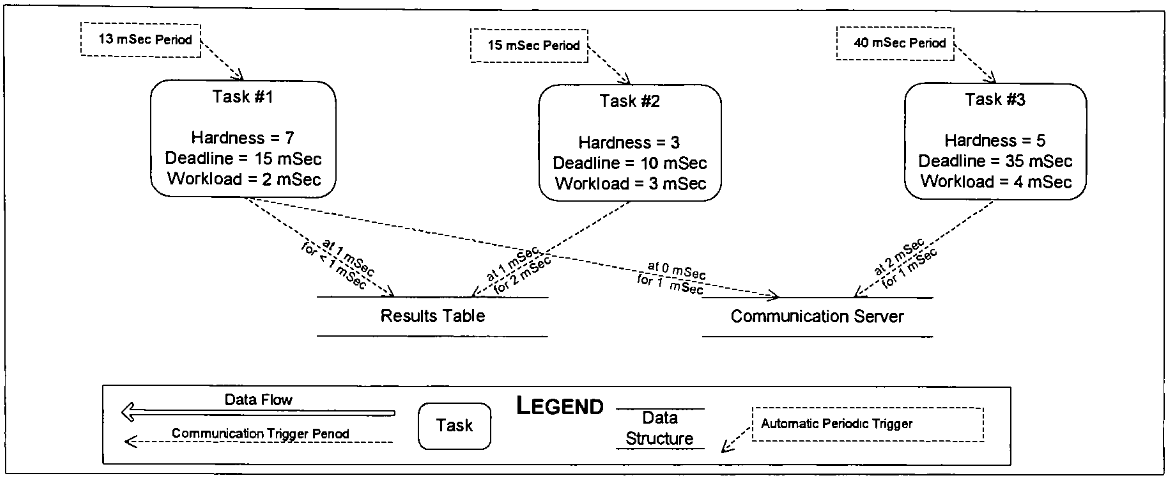



*Figure 21 - Real-Time Problem #2*

In the cases where task blocking can occur (as in this case) the schedulability test of *Equation 5* must be extended to include blocking time in addition to execution time (in other words, $C_i$ is replaced by $C_i+B_i$, where $B_i$ is the total amount of time that all other tasks can lock resources used by the task). This analysis represents the worst case task schedule. The actual task schedule used may eliminate some (or all) of the blocking times. The following analysis shows the upper and lower bounds for the task schedulability:

$$\frac{2}{13}+\frac{3}{15}+\frac{4}{40}\leq 3\cdot(2^{\frac{1}{3}}-1)$$

$$\text{Lower Bound } = 45.4\% \leq 77.9\% \text{ - Passed}$$

$$\frac{2+3}{13}+\frac{3+1}{15}+\frac{4+1}{40}\leq 3\cdot(2^{\frac{1}{3}}-1)$$

$$\text{Upper Bound } = 77.6\% \leq 77.9\% \text{ - Passed}$$

*Equation 7 - Schedulability, Problem #2*

It is obvious that even under the worst case task scheduling (when all blocking time is included), the tasks are still schedulable, albeit barely. The results from running `GRTOS-GA.EXE` are as follows:

| Rank | Fitness | Tasking Model | Timeslice (μSec) | Priority Inheritance | Priority Allocation | Priority Assignment |
|---|---|---|---|---|---|---|
| 1 | 76.68 | Preemptive | 1205 | Enabled | Rotating | Rate Monotonic |
| 2 | 75.81 | Preemptive | 1245 | Enabled | Rotating | Rate Monotonic |
| 3 | 74.41 | Preemptive | 1245 | Disabled | Rotating | Rate Monotonic |
| 4 | 72.78 | Preemptive | 1245 | Enabled | Rotating | Rate Monotonic |
| 5 | 54.67 | Cooperative | N/A (33299) | Enabled | Static | Rate Monotonic |

*Table 6 GRTOS-GA.EXE Results, Problem #2*

Again the results are not surprising, given that a schedule was possible under all conditions it is reasonable to expect that a rate monotonic assignment would perform best. It is interesting to note that the above results (see *Table 6 _ GRTOS-GA.EXE Results, Problem* #2) clearly show results that are very similar to the previous problem (see *Table 5 _ GRTOS-GA.EXE Results, Problem* #1). That is, a preemptive multitasking environment using priority inheritance with a timeslice of approximately 1.2 mSec provides the best results. These results point more strongly to the fact that rotating priorities may provide some added benefit, in fact that conclusion can clearly be made.

## 5.3 Test of Capabilities

Now that the GA has been verified to operate correctly, the next test is a simple problem for which there is no obvious solution. This represents the first opportunity to use the GA to find a solution for a real-time problem when the only other means of analysis is conjecture. The problem has been created by further augmenting the basic problem to include sporadic tasks modeled as periodic tasks.

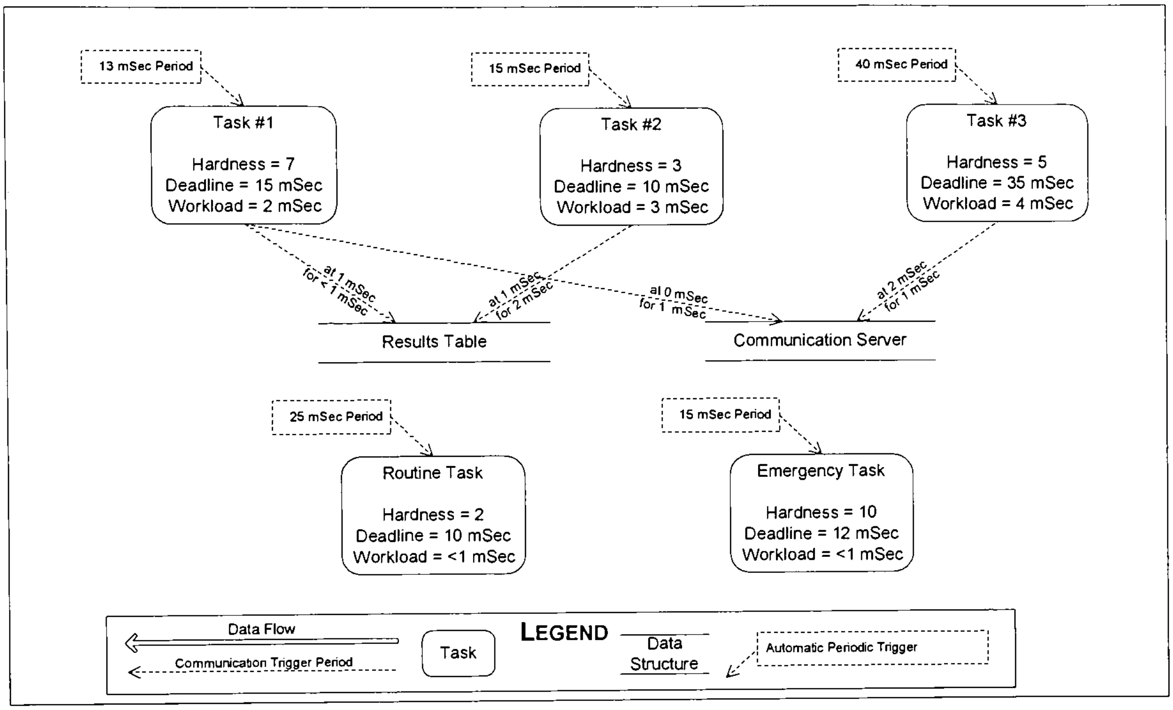



*Figure 22 Real-Time Problem #3*

As in the previous case, task blocking can occur, therefore the schedulability test of *Equation 5* is again extended to include blocking time in addition to execution time. The following shows the analysis:

$$\frac{2}{13}+\frac{3}{15}+\frac{4}{40}+\frac{1}{25}+\frac{1}{15}\leq 5\cdot(2^{\frac{1}{5}}-1)$$

Lower Bound $= 56.1\% \leq 74.3\%$ - Passed

$$\frac{2+3}{13}+\frac{3+1}{15}+\frac{4+1}{40}+\frac{1+0}{25}+\frac{1+0}{15}\leq 5\cdot(2^{\frac{1}{5}}-1)$$

Upper Bound $= 88.3\% \leq 74.3\%$ - Failed

*Equation 8 - Schedulability, Problem #3*

As this analysis clearly shows, this problem (i.e. the set of tasks) may or may not be schedulable. Whether the tasks are schedulable or not is a function of the RTOS configuration itself. This is evident from the fact that the lower bound on the task schedulability is below the threshold and the upper bound is above it. This means that under the worst possible scenario (i.e., RTOS configuration) the tasks are not schedulable. The obvious conclusion is that it is likely that an RTOS configuration exists which results in some level of blocking which yields a CPU utilization which is below the allowable threshold.

The results from running `GRTOS-GA.EXE` are as follows:

| Rank | Fitness | Tasking Model | Timeslice (μSec) | Priority Inheritance | Priority Allocation | Priority Assignment |
|---|---|---|---|---|---|---|
| 1 | 71.86 | Preemptive | 1005 | Enabled | Rotating | Deadline Monotonic |
| 2 | 71.01 | Cooperative | N/A (26878) | Disabled | Static | Random (-1704415732) |
| 3 | 69.85 | Cooperative | N/A (24732) | Enabled | Static | Random (1366122493) |
| 4 | 62.83 | Preemptive | 1879 | Enabled | Rotating | Deadline Monotonic |
| 5 | 61.61 | Preemptive | 44703 | Disabled | Static | Random (712905428) |

*Table 7 - GRTOS-GA.EXE Results, Problem #3*

Analyzing the characteristics of the 'random' priority assignments for this real-time application is very difficult. As the following graphs show (*Figure 23*, *Figure 24* and *Figure 25*), the task assignments that result from the random assignment algorithms cannot accurately be categorized.

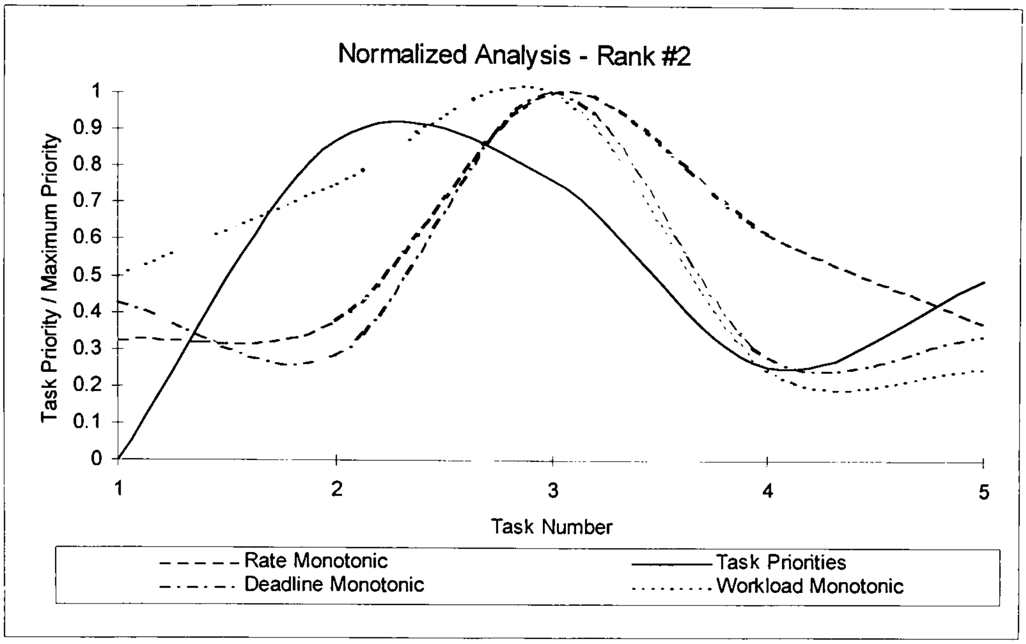



*Figure 23 Problem #3, 'Random' Assignment Analysis - Rank #2*

As can be clearly seen, the 'Task Priorities' curve does not resemble any of the other curves. This random task priority assignment does not share any general characteristics with the 'standard' task priority assignment algorithms. Therefore, this genotype cannot help in determining general trends for the priority assignment gene. The simple fact is that this random priority assignment outperformed most other priority assignment algorithms and cannot be described as anything other than random.

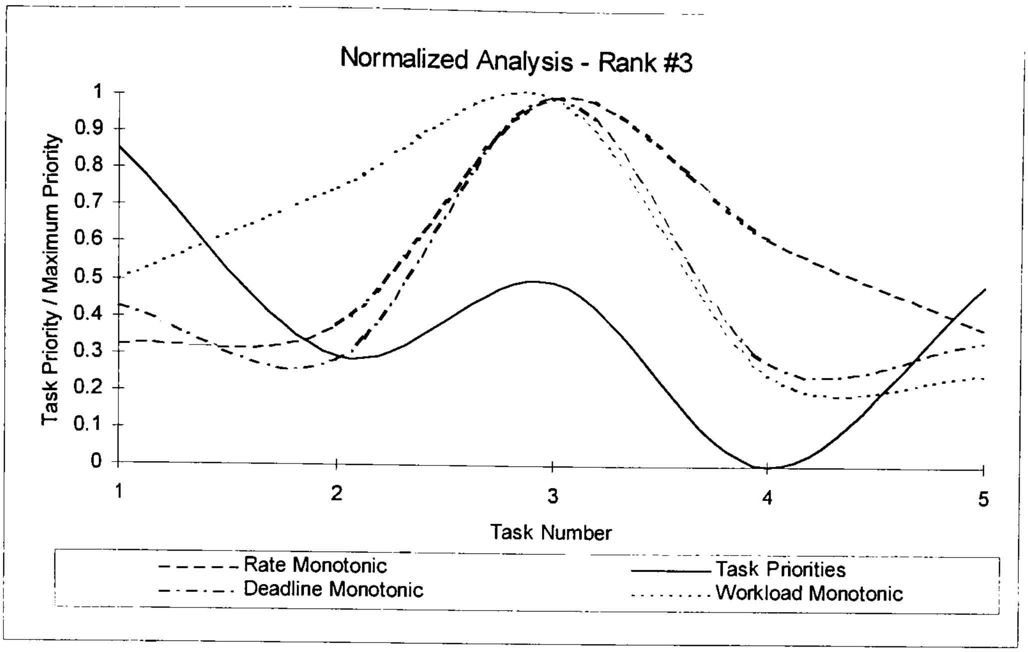



*Figure 24 Problem #3, 'Random' Assignment Analysis - Rank #3*

Although it is not obvious, the 'Task Priorities' curve above shares many common characteristics with the 'Deadline Monotonic' curve. The task priorities for this random assignment are more exaggerated than a pure deadline monotonic assignment but the relation between any two task priorities is the same. That is, for deadline monotonic assignment and the random assignment in this case, the following is true:

$$\forall \langle d_1, d_2 \rangle : d_1, d_2 \in [SetOfDeadlineMonotonicPriorities], \forall \langle r_1, r_2 \rangle : r_1, r_2 \in [SetOfRandomPriorities] \longrightarrow [(d_1 < d_2 \longleftrightarrow r_1 < r_2) \wedge (d_1 > d_2 \longleftrightarrow r_1 > r_2) \wedge (d_1 = d_2 \longleftrightarrow r_1 = r_2)]$$

*Equation 9 - Priority Assignment Characteristics*

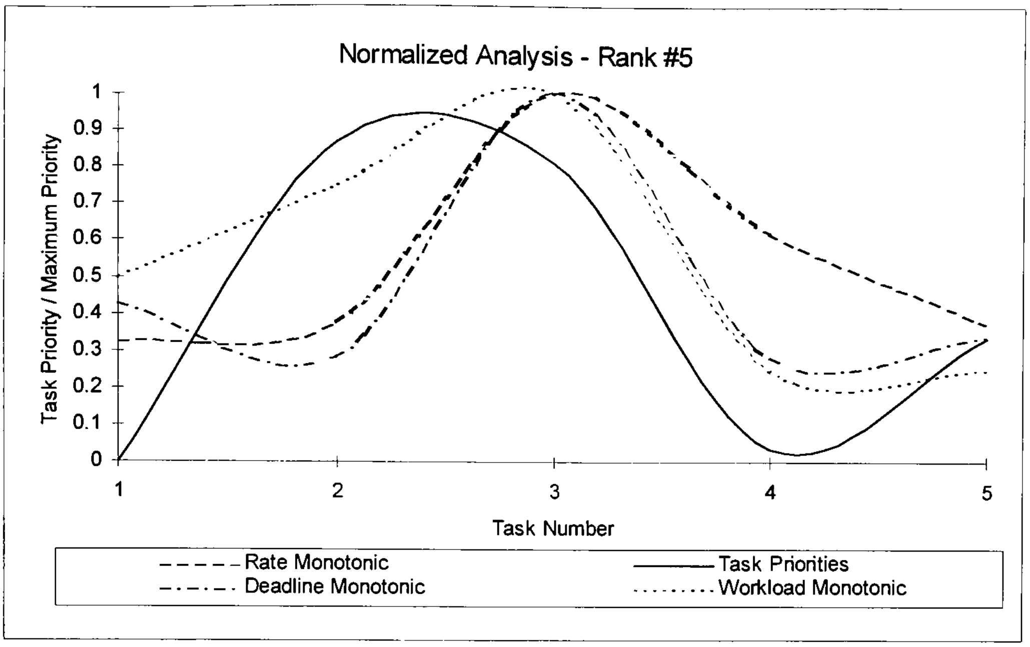



*Figure 25 - Problem #3, 'Random' Assignment Analysis - Rank #5*

Again, as can be clearly seen, the 'Task Priorities' curve does not entirely resemble any of the other curves. This random task priority assignment does share some general characteristics with the 'Workload Monotonic' curve but, that relationship is very weak. This weak association cannot accurately be used in determining general trends for the priority assignment gene.

From the results obtained it can be concluded that Deadline Monotonic task priority assignment yields the best results. This is, at least in part, surprising, since even worst case blocking analysis results in a nearly schedulable set of tasks (see *Equation 8 - Schedulability, Problem #3*). It is very promising that the genetic algorithm used here produced results that outperformed all others and would likely not have been derived by other means.

## 5.4 Test of Performance

The last stage in evaluating the GA was to use it to find a solution to a practical, complex problem. The important aspect to be evaluated in this case is the ability of the GA to converge to a solution for a complicated problem. With a practical complex problem it is very important that the GA be tested to ensure that it will be capable of converging. The actual real-time application used for this phase of the testing involves the operation and control of a complex digital radio transmitter.

This embedded, real-time application encompasses the main control of the digital radio. The primary functions of the application are: provide the user interface to the transmitter; monitor the operation of the transmitter to prevent dangerous operating condition from arising; and perform the necessary control system functions to maintain the transmitter output power within a specified limit. To accomplish these functions the transmitter software is broken down into a number of distinct but interrelated tasks. The responsibility of each of the tasks is listed below:

- Keypad Scan - The operator is able to control the transmission parameters of the radio (e.g., frequency, modulation mode, audio input source selection, data input source selection, etc.). This task is responsible for providing the operator with a means of interacting with the radio. The operator input is via an alpha-numeric keypad. This task is responsible for determining which key(s) are pressed by the operator and translating those keys pressed into internal radio commands.

- Front Panel Display - The operator is informed as to the status of the transmitter operation at all times (e.g., output power level, input audio level, fault messages, etc.). This task is responsible for updating the front panel liquid crystal display (LCD) with the present state of the radio transmitter.
- Remote Control - The transmitter is capable of being remotely controlled via a serial interface. This task is responsible for accepting serial data which represents operator commands (with the same controllable parameters as from the operator keypad). It is also responsible for providing radio status over the serial interface by responding to operator queries (with the same status parameters as on the LCD display).
- Emergency Monitor - The transmitter operation is monitored to ensure that dangerous signal levels and/or temperatures are not building up within the radio. This task is responsible for monitoring the internal radio temperature, voltage and current levels to ensure continuous safe operation.
- Analog Sampler - In order to provide control information to the rest of the radio processing software, this task is responsible for reading all analog to digital converters and providing those measurements in useful units of measure to the rest of the radio control software.
- Power Control - The output power of the transmitter is kept within specified limit during transmission. This is accomplished regardless of

the level of input audio to the transmitter. This task is responsible for adjusting the input audio level and the transmitter power gain to ensure that the output of the transmitter stays within limits.

- System State Control - The transmitter hardware is complex and in many cases mechanical in nature, this task is responsible for the absolute control of the various motors and relays used to tune and adjust the transmitter.
- Keyline Processing - Once the operator has configured the transmitter with the desired transmission parameters the system can be 'keyed' to begin transmission. The keyline is typically connected to the radio handset (i.e., 'Push-To-Talk' keyline) and causes the handset audio to begin being modulated over the air by the radio. This task is responsible for monitoring the selected key source and translating the actuation and release of the keyline into the appropriate commands to the radio hardware to start and terminate transmission.

The system design of the transmitter software is given in *Figure 26 - Real-Time Problem #4*, below.

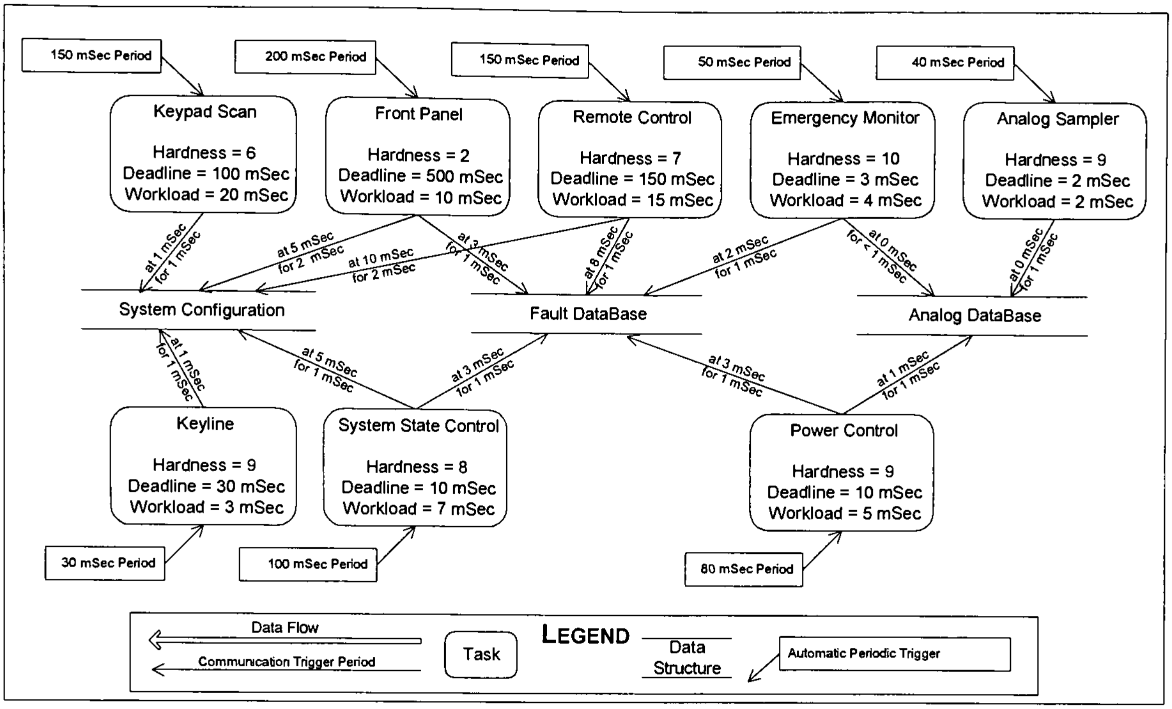



*Figure 26 - Real-Time Problem #4*

As in the previous case, task blocking can occur, therefore the schedulability test of *Equation 5* is again extended to include blocking time in addition to execution time. The following shows the analysis:

$$\frac{20}{150}+\frac{10}{200}+\frac{15}{150}+\frac{4}{50}+\frac{2}{40}+\frac{5}{80}+\frac{7}{100}+\frac{3}{30}\leq 8\cdot(2^{\frac{1}{8}}-1)$$

$$\text{Lower Bound} = 64.6\% \leq 72.4\% \text{ - Passed (barely)}$$

$$\frac{20+6}{150}+\frac{10+9}{200}+\frac{15+9}{150}+\frac{4+6}{50}+\frac{2+2}{40}+\frac{5+6}{80}+\frac{7+10}{100}+\frac{3+6}{30}\leq 8\cdot(2^{\frac{1}{8}}-1)$$

$$\text{Upper Bound} = 133.6\% \leq 72.4\% \text{ - Failed}$$

*Equation 10 - Schedulability, Problem #4*

As this analysis clearly shows, this problem (i.e. the set of tasks) may be schedulable, but very likely is not. Whether the tasks are schedulable or not is a function of the RTOS configuration itself. Note that there is only a small margin between the utilization lower bound and the schedulability threshold. The obvious conclusion is that it is not likely that an RTOS configuration exists which results in some level of blocking which yields a CPU utilization which is below the allowable threshold.

The results from running `GRTOS-GA.EXE` are as follows:

| Rank | Fitness | Tasking Model | Timeslice (μSec) | Priority Inheritance | Priority Allocation | Priority Assignment |
|---|---|---|---|---|---|---|
| 1 | 76.02 | Preemptive | 1227 | Disabled | Static | Deadline Monotonic |
| 2 | 75.94 | Preemptive | 2001 | Disabled | Static | Deadline Monotonic |
| 3 | 75.69 | Preemptive | 21060 | Enabled | Rotating | Random (-1675281081) |
| 4 | 73.57 | Preemptive | 2166 | Disabled | Rotating | Deadline Monotonic |
| 5 | 72.71 | Cooperative | N/A (47447) | Enabled | Rotating | Random (-4861458) |

*Table 8 - `GRTOS-GA.EXE` Results, Problem #4*

Analyzing the characteristics of the 'random' priority assignments for this real-time application is somewhat difficult. As the following graphs show (*Figure 27* and *Figure 28*), one of the task assignments that result from the random assignment algorithms can be accurately be categorized, and the other cannot.

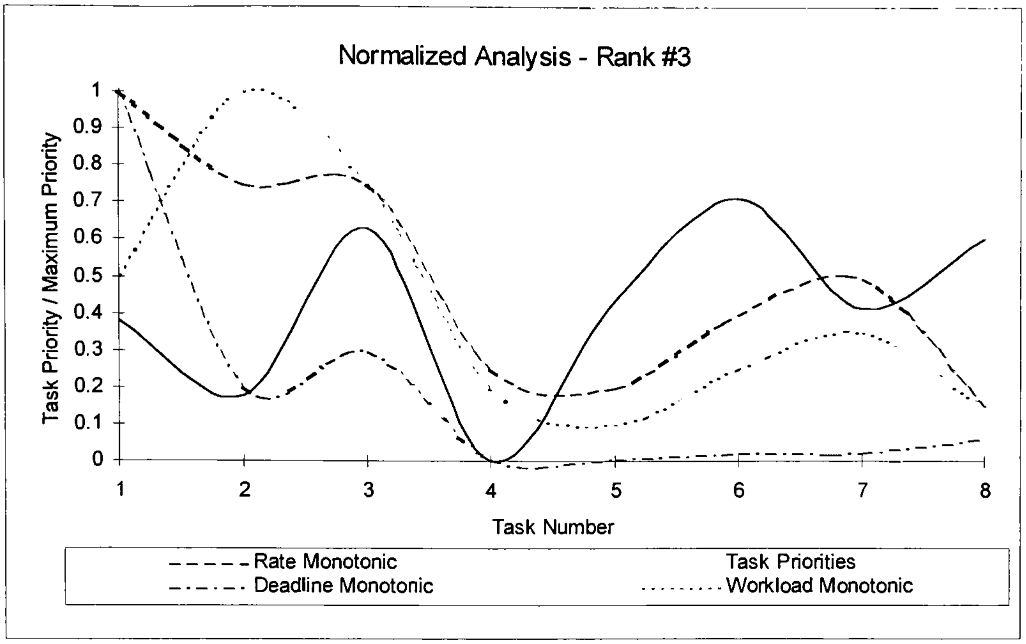



*Figure 27 - Problem #4, 'Random' Assignment Analysis - Rank #3*

Analyzing the priorities used in the random assignment above (ranked 3rd) shows that this random seed actually results in a (somewhat exaggerated) deadline monotonic assignment (see *Figure 27 - Problem #4, 'Random' Assignment Analysis - Rank #3*, above). Although not obvious, the 'Task Priorities' and 'Deadline Monotonic' curves posses the same general characteristics, again the relationships between task priorities are the same (see *Equation 9 - Priority Assignment Characteristics*). This supports the conclusion that this 'random' task priority assignment produces priority assignments which are similar to a deadline monotonic assignment scheme.

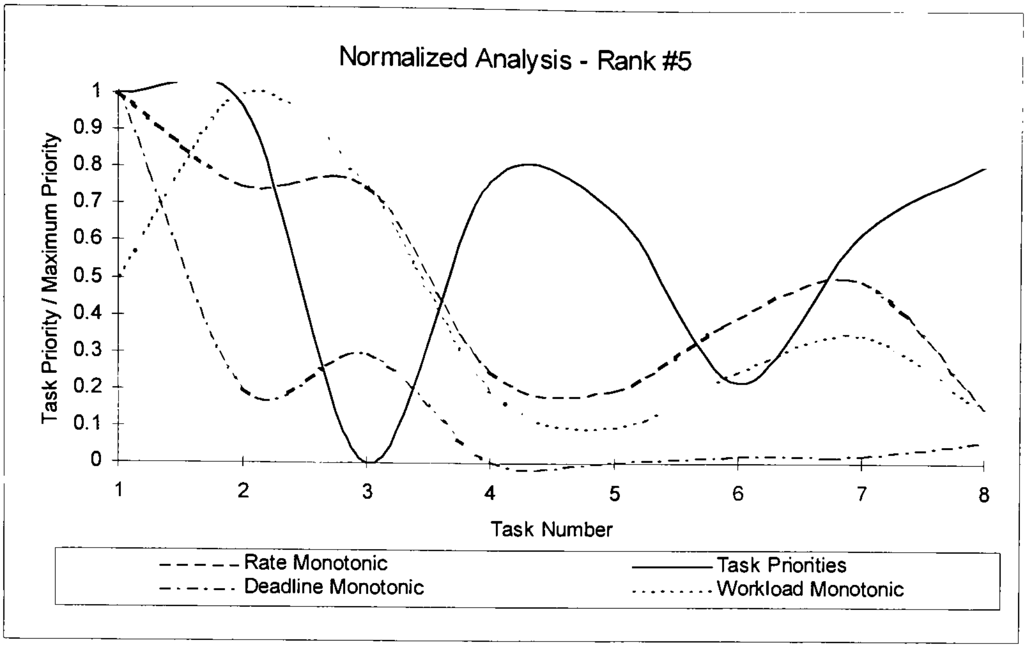



*Figure 28 - Problem #4, 'Random' Assignment Analysis - Rank #5*

As can be clearly seen, this 'Task Priorities' curve does not share any general characteristics with the 'standard' task priority assignment curves. There are simply no conclusions that can be drawn from this assignment with regard to priority assignment. The simple fact is that this random priority assignment outperformed many other priority assignment algorithms and cannot be described as anything other than random.

The results above (see *Table 8 -* `GRTOS-GA.EXE` *Results, Problem #4* and *Figure 27 - Problem #4, 'Random' Assignment Analysis - Rank #3*) clearly show that a preemptive multitasking environment, without priority inheritance, using deadline monotonic priority assignment and a timeslice of approximately 2 mSec provides the best results. The genetic algorithm produced these results for a problem specification which could not otherwise be solved.

# 6. Conclusions

From the results of the previous section it is reasonable to conclude that the genetic algorithm developed in this thesis can reliably be used to determine the 'best' real-time operating system configuration for an arbitrary real-time application. The tests performed verify that the GA converges to the correct solution in a case where the optimal solution is known. In addition, the GA converges to a reasonable solution in a case where the optimal solution can be predicted but not proven.

What is most interesting about the results obtained is that the task priority assignment algorithms which produced the best fitness for the problems chosen clearly fell into two categories. The problem specifications for 'schedulable' real-time problems resulted in rate monotonic assignment algorithms producing the best fitness values. This is not surprising since rate monotonic theory proves that this result produces an optimal solution. The problem specifications for 'non-schedulable' real-time problems resulted in deadline monotonic assignment algorithms producing the best fitness values. This is somewhat surprising but the data clearly supports this conclusion.

These results immediately raise the question: 'Does deadline monotonic task priority assignment produce the best results when the application tasks are likely to be non-schedulable?'. This is a very interesting question that would, unfortunately, require further research to answer. The data obtained very clearly implies this conclusion but the lack of a large number of diverse real-time application trials prevents that conclusion from being drawn. Opening this topic to additional research is probably the most significant result of this thesis.

There is an additional observation that must also be made. The most common industry practice when defining the RTOS configuration for a particular system is, by far, to use cooperative multitasking with uniform task priorities (this opinion is based on years of experience developing and studying fielded real-time systems). I believe that the reason for the choice of each of these configuration parameters is entirely different. As for the multitasking model, the reason that cooperative multitasking is often chosen is that cooperative multitasking systems are inherently simpler than preemptive multitasking systems. The argument for choosing uniform task priority assignment is that there is often little knowledge of the task execution profiles, when this is the case other priority assignment algorithms are not feasible.

The results of this thesis clearly show that preemptive multitasking is superior to cooperative multitasking in almost all situations, the added complexity of preemptive multitasking is more than outweighed by the performance improvement achieved. Also clearly evident from the results of this thesis is that uniform priority assignment always produces lower performance than any other assignment algorithm. Again, it is clear that analysis of the system design (i.e. task characteristics) is essential in order to determine the 'best' initial priority assignment algorithm for the real-time system.

# 7. Bibliography


Ref. 1 J. T. Baldwin, Predicting and Estimating Real-Time Performance, Embedded Systems Programming, pg 30-41, February 1995.

Ref. 2 S. Bodilsen, Scheduling Theory and Ada 9X, Embedded Systems Programming, pg 32-52, December 1994.

Ref. 3 M. F. Bramlette and E. E. Bouchard, Genetic Algorithms in Parametric Design of Aircraft, Handbook of Genetic Algorithms, Van Nostrand Reinhold, 1991.

Ref. 4 I. L. Bukatova and Y. V. Gulyaev, From Genetic Algorithms to Evolutionary Computer, Proceedings of the Fifth International Conference on Genetic Algorithms, pg 614-617, 1993.

Ref. 5 C. Darwin, The Origin Of Species, Mentor Edition, NAL Penguin Publishing, 1958.

Ref. 6 L. Davis, Handbook of Genetic Algorithms, Van Nostrand Reinhold, 1991.

Ref. 7 K. DeJong and W. Spears, On the State of Evolutionary Computation, Proceedings of the Fifth International Conference on Genetic Algorithms, pg 618-623, 1993.

Ref. 8 K. Ellison, Scheduling Algorithms for Systems with Hard Deadlines, Embedded Systems Programming, pg 24-31, April 1995.

Ref. 9 C. Heitmeyer and D. Mandrioli, Formal Methods for Real-Time Computing: An Overview, Formal Methods for Real-Time Computing, John Wiley & Sons, pg 1-32, 1996.

Ref. 10 E. D. Jensen, Eliminating the Hard/Soft Real-Time Dichotomy, Embedded Systems Programming, pg 28-33, October 1994.

Ref. 11 I. Lee, H. Ben-Abdallah and J-Y. Choi, A Process Algebraic Method for the Specification and Analysis of Real-Time Systems, Formal Methods for Real-Time Computing, John Wiley & Sons, pg 167-194, 1996.

Ref. 12 Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, 1992.

Ref. 13 M. Milenkovic, Operating Systems: Concepts and Design, McGraw-Hill, Inc., 1987.

Ref. 14 A. K. Mok, D. A. Stuart, and F. Jahanian, Specification and Analysis of Real-Time Systems: Modechart Language and Toolset, Formal Methods for Real-Time Computing, John Wiley & Sons, pg 33-54, 1996.

Ref. 15 R. Obenza, Guaranteeing Real-Time Performance Using RMA, Embedded Systems Programming, pg 26-40, May 1994.

Ref. 16 A. V. Oppenheim and A. S. Willsky, Signals and Systems, Prentice-Hall, 1983.

Ref. 17 N. J. Radcliff and F. A. W. George,A Study in Set Recombination Proceedings of the Fifth International Conference on Genetic Algorithms, pg 23-30, 1993.

Ref. 18 A. Tanenbaum, Modern Operating Systems, Prentice-Hall, 1992.

Ref. 19 D. M. Tate and A. E. Smith, Expected Allele Coverage and the Role of Mutation in Genetic Algorithms, Proceedings of the Fifth International Conference on Genetic Algorithms, pg 31-37, 1993.

# 8. Appendix A RTOS Source Code

## 8.1 TASKING.INT

```
(*********************************************************************)
(*                                                                   *)
(*                              TASKING                              *)
(*                                                                   *)
(*   This file contains the specification for a Turbo Pascal Unit which *)
(* extends Turbo Pascal by providing a tasking model.  Application tasks can *)
(* be dynamically created, destroyed and controlled.  Each task has access *)
(* to the entire global namespace of the application (although the mutual *)
(* exclusion mechanisms provided should be used to prevent simultaneous *)
(* access to shared variables).                                       *)
(*   The tasking model can be either 'cooperative' or 'preemptive' multi- *)
(* tasking.  For cooperative multi-tasking applications, all tasks must *)
(* block (or access a TASKING service in some way) at some point during *)
(* their execution to allow other tasks to run.  For preemptive multi- *)
(* tasking, control will pass among ready tasks for every timeslice, for *)
(* this reason tasks do not need to block at all because control will be *)
(* taken from them when their timeslice has expired.                  *)
(*   In addition to general task management facilities, the following multi- *)
(* tasking constructs are supported: Counting ('Dijkstra') Semaphores, *)
(* Events, Binary Semaphores, Condition Variables, Message Passing, Blocking *)
(* Mouse activities and Blocking Keyboard Input operations.           *)
(*                                                                   *)
(*********************************************************************)

interface

uses  MS_MOUSE;

type  (*********************************************************************)
      (*   The policy that TASKING uses when performing task scheduling is *)
      (* based on task priorities.  The task priorities can be static or *)
      (* dynamic (rotating).                                             *)
      (*********************************************************************)
      PRIORITY_SCHEDULING_POLICIES = (
        (*******************************************************************)
        (*  Priorities are not changed by TASKING in any way, the program is *)
        (* allowed to change task priorities but TASKING schedules task *)
        (* execution in a pure round-robin manner.                       *)
        (*******************************************************************)
        STATIC_PRIORITIES,
        (*******************************************************************)
        (*  Priorities are rotated after each context switch such that the *)
        (* priority of the task being switched out becomes the lowest. The *)
        (* actual priority values for the tasks are not changed but the *)
        (* logical meaning associated with that priority level is rotated. *)
        (*******************************************************************)
        ROTATING_PRIORITIES);
      (*********************************************************************)

      (*********************************************************************)
      (*   To aid in application development, TASKING can generate reports *)
      (* which contain useful statistical information about the tasks.   *)
      (*********************************************************************)
      STATISTIC_OPTIONS = (
        (*******************************************************************)
        (*  No statistics are gathered, this should be used for once the *)
        (* application debugging is complete.                            *)
        (*******************************************************************)
        NO_STATISTICS,
        (*******************************************************************)
        (*  Gather statistics about the tasks: stack utilization (absolute *)
        (* and percent), CPU utilization (percent).  Statistics concerned *)
        (* with the application as a whole are also reported.            *)
        (*  This information is contained in the directory from which the *)
        (* program was run, in the file 'TASKING.RPT'.                   *)
        (*******************************************************************)
        TASK_STATISTICS,
        (*******************************************************************)
        (*  Gather statistics about the MS-DOS services that were used by *)
        (* the application.                                              *)
        (*  This information is contained in the directory from which the *)
        (* program was run, in the file 'IRQ.RPT'.                       *)
        (*******************************************************************)
        MSOOS_SERVICES_STATISTICS,
        (*******************************************************************)
        (*  Gather statistics about the hardware interrupts that were *)
        (* handled while the program was executing.                      *)
        (*  This information is contained in the directory from which the *)
        (* program was run, in the file 'SERVICES.RPT'.                  *)
        (*******************************************************************)
        HARDWARE_INTERRUPT_STATISTICS,
        (*******************************************************************)
        (*  Gather all of the above statistics.                          *)
        (*******************************************************************)
        ALL_STATISTICS);
      (*********************************************************************)

      (*********************************************************************)
      (*   Two distinctly different multi-tasking models are supported by *)
      (* TASKING, Cooperative and Preemptive.                            *)
      (*********************************************************************)
      TASKING_MODELS = (
        (*******************************************************************)
        (*  For cooperative multitasking, context switches only occur when a *)
        (* task surrenders control of the CPU by making a TASKING call.  *)
        (*******************************************************************)
        COOPERATIVE,
        (*******************************************************************)
        (*  For preemptive multutasking, TASKING seizes control of the CPU *)
        (* from the task and re-assigns it to another ready task it at *)
        (* 'timeslice' intervals.                                        *)
        (*******************************************************************)
        PREEMPTIVE);
      (*********************************************************************)

      (*********************************************************************)
      (*  This data structure contains all of the user configurable TASKING *)
      (* parameters.  TASKING examines the values in the USER_CONFIGURATION *)
      (* variable when 'main()' terminates, therefore all changes to the *)
      (* TASKING configuration must be made in the main body of the program *)
      (* before multitasking actually starts).                           *)
      (*********************************************************************)
      CONFIGURATION = record
        (*******************************************************************)
        (*  TASKING can search the INI file of the application program for *)
        (* all (or some) configuration parameters.  See TASKING.INI for a *)
        (* complete description of the format of INI file entries.       *)
        (*******************************************************************)
        USE_INI_FILE_TASKING_PARAMETERS : boolean;
        (*******************************************************************)
        (*  In order to further refine the TASKING characteristics, the way *)
        (* that TASKING handles priorities (and hence scheduling) can be *)
        (* controlled.                                                   *)
        (*******************************************************************)
        PRIORITY_SCHEDULING_POLICY : PRIORITY_SCHEDULING_POLICIES;
        (*******************************************************************)
        (*  Priority inheretance is a mechanism whereby priority inversion *)
        (* can be guranteed not to exist.  Priority inversion occurs when a *)
        (* high priority task is forced to wait while a low priorty task *)
        (* (with a locked resource) is preempted by an intermediate priorty *)
        (* task.                                                         *)
        (*******************************************************************)
        PRIORITY_INHERITANCE_ENABLED : boolean;
        (*******************************************************************)
        (*  During application development is will often be useful to 'peek' *)
        (* into the resource utilization of the system and/or the tasks. *)
        (*******************************************************************)
        STATISTICS : STATISTIC_OPTIONS;
        (*******************************************************************)
        (*  TASKING supports two completely different multitasking models: *)
        (* cooperative and preemptive.                                   *)
        (*******************************************************************)
        case TASKING_MODEL : TASKING_MODELS of
          (*****************************************************************)
          (*  For cooperative multitasking, context switches only occur when *)
          (* a task surrenders control of the CPU by making a TASKING call. *)
          (*****************************************************************)
          COOPERATIVE :
            (***************************************************************)
            (*  No additional parameters.                                  *)
            (***************************************************************)
            ();
          (*****************************************************************)
          (*  For preemptive multutasking, TASKING seizes control of the CPU *)
          (* from the task and re-assigns it to another ready task it at *)
          (* 'timeslice' intervals.                                        *)
          (*****************************************************************)
          PREEMPTIVE :
            (***************************************************************)
            (*  If the tasking model is preemptive, then the minimum amount *)
            (* of time that a task should be allowed to run before control *)
            (* is passed to another ready task.  The actual timeslice that *)
            (* is achieved may differ from the value specified based on the *)
            (* task characteristics and activities.                        *)
            (***************************************************************)
            (TARGET_TIMESLICE : longint); (* uSec *)
      end; (* CONFIGURATION *)

const (*********************************************************************)
      (*  This variable can be directly modified within the main body of the *)
      (* application, changes at any other time have no impact on TASKING. *)
      (*********************************************************************)
      TASKING_CONFIGURATION : CONFIGURATION =
        (
         USE_INI_FILE_TASKING_PARAMETERS : false;
         PRIORITY_SCHEDULING_POLICY      : STATIC_PRIORITIES;
         PRIORITY_INHERITANCE_ENABLED    : false;
         STATISTICS                      : NO_STATISTICS;
         TASKING_MODEL                   : PREEMPTIVE;
         TARGET_TIMESLICE                : 10000 (uSEC)
        );

      (*********************************************************************)
      (*  Timeslice values must fall within the range of fifty microseconds *)
      (* to sixty seconds, TASKING will adjust the target timeslice value to *)
      (* enforce this restriction if necessary.                          *)
      (*********************************************************************)
      MINIMUM_TIMESLICE = 50       (uSec);
      MAXIMUM_TIMESLICE = 60000000 (uSec);

      (*********************************************************************)
      (*  This is the restriction that TASKING places on the number of tasks *)
      (* that are 'active' at any point in time.                         *)
      (*********************************************************************)
      MAXIMUM_NUMBER_OF_TASKS = 2047;

type  (*********************************************************************)
      (*  Since tasks can be created/destroyed dynamically, an application *)
      (* can be composed of more than MAX_NUMBER_OF_TASKS, but no more than *)
      (* MAX_NUMBER_OF_TASKS tasks can be active at any point in time. *)
      (* Task identifiers will be created/allocated/managed by TASKING, *)
      (* under no circumstances should a TASK_ID be assigned a value by the *)
      (* application.                                                    *)
      (*********************************************************************)
      TASK_IDS = 0..$FFFF;

      (*********************************************************************)
      (*  All user tasks must be assigned a priority from this range.    *)
      (*********************************************************************)
      USER_PRIORITIES = 1..10000;

      (*********************************************************************)
      (*  All application tasks MUST be procedures which match this task *)
      (* specification.  The parameters to the task will be passed into the *)
      (* task when it begins executing.  Since these values will exists on *)
      (* the stack frame of the task (which never goes away) the task will *)
      (* have access to them whenever it is running.  Modifying either of *)
      (* these variables will have NO effect on the execution of the task. *)
      (* The variables are provided by TASKING to the task but are in no way *)
      (* used by TASKING (a mechanism is provided to alter the priority of a *)
      (* task, see the CHANGE_PRIORITY() procedure).                     *)
      (* NOTE: Tasks must be declared 'far' either implicitly with $F+ or *)
      (*       explicitly by adding 'far;' after the procedure header.   *)
      (*********************************************************************)
      TASK_TYPE = procedure(TASK_ID : TASK_IDS; PRIORITY   USER_PRIORITIES);

      (*********************************************************************)
      (*  TASKING ensures that the operations requested by a task are valid *)
      (* within the context of the request.  Because of these validity *)
      (* checks, errors of these categories can be detected.             *)
      (*********************************************************************)
      SYSTEM_ERRORS =
        (
         TASK_ALREADY_ACTIVE,
         INSUFFICIENT_RESOURCES,
         TASK_IS_NOT_ACTIVE,
         TASK_ALREADY_SUSPENDED,
         ILLEGAL_TASK_ID,
         ILLEGAL_OPERATION
        );

      (*********************************************************************)
      (*  Each application task can have different and/or common error *)
      (* handling routines.  Any error handler which the user decides to *)
      (* implement must conform to this specification.                   *)
      (*********************************************************************)
      HANDLER_PROCEDURE = procedure(TASK_ID : TASK_IDS);

      (*********************************************************************)
      (*  This data structure is used to for all time specifications within *)
      (* TASKING.                                                        *)
      (*********************************************************************)
      TIME = record
        DAYS            u..31;
```

```
    HOURS        : 0..23;
    MINUTES      : 0..59;
    SECONDS      : 0..59;
    MILLISECONDS : 0..999;
  end; (* TIME *(

  (*******************************************************************(
  (*  This is the single data structure that must be configured for each *(
  (* application task.                                                  *(
  (*******************************************************************(
  TASK_ATTRIBUTES = record
    (*****************************************************************(
    (*  This priority is subject to the 'Priority Scheduling Policy' *(
    (* established in the USER_CONFIGURATION data structure.  Note that *(
    (* higher numbers are higher priority.                              *(
    (*****************************************************************(
    PRIORITY : USER_PRIORITIES;
    (*****************************************************************(
    (*  This is the number of words that the task requires for stack *(
    (* space.  There is no easy way to determine this value accurately. *(
    (* Turning on the TASK_STATISTICS is the best method of determining *(
    (* stack requirements.  Note that if the application 'locks up' it *(
    (* is very likely that one (or more( of the task stacks is not big *(
    (* enough.                                                          *(
    (*****************************************************************(
    STACK_WORDS_NEEDED : word;
    (*****************************************************************(
    (* If an application does not assign error handlers then TASKING *(
    (* will use internal default handlers which provide some diagnostics *(
    (* and attempt to allow the application to continue running.  It is *(
    (* unlikely that the application will be able to run in the presence *(
    (* of errors, but all attempts are made to do so.  If the default *(
    (* error handlers are to be used then the corresponding *(
    (* ERROR_HANDLERS element should be set to 'nil'                    *(
    (*****************************************************************(
    ERROR_HANDLERS : array (SYSTEM_ERRORS) of HANDLER_PROCEDURS;
  end; (* TASK_ATTRIBUTES *(

  (*******************************************************************(
  (*                  Counting ('Dijkstra'( Semaphores                   *)
  (*                                                                    *(
  (*  All semaphores used by the application must be variables declared *)
  (* (or dynamically created( of this type.  'Dijkstra' semaphores can *(
  (* retain signals because information is maintained so that TASKING *(
  (* knows how may signals are stored in the semaphore at all times.   *(
  (*******************************************************************(
  SEMAPHORE = 0..$FFFF;

  (*******************************************************************(
  (*                        Software Events                             *)
  (*                                                                    *(
  (*  All events used by the application must be variables declared (or *(
  (* dynamically created) of this type.  Events can retain only a single *(
  (* signal.  Once an event has been signaled all subsequent signals *(
  (* are lost (i.e., the event remains signaled(.                      *(
  (*******************************************************************(
  EVENT = (UNSIGNALED, SIGNALED(;

  (*******************************************************************(
  (*                       Binary Semaphores                            *(
  (*                                                                    *(
  (*  All binary semaphores used by the application must be variables *(
  (* declared (or dynamically created( of this type.  Binary semaphores *(
  (* are used to ensure mutually exclusive access to a resource which is *(
  (* shared between tasks.  It is logically an error to signal a *)
  (* signaled binary semaphore, TASKING will detect this error as an *)
  (* illegal operation.                                                 *(
  (*******************************************************************(
  BINARY_SEMAPHORE = 0..1;

  (*******************************************************************(
  (*                      Condition Variables                           *(
  (*                                                                    *(
  (*  All condition variables used by the application must be variables *(
  (* declared (or dynamically created( of this type.  It is not likely *(
  (* that applications will use condition variables (they are most *(
  (* useful in the implementation of Hoare Monitors(.  Condition *(
  (* variables do not have any memory associated with them, only if a *(
  (* task is already waiting on a condition variable when it is signaled *(
  (* will that task become unblocked.  If a condition variable is *(
  (* signaled and there is no task waiting on it then the signal is lost.*(
  (*******************************************************************(
  CONDITION_VARIABLE = (CONDITION_VARIABLES_DO_NOT_HAVE_VALUES);

(*********************************************************************(
(*                    Memory Management Operations                     *(
(*********************************************************************(

procedure GETMEM
(*********************************************************************(
  (var PTR : pointer; SIZE : word);
(*-------------------------------------------------------------------*(
(*  This procedure replaces the standard Turbo Pascal memory allocation *(
(* procedure of the same name.  Since TASKING must perform all memory *(
(* allocation and de-allocation, this routine *must* be used instead of the *(
(* system.getmem() procedure.  If 'type P : ^T;' is declared, then the *(
(* correct usage is 'getmem(pointer(P(, sizeof(T();'.                  *)
(*********************************************************************(

procedure FREEMEM
(*********************************************************************(
  (PTR : pointer; SIZE : word(;
(*-------------------------------------------------------------------*(
(*  This procedure replaces the standard Turbo Pascal memory allocation *)
(* procedure of the same name.  Since TASKING must perform all memory *)
(* allocation and de-allocation, this routine *must* be used instead of the *)
(* system.freemem() procedure.  If 'type P : ^T;' is declared, then the *(
(* correct usage is 'freemem(pointer(P(, sizeof(T));'.                 *(
(*********************************************************************(

(*********************************************************************)
(*                General Task Management Operations                   *(
(*********************************************************************(

procedure CREATE
(*********************************************************************)
  (var TASK : TASK_IDS; ATTR : TASK_ATTRIBUTES; ENTRY_POINT : TASK_TYPE(;
(*-------------------------------------------------------------------*)
(*  This routine creates all system data structures for a parallel *)
(* executing task.  The task is made ready to run and will be available for *)
(* execution.  Actual execution will occur when the scheduler activates the *(
(* task (i.e. precise execution time is unknown and depends on availability *(
(* of system resources and system load).  In addition NO tasks will be *(
(* scheduled until the application 'main' has terminated.              *)
(*********************************************************************)

procedure DESTROY
(*********************************************************************(
  (TASK : TASK_IDS);
(*-------------------------------------------------------------------*(
(*  This routine terminates and (if appropriate( releases all of the task's *(
(* resources to be used by other tasks and then removes all evidence of the *(
(* task ever having existed.  Tasks may only 'destroy' other tasks, not *(
(* themselves.  This operation should only be performed when the full *(
(* consequences of task destruction are known and the result of the task '(
(*********************************************************************(

procedure SUSPEND
(*********************************************************************(
  (TASK : TASK_IDS(;
(*-------------------------------------------------------------------*(
(*  This routine causes a previously running or ready task to be blocked *(
(* unconditionally.  The only way that a task suspended by this call can run *(
(* again is if another task RESUME(('s it.                             *(
(*********************************************************************(

procedure RESUME
(*********************************************************************(
  (TASK : TASK_IDS(;
(*-------------------------------------------------------------------*(
(*  This routine causes a previously suspended task to be made ready.  The *(
(* task is not necessarily the next to run, that is based on the priorities *(
(* of the other ready tasks.                                           *(
(*********************************************************************(

function GET_MILLISECOND_TICKS : longint;
(*********************************************************************(
(*  This routine retuns the number of milliseconds that have elapsed since *(
(* the program started executing.                                      *(
(*********************************************************************(

procedure WAIT_FOR_DELAY
(*********************************************************************(
  (DURATION : TIME(;
(*-------------------------------------------------------------------*(
(*  This system call causes the currently running task to become blocked *(
(* for the specified length of time.  There is no guarantee that the task *(
(* will begin executing when the time expires, the only guarantee is that *(
(* the task will become ready to execute at that time.                 *(
(*********************************************************************(

procedure PREEMPTABLE_DELAY
(*********************************************************************(
  (DURATION : TIME(;
(*-------------------------------------------------------------------*(
(*  This system call causes the currently running task to be delayed by the *(
(* specified length of time.                                           *(
(*********************************************************************(

procedure CHANGE_PRIORITY
(*********************************************************************(
  (PRIORITY : USER_PRIORITIES(;
(*-------------------------------------------------------------------*)
(*  This routine simply changes the priority of the currently running task *)
(* and possible reschedules the running task (if there is a new highest *(
(* priority task ready to run).                                        *(
(*********************************************************************(

(*********************************************************************(
(*            Counting ('Dijkstra'( Semaphore Operations              *(
(*********************************************************************(

procedure SIGNAL_SEMAPHORE
(*********************************************************************)
  (var SEM : SEMAPHORE(;
(*-------------------------------------------------------------------*(
(*  This routine performs an Up(( operation on the specified semaphore.  If *(
(* there are tasks waiting on the semaphore then one is awakened.  If there *(
(* are none waiting then the semaphore is incremented.                 *)
(*********************************************************************)

procedure WAIT_ON_SEMAPHORE
(*********************************************************************)
  (var SEM : SEMAPHORE);
(*-------------------------------------------------------------------*(
(*  This routine performa a Down(( operation on the specified semaphore. *(
(* If the semaphore is zero then the running task is suspended.  If it is *(
(* non-zero then the semaphore is decremented.                         *(
(*********************************************************************)

(*********************************************************************(
(*                   Software Event Operations                         *(
(*********************************************************************(

procedure SIGNAL_EVENT
(*********************************************************************(
  (var THE_EVENT : EVENT(;
(*-------------------------------------------------------------------*(
(*  This routine signals the event specified, if there are tasks waiting on *(
(* the event then one task is made ready.  If there are no tasks waiting on *(
(* the event then the signal is saved, only one signal is maintained no *(
(* matter how many times the event is signaled.                        *(
(*********************************************************************(

procedure BROADCAST_EVENT
(*********************************************************************(
  (var THE_EVENT : EVENT(;
(*-------------------------------------------------------------------*(
(*  This routine signals the event specified, if there are tasks waiting on *)
(* the event then they are ALL made ready.  If there are no tasks waiting on *)
(* the event then the signal is saved and the next SINGLE task to wait on *)
(* the event will consume the event.  Only one signal is maintained no *)
(* matter how many times the event is signaled.                        *(
(*********************************************************************(

procedure START_PERIODIC_EVENT
(*********************************************************************(
  (var THE_EVENT : EVENT; INTERVAL : TIME);
(*-------------------------------------------------------------------*(
(*  This routine starts the periodic signalling (by TASKING( of the event *(
(* specified at the specified interval.  This routine cannot cause a task to *(
(* be suspended but a re-schedule may occur.                           *(
(*********************************************************************(

procedure STOP_PERIODIC_EVENT
(*********************************************************************(
  (var THE_EVENT : EVENT(;
(*-------------------------------------------------------------------*(
(*  This routine stops the periodic signalling (by TASKING( of the event *)
(* specified.  This routine cannot cause a task to be suspended but a *)
(* re-schedule may occur.                                              *)
(*********************************************************************(

procedure WAIT_ON_EVENT
(*********************************************************************(
  (var THE_EVENT : EVENT);
(*-------------------------------------------------------------------*(
(*  This routine causes the calling task to be suspended waiting for the *)
(* specified event to occur.  If the event has already occurred then the *)
(* calling task will immediately become ready, although a re-schedule may *)
(* occur.                                                              *)
(*********************************************************************)

(*********************************************************************)
(*                 Binary Semaphore Operations                         *(
(*********************************************************************(

procedure SIGNAL_BINARY_SEMAPHORE
(*********************************************************************(
  (var SEM : BINARY_SEMAPHORE);
(*-------------------------------------------------------------------*(
(*  This routine performa an Up(( operation on the specified semaphore.  If *(
(* there are tasks waiting on the semaphore then one is awakened.  If there *)
(* are none waiting then the semaphore is set.                         *)
(*********************************************************************(

procedure WAIT_ON_BINARY_SEMAPHORE
(*********************************************************************(
```

```
  |var SEM : BINARY_SEMAPHORE|;
|*-----------------------------------------------------------------------*|
|*   This routine performs a Down|| operation on  the  specified  semaphore. *|
|* If the semaphore is zero then the running task is suspended.   If  it  is *|
|* non-zero then the semaphore is cleared.                                  *|
|**************************************************************************|

|**************************************************************************|
|*                      Condition Variable Dperations                      *|
|**************************************************************************|

procedure SIGNAL_CONDITIDN_VARIABLE
|**************************************************************************|
  |var C_VAR : CONDITION_VARIABLE|;
|*-----------------------------------------------------------------------*|
|*   This routine signals the condition variable specified, if  there  is  a *|
|* task waiting on the condition variable then it is made ready |or  one  of *|
|* the multiple waiting tasks is made ready|.  If there are no tasks waiting *|
|* on the condition variable then the signal is lost.                        *|
|**************************************************************************|

procedure WAIT_ON_CONDITION_VARIABLE
|**************************************************************************|
  |var C_VAR : CDNDITION_VARIABLE|;
|*-----------------------------------------------------------------------*|
|*   This routine causes  the  calling  task  to  become  blocked.   Because *|
|* condition variables are not stored by the system,  the  calling  task  is *|
|* guaranteed to become blocked.   As  soon  as  another  task  signals  the *|
|* condition variable the waiting task will be made ready  |or  one  of  the *|
|* multiple waiting tasks will be made ready|.                               *|
|**************************************************************************|

|**************************************************************************|
|*                        Message Passing Operations                       *|
|**************************************************************************|

procedure SEND_MESSAGE_TO
|**************************************************************************|
  |RECIPIENT_TASK : TASK_IDS; XMIT_MESSAGE_PTR : pointer|;
|*-----------------------------------------------------------------------*|
|*   This routine sends the message  |actually  the  pointer|  to  the  task *|
|* by placing it into the task's mailbox |which is automatically created  if *|
|* necessary|.  Although the calling task cannot  block  while  sending  the *|
|* message it is possible for a reschedule to occur.  Typically messages are *|
|* dynamically allocated before being sent and the receiver  is  responsible *|
|* for disposing of them after they have been used.                          *|
|**************************************************************************|

procedure RECEIVE_MESSAGE
|**************************************************************************|
  |var RCV_MESSAGE_PTR : pointer|;
|*-----------------------------------------------------------------------*|
|*   This routine retrieves a message |actually a pointer to a message| from *|
|* the caller's mailbox |which is created if necessary|,  if  there  are  no *|
|* messages in the mailbox then nil is returned.   The  task  cannot  become *|
|* blocked by calling this routine but a reschedule is possible.   Typically *|
|* the message will be disposed of after it has been used by  the  receiving *|
|* task.                                                                     *|
|**************************************************************************|

procedure WAIT_AND_RECEIVE_MESSAGE
|**************************************************************************|
  |var RCV_MESSAGE_PTR : pointer|;
|*-----------------------------------------------------------------------*|
|*   This routine retrieves a message |actually a pointer to a message| from *|
|* the caller's mailbox |which is created if necessary|,  if  there  are  no *|
|* messages in the mailbox then the task is blocked waiting for the  mailbox *|
|* to become non-empty, because of this the return pointer can never be nil. *|
|* Typically the message will be disposed of after it has been used  by  the *|
|* receiving task.                                                           *|
|**************************************************************************|

|**************************************************************************|
|*                       MS-Mouse Handling Operations                      *|
|**************************************************************************|

procedure ENABLE_MOUSE_ACTIONS
|**************************************************************************|
  |MOUSE_ACTIONS : word|;
|*-----------------------------------------------------------------------*|
|*   This routine enables the specified MS-Mouse events to  be  received  by *|
|* the task which is |or will be| waiting to receive mouse actions.          *|
|**************************************************************************|

procedure WAIT_AND_RECEIVE_MOUSE_ACTIONS
|**************************************************************************|
  |var MOUSE_INFO : MOUSE_PARAMETERS|;
|*-----------------------------------------------------------------------*|
|*  If an MS-Mouse handler is to be created then communication  between  the *|
|* mouse driver and the 'mouse handling' task is done through a variable  of *|
|* type MS_MOUSE.MOUSE_PARAMETERS |which  can  be  declared  or  dynamically *|
|* created|.                                                                *|
|*   This routine receives MS-Mouse  action  parameters  from  the  MS-Mouse *|
|* driver.  If none of the enabled mouse actions has occurred then the  task *|
|* is blocked waiting for one |or more| action to occur.                     *|
|**************************************************************************|

|**************************************************************************|
|*                       Keyboard Handling Operations                      *|
|**************************************************************************|

function WAIT_ON_READKEY : char;
|**************************************************************************|
|*   This routine provides a means for an application task to perform I/D in *|
|* a 'blocking' manner.  This means that while there is no  input  from  the *|
|* keyboard the calling task is suspended, as soon as input arrives from the *|
|* keyboard the caller is awakened and given the ASCII code of the key  that *|
|* was pressed by the user |i.e., a blocking eguivalent of 'readkey'|.       *|
|*   An added benefit of using blocked keyboard input is that  this  version *|
|* of 'readkey' is able to detect |and pass back  to  the  application|  the *|
|* extended function keys F11 & F12 |as well as shift, alt & ctrl versions|. *|
|* The Turbo Pascal 'readkey' function is unable to recognize F11 & F12.     *|
|*                                                                           *|
|*   WARNING:                                                                *|
|*     Some add-on utilities that extended the size of the  keyboard  buffer *|
|*   will actually 'steal'  the  keys  from  the  blocked  task.   All  such *|
|*   utilities should be disabled when using this blocking  I/O  feature  of *|
|*   TASKING.                                                                *|
|*     Using blocking I/O causes the application to be unable to distinguish *|
|*   between multiple characters arriving from the pressing of  an  ANSI.SYS *|
|*   re-programmed key on the keyboard from the  actual  re-programmed  key. *|
|*   That is, if a keyboard key is re-programmed  using  ANSI.SYS  then  the *|
|*   application will never be able to see that key, it will  only  see  the *|
|*   re-programmed value whenever the key is pressed  |the  key  translation *|
|*   by ANSI.SYS occurs before the 'key' is detectable by software, although *|
|*   the standard Turbo Pascal 'readkey' is able  to  see  the  'base key'|. *|
|*   This anomaly can create synchronization problems between  the  keyboard *|
|*   and TASKING if the key has been reprogrammed as multiple keys |which is *|
|*   usually the case|.  Under these conditions the keyboard buffer can  end *|
|*   up with more characters in it than are  reported  to  the  application. *|
|*   ANSI.SYS re-programming should be disabled when using this blocking I/O *|
|*   feature of TASKING.                                                     *|
|**************************************************************************|

|**************************************************************************|
|*                      Scheduler Handling Operations                      *|
|**************************************************************************|

procedure LOCK_SCHEDULER;
|**************************************************************************|
|*   This procedure 'locks' the  scheduler and  prevents  all  further  task *|
|* rescheduling until UNLOCK_SCHEDULER is called.                            *|
|*   One use of this routine is when multiple tasks need  to  write  to  the *|
|* display and it is desired that the output from each  task  occur  without *|
|* the output from other tasks being intermixed with it.   During  the  time *|
|* that the scheduler is  'locked' there will be *no* preemptive/cooperative *|
|* task rescheduling.  The application should avoid 'locking' the  scheduler *|
|* if possible, if that is  not  feasible  then  the  time  spent  with  the *|
|* scheduler 'locked' should be an absolute minimum.                         *|
|*   It is imperative the *each* call to LOCK_SCHEDULER have a corresponding *|
|* call to UNLOCK_SCHEDULER, successive |nested| calls to LOCK_SCHEDULER are *|
|* allowed as long as each has a corresponding call to UNLOCK_SCHEDULER.     *|
|**************************************************************************|

procedure UNLOCK_SCHEDULER;
|**************************************************************************|
|*   This procedure 'unlocks' the scheduler and allows task rescheduling  to *|
|* continue based on  the  previously  existing  configuration.   Note  that *|
|* LOCK_SCHEDULER must previously have been called.                          *|
|**************************************************************************|
```

# 8.2 TASKING.PAS

```
unit TASKING;
(****************************************************************************)
(*   Extends Turbo Pascal by adding  a  preemptive  or  cooperative  tasking *)
(* model to the language.  The multitasking extensions are  fully  supported *)
(* while running under MS-DOS.                                               *)
(*-------------------------------------------------------------------------*)
(*                       Compiler Options (Ver. 7.0)                         *)
(*$A+                                Word Alignment                          *)
(*$B-                                Short Circuit Boolean Evaluation        *)
(*$D+                                Debug Code Generation ON (Sort of)      *)
(*                                    Requires /V option to TPC to activate  *)
(*$L+                                Local Debug symbols ON (Sort of)        *)
(*                                    Requires /V option to TPC to activate  *)
(*$F-                                Far calls only as necessary             *)
(*$G-                                Generic 80x86 code only                 *)
(*$I-                                I/O Checking OFF                        *)
(*$N-                                Do Not allow 80x87 instructions         *)
(*$O-                                Overlays NOT allowed                    *)
(*$P-                                Standard 'string' parameters            *)
(*$Q-                                Overflow Checking OFF                   *)
(*$R-                                Range Checking OFF                      *)
(*SS-                                Stack Checking OFF                      *)
(*$T-                                Do Not Force Typed '@' references       *)
(*$V-                                Var-string Checking OFF                 *)
(*SX-                                Disable Extended syntax                 *)
(****************************************************************************)

(*$I TASKING.INT - Filename of the unit interface (i.e., 'public' stuff)     *)

implementation

uses  dos;

const (**********************************************************************)
      (*  This is the ID of the TASKING task which is executed when there is *)
      (* nothing else to execute (this is reserved and cannot be assigned to *)
      (* user tasks).                                                        *)
      (**********************************************************************)
      NULL_TASK_ID = low(TASK_IDS);

      (**********************************************************************)
      (*  This is used to document the code where the task ID parameter to a *)
      (* procedure is needed by the compiler but not used by the code        *)
      (* because of the context of the call.                                 *)
      (**********************************************************************)
      ANY_TASK = NULL_TASK_ID;

const (**********************************************************************)
      (*  These priority declarations allow TASKING to create tasks that are *)
      (* guaranteed to have higher and lower  priorities  than  user  tasks. *)
      (* The obvious use of this is the null  task,  which  has  a  priority *)
      (* lower than all user tasks, and  tasks  which  are  executing  in  a *)
      (* scheduler-locked code region which have a priority higher than  all *)
      (* others.                                                             *)
      (**********************************************************************)
      HIGHEST_PRIORITY = succ(high(USER_PRIORITIES));
      LOWEST_PRIORITY  = pred(low(USER_PRIORITIES));

type  (**********************************************************************)
      (*  General task priority type which includes both user priorities and *)
      (* TASKING reserved priorities.                                        *)
      (**********************************************************************)
      PRIORITIES = LOWEST_PRIORITY..HIGHEST_PRIORITY;

const (**********************************************************************)
      (*  This variable keeps track of the  highest  priority  when  dynamic *)
      (* priority rotation is enabled (i.e., dynamic priorities).            *)
      (**********************************************************************)
      HIGHEST_DYNAMIC_PRIORITY : PRIORITIES = HIGHEST_PRIORITY;

type  (**********************************************************************)
      (*  This list enumerates the allowable priority comparisons  (used  to *)
      (* determine the next highest priority task to execute).               *)
      (**********************************************************************)
      COMPARISONS = (GREATER_THAN, LESS_THAN, EQUAL_TO);

const (**********************************************************************)
      (*  This is used to document the code where the address of  the  block *)
      (* parameter to a procedure is needed by the compiler but not used  by *)
      (* the code because of the context of the call.                        *)
      (**********************************************************************)
      THERE_IS_NO_BLOCK_TO_MATCH = nil;

      (**********************************************************************)
      (*  These are the hardware specific constants necessary to program the *)
      (* system clock chip (i8254) for TASKING and MS-DOS operation.         *)
      (**********************************************************************)
      TIMER_CLOCK_FREQUENCY      = 1193180;
      MSDOS_TIMER_0_VALUE        = $FFFF;
      ONE_MSEC_TIMER_0_VALUE     = round(TIMER_CLOCK_FREQUENCY * 0.001);
      DEFAULT_MSDOS_CLOCK_PERIOD = MSDOS_TIMER_0_VALUE / TIMER_CLOCK_FREQUENCY;
      TIMER_CONTROL_PORT         = $43;
      TIMER_0_DATA_PORT          = $40;
      SYSTEM_TIMER_CONTROL_WORD  = $34;

      (**********************************************************************)
      (*  Allows TASKING application to run over a month boundary and report *)
      (* the correct execution time.                                         *)
      (**********************************************************************)
      DAYS_IN : array[1..12] of integer=(31,28,31,30,31,30,31,31,30,31,30,31);

      (**********************************************************************)
      (*  This is used as a special delay value  which  indicates  that  the *)
      (* task is no longer (or never was) waiting for a delay to expire.     *)
      (**********************************************************************)
      NO_TIME_DELAY = -1;

      (**********************************************************************)
      (*  These values are used to compute the number of milliseconds that a *)
      (* task must 'wait' when it requests a WAIT_FOR_DELAY().               *)
      (**********************************************************************)
      MILLISECONDS_PER_SECOND = 1000;
      MILLISECONDS_PER_MINUTE = MILLISECONDS_PER_SECOND * 60;
      MILLISECONDS_PER_HOUR   = MILLISECONDS_PER_MINUTE * 60;
      MILLISECONDS_PER_DAY    = MILLISECONDS_PER_HOUR * 24;

      (**********************************************************************)
      (*  This is an interrupt that is revectored from its  original  vector *)
      (* to allow TASKING to monitor/control  software  interrupts.   MS-DOS *)
      (* allows user applications to use interrupts 0xF0-0xFF.  This  allows *)
      (* TASKING to prevent preempting MS-DOS (which is guaranteed to  crash *)
      (* if reentered).                                                      *)
      (**********************************************************************)
      REVECTORED_INTERRUPT_NUMBER = $FF;

      (**********************************************************************)
      (*  This allows the statistics functions to detemine how much  of  the *)
      (* task stack was used by the stack and how much is wasted.            *)
      (**********************************************************************)
      DEFAULT_STACK_VALUE = $5A5A;

      (**********************************************************************)
      (*  80x86  FLAGS  register  bit  definitions  used  to  test/set/clear *)
      (* individual bits in the register.                                    *)
      (**********************************************************************)
      CARRY_FLAG     = $0001;  PARITY_FLAG    = $0004;  AUXILIARY_FLAG = $0010;
      ZERO_FLAG      = $0040;  SIGN_FLAG      = $0080;  TRAP_FLAG      = $0100;
      INTERRUPT_FLAG = $0200;  DIRECTION_FLAG = $0400;  OVERFLOW_FLAG  = $0800;

      (**********************************************************************)
      (*  MS-DOS, system and hardware specific interrupt declarations (i.e., *)
      (* interrupt vector table entries).  This information was derived from *)
      (* a number of different sources and represents the most comprehensive *)
      (* PC interrupt usage description that could be compiled.              *)
      (**********************************************************************)
      DIVIDE_BY_ZERO_INTERRUPT_NUMBER          = $00;
      SINGLE_STEP_INTERRUPT_NUMBER             = $01;
      NMI_INTERRUPT_NUMBER                     = $02;
      BREAKPOINT_INTERRUPT_NUMBER              = $03;
      OVERFLOW_INTERRUPT_NUMBER                = $04;
      PRINT_SCREEN_INTERRUPT_NUMBER            = $05;
      INVALID_OPCODE_INTERRUPT_NUMBER          = $06;
      SYSTEM_TIMER_INTERRUPT_NUMBER            = $08;
      KEYBOARD_INTERRUPT_NUMBER                = $09;
      IRQ_3_INTERRUPT_NUMBER                   = $0B;
      IRQ_4_INTERRUPT_NUMBER                   = $0C;
      IRQ_5_INTERRUPT_NUMBER                   = $0D;
      IRQ_6_INTERRUPT_NUMBER                   = $0E;
      IRQ_7_INTERRUPT_NUMBER                   = $0F;
      VIDEO_SERVICES_INTERRUPT_NUMBER          = $10;
      EQUIPMSNT_CHECK_INTERRUPT_NUMBER         = $11;
      MEMORY_SIZE_INTERRUPT_NUMBER             = $12;
      DISK_SERVICES_INTERRUPT_NUMBER           = $13;
      SERIAL_COMM_INTERRUPT_NUMBER             = $14;
      CASSETTE_IO_INTERRUPT_NUMBER             = $15;
      KEYBOARD_IO_INTERRUPT_NUMBER             = $16;
      PRINTER_IO_INTERRUPT_NUMBER              = $17;
      BOOTSTRAP_INTERRUPT_NUMBER               = $19;
      TIME_OF_DAY_INTERRUPT_NUMBER             = $1A;
      KEYBOARD_BREAK_INTERRUPT_NUMBER          = $1B;
      CLOCK_INTERRUPT_NUMBER                   = $1C;
      VIDEO_PARAMS_POINTER                     = $1D;
      FLOPPY_DISK_PARAMS_POINTER               = $1E;
      VIDEO_GRAPHICS_POINTER                   = $1F;
      PROGRAM_TERMINATION_INTERRUPT_NUMBER     = $20;
      MSDOS_BIOS_FUNCTION_INTERRUPT_NUMBER     = $21;
      PROGRAM_TERMINATION_ADDRESS              = $22;
      CONTROL_C_ADDRESS                        = $23;
      CRITICAL_ERROR_ADDRESS                   = $24;
      ABSOLUTE_DISK_READ_INTERRUPT_NUMBER      = $25;
      ABSOLUTE_DISK_WRITE_INTERRUPT_NUMBER     = $26;
      TSR_INTERRUPT_NUMBER                     = $27;
      IDLE_HANDLER_INTERRUPT_NUMBER            = $28;
      MSDOS_TTY_HANDLER_INTERRUPT_NUMBER       = $29;
      MSDOS_NETWORK_INTERRUPT_NUMBER           = $2A;
      BATCH_EXECUTE_INTERRUPT_NUMBER           = $2E;
      MULTIPLEX_INTERRUPT_NUMBER               = $2F;
      CPM_JUMP_PART_1_POINTER                  = $30;
      CPM_JUMP_PART_2_POINTER                  = $31;
      MS_MOUSE_INTERRUPT_NUMBER                = $33;
      DISKETTE_REQUEST_INTERRUPT_NUMBER        = $40;
      FIXED_DISK_PARAMS_1_POINTER              = $41;
      EGA_GRAPHICS_POINTER                     = $43;
      FIXED_DISK_PARAMS_2_POINTER              = $46;
      USER_ALARM_ADDRESS                       = $4A;
      ROM_BIOS_ALARM_INTERRUPT_NUMBER          = $50;
      NETWORK_FUNCTIONS_INTERRUPT_NUMBER       = $5A;
      REVECTORED_19H_INTERRUPT_NUMBER          = $5B;
      NETWORK_USE_INTERRUPT_NUMBER             = $5C;
      LIM_EMS_INTERRUPT_NUMBER                 = $67;
      REAL_TIME_CLOCK_INTERRUPT_NUMBER         = $70;
      IRQ_2_REDIRECT_INTERRUPT_NUMBER          = $71;
      IRQ_10_INTERRUPT_NUMBER                  = $72;
      IRQ_11_INTERRUPT_NUMBER                  = $73;
      IBM_PS2_MOUSE_EVENT_INTERRUPT_NUMBER     = $74;
      COPROCESSOR_ERROR_INTERRUPT_NUMBER       = $75;
      HARD_DISK_EVENT_INTERRUPT_NUMBER         = $76;
      IRQ_15_INTERRUPT_NUMBER                  = $77;

      (**********************************************************************)
      (*  This set of interrupts are supposedly used by BASIC.  There is  no *)
      (* other  documentation  to  indicate  which  are  actually  interrupt *)
      (* handlers, in fact examining a 'standard' PC vector table shows that *)
      (* most of these are actually nil (i.e., 0000:0000).                   *)
      (**********************************************************************)
      BASIC_INTERRUPT_NUMBERS = [$80..$F0[;

      (**********************************************************************)
      (*  These interrupts are not used by MS-DOS and documented  as  usable *)
      (* by MS-DOS application programs.                                     *)
      (**********************************************************************)
      PROGRAM_USABLE_INTERRUPT_NUMBERS = [$60..$66, $F1..$FF];

      (**********************************************************************)
      (*  Members of this set of interrupts are only generated by the  80x86 *)
      (* hardware in response to abnormal program operation.                 *)
      (**********************************************************************)
      SYSTEM_INTERRUPT_NUMBERS =
        [
         DIVIDE_BY_ZERO_INTERRUPT_NUMBER,
         SINGLE_STEP_INTERRUPT_NUMBER,
         NMI_INTERRUPT_NUMBER,
         BREAKPOINT_INTERRUPT_NUMBER,
         OVERFLOW_INTERRUPT_NUMBER,
         INVALID_OPCODE_INTERRUPT_NUMBER
        ];

      (**********************************************************************)
      (*  This set of interrupts comprises the Operating Service handlers of *)
      (* MS-DOS.  This includes 'add-on' services provided by device drivers *)
      (* and/or TSR programs.  They are all software  interrupts  which  can *)
      (* only be accessed by the 80x86 machine instruction 'INT nn'.         *)
      (**********************************************************************)
      OS_SERVICES_INTERRUPT_NUMBERS =
        [
         PRINT_SCREEN_INTERRUPT_NUMBER,
         VIDEO_SERVICES_INTERRUPT_NUMBER,
         EQUIPMENT_CHECK_INTERRUPT_NUMBER,
         MEMORY_SIZE_INTERPUPT_NUMBER,
         DISK_SERVICES_INTERRUPT_NUMBER,
         SERIAL_COMM_INTERRUPT_NUMBER,
         CASSETTE_IO_INTERRUPT_NUMBER,
         KEYBOARD_IO_INTERRUPT_NUMBER,
         PRINTER_IO_INTERRUPT_NUMBER,
         TIME_OF_DAY_INTERRUPT_NUMBER,
         KEYBOARD_BREAK_INTERRUPT_NUMBER,
         CLOCK_INTERRUPT_NUMBER,
         PROGRAM_TERMINATION_INTERRUPT_NUMBER,
         MSDOS_BIOS_FUNCTION_INTERRUPT_NUMBER,
         ABSOLUTE_DISK_READ_INTERRUPT_NUMBER,
         ABSOLUTE_DISK_WRITE_INTERRUPT_NUMBER,
         TSR_INTERRUPT_NUMBER,
         IDLE_HANDLER_INTERRUPT_NUMBER,
         MSDOS_TTY_HANDLER_INTERRUPT_NUMBER,
         MSDOS_NETWORK_INTERRUPT_NUMBER,
         BATCH_EXECUTE_INTERRUPT_NUMBER,
         MULTIPLEX_INTERRUPT_NUMBER,
         MS_MOUSE_INTERRUPT_NUMBER,
         DISKETTE_REQUEST_INTERRUPT_NUMBER,
         ROM_BIOS_ALARM_INTERRUPT_NUMBER,
```

```
    NETWORK_FUNCTIONS_INTERRUPT_NUMBER,
    REVECTORED_19H_INTERRUPT_NUMBER,
    NETWORK_USE_INTERRUPT_NUMBER,
    LIM_EMS_INTERRUPT_NUMBER
   );

  (*****************************************************************)
  (*  The members of this set  of  interrupts  are  only  documented  as *)
  (* 'RESERVED'.  The use  of  any  of  these  interrupts  is  therefore *)
  (* forbidden (in order to maintain 100% compatibility).  Unfortunately *)
  (* some programs do use interrupts from this set.  An example of  this *)
  (* is DEC PathWorks Network driver which uses  $68.   Note  that  this *)
  (* unauthorized use can  cause  incompatibilities  with  TASKING  (and *)
  (* other programs).                                                    *)
  (*****************************************************************)
  RESERVED_INTERRUPT_NUMBERS =
    [
     $07, $2B..$2D, $32, $34..$3F, $42, $44..$45,
     $47..$49, $4B..$5B, $5D..$5F, $68..$6F, $78..$7F
    ];

  (*****************************************************************)
  (*  The members of this set of interrupts are used in an  incompatible *)
  (* manner.  These incompatibilities were discovered by testing,  there *)
  (* are no documents which describe the way in which the interrupts are *)
  (* used.  These interrupts are supposed to be used for other  purposes *)
  (* but are 'taken over' by the applications listed below and used in a *)
  (* manner that is not standard and is not completely understood.       *)
  (*****************************************************************)
  INCOMPATIBLE_INTERRUPT_NUMBERS =
    [
     (*---------------------------------------------------------------*)
     (*                   DEC PathWorks Network Driver Software         *)
     (*  It appears that as part of handling this interrupt  the  driver *)
     (* modifies the vector address so that the next interrupt goes to a *)
     (* different location in the driver.  This is definitely not a good *)
     (* approach and TASKING cannot deal with it.  Because  TASKING  has *)
     (* takes over the interrupt and the passes control  to  the  actual *)
     (* handler, if the handler then wants  the  next  interrupt  to  go *)
     (* somewhere else TASKING has no way of intercepting it.  The  only *)
     (* alternative is to let the driver 'fend for itself' and hope that *)
     (* it can handle reentrancy or that reentrancy never occurrs.       *)
     $68
     (*---------------------------------------------------------------*)
    ];

  (*****************************************************************)
  (*  The i8259A Programmable Interrupt Controller (PIC)  chip  must  be *)
  (* directly accessed by TASKING in order to support preemptive  multi- *)
  (* tasking.  These declarations  provide  the  necessary  PC  hardware *)
  (* specific information.                                               *)
  (*****************************************************************)
  NON_SPECIFIC_END_OF_INTERRUPT          = $20;
  PROGRAMMABLE_INTERRUPT_CONTROLLER_PORT = $20;

  (*****************************************************************)
  (*  Areas of MS-DOS Data area used to support blocked keyboard input.  *)
  (*****************************************************************)
  DOS_DATA_SEGMENT            = $0040;
  KYBD_BUFFER_HEAD_PTR_OFS    = $001A;
  KYBD_BUFFER_TAIL_PTR_OFS    = $001C;
  KYBD_BUFFER_BEGIN_PTR_OFS = $0080;
  KYBD_BUFFER_END_PTR_OFS     = $0082;

  (*****************************************************************)
  (*  General text processing control characters.                        *)
  (*****************************************************************)
  CR        = chr($0D);  (* Carriage Return                      *)
  LF        = chr($0A);  (* Line Feed                            *)
  NULL_CHR  = chr($00);  (* Marks the beginning of an extended key *)

type
  (*****************************************************************)
  (*  In order to allow efficient allocation/deallocation  of  TASK_IDs, *)
  (* TASKING maintains a list of sets of TASK_IDs.  Since each  set  can *)
  (* hold 256 elements, this list defines the number (and range) of such *)
  (* sets.  Note that '0' must be the start of the range.                *)
  (*****************************************************************)
  TASK_ID_SETS = 0..((MAXIMUM_NUMBER_OF_TASKS div 256) + 1);

  (*****************************************************************)
  (*  When tasks use message passing then this data structure is used to *)
  (* manage the message lists within each  mailbox.   The  size  of  the *)
  (* mailbox is dependent  upon  system  heap  space  only  because  the *)
  (* messaged are managed by this linked list structure.                 *)
  (*****************************************************************)
  MESSAGE_PTR = ^MSSSAGE;
  MESSAGE = record
    INFO_ADDRESS : pointer;
    NEXT         : MESSAGE_PTR;
  end; (* MESSAGE *)

  (*****************************************************************)
  (*  When tasks use message passing then this data structure is used to *)
  (* maintain the list of mailboxes.  Each mailbox must have  an  owner, *)
  (* i.e. the task that will receive the mail and each task can have  at *)
  (* most one mailbox.  Messages are managed  using  semaphores.   The *)
  (* number of tasks receiving mail is dependent upon  heap  space  only *)
  (* because the mailboxes are managed by this  linked  list  structure. *)
  (*  For performance improvements this could be a binary tree sorted by *)
  (* the OWNER field.  Performance will only become an issue  with  very *)
  (* large applications with many  message  passing  tasks.   Note  that *)
  (* performance will only degrade if the number of tasks  that  receive *)
  (* messages is large NOT if the number of  messages  being  passed  is *)
  (* large.                                                              *)
  (*****************************************************************)
  MAILBOX_PTR = ^MAILBOX;
  MAILBOX = record
    OWNER    : TASK_IDS;
    SEM      : SEMAPHORE;
    CONTENTS : MESSAGE_PTR;
    NEXT     : MAILBOX_PTR;
  end; (* MAILBOX *)

  (*****************************************************************)
  (*  The task context  is  saved  into  and  restored  from  this  data *)
  (* structure.  It is basically a representation of the 8086 registers. *)
  (* Remember all applications under DOS actually  run  on  an  8086  no *)
  (* matter what CPU is used in the PC.                                  *)
  (*****************************************************************)
  CONTEXT_REGISTERS = record
    AX : word; BX : word; CX : word; DX : word;
    BP : word; SI : word; DI : word;
    ES : word; DS : word;
    SS : word; SP : word;
    CS : word; IP : word;
    FLAGS     : word;
  end; (* CONTEXT_REGISTERS *)

  (*****************************************************************)
  (*  Basic pointer to a semaphore, used to find the semaphore that  the *)
  (* tasks are waiting on.  Internally semaphores are  distinguished  by *)
  (* their addresses.                                                    *)
  (*****************************************************************)
  SEMAPHORE_PTR = ^SEMAPHORE;

  (*****************************************************************)
  (*  This data structure is used to maintain the information about what *)
  (* (and if) a task is waiting for.  Since everything  except  time  is *)
  (* implemented as a semaphore a task waiting for a mail  message  (for *)
  (* example) would have a semaphore entry.  A task waiting for  a  time *)
  (* delay would have an absolute time entry.                            *)
  (*****************************************************************)
  WAITING_PTR = ^WAITS;
  WAITS = record
    ABSOLUTE_TIME : longint;
    SEM_PTR       : SEMAPHORE_PTR;
  end; (* WAITS *)

  (*****************************************************************)
  (*  This data structure is used to maintain a linked list of  periodic *)
  (* events used by the application.  Elements  are  added  and  removed *)
  (* from the list as a result of starting and stopping periodie events. *)
  (*****************************************************************)
  PERIODIC_EVENTS_PTR = ^PERIODIC_EVENTS;
  PERIODIC_EVENTS = record
    PERIOD    : longint;
    EVENT_PTR : ^EVENT;
    NEXT      : PERIODIC_EVENTS_PTR;
  end; (* PERIODIC_EVENTS *)

  (*****************************************************************)
  (*  This is the main TASKING data structure.   Each  task  created  is *)
  (* given a task control block, this contains all necessary information *)
  (* about the task including: save context (if it is blocked), priority,*)
  (* task identifier, personal stack and personal error handlers.        *)
  (*****************************************************************)
  TASK_CONTRDL_BLOCK_PTR = ^TASK_CDNTROL_BLOCK;
  TASK_CONTROL_BLOCK = record
    TASK_ID       : TASK_IDS;
    CLOCK_TICKS   : longint;
    PRIORITY      : PRIORITIES;
    STACK_PTR     : pointer;
    STACK_SIZE    : word;
    CONTEXT       : CONTEXT_REGISTEAS;
    WAITING_FOR   : WAITING_PTR;
    ERROR_HANDLER : array [SYSTEM_ERRORS] of HANDLER_PROCEDURE;
    NEXT          : TASK_CONTROL_BLOCK_PTR;
    PREVIOUS      : TASK_CONTROL_BLOCK_PTR;
  end; (* TASK_CONTROL_BLOCK *)

  (*****************************************************************)
  (*  This data structure is used to maintain a linked list  of  TASKING *)
  (* constructs used by the application.  This list is used to implement *)
  (* priority inheritance (which can be used to guarentee that  priority *)
  (* inversion is controlled).                                           *)
  (*****************************************************************)
  LOCKED_RESOURCE_PTR = ^LOCKED_RSSOURCES;
  LOCKED_RESOURCES = record
    OWNER         : TASK_CONTROL_BLOCK_PTR;
    PRIORITY      : PRIORITIES;
    SEMAPHORE_PTR : ^SEMAPHORE;
    NEXT          : LOCKED_RESOURCE_PTR;
  end; (* LOCKED_RESDURCES *)

  (*****************************************************************)
  (*  Tasks can be in any one of these states if they are  not  actually *)
  (* running.  That is, at all times one task is running and all  others *)
  (* can be found in one of these states.                                *)
  (*  BLOCKED means that the task is waiting for another task to do some-*)
  (* thing which will make it READY (signal event, semaphore, etc.).     *)
  (*  DELAYED means that the task is waiting for a time-out to occur.    *)
  (*  READY means that the task has no reason to be not  running  except *)
  (* that the scheduler has not given control to it yet (this  might  be *)
  (* because its priority is lower than another READY task).             *)
  (*****************************************************************)
  TASK_STATES = (BLOCKED, OELAYED, READY);

  (*****************************************************************)
  (*  These values are used to allow a single queue  management  routine *)
  (* to know why a task is being  removed  from  a  queue.   The  search *)
  (* criteria depends on the reason for the removal.                     *)
  (*****************************************************************)
  QUEUE_REMOVAL_REASONS =
    (
     BECAUSE_THE_BLOCK_WAS_SIGNALLED,
     BECAUSE_THE_DELAY_HAS_COMPLETED,
     UNCONDITIONALLY
    );

  (*****************************************************************)
  (*  This data structure is used to maintain the  priority  queues  for *)
  (* each of the allowable task states.                                  *)
  (*****************************************************************)
  QUEUES = record HEAD, TAIL : TASK_CONTROL_BLOCK_PTR; end; (* QUEUES *)

  (*****************************************************************)
  (*  This structure is used to manage the interrupts that TASKING  must *)
  (* chain into.  Because it is a variant record the fields overlap  and *)
  (* allow direct decomposition of a pointer into segment and offset.    *)
  (*****************************************************************)
  HANDLEAS = record
    case integer of
      0 : (OFFSET, SEGMENT : word);
      1 : (VECTOR          : pointer);
    end; (* HANDLER *)

const
  (*****************************************************************)
  (*  This variable contains the doubly  linked  list  of  task  control *)
  (* blocks for tasks which are ready to run.                            *)
  (*****************************************************************)
  READY_QUEUE : QUEUES   = (HEAD : nil; TAIL : nil);

  (*****************************************************************)
  (*  This variable contains the doubly  linked  list  of  task  control *)
  (* blocks for tasks which are blocked waiting for a semaphore,  event, *)
  (* binary semaphore, condition variable, message, mouse  action  or *)
  (* keyboard action.                                                    *)
  (*****************************************************************)
  BLOCKED_QUEUE : QUEUES   = (HEAD : nil; TAIL : nil);

  (*****************************************************************)
  (*  This variable contains the doubly  linked  list  of  task  control *)
  (* blocks for tasks which are waiting for a delay to expire.           *)
  (*****************************************************************)
  DELAYED_QUEUE : QUEUES   = (HEAD : nil; TAIL : nil);

  (*****************************************************************)
  (*  This pointer will be nil unless the user has installed a  task  to *)
  (* handle the mouse, at that time an event will be  created  and  this *)
  (* pointer will then point to it.  The user never knows that the mouse *)
  (* is handled as an event.                                             *)
  (*****************************************************************)
  USER_MOUSE_EVENT_PTR : ^EVENT = nil;

  (*****************************************************************)
  (*  This variable is used when blocked console I/O is being used.  The *)
  (* semaphore is created and maintained by TASKING, the application  is *)
  (* only aware of the blocking, not the method used to achieve it.      *)
  (*****************************************************************)
  USER_KEYBOARD_SEM_PTR : SEMAPHDRE_PTR = nil;

  (*****************************************************************)
  (*  This variable is the head of the linked list of periodic events.   *)
  (*****************************************************************)
  PERIODIC_EVENTS_LIST_PTR : PERIDDIC_EVENTS_PTR = nil;

  (*****************************************************************)
  (*  This variable is the head of the linked list of TASKING constructs *)
```

```
      (* used by the application (used when priority inheritance is enabled).*)
      (**********************************************************************)
      LOCKED_RESOURCE_LIST_PTR : LOCKED_RESOURCE_PTR = nil;

      (**********************************************************************)
      (*  These variables are used when TASKING statistics are gathered to *)
      (* provide the user with various system parameters which can be of *)
      (* interest to the TASKING application developer.                   *)
      (**********************************************************************)
      TASKING_STATISTICS : record
        START_TIME                        : real;
        NUMBER_OF_PERIODIC_EVENTS         : longint;
        NUMBER_OF_PERIODIC_EVENTS_MISSED  : longint;
        CONTEXT_SWITCHES                  : longint;
        COOPERATIVE_CONTEXT_SWITCHES      : longint;
        TOTAL_AVAILABLE_HEAP              : longint;
        SERVICE_CALLS                     : array[$00..$FF] of longint;
        HARDWARE_INTERRUPTS               : array[$00..$0F] of longint;
      end (* TASKING_STATISTICS *) =
        (
         START_TIME                        : 0.0;
         NUMBER_OF_PERIODIC_EVENTS         : 0;
         NUMBER_OF_PERIODIC_EVENTS_MISSED  : 0;
         CONTEXT_SWITCHES                  : 0;
         COOPERATIVE_CONTEXT_SWITCHES      : 0;
         TOTAL_AVAILABLE_HEAP              : 0;
         SERVICE_CALLS                     :
           (
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
           );
         HARDWARE_INTERRUPTS                   (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
        );
      (**********************************************************************)
      (*  This variable keeps track of the nesting level of the scheduler *)
      (* 'locks' that have occured, its value can not be negative.        *)
      (**********************************************************************)
      IN_PREEMPTABLE_REGION : integer = 0;

(**************************************************************************)
(*                     Default Error Handlers (Preview)                    *)
(*$F+  Handlers must be 'far' because they are used as procedural variables *)
(**************************************************************************)

var   (**********************************************************************)
      (*  This file is created (in the same directory as the application *)
      (* executable if LOG_ERRORS_TO_FILE is true) and contains a log of all *)
      (* errors and warnings detected by TASKING.                          *)
      (**********************************************************************)
      ERROR_LOG : text;

procedure TASK_ALREADY_ACTIVE_ERROR_HANDLER     (TASK_ID : TASK_IDS); forward;
procedure INSUFFICIENT_RESOURCES_ERROR_HANDLER  (TASK_ID : TASK_IDS); forward;
procedure TASK_IS_NOT_ACTIVE_ERROR_HANDLER      (TASK_ID : TASK_IDS); forward;
procedure TASK_ALREADY_SUSPENDED_ERROR_HANDLER  (TASK_ID : TASK_IDS); forward;
procedure ILLEGAL_TASK_ID_ERROR_HANDLER         (TASK_ID : TASK_IDS); forward;
procedure ILLEGAL_OPERATION_ERROR_HANDLER       (TASK_ID : TASK_IDS); forward;
procedure DEADLOCK_DETECTED_ERROR_HANDLER;                            forward;

(**************************************************************************)
(*                  End of Default Error Handlers (Preview)                *)
(*$F-                                                                      *)
(**************************************************************************)

(**************************************************************************)
(*                           Null Task Definition                          *)
(**************************************************************************)
procedure REQUEUE_AND_RESCHEDULE
  (TASK : TASK_IDS; RESULTING_STATE   TASK_STATES); forward;

procedure NULL_TASK;
(**************************************************************************)
(*   This task is used only to provide a place for the CPU to execute when *)
(* it has absolutely nothing else to do. All that this task tries to do is *)
(* pass control to tasks which are ready to run by continuously rescheduling *)
(* itself, since the null task has the lowest possible priority, all tasks *)
(* which are READY to run will be scheduled as soon as possible.           *)
(*   When it is not possible to pass control to another task (i.e., there *)
(* are no other tasks) then the application is terminated.                 *)
(**************************************************************************)

  begin (* NULL_TASK *)
    repeat
      asm sti end;
      (**********************************************************************)
      (*   If there are no other user tasks which are still active or *)
      (* blocked, then the only remaining task is the null task. If the *)
      (* null task is the only one left then terminate everything, else *)
      (* pass control to some other task (or at least attempt to).         *)
      (**********************************************************************)
      if (READY_QUEUE.HEAD   = nil) and (READY_QUEUE.TAIL   = nil) and
         (DELAYED_QUEUE.HEAD = nil) and (DELAYED_QUEUE.TAIL = nil)
        then
          if (BLOCKED_QUEUE.HEAD = nil) and (BLOCKED_QUEUE.TAIL = nil)
            then
              (**************************************************************)
              (*   Shut down everything (via the exit procedure chain).     *)
              (**************************************************************)
              halt(0)
            else
              (**************************************************************)
              (*   Deadlock is defined as the condition where all tasks that *)
              (* exist are blocked waiting for something to be signaled by *)
              (* another task. Since all tasks are blocked (and will not be *)
              (* scheduled to run) the action necessary to unblock any of *)
              (* the tasks cannot be satisfied. In TASKING, this deadlocked *)
              (* condition exists when there are no READY or DELAYED tasks *)
              (* and all existing tasks are BLOCKED on things *other than* *)
              (* keyboard or mouse events (which can be signaled as a result *)
              (* of interrupts).                                             *)
              (**************************************************************)
              if (USER_KEYBOARD_SEM_PTR = nil) and
                 (USER_MOUSE_EVENT_PTR = nil)
                then
                  DEADLOCK_DETECTED_ERROR_HANDLER
                else
                  (* for the dangling 'else' *)
        else
          (******************************************************************)
          (*   Continuously attempt to surrender the CPU to another task.  *)
          (******************************************************************)
          REQUEUE_AND_RESCHEDULE(NULL_TASK_ID, READY);
    until false;
  end; (* NULL_TASK *)

const NULL_TASK_STACK_SIZE = 511;

var   NULL_TASK_STACK : array[0..NULL_TASK_STACK_SIZE] of word;

const NULL : TASK_CONTROL_BLOCK =
        (
         TASK_ID       : NULL_TASK_ID;
         CLOCK_TICKS   : 0;
         PRIORITY      : LOWEST_PRIORITY;
         STACK_PTR     : addr(NULL_TASK_STACK);
         STACK_SIZE    : sizeof(NULL_TASK_STACK);
         CONTEXT       :
           (
            AX    : $0000; BX : $0000; CX : $0000; DX : $0000;
            BP    : $0000; SI : $0000; DI : $0000;
            ES    : $0000; DS : seg(NULL_TASK_STACK);
            SS    : seg(NULL_TASK_STACK[NULL_TASK_STACK_SIZE]);
            SP    : ofs(NULL_TASK_STACK[NULL_TASK_STACK_SIZE]);
            CS    : seg(NULL_TASK); IP : ofs(NULL_TASK);
            FLAGS : INTERRUPT_FLAG
           );
         WAITING_FOR   : nil;
         ERROR_HANDLER :
           (
            TASK_ALREADY_ACTIVE_ERROR_HANDLER,
            INSUFFICIENT_RESOURCES_ERROR_HANDLER,
            TASK_IS_NOT_ACTIVE_ERROR_HANDLER,
            TASK_ALREADY_SUSPENDED_ERROR_HANDLER,
            ILLEGAL_TASK_ID_ERROR_HANDLER,
            ILLEGAL_OPERATION_ERROR_HANDLER
           );
         NEXT          : nil;
         PREVIOUS      : nil
        );
(**************************************************************************)
(*                        End of Null Task Definition                      *)
(**************************************************************************)

const (**********************************************************************)
      (*  This variable always points to the currently running task's task *)
      (* control block. There can only be one task which is running at any *)
      (* point in time. This variable will never be nil.                  *)
      (**********************************************************************)
      RUNNING_TASK_PTR . TASK_CONTROL_BLOCK_PTR = @NULL;

      (**********************************************************************)
      (*  This variable contains a pointer to the beginning of the list of *)
      (* mailboxes, of course until a message is sent or a receive is *)
      (* attempted there are no mailboxes.                                 *)
      (**********************************************************************)
      MAILBOX_LIST : MAILBOX_PTR = nil;

      (**********************************************************************)
      (*  This variable is used to hold the number of milliseconds which *)
      (* elapsed since the program started running. It is the basis for all *)
      (* task delay calculations.                                          *)
      (**********************************************************************)
      MILLISECOND_TICKS   longint = 0;

      (**********************************************************************)
      (*  This variable is used to prevent timer interrupts from occurring *)
      (* until all processing for the present interrupt has completed. This *)
      (* ensures that TASKING internal data structures are not compromised *)
      (* due to re-entrant timer interrupts (this can only occur when *)
      (* interrupt handling takes longer than the clock timer period).     *)
      (**********************************************************************)
      TIMER_NSEOI_NEEDS_TO_BE_DONE : boolean = false;

      (**********************************************************************)
      (*  This variable is used to allow TASKING to know when it is 'fair' *)
      (* to preempt a running task. This ensures that all tasks get their *)
      (* full timeslice every time (of course this is only used when the *)
      (* tasking mode is preemptive).                                      *)
      (**********************************************************************)
      PREEMPTIVE_TIMESLICE : real = 0.0; (* Microseconds *)

      (**********************************************************************)
      (*  This is used to account for the possible inaccuracy between the *)
      (* target timeslice and the time period that the clock is capable of *)
      (* providing (due to integer count values).                          *)
      (**********************************************************************)
      _95_PERCENT = 0.95;

      (**********************************************************************)
      (*  This counter is used to prevent the TASKING application from *)
      (* terminating if there are child programs still executing. Failure *)
      (* to do this will almost certainly crash MS-DOS because a child will *)
      (* live past its parent, MS-DOS never expects this and internal MS-DOS *)
      (* data structures and memory allocation schemes do not support it.  *)
      (**********************************************************************)
      NUMBER_OF_CHILD_PROGRAMS : integer = 0;

      (**********************************************************************)
      (*  This flag is set once the application 'main' has terminated. This *)
      (* ensures that a task created in 'main' does not cause a reschedule *)
      (* (rescheduling 'main' is meaningless because it is not a task).    *)
      (**********************************************************************)
      ALLOW_RESCHEDULE_IN_CREATE : boolean = false;

      (**********************************************************************)
      (*  When TASKING is started there are no mouse events to report (hence *)
      (* the initialization values). This variable is used to transfer *)
      (* MS-Mouse parameters from the TASKING mouse handler to the user task *)
      (* (if and only if a mouse task has been installed).                 *)
      (**********************************************************************)
      MOUSE_INFO_FROM_HANDLER : MS_MOUSE.MOUSE_PARAMETERS =
        (
         ACTIVITY_MASK            : 0;
         BUTTON_STATES            : 0;
         VERTICAL_TEXT_POSITION   : 0;
         HORIZONTAL_TEXT_POSITION : 0
        );

      (**********************************************************************)
      (*  This set of interrupts are *all* revectored to the same interrupt *)
      (* handler which determines the actual interrupt number, disables *)
      (* preemption and then dispatches to the actual interrupt handler. *)
      (* Only *software* interrupts can be included in this set.           *)
      (**********************************************************************)
      PROTECTED_INTERRUPTS =
          OS_SERVICES_INTERRUPT_NUMBERS          (* System Software Interrupts *)
        + PROGRAM_USABLE_INTERRUPT_NUMBERS       (* User        "          "   *)
        - SYSTEM_INTERRUPT_NUMBERS               (* System Hardware        "   *)
        + RESERVED_INTERRUPT_NUMBERS             (* Unused      ?          "   *)
        + BASIC_INTERRUPT_NUMBERS                (* BASIC       ?          "   *)
        - INCOMPATIBLE_INTERRUPT_NUMBERS         (* Non-standard usage     "   *)
        - [REVECTORED_INTERRUPT_NUMBER];         (* TASKING Generic Interrupt  *)

var   (**********************************************************************)
      (*  When TASKING is started there are no TASK_IDs allocated so the set *)
      (* of allocated task identifiers must be the empty set. The task *)
      (* identifier of the null task is not included in this set because it *)
      (* cannot be allocated to user tasks.                                *)
      (**********************************************************************)
      ALLOCATED_TASK_IDS : array[TASK_ID_SETS] of set of byte;

      (**********************************************************************)
      (*  Loop control variable to initialize allocated TASK_ID sets.      *)
      (**********************************************************************)
      TASK_ID_SET : TASK_ID_SETS;

      (**********************************************************************)
      (*  This variable contains the address of the next exit procedure in *)
      (* the exit call chain. This value is restored when TASKING *)
      (* terminates.                                                       *)
      (**********************************************************************)
      SAVE_EXIT : pointer;
```

```pascal
    (*****************************************************************)
    (*  Pointer to original clock interrupt handler, used to allow MS-DOS *)
    (* to see clock ticks at 18.2 Hz rate (initialized at startup).      *)
    (*****************************************************************)
    CLOCK_VECTOR : pointer;

    (*****************************************************************)
    (*  This variable contains the original address of the interrupt that *)
    (* TASKING 'takes over', it is saved here during initialization so *)
    (* that it can be restored upon termination.                         *)
    (*****************************************************************)
    OLD_REVECTORED_INTERRUPT : HANDLERS;

    (*****************************************************************)
    (*  This variable contains information used to manage interrupts which *)
    (* originate from hardware sources.  This is necessary because add-on *)
    (* card driver software may (or may not) support preemption, this *)
    (* guarantees that it won't be preempted in case reentrancy will cause *)
    (* the driver to crash.                                               *)
    (*****************************************************************)
    HARDWARE_HANDLER : array[D..15] of HANDLERS;

    (*****************************************************************)
    (*  This variable contains the addresses of the next MS-DOS function *)
    (* dispatcher in the call chains.  These are restored when TASKING *)
    (* terminates.                                                        *)
    (*****************************************************************)
    INTERRUPT_VECTOR_HANDLER : array[$DD..$FF] of HANDLERS;

    (*****************************************************************)
    (*  These variables define TASKING's view of clock timer ticks which *)
    (* TASKING calculates based on the requested timeslice value.  (Don't *)
    (* worry...before termination it is restored back to what MS-DOS *)
    (* expects it to be).                                                 *)
    (*****************************************************************)
    TIMER_PERIOD : real; (* Microseconds *)
    NEW_TIMER_D_VALUE : word;

    (*****************************************************************)
    (*  The TASKING configuration is copied from the default (user) values *)
    (* just prior to beginning multi-tasking, this variable holds the *)
    (* configuration parameters used by TASKING.                         *)
    (*****************************************************************)
    ACTUAL_TASKING_CONFIGURATION : CONFIGURATION;

(*********************************************************************)
(*                           Non-User functions                       *)
(*********************************************************************)

function COMPARE_PRIORITIES
(*********************************************************************)
  (
   PRIORITY_A : PRIORITIES;
   COMPARISON : COMPARISONS;
   PRIORITY_B : PRIORITIES
  ) : boolean;
(*-------------------------------------------------------------------*)
(*   This routine compares the relative priorities passed in based on the *)
(* highest dynamic priority *at this time*.  This is necessary so that a *)
(* priority rotation scheme can be used.  Note that for static priorities *)
(* this routine will also return meaningful results (although direct *)
(* comparisons are possible).                                            *)
(*********************************************************************)

  var   (*************************************************************)
        (*  Converted priorities which can then be directly compared for *)
        (* relative value.                                                *)
        (*************************************************************)
        NORMALIZED_A, NORMALIZED_B : integer;

  begin (* COMPARE_PRIORITIES *)
    case PRIORITY_A of
      LOWEST_PRIORITY  : NORMALIZED_A := -maxint;
      HIGHEST_PRIORITY : NORMALIZED_A := +maxint;
      else
        if PRIORITY_A > HIGHEST_DYNAMIC_PRIORITY
          then
            NORMALIZED_A := integer(PRIORITY_A) mod
                            integer(HIGHEST_DYNAMIC_PRIORITY)
          else if PRIORITY_A = HIGHEST_DYNAMIC_PRIORITY then
            NORMALIZED_A := integer(HIGHEST_DYHANIC_PRIORITY)
          else (* PRIORITY_A < HIGHEST_DYNAMIC_PRIORITY *)
            NORMALIZED_A := PRIORITY_A + (HIGHEST_PRIORITY - 1 -
                                          integer(HIGHEST_DYNAMIC_PRIORITY));
    end; (* case...of *)
    case PRIORITY_B of
      LOWEST_PRIORITY  : NORMALIZED_B := -maxint;
      HIGHEST_PRIORITY : NORMALIZED_B := +maxint;
      else
        if PRIORITY_A > HIGHEST_DYNAMIC_PRIORITY
          then
            NORMALIZED_B := integer(PRIORITY_B) mod
                            integer(HIGHEST_DYNAMIC_PRIORITY)
          else if PRIORITY_B = HIGHEST_DYNAMIC_PRIORITY then
            NORMALIZED_B := integer(HIGHEST_DYHANIC_PRIORITY)
          else (* PRIORITY_B < HIGHEST_DYNAMIC_PRIORITY *)
            NORMALIZED_B := PRIORITY_B + (HIGHEST_PRIORITY - 1 -
                                          integer(HIGHEST_DYNAMIC_PRIORITY));
    end; (* case...of *)

    case COMPARISON of
      GREATER_THAN : COMPARE_PRIORITIES := NORMALIZED_A > NORMALIZED_B;
      LESS_THAN    : COMPARE_PRIORITIES := NORMALIZED_A < NORMALIZED_B;
      EQUAL_TO     : COMPARE_PRIORITIES := NORMALIZED_A = NORMALIZED_B;
    end; (* case...of *)
  end; (* COMPARE_PRIORITIES *)

procedure INSERT_INTO
(*********************************************************************)
  (var QUEUE : QUEUES; NEW_TCB_PTR : TASK_CONTROL_BLOCK_PTR);
(*-------------------------------------------------------------------*)
(*   This routine inserts the specified task control block into the queue *)
(* specified using the task priority to insert the task into the correct *)
(* place in the priority gueue.                                          *)
(*********************************************************************)

  var   (*************************************************************)
        (*  Task Control Block pointer which is 'walked' through the queue *)
        (* in search of the proper insertion point.                       *)
        (*************************************************************)
        WALKING_PTR : TASK_CONTROL_BLOCK_PTR;

        (*************************************************************)
        (*  Temporary pointer needed to perform insertion between two task *)
        (* control blocks.                                                *)
        (*************************************************************)
        TEMP_PTR : TASK_CONTROL_BLOCK_PTR;

  begin (* INSERT_INTO *)
    asm
        pushf
        cli
    end; (* asm *)
    (* Find place in queue to insert into *)
    WALKING_PTR := QUEUE.TAIL;
    while (WALKING_PTR <> nil) and
          COMPARE_PRIORITIES(NEW_TCB_PTR^.PRIORITY, GREATER_THAN,
                             WALKING_PTR^.PRIORITY] do
      WALKING_PTR := WALKING_PTR^.PREVIOUS;

    (* Insert element *)
    if WALKING_PTR = nil
      then (* This task is highest priority *)
        if QUEUE.HEAD = nil
          then (* the list is empty *)
            begin
              QUEUE.HEAD             := NEW_TCB_PTR;
              QUEUE.HEAD^.PREVIOUS   := nil;
              QUEUE.TAIL             := NEW_TCB_PTR;
              QUEUE.TAIL^.NEXT       := nil;
            end (* if...then *)
          else
            begin
              NEW_TCB_PTR^.NEXT      := QUEUE.HEAD;
              QUEUE.HEAD^.PREVIOUS   := NEW_TCB_PTR;
              QUEUE.HEAD             := NEW_TCB_PTR;
              QUEUE.HEAD^.PREVIOUS   := nil;
            end (* if...then...else *)
      else
        if WALKING_PTR = QUEUE.TAIL
          then (* This task is lowest priority *)
            begin
              QUEUE.TAIL^.NEXT       := NEW_TCB_PTR;
              NEW_TCB_PTR^.PREVIOUS  := QUEUE.TAIL;
              QUEUE.TAIL             := NEW_TCB_PTR;
              QUEUE.TAIL^.NEXT       := nil;
            end (* if...then *)
          else (* This task is somewhere in the list *)
            begin
              TEMP_PTR               := WALKING_PTR^.NEXT;
              WALKING_PTR^.NEXT      := NEW_TCB_PTR;
              NEW_TCB_PTR^.PREVIOUS  := WALKING_PTR;
              NEW_TCB_PTR^.NEXT      := TEMP_PTR;
              TEMP_PTR^.PREVIOUS     := NEW_TCB_PTR;
            end; (* if...then...else *)
    asm popf end;
  end; (* INSERT_INTO *)

function REMOVE_FROM_HEAD
(*********************************************************************)
  (var QUEUE : QUEUES) : TASK_CONTROL_BLOCK_PTR;
(*-------------------------------------------------------------------*)
(*   This routine removes one task control block from the head of the *)
(* specified queue and returns that pointer.  Note that this works even if *)
(* the queue is empty because then nil is returned.                      *)
(*********************************************************************)

  var   (*************************************************************)
        (*  Temporary pointer necessary for knowing the value of the 'head' *)
        (* before the removal was performed.                              *)
        (*************************************************************)
        TEMP_PTR : TASK_CONTROL_BLOCK_PTR;

  begin (* REMOVE_FROM_HEAD *)
    asm
        pushf
        cli
    end; (* asm *)
    TEMP_PTR := QUEUE.HEAD;
    if QUEUE.HEAD <> QUEUE.TAIL
      then
        begin
          QUEUE.HEAD           := QUEUE.HEAD^.NEXT;
          QUEUE.HEAD^.PREVIOUS := nil;
        end (* if...then *)
      else
        begin
          QUEUE.HEAD := nil;
          QUEUE.TAIL := nil;
        end; (* if...then *)
    if TEMP_PTR <> nil then
      begin
        TEMP_PTR^.NEXT     := nil;
        TEMP_PTR^.PREVIOUS := nil;
      end; (* if...then *)
    REMOVE_FROM_HEAD := TEMP_PTR;
    asm popf end;
  end; (* REMOVE_FROM_HEAD *)

function FIND_AND_REMOVE
(*********************************************************************)
  (    TASK    : TASK_IDS;
   var QUEUE   : QUEUES;
       REASON  : QUEUE_REMOVAL_REASONS;
       SEM_PTR : SEMAPHORE_PTR
  ) : TASK_CONTROL_BLOCK_PTR;
(*-------------------------------------------------------------------*)
(*   This routine searches the specified queue for the task control block *)
(* which matches the search criteria implied by the reason that the task is *)
(* being removed.  If a matching task control block is found then it is *)
(* returned, nil is returned if no match is found in the specified queue. *)
(*********************************************************************)

  var   (*************************************************************)
        (*  Task Control Block pointer which is 'walked' through the queue *)
        (* in search of the block which is to be removed.                 *)
        (*************************************************************)
        WALKING_PTR : TASK_CONTROL_BLOCK_PTR;

  begin (* FIND_AND_REMOVE *)
    asm
        pushf
        cli
    end; (* asm *)
    WALKING_PTR := QUEUE.HEAD;
    case REASON of
      UNCONDITIONALLY :
        while (WALKING_PTR <> nil) and (WALKING_PTR^.TASK_ID <> TASK) do
          WALKING_PTR := WALKING_PTR^.NEXT;
      BECAUSE_THE_BLOCK_WAS_SIGNALLED :
        while (WALKING_PTR <> nil) and
              ((WALKING_PTR^.WAITING_FOR = nil) or
               ((WALKING_PTR^.WAITING_FOR <> nil) and
                (WALKING_PTR^.WAITING_FOR^.SEM_PTR <> SEM_PTR)
               )
              ) do
          WALKING_PTR := WALKING_PTR^.NEXT;
      BECAUSE_THE_DELAY_HAS_COMPLETED :
        while (WALKING_PTR <> nil) and
              ((WALKING_PTR^.WAITING_FOR^.ABSOLUTE_TIME = NO_TIME_DELAY) or
               (WALKING_PTR^.WAITING_FOR^.ABSOLUTE_TIME > MILLISECOND_TICKS)
              ) do
          WALKING_PTR := WALKING_PTR^.NEXT;
    end; (* case...of *)
    if ((REASON <> BECAUSE_THE_BLOCK_WAS_SIGNALLED) and
        (WALKING_PTR <> nil)
       ) or
       ((REASON = BECAUSE_THE_BLOCK_WAS_SIGNALLED) and
        (WALKING_PTR^.WAITING_FOR <> nil) and
        (WALKING_PTR^.WAITING_FOR^.SEM_PTR = SEM_PTR)
       )
      then
        begin
          if (WALKING_PTR^.PREVIOUS <> nil] and (WALKING_PTR^.NEXT <> nil)
            then
              begin
                WALKING_PTR^.PREVIOUS^.NEXT := WALKING_PTR^.NEXT;
```

```
          WALKING_PTR^.NEXT^.PREVIOUS := WALKING_PTR^.PREVIOUS;
        end |* if...then *|
      else if |WALKING_PTR^.PREVIOUS = nil| and
              |WALKING_PTR^.NEXT <> nil|
             then
               begin
                 QUEUE.HEAD            := QUEUE.HEAD^.NEXT;
                 QUEUE.HEAD^.PREVIOUS := nil;
               end |* if...then *|
             else if |WALKING_PTR^.PREVIOUS <> nil| and
                     |WALKING_PTR^.NEXT = nil|
                    then
                      begin
                        QUEUE.TAIL       := QUEUE.TAIL^.PREVIOUS;
                        QUEUE.TAIL^.NEXT := nil;
                      end |* if...then *|
                    else
                      begin
                        QUEUE.HEAD := nil;
                        QUEUE.TAIL := nil;
                      end; |* if...then...else *|
    WALKING_PTR^.NEXT     := nil;
    WALKING_PTR^.PREVIOUS := nil;
    if REASON <> UNCONDITIONALLY then
      begin
        WALKING_PTR^.WAITING_FOR^.ABSOLUTE_TIME := NO_TIME_DELAY;
        WALKING_PTR^.WAITING_FOR^.SEM_PTR       := nil;
      end; |* if...then *|
    FIND_AND_REMOVE := WALKING_PTR;
  end |* if...then *|
else
  FIND_AND_REMOVE := nil;
asm popf end;
end; |* FIND_AND_REMOVE *|

function FIND_OR_CREATE_MBOX_FOR
|**************************************************************************|
  |TASK · TASK_IDS| : MAILBOX_PTR;
|*------------------------------------------------------------------------*|
|*   This routine searches the system mailbox list and returns a pointer  to *|
|* the mailbox owned by the specified task.  If the task does not yet own  a *|
|* mailbox then one is created and added to the mailbox list.              *|
|**************************************************************************|

  var   |******************************************************************|
        |*  Pointer to mailbox structures which is 'walked* through the list *|
        |* in search of the matching owner.                                 *|
        |******************************************************************|
        WALKING_PTR : MAILBOX_PTR;

        |******************************************************************|
        |*  Temporary return value |function name cannot be  used  on  right *|
        |* hand side of an assignment statement|.                           *|
        |******************************************************************|
        TEMP_PTR : MAILBOX_PTR;

  begin |* FIND_OR_CREATE_MBOX_FOR *|
    asm
        pushf
        cli
    end; |* asm *|
    WALKING_PTR := MAILBOX_LIST;
    while |WALKING_PTR <> nil| and |WALKING_PTR^.OWNER <> TASK| do
      WALKING_PTR := WALKING_PTR^.NEXT;
    if WALKING_PTR^.OWNER = TASK
      then
        TEMP_PTR := WALKING_PTR
      else
        begin
          if WALKING_PTR = nil
            then
              new|WALKING_PTR|
            else
              begin
                new|WALKING_PTR^.NEXT|;
                WALKING_PTR := WALKING_PTR^.NEXT;
              end; |* if...then...else *|
          TEMP_PTR := MAILBOX_LIST;
          MAILBOX_LIST := WALKING_PTR;
          MAILBOX_LIST^.NEXT := TEMP_PTR;
          TEMP_PTR := MAILBOX_LIST;
          with TEMP_PTR^ do
            begin
              OWNER    := TASK;
              SEM      := 0;
              CONTENTS := nil;
            end; |* with...do *|
        end; |* if..then...else *|
    FIND_OR_CREATE_MBOX_FOR := TEMP_PTR;
    asm popf end;
  end; |* FIND_OR_CREATE_MBOX_FOR *|

procedure REMOVE_MBOX_OF
|**************************************************************************|
  |TASK : TASK_IDS|;
|*------------------------------------------------------------------------*|
|*   This routine searches the system mailbox list and removes  the  mailbox *|
|* owned by the specified task.  If the task does not yet own a mailbox then *|
|* nothing is done.                                                         *|
|**************************************************************************|

  var   |******************************************************************|
        |*  Pointer to mailbox structures which is *walked' through the list *|
        |* in search of the matching owner.                                 *|
        |******************************************************************|
        WALKING_PTR : MAILBOX_PTR;

        |******************************************************************|
        |*  Temporary pointer used to re-arrange mailbox list links.         *|
        |******************************************************************|
        TEMP_PTR : MAILBOX_PTR;

  begin |* REMOVE_MBOX_OF *|
    asm
        pushf
        cli
    end; |* asm *|
    WALKING_PTR := MAILBOX_LIST;
    while |WALKING_PTR <> nil| and |WALKING_PTR^.OWNER <> TASK| do
      WALKING_PTR := WALKING_PTR^.NEXT;
    if WALKING_PTR^.OWNER = TASK then
      begin
        TEMP_PTR := WALKING_PTR;

        |* Patch link past element to delete *|
        WALKING_PTR := MAILBOX_LIST;
        while WALKING_PTR <> TEMP_PTR do
          WALKING_PTR := WALKING_PTR^.NEXT;
        WALKING_PTR^.NEXT := TEMP_PTR^.NEXT;

        |* Free memory allocated to deleted mailbox *|
        dispose|TEMP_PTR|;
      end; |* if..then *|
    asm popf end;
  end; |* REMOVE_MBOX_OF *|

procedure CLOCK_INTERCEPTOR; interrupt;
|**************************************************************************|
|*   This routine checks to see if the DELAYED queue is empty and if  it  is *|
|* not then it attempts to remove all tasks that have waited long enough and *|
|* put them into the READY queue.  After that has been  done  it  implements *|
|* preemptive multitasking if appropriate.                                  *|
|**************************************************************************|

  const |******************************************************************|
        |*  This  counter  is  used  to  determine  when  the  MS-DOS  clock *|
        |* interrupt handler should be called so that MS-DOS thinks that the *|
        |* timer is running at a 55 msec period.                            *|
        |******************************************************************|
        MSDOS_CLOCK_PERIOD : real = DEFAULT_MSDOS_CLOCK_PERIOD |Sec| * 1e6 / 2;

        |******************************************************************|
        |*  This counter is used to generate TASKING millisecond clock ticks *|
        |******************************************************************|
        MILLISECOND_PERIOD : real = 0.0; |uSec|

  var   |******************************************************************|
        |*  Task Control Block pointer used to release all tasks  that  have *|
        |* been delayed for the proper amount of time.                      *|
        |******************************************************************|
        DELAY_COMPLETED_TASK_PTR : TASK_CONTROL_BLOCK_PTR;

        |******************************************************************|
        |*  Pointer to the list of periodic events |possibly nil|.           *|
        |******************************************************************|
        WALKING_PTR : PERIODIC_EVENTS_PTR;

        |******************************************************************|
        |*  Pointer to Task Control Block of waiting task to be made ready.  *|
        |******************************************************************|
        WAITING_TASK_PTR : TASK_CONTROL_BLOCK_PTR;


  begin |* CLOCK_INTERCEPTOR *|
    LOCK_SCHEDULER;
    if ACTUAL_TASKING_CONFIGURATION.STATISTICS <> NO_STATISTICS then
      begin
        inc|TASKING_STATISTICS.HARDWARE_INTERRUPTS|0||;
        inc|RUNNING_TASK_PTR^.CLOCK_TICKS|;
      end; |* if..then *|

    |* Perform all local processing *|
    MILLISECOND_PERIOD := MILLISECOND_PERIOD + TIMER_PERIOD;
    if MILLISECOND_PERIOD >= ||1000 |uSec|| * _95_PERCENT| then
      begin
        MILLISECOND_PERIOD := MILLISECOND_PERIOD - |1000.0 |uSec||;
        inc|MILLISECOND_TICKS|;
      end; |* if...then *|

    if DELAYED_QUEUE.HEAD <> nil then
      begin
        DELAY_COMPLETED_TASK_PTR := FIND_AND_REMOVE
                                    |
                                     ANY_TASK,
                                     |From the| DELAYED_QUEUE,
                                     BECAUSE_THE_DELAY_HAS_COMPLETED,
                                     THERE_IS_NO_BLOCK_TO_MATCH
                                    |;
          if DELAY_COMPLETED_TASK_PTR <> nil then
            INSERT_INTO|READY_QUEUE, DELAY_COMPLETED_TASK_PTR|;
      end; |* if...then *|

    |* Find the place in the list to insert the event *|
    WALKING_PTR := PERIODIC_EVENTS_LIST_PTR;
    while WALKING_PTR <> nil do
      begin
        if |MILLISECOND_TICKS mod WALKING_PTR^.PERIOD| = 0 then
          with WALKING_PTR^ do
            begin
              if EVENT_PTR^ = SIGNALED then
                inc|TASKING_STATISTICS.NUMBER_OF_PERIODIC_EVENTS_MISSED|;
              WAITING_TASK_PTR := FIND_AND_REMOVE
                                  |
                                   ANY_TASK,
                                   |From the| BLOCKED_QUEUE,
                                   BECAUSE_THE_BLOCK_WAS_SIGNALLEO,
                                   |At| addr|EVENT_PTR^|
                                  |;
              if WAITING_TASK_PTR = nil
                then
                  EVENT_PTR^ := SIGNALED
                else
                  INSERT_INTO|READY_QUEUE, WAITING_TASK_PTR|;
              inc|TASKING_STATISTICS.NUMBER_OF_PERIODIC_EVENTS|;
            end; |* with...do *|
        WALKING_PTR := WALKING_PTR^.NEXT;
      end; |* while...do *|

    |* Call original interrupt handler? |once every 55 msec| *|
    MSDOS_CLOCK_PERIOD := MSDOS_CLOCK_PERIOD + TIMER_PERIOD;
    if MSDOS_CLOCK_PERIOD >= |DEFAULT_MSDOS_CLOCK_PERIOD * _95_PERCENT|
      then
        begin
          MSDOS_CLOCK_PERIOD := MSDOS_CLOCK_PERIOD -
                                DEFAULT_MSDOS_CLOCK_PERIOD * 1e6;

          asm
              pushf
              call  dword ptr ds:CLOCK_VECTOR
              cli
          end; |* asm *|
        end |* if...then *|
      else
        TIMER_NSEOI_NEEDS_TO_BE_DONE := true;
    UNLOCK_SCHEDULER;

    if |ACTUAL_TASKING_CONFIGURATION.TASKING_MODEL = PREEMPTIVE| and
       |IN_PREEMPTABLE_REGION = 0| then
      begin
        PREEMPTIVE_TIMBSLICE := PREEMPTIVE_TIMESLICE + TIMER_PERIOD;
        if PREEMPTIVE_TIMESLICE >=
           ACTUAL_TASKING_CONFIGURATION.TARGET_TIMESLICE
          then
            REQUEUE_AND_RESCHEDULE|RUNNING_TASK_PTR^.TASK_ID, READY|;
      end; |* if...then *|

    if TIMER_NSEOI_NEEDS_TO_BE_DONE then
      begin
        |* Issue NSEOI to PIC *|
        port[PROGRAMMABLE_INTERRUPT_CONTROLLER_PORT] :=
             NON_SPECIFIC_END_OF_INTERRUPT;
        TIMER_NSEOI_NEEDS_TO_BE_DONE := false;
      end; |* if...then *|
  end; |* CLOCK_INTERCEPTOR *|

procedure KEYBOARD_INTERCEPTOR; interrupt;
|**************************************************************************|
|*   This routine checks to see if the  keyboard  interrupt  that  was  just *|
|* detected corresponds to an  incoming  character  or  just  some  type  of *|
|* keyboard activity |like pressing the SHIFT, ALT, CTRL or *lock' key|.  If *|
|* the interrupt was because of a key pressed then the controlling semaphore *|
|* is signaled |if and only if the semaphore was created as a  result  of  a *|
|* task waiting for a key to be pressed|.                                   *|
|**************************************************************************|

  const |******************************************************************|
        |*  Pointer to pointer to the head of the keyboard buffer.          *|
        |******************************************************************|
        HEAD_PTR = ptr|DOS_DATA_SEGMENT, KYBD_BUFFER_HEAD_PTR_OFS|;
```

```
      (*********************************************************************)
      (*  Pointer to pointer to the tail of the keyboard buffer.          *)
      (*********************************************************************)
      TAIL_PTR = ptr(DOS_DATA_SEGMENT, KYBD_BUFFER_TAIL_PTR_OFS);

      (*********************************************************************)
      (*  Pointer to the beginning of the keyboard buffer (not the head).  *)
      (*********************************************************************)
      BUFFER_START = ptr(DOS_DATA_SEGMENT, KYBD_BUFFER_BEGIN_PTR_OFS);

      (*********************************************************************)
      (*  Pointer to the end of the keyboard buffer (not the tail).        *)
      (*********************************************************************)
      BUFFER_END = ptr(DOS_DATA_SEGMENT, KYBD_BUFFER_END_PTR_OFS);

  var (*********************************************************************)
      (*  Pointer to Task Control Block which is waiting on  the  keyboard *)
      (* semaphore (may very well be nil).                                 *)
      (*********************************************************************)
      WAITING_TASK_PTR : TASK_CONTROL_BLOCK_PTR;

      (*********************************************************************)
      (*  Offset into the buffer of the tail  *before*  the  interrupt  is *)
      (* processed by the MS-DOS handler (used to determine if a character *)
      (* was received).                                                    *)
      (*********************************************************************)
      TAIL : word;

      (*********************************************************************)
      (*  The number of characters received into the buffer, probably  one *)
      (* for each interrupt but could be more.                             *)
      (*********************************************************************)
      NUM_CHARS : integer;

      (*********************************************************************)
      (*  Pointer to original keyboard handler.                           *)
      (*********************************************************************)
      VECTOR : pointer;

  begin (* KEYBOARD_INTERCEPTOR *)
    VECTOR := HARDWARE_HANDLER[$1].VECTOR;
    if ACTUAL_TASKING_CONFIGURATION.STATISTICS <> NO_STATISTICS then
      inc(TASKING_STATISTICS.HARDWARE_INTERRUPTS[$1]);

    TAIL := word(TAIL_PTR^);

    asm
      (* Call original interrupt handler *)
      pushf
      call  dword ptr ss:VECTOR
      cli
    end; (* asm *)
    if (TAIL <> word(TAIL_PTR^)) and (USER_KEYBOARD_SEM_PTR <> nil) then
      begin (* tail changed, therefore a character was received *)
        WAITING_TASK_PTR := FIND_AND_REMOVE
                            (
                             ANY_TASK,
                             (From the) BLOCKED_QUEUE,
                             BECAUSE_THE_BLOCK_WAS_SIGNALLED,
                             (At) addr(USER_KEYBOARD_SEM_PTR^)
                            );
        if TAIL < word(TAIL_PTR^)
          then
            NUM_CHARS := word(TAIL_PTR^) - TAIL
          else
            NUM_CHARS := (word(BUFFER_END^) - TAIL) +
                         (word(TAIL_PTR^) - word(BUFFER_START^));
        (* Convert bytes in buffer to characters *)
        NUM_CHARS := NUM_CHARS div 2;
        if WAITING_TASK_PTR = nil
          then
            inc(USER_KEYBOARD_SEM_PTR^)
          else
            INSERT_INTO(READY_QUEUE, WAITING_TASK_PTR);
        inc(USER_KEYBOARD_SEM_PTR^, NUM_CHARS - 1);
      end; (* if...then *)
  end; (* KEYBOARD_INTERCEPTOR *)

procedure IRQ_2_INTERCEPTOR; interrupt;
(*************************************************************************)
(*  Hardware interrupt handler, ensures  that  preemption  is  disabled  and *)
(* passes control to the actual interrupt routine.                      *)
(*************************************************************************)

  var (*********************************************************************)
      (*  Pointer to original hardware interrupt handler.                 *)
      (*********************************************************************)
      VECTOR : pointer;

  begin (* IRQ_2_INTERCEPTOR *)
    LOCK_SCHEDULER;

    VECTOR := HARDWARE_HANDLER[$2].VECTOR;
    if ACTUAL_TASKING_CONFIGURATION.STATISTICS <> NO_STATISTICS then
      inc(TASKING_STATISTICS.HARDWARE_INTERRUPTS[$2]);

    asm
      (* Call original interrupt handler *)
      pushf
      call  dword ptr ss:VECTOR
      cli
    end; (* asm *)

    UNLOCK_SCHEDULER;
  end; (* IRQ_2_INTERCEPTOR *)

procedure IRQ_3_INTERCEPTOR; interrupt;
(*************************************************************************)
(*  Hardware interrupt handler, ensures  that  preemption  is  disabled  and *)
(* passes control to the actual interrupt routine.                      *)
(*************************************************************************)

  var (*********************************************************************)
      (*  Pointer to original hardware interrupt handler.                 *)
      (*********************************************************************)
      VECTOR   pointer;

  begin (* IRQ_3_INTERCEPTOR *)
    LOCK_SCHEDULER;

    VECTOR := HARDWARE_HANDLER[$3].VECTOR;
    if ACTUAL_TASKING_CONFIGURATION.STATISTICS <> NO_STATISTICS then
      inc(TASKING_STATISTICS.HARDWARE_INTERRUPTS[$3]);

    asm
      (* Call original interrupt handler *)
      pushf
      call  dword ptr ss:VECTOR
      cli
    end; (* asm *)

    UNLOCK_SCHEDULER;
  end; (* IRQ_3_INTERCEPTOR *)

procedure IRQ_4_INTERCEPTOR; interrupt;
(*************************************************************************)
(*  Hardware interrupt handler, ensures  that  preemption  is  disabled  and *)
(* passes control to the actual interrupt routine.                      *)
(*************************************************************************)

  var (*********************************************************************)
      (*  Pointer to original hardware interrupt handler.                 *)
      (*********************************************************************)
      VECTOR : pointer;

  begin (* IRQ_4_INTERCEPTOR *)
    LOCK_SCHEDULER;

    VECTOR := HARDWARE_HANDLER[$4].VECTOR;
    if ACTUAL_TASKING_CONFIGURRTION.STATISTICS <> NO_STATISTICS then
      inc(TASKING_STATISTICS.HARDWARE_INTERRUPTS[$4]);

    asm
      (* Call original interrupt handler *)
      pushf
      call  dword ptr ss:VECTOR
      cli
    end; (* asm *)

    UNLOCK_SCHEDULER;
  end; (* IRQ_4_INTERCEPTOR *)

procedure IRQ_5_INTERCEPTOR; interrupt;
(*************************************************************************)
(*  Hardware interrupt handler, ensures  that  preemption  is  disabled  and *)
(* passes control to the actual interrupt routine.                      *)
(*************************************************************************)

  var (*********************************************************************)
      (*  Pointer to original hardware interrupt handler.                 *)
      (*********************************************************************)
      VECTOR : pointer;

  begin (* IRQ_5_INTERCEPTOR *)
    LOCK_SCHEDULER;

    VECTOR := HARDWARE_HANDLER[$5].VECTOR;
    if ACTUAL_TASKING_CONFIGURATION.STATISTICS <> NO_STATISTICS then
      inc(TASKING_STATISTICS.HARDWARE_INTERRUPTS[$5]);

    asm
      (* Call original interrupt handler *)
      pushf
      call  dword ptr ss:VECTOR
      cli
    end; (* asm *)

    UNLOCK_SCHEDULER;
  end; (* IRQ_5_INTERCEPTOR *)

procedure IRQ_6_INTERCEPTOR; interrupt;
(*************************************************************************)
(*  Hardware interrupt handler, ensures  that  preemption  is  disabled  and *)
(* passes control to the actual interrupt routine.                      *)
(*************************************************************************)

  var (*********************************************************************)
      (*  Pointer to original hardware interrupt handler.                 *)
      (*********************************************************************)
      VECTOR : pointer;

  begin (* IRQ_6_INTERCEPTOR *)
    LOCK_SCHEDULER;

    VECTOR := HARDWARE_HANDLER[$6].VECTOR;
    if ACTUAL_TASKING_CONFIGURATION.STATISTICS <> NO_STATISTICS then
      inc(TASKING_STATISTICS.HARDWARE_INTERRUPTS[$6]);

    asm
      (* Call original interrupt handler *)
      pushf
      call  dword ptr ss:VECTOR
      cli
    end; (* asm *)

    UNLOCK_SCHEDULER;
  end; (* IRQ_6_INTERCEPTOR *)

procedure IRQ_7_INTERCEPTOR; interrupt;
(*************************************************************************)
(*  Hardware interrupt handler, ensures  that  preemption  is  disabled  and *)
(* passes control to the actual interrupt routine.                      *)
(*************************************************************************)

  var (*********************************************************************)
      (*  Pointer to original hardware interrupt handler.                 *)
      (*********************************************************************)
      VECTOR : pointer;

  begin (* IRQ_7_INTERCEPTOR *)
    LOCK_SCHEDULER;

    VECTOR := HARDWARE_HANDLER[$7].VECTOR;
    if ACTUAL_TASKING_CONFIGURATION.STATISTICS <> NO_STATISTICS then
      inc(TASKING_STATISTICS.HARDWARE_INTERRUPTS[$7]);

    asm
      (* Call original interrupt handler *)
      pushf
      call  dword ptr ss:VECTOR
      cli
    end; (* asm *)

    UNLOCK_SCHEDULER;
  end; (* IRQ_7_INTERCEPTOR *)

procedure IRQ_8_INTERCEPTOR; interrupt;
(*************************************************************************)
(*  Hardware interrupt handler, ensures  that  preemption  is  disabled  and *)
(' passes control to the actual interrupt routine.                      *)
(*************************************************************************)

  var (*********************************************************************)
      (*  Pointer to original hardware interrupt handler.                 *)
      (*********************************************************************)
      VECTOR : pointer;

  begin (* IRQ_8_INTERCEPTOR *)
    LOCK_SCHEDULER;

    VECTOR := HARDWARE_HANDLER[$8].VECTOR;
    if ACTUAL_TASKING_CONFIGURATION.STATISTICS <> NO_STATISTICS then
      inc(TASKING_STATISTICS.HARDWARE_INTERRUPTS[$8]);

    asm
      (* Call original interrupt handler *)
      pushf
      call  dword ptr ss:VECTOR
      cli
    end; (* asm *)

    UNLOCK_SCHEDULER;
  end; (* IRQ_8_INTERCEPTOR *)

procedure IRQ_9_INTERCEPTOR; interrupt;
(*************************************************************************)
(*  Hardware interrupt handler, ensures  that  preemption  is  disabled  and *)
```

```
(* passes control to the actual interrupt routine.                         *)
(***************************************************************************)

  var    (*****************************************************************)
         (*  Pointer to original hardware interrupt handler.              *)
         (*****************************************************************)
         VECTOR : pointer;

  begin (* IRQ_9_INTERCEPTOR *)
    LOCK_SCHEDULER;

    VECTOR := HARDWARE_HANDLER($9).VECTOR;
    if ACTUAL_TASKING_CONFIGURATION.STATISTICS <> NO_STATISTICS then
      inc(TASKING_STATISTICS.HARDWARE_INTERRUPTS($9));

    asm
      (* Call original interrupt handler *)
        pushf
        call  dword ptr ss:VECTOR
        cli
    end; (* asm *)

    UNLOCK_SCHEDULER;
  end; (* IRQ_9_INTERCEPTOR *)

procedure IRQ_A_INTERCEPTOR; interrupt;
(***************************************************************************)
(*  Hardware interrupt handler, ensures  that  preemption  is  disabled  and *)
(* passes control to the actual interrupt routine.                         *)
(***************************************************************************)

  var    (*****************************************************************)
         (*  Pointer to original hardware interrupt handler.              *)
         (*****************************************************************)
         VECTOR : pointer;

  begin (* IRQ_A_INTERCEPTOR *)
    LOCK_SCHEDULER;

    VECTOR := HARDWARE_HANDLER($A).VECTOR;
    if ACTUAL_TASKING_CONFIGURATION.STATISTICS <> NO_STATISTICS then
      inc(TASKING_STATISTICS.HARDWARE_INTERRUPTS($A));

    asm
      (* Call original interrupt handler *)
        pushf
        call  dword ptr ss:VECTOR
        cli
    end; (* asm *)

    UNLOCK_SCHEDULER;
  end; (* IRQ_A_INTERCEPTOR *)

procedure IRQ_B_INTERCEPTOR; interrupt;
(***************************************************************************)
(*  Hardware interrupt handler, ensures  that  preemption  is  disabled  and *)
(* passes control to the actual interrupt routine.                         *)
(***************************************************************************)

  var    (*****************************************************************)
         (*  Pointer to original hardware interrupt handler.              *)
         (*****************************************************************)
         VECTOR : pointer;

  begin (* IRQ_B_INTERCEPTOR *)
    LOCK_SCHEDULER;

    VECTOR := HARDWARE_HANDLER($B).VECTOR;
    if ACTUAL_TASKING_CONFIGURATION.STATISTICS <> NO_STATISTICS then
      inc(TASKING_STATISTICS.HARDWARE_INTERRUPTS($B));

    asm
      (* Call original interrupt handler *)
        pushf
        call  dword ptr ss:VECTOR
        cli
    end; (* asm *)

    UNLOCK_SCHEDULER;
  end; (* IRQ_B_INTERCEPTOR *)

procedure IRQ_C_INTERCEPTOR; interrupt;
(***************************************************************************)
(*  Hardware interrupt handler, ensures  that  preemption  is  disabled  and *)
(* passes control to the actual interrupt routine.                         *)
(***************************************************************************)

  var    (*****************************************************************)
         (*  Pointer to original hardware interrupt handler.              *)
         (*****************************************************************)
         VECTOR : pointer;

  begin (* IRQ_C_INTERCEPTOR *)
    LOCK_SCHEDULER;

    VECTOR := HARDWARE_HANDLER($C).VECTOR;
    if ACTUAL_TASKING_CONFIGURATION.STATISTICS <> NO_STATISTICS then
      inc(TASKING_STATISTICS.HARDWARE_INTERRUPTS($C));

    asm
      (* Call original interrupt handler *)
        pushf
        call  dword ptr ss:VECTOR
        cli
    end; (* asm *)

    UNLOCK_SCHEDULER;
  end; (* IRQ_C_INTERCEPTOR *)

procedure IRQ_D_INTERCEPTOR; interrupt;
(***************************************************************************)
(*  Hardware interrupt handler, ensures  that  preemption  is  disabled  and *)
(* passes control to the actual interrupt routine.                         *)
(***************************************************************************)

  var    (*****************************************************************)
         (*  Pointer to original hardware interrupt handler.              *)
         (*****************************************************************)
         VECTOR : pointer;

  begin (* IRQ_D_INTERCEPTOR *)
    LOCK_SCHEDULER;

    VECTOR := HARDWARE_HANDLER($D).VECTOR;
    if ACTUAL_TASKING_CONFIGURATION.STATISTICS <> NO_STATISTICS then
      inc(TASKING_STATISTICS.HARDWARE_INTERRUPTS($D));

    asm
      (* Call original interrupt handler *)
        pushf
        call  dword ptr ss:VECTOR
        cli
    end; (* asm *)

    UNLOCK_SCHEDULER;
  end; (* IRQ_D_INTERCEPTOR *)

procedure IRQ_E_INTERCEPTOR; interrupt;
(***************************************************************************)
(*  Hardware interrupt handler, ensures  that  preemption  is  disabled  and *)
(* passes control to the actual interrupt routine.                         *)
(***************************************************************************)

  var    (*****************************************************************)
         (*  Pointer to original hardware interrupt handler.              *)
         (*****************************************************************)
         VECTOR : pointer;

  begin (* IRQ_E_INTERCEPTOR *)
    LOCK_SCHEDULER;

    VECTOR := HARDWARE_HANDLER($E).VECTOR;
    if ACTUAL_TASKING_CONFIGURATION.STATISTICS <> NO_STATISTICS then
      inc(TASKING_STATISTICS.HARDWARE_INTERRUPTS($E));

    asm
      (* Call original interrupt handler *)
        pushf
        call  dword ptr ss:VECTOR
        cli
    end; (* asm *)

    UNLOCK_SCHEDULER;
  end; (* IRQ_E_INTERCEPTOR *)

procedure IRQ_F_INTERCEPTOR; interrupt;
(***************************************************************************)
(*  Hardware interrupt handler, ensures  that  preemption  is  disabled  and *)
(* passes control to the actual interrupt routine.                         *)
(***************************************************************************)

  var    (*****************************************************************)
         (*  Pointer to original hardware interrupt handler.              *)
         (*****************************************************************)
         VECTOR : pointer;

  begin (* IRQ_F_INTERCEPTOR *)
    LOCK_SCHEDULER;

    VECTOR := HARDWARE_HANDLER($F).VECTOR;
    if ACTUAL_TASKING_CONFIGURATION.STATISTICS <> NO_STATISTICS then
      inc(TASKING_STATISTICS.HARDWARE_INTERRUPTS($F));

    asm
      (* Call original interrupt handler *)
        pushf
        call  dword ptr ss:VECTOR
        cli
    end; (* asm *)

    UNLOCK_SCHEDULER;
  end; (* IRQ_F_INTERCEPTOR *)

procedure SW_INTERRUPT_INTERCEPTOR
(***************************************************************************)
  (_FLAGS, _CS, _IP, _AX, _BX, _CX, _DX, _SI, _DI, _DS, _ES, _BP : word);
  interrupt;
(*-------------------------------------------------------------------------*)
(*   This routine is used to intercept *all* software  interrupts  (accessed *)
(* via 'INT nn' instruction).  By inspecting the actual  opcode  before  the *)
(* return address,  the  interrupt  number  is  determined  and  control  is *)
(* transferred to that handler.  This is done to ensure that all MS-DOS (and *)
(* add-on) software is not preempted (i.e. reentered).                     *)
(***************************************************************************)

  const  (*****************************************************************)
         (*  Hexadecimal value for Intel 80x86 'INT' instruction.         *)
         (*****************************************************************)
         SW_INT_OPCODE = $CD;

         (*****************************************************************)
         (*  Hexadecimal value for MS-DOS 'exec()' function.              *)
         (*****************************************************************)
         EXEC_FUNCTION_CODE = $4B00;

  var    (*****************************************************************)
         (*  Intel 80x86 register variables used to  pass  original  register *)
         (* values to software interrupt handler.                         *)
         (*****************************************************************)
         REGS : registers;

         (*****************************************************************)
         (*  Pointer to position in interrupt vector table where the original *)
         (* handler address must be installed prior to passing control to the *)
         (* handler (via software interrupt).                             *)
         (*****************************************************************)
         REVECTOR : pointer absolute $0000:(REVECTORED_INTERRUPT_NUMBER shl 2);

         (*****************************************************************)
         (*  Flag that indicates that an call to the MS-DOS  function  'exec' *)
         (* is being performed, this must be re-enterant  to  allow  parallel *)
         (* execution of another program.                                 *)
         (*****************************************************************)
         MSDOS_EXEC_FUNCTION_CALL : boolean;

         (*****************************************************************)
         (*  Value to hold the software interrupt number being called.    *)
         (*****************************************************************)
         INTERRUPT_NUMBER : byte;

  begin (* SW_INTERRUPT_INTERCEPTOR *)
    asm
        pushf
        cli
    end; (* asm *)

    if mem[_CS : _IP - 2] = SW_INT_OPCODE
      then
        begin
          INTERRUPT_NUMBER := mem[_CS : _IP - 1];
          REVECTOR := INTERRUPT_VECTOR_HANDLER(INTERRUPT_NUMBER).VECTOR;

          (* Gather statistics? *)
          if ACTUAL_TASKING_CONFIGURATION.STATISTICS <> NO_STATISTICS then
            inc(TASKING_STATISTICS.SERVICE_CALLS(INTERRUPT_NUMBER));

          (* Allow rescheduling during interrupt? *)
          MSDOS_EXEC_FUNCTION_CALL :=
            (INTERRUPT_NUMBER = MSDOS_BIOS_FUNCTION_INTERRUPT_NUMBER) and
            (_AX = EXEC_FUNCTION_CODE);

          if MSDOS_EXEC_FUNCTION_CALL
            then
              inc(NUMBER_OF_CHILD_PROGRAMS)
            else
              LOCK_SCHEDULER;
          with REGS do
            begin
              AX := _AX; BX := _BX; CX := _CX; DX := _DX;
              BP := _BP; SI := _SI; DI := _DI;
              DS := _DS; ES := _ES;
              FLAGS := _FLAGS;
              intr(REVECTORED_INTERRUPT_NUMBER, REGS);
              asm cli end;
              _AX := AX; _BX := BX; _CX := CX; _DX := DX;
              _BP := BP; _SI := SI; _DI := DI;
              _DS := DS; _ES := ES;
              _FLAGS := FLAGS or INTERRUPT_FLAG;
```

```
        end; (* with...do *)
      if MSDOS_EXEC_FUNCTION_CALL
        then
          dec(NUMBER_OF_CHILD_PROGRAMS)
        else
          UNLOCK_SCHEDULER;
    end (* if...then *)
  else
    with RUNNING_TASK_PTR^ do ERROR_HANDLER[ILLEGAL_OPERATION)(TASK_ID);

  asm popf end;
end; (* SW_INTERRUPT_INTERCEPTOR *)

procedure DISPATCH_TASK;
(****************************************************************************)
(*   This routine causes the task specified  by  the  RUNNING_TASK_PTR  task *)
(* control block to begin running.                                           *)
(****************************************************************************)

  const (*******************************************************************)
        (*  Temporary variable used to switch contexts (all registers!).    *)
        (* NOTE: MUST be constant to force it into the DSeg!                *)
        (*******************************************************************)
        VALUE : word = 0;

  begin (* DISPATCH_TASK *)
    asm cli end;
    (* Reset preemptive timeslice counter *)
    PREEMPTIVE_TIMESLICE := PREEMPTIVE_TIMESLICE -
                            ACTUAL_TASKING_CONFIGURATION.TARGET_TIMESLICE;

    (* Dispatch a new task *)

    (* Switch Stacks *)
    VALUE := RUNNING_TASK_PTR^.CONTEKT.SS;
    asm mov bx,VALUE end;

    VALUE := RUNNING_TASK_PTR^.CONTEKT.SP;
    asm
        mov  sp,VALUE
        mov  ss,bx
    end; (* asm *)

    (* Setup new context, enabling interrupts *)
    VALUE := RUNNING_TASK_PTR^.CONTEXT.FLAGS or INTERRUPT_FLAG;
    asm push VALUE end;
    VALUE := RUNNING_TASK_PTR^.CONTEXT.CS;
    asm push VALUE  end;
    VALUE := RUNNING_TASK_PTR^.CONTEXT.IP;
    asm push VALUE end;
    VALUE := RUNNING_TASK_PTR^.CONTEXT.AX;
    asm push VALUE end;
    VALUE := RUNNING_TASK_PTR^.CONTEXT.BK;
    asm push VALUE end;
    VALUE := RUNNING_TASK_PTR^.CONTEXT.CX;
    asm push VALUE end;
    VALUE := RUNNING_TASK_PTR^.CONTEXT.DX;
    asm push VALUE end;
    VALUE := RUNNING_TASK_PTR^.CONTEXT.BP;
    asm push VALUE end;
    VALUE := RUNNING_TASK_PTR^.CONTEXT.SI;
    asm push VALUE end;
    VALUE := RUNNING_TASK_PTR^.CONTEXT.DI;
    asm push VALUE end;
    VALUE := RUNNING_TASK_PTR^.CONTEXT.ES;
    asm push VALUE end;
    VALUE := RUNNING_TASK_PTR^.CONTEXT.DS;
    asm push VALUE end;

    if TIMER_NSEOI_NEEDS_TO_BE_DONE then
      begin
        (* Issue NSEOI to PIC *)
        port[PROGRAMMABLE_INTERRUPT_CONTROLLER_PORT] :=
             NON_SPECIFIC_END_OF_INTERRUPT;
        TIMER_NSEOI_NEEDS_TO_BE_DONE := false;
      end; (* if...then *)

    (* Switch Contexts *)
    asm
        pop  ds
        pop  es
        pop  di
        pop  si
        pop  bp
        pop  dx
        pop  cx
        pop  bx
        pop  ax
        iret
    end; (* asm *)
  end; (* DISPATCH_TASK *)

procedure REQUEUE_AND_RESCHEDULE
(****************************************************************************)
  (TASK : TASK_IDS; RESULTING_STATE : TASK_STATES);
(*-------------------------------------------------------------------------*)
(*   This routine is used to preempt a task (which must  either  be  running *)
(* or ready) and put it into the specified queue (either blocked or  ready). *(
(* This is useful for 'blocking' a task or for allowing another  ready  task *)
(* to run (effectively used for context switching).                          *)
(****************************************************************************)

  label (*******************************************************************)
        (*  Label within procedure for the return (i.e. 'Dispatch')  address *)
        (* of a task which is going from RUNNING to some other state.        *)
        (*******************************************************************)
        REQUEUE_AND_RESCHEDULE_CONTINUE;

  const (*******************************************************************)
        (*  Speed vs Storage trade-off.  The queues are kept in an array  so *)
        (* that no run-time decisions (case...of) are needed to  access  the *)
        (* proper queue.                                                     *)
        (*******************************************************************)
        QUEUE_OF : array [TASK_STATES) of ^QUEUES =
                   (@BLOCRED_QUEUE, @DELAYED_QUEUE, @READY_QUEUE);

  var   (*******************************************************************)
        (*  Temporary storage used to determine the contents  of  the  80x86 *)
        (* registers.                                                        *)
        (*******************************************************************)
        VALUE : word;

        (*******************************************************************)
        (*  Temporary pointer to a Task Control Block which 'was/is' running *)
        (* but 'is/will be' rescheduled.                                     *)
        (*******************************************************************)
        RESCHEDULED_TASK_PTR : TASK_CONTROL_BLOCK_PTR;

        (*******************************************************************)
        (*  Task ID of the task which was running but is  now  going  to  be *)
        (* rescheduled.                                                      *)
        (*******************************************************************(
        PREVIOUSLY_RUNNING_TASK_ID : TASK_IDS;

  begin (* REQUEUE_AND_RESCHEDULE *)
    asm
        pushf
        cli
    end; (* asm *)

    (* Don't reschedule a task in a 'locked' region *)
    if COMPARE_PRIORITIES(RUNNING_TASK_PTR^.PRIORITY, EQUAL_TO,
                          HIGHEST_PRIORITY) then
      begin
        asm popf end;
        exit;
      end; (* if...then *(

    PREVIOUSLY_RUNNING_TASK_ID := RUNNING_TASK_PTR^.TASK_ID;
    if RUNNING_TASK_PTR^.TASK_ID = TASK
      then
        begin
          (* Save context of the running task *)
          asm
              push ax
              pop  VALUE
          end; (* asm *)
          RUNNING_TASK_PTR^.CONTEXT.AX := VALUE;
          asm
              push bx
              pop  VALUE
          end; (* asm *)
          RUNNING_TASK_PTR^.CONTEKT.BK := VALUE;
          asm
              push cx
              pop  VALUE
          end; (* asm *)
          RUNNING_TASK_PTR^.CONTEXT.CK := VALUE;
          asm
              push dx
              pop  VALUE
          end; (* asm *)
          RUNNING_TASK_PTR^.CONTEXT.DK := VALUE;
          asm
              push bp
              pop  VALUE
          end; (* asm *)
          RUNNING_TASK_PTR^.CONTEXT.BP := VALUE;
          asm
              push si
              pop  VALUE
          end; (* asm *)
          RUNNING_TASK_PTR^.CONTEXT.SI := VALUE;
          asm
              push di
              pop  VALUE
          end; (* asm *)
          RUNNING_TASK_PTR^.CONTEXT.DI := VALUE;
          asm
              push es
              pop  VALUE
          end; (* asm *)
          RUNNING_TASK_PTR^.CONTEXT.ES := VALUE;
          asm
              push ds
              pop  VALUE
          end; (* asm *)
          RUNNING_TASK_PTR^.CONTEXT.DS := VALUE;
          RUNNING_TASK_PTR^.CONTEXT.SS := sseg;
          RUNNING_TASK_PTR^.CONTEXT.SP := sptr;
          asm
              mov  ax,seg REQUEUE_AND_RESCHEDULE_CONTINUE
              push ax
              pop  VALUE
          end; (* asm *)
          RUNNING_TASK_PTR^.CONTEXT.CS := VALUE;
          asm
              mov  ax,offset REQUEUE_AND_RESCHEDULE_CONTINUE
              push ax
              pop  VALUE
          end; (* asm *)
          RUNNING_TASK_PTR^.CONTEXT.IP := VALUE;
          RESCHEDULED_TASK_PTR := RUNNING_TASK_PTR;
        end (* if...then *)
      else (* The task must be in the Ready Queue *(
        RESCHEDULED_TASK_PTR := FIND_AND_REMOVE
                                 (
                                  TASK,
                                  (From the) READY_QUEUE,
                                  UNCONDITIONALLY,
                                  THERE_IS_NO_BLOCK_TO_MATCH
                                 );
    if RESCHEDULED_TASK_PTR <> nil
      then
        begin
          RUNNING_TASK_PTR := REMOVE_FROM_HEAD(READY_QUEUE);
          if (RUNNING_TASK_PTR <> nil) and
             (RUNNING_TASK_PTR^.TASK_ID = NULL_TASK_ID) then
            INSERT_INTO(READY_QUEUE, RUNNING_TASK_PTR);

          INSERT_INTO(QUEUE_OF[RESULTING_STATE)^, RESCHEDULED_TASK_PTR);

          if (RUNNING_TASK_PTR = nil) or
             (RUNNING_TASK_PTR^.TASK_ID = NULL_TASK_ID) then
            RUNNING_TASK_PTR := REMOVE_FROM_HEAD(READY_QUEUE);
          if (ACTUAL_TASKING_CONFIGURATION.STATISTICS <> NO_STATISTICS) and
             (RUNNING_TASK_PTR^.TASK_ID <> PREVIOUSLY_RUNNING_TASK_ID) then
            begin
              inc(TASKING_STATISTICS.CONTEKT_SWITCHES);
              if PREEMPTIVE_TIMESLICE <
                 ACTUAL_TASKING_CONFIGURATION.TARGET_TIMESLICE
                then
                  inc(TASKING_STATISTICS.COOPERATIVE_CONTEXT_SWITCHES);
            end; (* if...then *)

          if ACTUAL_TASKING_CONFIGURATION.PRIORITY_SCHEDULING_POLICY =
             ROTATING_PRIORITIES then
            begin
              if HIGHEST_DYNAMIC_PRIORITY = low(USER_PRIORITIES)
                then
                  HIGHEST_DYNAMIC_PRIORITY := high(USER_PRIORITIES)
                else
                  HIGHEST_DYNAMIC_PRIORITY := HIGHEST_DYNAMIC_PRIORITY - 1;
            end; (* if...then *)

          DISPATCH_TASK;
        end (* if...then *!
      else
        with RUNNING_TASK_PTR^ do ERROR_HANDLER[TASK_IS_NOT_ACTIVE)(TASK_ID);

    REQUEUE_AND_RESCHEDULE_CONTINUE:
    asm popf end;
  end; (* REQUEUE_AND_RESCHEDULE *(

procedure MOUSE_EVENT_HANDLER
(****************************************************************************)
  (MOUSE : MOUSE_PARAMETERE(; far;
(*-------------------------------------------------------------------------*)
(*   This routine is called by the MS-Mouse driver in response to registered *)
(* mouse actions.  The TASKING mouse event is signaled (or the waiting  task *)
(* is made ready, whichever is appropriate).                                 *)
(****************************************************************************)

  var   (*******************************************************************)
        (*  Pointer to Task Control Block which is waiting on the  MS  Mouse *)
        (* semaphore (may very well be nil).                                 *)
        (*******************************************************************)
        WAITING_TASK_PTR : TASK_CONTROL_BLOCK_PTR;
```

```
  begin (* MOUSE_EVENT_HANDLER *)
    asm
        pushf
        cli
    end; (* asm *)
    MOUSE_INFO_FROM_HANDLER := MOUSE;
    if USER_MOUSE_EVENT_PTR <> nil then
      begin
        WAITING_TASK_PTR := FIND_AND_REMOVE
                              (
                               ANY_TASK,
                               (From the) BLOCKED_QUEUE,
                               BECAUSE_THE_BLOCK_WAS_SIGNALLEO,
                               (At) addr(USER_MOUSE_EVENT_PTR^)
                              );
        if WAITING_TASK_PTR = nil
          then
            USER_MOUSE_EVENT_PTR^ := SIGNALED
          else
            INSERT_INTO(READY_QUEUE, WAITING_TASK_PTR);
      end; (* if...then *)
    asm popf end;
  end; (* MOUSE_EVENT_HANDLER *)

procedure CHECK_AND_SETUP_BLOCK
(************************************************************************)
  (DELAY_UNTIL_ABSOLUTE_TIME : longint; SEMAPHORE_ADORESS : pointer);
(*----------------------------------------------------------------------*)
(*  This routine is used to establish the currently running  task  as  being *)
(* blocked on a semaphore (for any number of reasons) or  as  being  delayed *)
(* for some time period.  These functions are integrated because many of the *)
(* tasking operations require this service.                                  *)
(************************************************************************)

  begin (* CHECK_AND_SETUP_BLOCK *)
    asm
        pushf
        cli
    end; (* asm *)
    if (RUNNING_TASK_PTR^.WAITING_FOR <> nil) and
       (RUNNING_TASK_PTR^.WAITING_FOR^.SEM_PTR <> nil)
      then
        begin
          with RUNNING_TASK_PTR^ do
            ERROR_HANDLER[TASK_ALREADY_SUSPENDED](TASK_ID);
          REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID, READY);
        end (* if...then *)
      else
        begin
          if RUNNING_TASK_PTR^.WAITING_FOR = nil then
            new(RUNNING_TASK_PTR^.WAITING_FOR);

          with RUNNING_TASK_PTR^.WAITING_FOR^ do
            begin
              ABSOLUTE_TIME := DELAY_UNTIL_ABSOLUTE_TIME;
              SEM_PTR := SEMAPHORE_ADDRESS;
              if SEM_PTR <> nil
                then
                  REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID, BLOCKED)
                else
                  if ABSOLUTE_TIME <> NO_TIME_DELAY
                    then
                      REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID,DELAYED)
                    else
                      with RUNNING_TASK_PTR^ do
                        ERROR_HANDLER[ILLEGAL_OPERATION](TASK_ID);
                      REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID, READY);
            end; (* with...do *)
        end; (* if...then...else *)
    asm popf end;
  end; (* CHECK_AND_SETUP_BLOCK *)

procedure CHECK_OR_ADD_RESOURCE_TO_LOCKED_LIST
(******************************************/*****************************)
  (SEMAPHORE_ADDRESS : pointer);
(*----------------------------------------------------------------------*)
(*  This routine is used (when priority inheritance is enabled) to check  to *)
(* see if the priority of a task needs to be promoted as a result of a  task *)
(* attempting to lock a resource that is already locked.  If the resource is *)
(* being locked for the first time them the resource is added to the  linked *)
(* list of resources.                                                        *)
(************************************************************************)

  var   (*****************************************************************)
        (*  Pointer to locked resource that is 'walked' through the list  of *)
        (* all locked resources.                                           *)
        (*****************************************************************)
        WALKING_PTR : LOCKED_RESOURCE_PTR;

        (*****************************************************************)
        (*  Pointer to Task Control Block for the task that  is  having  its *)
        (* priority promoted (as a result of priority inheritance).        *)
        (*****************************************************************)
        PROMOTED_TASK_PTR : TASK_CONTROL_BLOCK_PTR;

  begin (* CHECK_OR_ADD_RESOURCE_TO_LOCKED_LIST *)
    asm
        pushf
        cli
    end; (* asm *)
    WALKING_PTR := LOCKED_RESOURCE_LIST_PTR;
    while (WALKING_PTR <> nil) and
          (WALKING_PTR^.SEMAPHORE_PTR <> SEMAPHORE_ADDRESS) do
      WALKING_PTR := WALKING_PTR^.NEXT;

    if (WALKING_PTR <> nil) and
       (WALKING_PTR^.SEMAPHORE_PTR = SEMAPHORE_ADDRESS)
      then
        begin
          if WALKING_PTR^.OWNER = nil
            then
              with WALKING_PTR^ do
                begin
                  OWNER := RUNNING_TASK_PTR;
                  PRIORITY := RUNNING_TASK_PTR^.PRIORITY;
                end (* with...do *)
            else
              begin
                (* Promote priority? *)
                if COMPARE_PRIORITIES(WALKING_PTR^.OWNER^.PRIORITY, LESS_THAN,
                                      RUNNING_TASK_PTR^.PRIORITY) then
                  begin
                    PROMOTED_TASK_PTR := FIND_AND_REMOVE
                                           (
                                            WALKING_PTR^.OWNER^.TASK_ID,
                                            (From the) READY_QUEUE,
                                            UNCONDITIONALLY,
                                            THERE_IS_NO_BLOCK_TO_MATCH
                                           );
                    PROMOTED_TASK_PTR^.PRIORITY := RUNNING_TASK_PTR^.PRIORITY;
                    INSERT_INTO(READY_QUEUE, PROMOTED_TASK_PTR);
                  end; (* if...then *)
              end; (* if...then...else *)
        end (* if...then *)
      else
        begin
          (* Find place to insert list element *)
          WALKING_PTR := LOCKED_RESOURCE_LIST_PTR;
          if WALKING_PTR = nil
            then
              begin
                new(WALKING_PTR);
                LOCKED_RESOURCE_LIST_PTR := WALKING_PTR;
              end (* if...then *)
            else
              begin
                while WALKING_PTR^.NEXT <> nil do
                  WALKING_PTR := WALKING_PTR^.NEXT;
                new(WALKING_PTR^.NEXT);
                WALKING_PTR := WALKING_PTR^.NEXT;
              end; (* if...then...else *)

          (* Add new locked resource *)
          with WALKING_PTR^ do
            begin
              OWNER         := RUNNING_TASK_PTR;
              PRIORITY      := RUNNING_TASK_PTR^.PRIORITY;
              SEMAPHORE_PTR := SEMAPHORE_ADDRESS;
              NEXT          := nil;
            end; (* with...do *)
        end; (* if...then...else *)
    asm popf end;
  end; (* CHECK_OR_ADD_RESOURCE_TO_LOCKED_LIST *)

procedure FIND_AND_REMOVE_LOCKED_RESOURCE
(************************************************************************)
  (SEMAPHORE_ADORESS : pointer);
(*----------------------------------------------------------------------*)
(*  This routine is used to remove the ownership of the  resource  from  any *)
(* task.  This is necessary after the resource has been  'unlocked'  and  is *)
(* therefore no longer 'owned'.                                              *)
(************************************************************************)

  var   (*****************************************************************)
        (*  Pointer to locked resource that is 'walked' through the list  of *)
        (* all locked resources.                                           *)
        (*****************************************************************)
        WALKING_PTR : LOCKED_RESOURCE_PTR;

  begin (* FIND_AND_REMOVE_LOCKED_RESOURCE *)
    asm
        pushf
        cli
    end; (* asm *)
    WALKING_PTR := LOCKED_RESOURCE_LIST_PTR;
    while (WALKING_PTR <> nil) and
          (WALKING_PTR^.SEMAPHORE_PTR <> SEMAPHORE_ADDRESS) do
      WALKING_PTR := WALKING_PTR^.NEXT;

    if (WALKING_PTR <> nil) and (WALKING_PTR^.OWNER <> nil) and
       (WALKING_PTR^.SEMAPHORE_PTR = SEMAPHORE_ADDRESS) then
      begin
        if COMPARE_PRIORITIES(RUNNING_TASK_PTR^.PRIORITY, GREATER_THAN,
                              WALKING_PTR^.PRIORITY) and
           (WALKING_PTR^.OWNER = RUNNING_TASK_PTR) then
          RUNNING_TASK_PTR^.PRIORITY := WALKING_PTR^.PRIORITY;
        WALKING_PTR^.OWNER := nil;
      end; (* if...then *)
    asm popf end;
  end; (* FIND_AND_REMOVE_LOCKED_RESOURCE *)

(************************************************************************)
(*                         End of non-user functions                    *)
(************************************************************************)

(************************************************************************)
(*                              User functions                          *)
(************************************************************************)

procedure GETMEM
(************************************************************************)
  (var PTR : pointer; SIZE : word);
(*----------------------------------------------------------------------*)
(*   This procedure replaces the standard  Turbo  Pascal  memory  allocation *)
(* procedure by the same  name.   Since  TASKING  must  perform  all  memory *)
(* allocation and de-allocation, this routine *must* be used instead of  the *)
(* system.getmem() procedure.   If  'type P : ^T;'  is  declared,  then  the *)
(* correct usage is 'getmem(pointer(P), sizeof(T));'.  Within  TASKING,  the *)
(* system.new() procedure can also be used because it is  always  used  when *)
(* interrupts are disabled.                                                  *)
(************************************************************************)

  begin (* GETMEM *)
    asm
      pushf
      cli
    end; (* asm *)
    system.getmem(PTR, SIZE);
    asm popf end;
  end; (* GETMEM *)

procedure FREEMEM
(************************************************************************)
  (PTR : pointer; SIZE : word);
(*----------------------------------------------------------------------*)
(*   This procedure replaces the standard  Turbo  Pascal  memory  allocation *)
(* procedure by the same  name   Since  TASKING  must  perform  all  memory *)
(* allocation and de-allocation, this routine *must* be used instead of  the *)
(* system.freemem() procedure.  If  'type P : ^T;'  is  declared,  then  the *)
(* correct usage is 'freemem(pointer(P), sizeof(T));'.  Within TASKING,  the *)
(* system.dispose() procedure can also be used because  it  is  always  used *)
(* when interrupts are disabled.                                             *)
(************************************************************************)

  begin (* FREEMEM *)
    asm
      pushf
      cli
    end; (* asm *)
    system.freemem(PTR, SIZE);
    asm popf end;
  end; (* FREEMEM *)

procedure CREATE
(************************************************************************)
  (var TASK : TASK_IDS; ATTR : TASK_ATTRIBUTES; ENTRY_POINT : TASK_TYPE);
(*----------------------------------------------------------------------*)
(*   This  routine  creates  all  system  data  structures  for  a  parallel *)
(* executing task.  The task is made ready to run and will be available  for *)
(* execution.  Actual execution will occur when the scheduler activates  the *)
(* task (i.e. precise execution time is unknown and depends on  availability *)
(* of system resources and system load).                                     *)
(************************************************************************)

  const (*****************************************************************)
        (*  When a task implicitely terminates there must be room for  calls *)
        (* to routines necessary for proper TASKING clean-up.  This constant *)
        (* must be larger than the maximum bytes needed for  these  clean-up *)
        (* routines to execute properly.                                   *)
        (*****************************************************************)
        STACK_HEADROOM = 48;

  var   (*****************************************************************)
        (*  Pointer to the to-be-created Task Control Block.               *)
        (*****************************************************************)
        NEW_TCB_PTR : TASK_CONTROL_BLOCK_PTR;
```

```
      (*****************************************************************)
      (* Pointer to the top of the task stack (as opposed to the first *)
      (* byte of the stack which is used to reference the Turbo Pascal *)
      (* variable returned when the memory is allocated).              *)
      (*****************************************************************)
      TOP_OF_STACK_PTR : pointer;

      (*****************************************************************)
      (* Loop control variable of possible errors used to initialize the *)
      (* task specific error handlers.                                 *)
      (*****************************************************************)
      ERROR : SYSTEM_ERRORS;

function NEW_TASK_ID : TASK_IDS;
(*********************************************************************)
(*  This function determines what the next available task identifier is, *)
(* marks it as allocated and returns it to the caller.               *)
(*********************************************************************)

  var   (*************************************************************)
        (*  Temporary variable used to hold the potential new task ID. *)
        (*************************************************************)
        THE_NEW_TASK_ID : TASK_IDS;

        (*************************************************************)
        (*  Loop control variable to search arrays of TASK_ID sets.  *)
        (*************************************************************)
        TASK_ID_SET : TASK_ID_SETS;

  begin (* NEW_TASK_ID *)
    TASK_ID_SET := low(TASK_ID_SETS);

    THE_NEW_TASK_ID := succ(NULL_TASK_ID);
    while (THE_NEW_TASK_ID in ALLOCATED_TASK_IDS[TASK_ID_SET]) and
          (THE_NEW_TASK_ID <= high(byte)) and
          (TASK_ID_SET <= high(TASK_ID_SETS)) do
      begin
        inc(THE_NEW_TASK_ID);
        if (THE_NEW_TASK_ID > high(byte)) and
           (TASK_ID_SET < high(TASK_ID_SETS)) then
          begin
            TASK_ID_SET := succ(TASK_ID_SET);
            THE_NEW_TASK_ID := low(TASK_IDS);
          end; (* if...then *)
      end; (* while...do *)

    if not (THE_NEW_TASK_ID in ALLOCATED_TASK_IDS[TASK_ID_SET])
      then
        begin
          ALLOCATED_TASK_IDS[TASK_ID_SET] :=
                ALLOCATED_TASK_IDS[TASK_ID_SET] + [THE_NEW_TASK_ID];
          NEW_TASK_ID := (TASK_ID_SET shl 8) or THE_NEW_TASK_ID;
        end (* if...then *)
      else
        with RUNNING_TASK_PTR^ do
          ERROR_HANDLER[INSUFFICIENT_RESOURCES](TASK_ID);
  end; (* NEW_TASK_ID *)

function INC_PTR
(*********************************************************************)
  (THE_PTR : pointer; INCREMENT : word) : pointer;
(*-------------------------------------------------------------------*)
(*  This routine merely increments a double word pointer by the value *)
(* specified.                                                        *)
(*********************************************************************)

  var   (*************************************************************)
        (*  Temporary variables to hold intermediate result.         *)
        (*************************************************************)
        SEGMENT, OFFSET : word;
        SUM_OFS  : longint;

  begin (* INC_PTR *)
    SUM_OFS := ofs(THE_PTR^) + INCREMENT;
    SEGMENT := seg(THE_PTR^);
    OFFSET  := SUM_OFS and $0000FFFF;
    if (SUM_OFS and $FFFF0000) <> 0 then
      (* Overflowed 64k boundary by 1 (because INCREMENT is a word!) *)
      inc(SEGMENT, $1000);
    INC_PTR := ptr(SEGMENT, OFFSET);
  end; (* INC_PTR *)

begin (* CREATE *)
  asm
    pushf
    cli
  end; (* asm *)
  TASK := NEW_TASK_ID;
  new(NEW_TCB_PTR);
  NEW_TCB_PTR^.STACK_SIZE := ATTR.STACK_WORDS_NEEDED * 2 (* Bytes/word *);
  with NEW_TCB_PTR^ do
    begin
      getmem(STACK_PTR, STACK_SIZE);
      (* Get pointer to Top-Of-Stack *)
      TOP_OF_STACK_PTR := INC_PTR(STACK_PTR, STACK_SIZE-12-STACK_HEADROOM);
      TASK_ID          := TASK;
      CLOCK_TICKS      := 0;
      PRIORITY         := ATTR.PRIORITY;
      WAITING_FOR      := nil;
      NEXT             := nil;
      PREVIOUS         := nil;
      for ERROR := low(SYSTEM_ERRORS) to high(SYSTEM_ERRORS) do
        if addr(ATTR.ERROR_HANDLERS[ERROR]) = nil
          then
            ERROR_HANDLER[ERROR] := NULL.ERROR_HANDLER[ERROR]
          else
            ERROR_HANDLER[ERROR] := ATTR.ERROR_HANDLERS[ERROR];
      with CONTEXT do
        begin
          AX    := $0000;  BX := $0000;  CX := $0000;  DX := $0000;
          BP    := $0000;  SI := $0000;  DI := $0000;  ES := $0000;
          DS    := dseg;
          SS    := seg(TOP_OF_STACK_PTR^);
          SP    := (ofs(TOP_OF_STACK_PTR^) + 1) and $FFFE;
          CS    := seg(ENTRY_POINT);         IP := ofs(ENTRY_POINT);
          FLAGS := INTERRUPT_FLAG;
        end; (* with...do *)
      (* Clear the stack *)
      fillchar(STACK_PTR^, STACK_SIZE-1, byte(DEFAULT_STACK_VALUE));
      (* Install task identifier and priority onto stack *)
      memw[CONTEXT.SS : CONTEXT.SP +  4] := PRIORITY;
      memw[CONTEXT.SS : CONTEXT.SP +  6] := TASK_ID;
      (* Install implied call to DESTROY() onto task stack *)
      memw[CONTEXT.SS : CONTEXT.SP +  0] := ofs(DESTROY);
      memw[CONTEXT.SS : CONTEXT.SP +  2] := seg(DESTROY);
      memw[CONTEXT.SS : CONTEXT.SP + 12] := TASK_ID;
    end; (* with...do *)
  INSERT_INTO(READY_QUEUE, NEW_TCB_PTR);

  if ALLOW_RESCHEDULE_IN_CREATE then
    REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID, READY);
  asm popf end;
end; (* CREATE *)

procedure DESTROY
(*********************************************************************)
  (TASK : TASK_IDS);
(*-------------------------------------------------------------------*)
(*  This routine terminates (if appropriate) releases all of the task's *)
```

```
(* resources to be used by other tasks and then removes all evidence of the *)
(* task ever having existed.  An implied call to this routine is setup at *)
(* the time that a task is created (by setting up the stack to return here). *)
(* This implied call ensures that a task that reaches its 'end' statement is *)
(* properly terminated and does not become an orphan (or a zombie).    *)
(*********************************************************************)

  const (*************************************************************)
        (*  Pointer to the Task Control Block that is being destroyed. *)
        (* NOTE: MUST be constant to force it into the DSeg!          *)
        (*************************************************************)
        DESTROYED_TASK_PTR : TASK_CONTROL_BLOCK_PTR = nil;

        (*************************************************************)
        (*  Flag to indicate that the task that is being destroyed 'was/is' *)
        (* running, this has stack access consequences.              *)
        (* NOTE: MUST be constant to force it into the DSeg!          *)
        (*************************************************************)
        DESTROYED_TASK_WAS_RUNNING : boolean = false;

        (*************************************************************)
        (*  Temporary variable used to switch stacks (SS and SP).    *)
        (* NOTE: MUST be constant to force it into the DSeg!          *)
        (*************************************************************)
        VALUE : word = 0;

  procedure FREE_TASK_ID
  (*******************************************************************)
    (TASK_ID : TASK_IDS);
  (*-----------------------------------------------------------------*)
  (*  This routine merely puts a task identifier back into the free list so *)
  (* that it can be re-allocated if necessary.                       *)
  (*******************************************************************)

    begin (* FREE_TASK_ID *)
      ALLOCATED_TASK_IDS[hi(TASK_ID) shr 8] :=
          ALLOCATED_TASK_IDS[hi(TASK_ID) shr 8] - [lo(TASK_ID)];
    end; (* FREE_TASK_ID *)

  begin (* DESTROY *)
    asm
      pushf
      cli
    end; (* asm *)

    (* The null task cannot be destroyed! *)
    if TASK = NULL_TASK_ID then
      begin
        RUNNING_TASK_PTR^.ERROR_HANDLER[ILLEGAL_OPERATION](TASK);
        asm popf end;
        exit;
      end; (* if...then *)

    if TASK = RUNNING_TASK_PTR^.TASK_ID
      then
        begin
          DESTROYED_TASK_PTR := RUNNING_TASK_PTR;
          DESTROYED_TASK_WAS_RUNNING := true;
        end (* if...then *)
      else
        begin
          DESTROYED_TASK_PTR := FIND_AND_REMOVE
                                  (
                                   TASK,
                                   (From the) READY_QUEUE,
                                   UNCONDITIONALLY,
                                   THERE_IS_NO_BLOCK_TO_MATCH
                                  );
          if DESTROYED_TASK_PTR = nil then
            begin
              DESTROYED_TASK_PTR := FIND_AND_REMOVE
                                      (
                                       TASK,
                                       (From the) BLOCKED_QUEUE,
                                       UNCONDITIONALLY,
                                       THERE_IS_NO_BLOCK_TO_MATCH
                                      );
              if DESTROYED_TASK_PTR = nil then
                DESTROYED_TASK_PTR := FIND_AND_REMOVE
                                        (
                                         TASK,
                                         (From the) DELAYED_QUEUE,
                                         UNCONDITIONALLY,
                                         THERE_IS_NO_BLOCK_TO_MATCH
                                        );
            end; (* if...then *)
          DESTROYED_TASK_WAS_RUNNING := false;
        end; (* if...then...else *)

    if DESTROYED_TASK_PTR = nil
      then
        begin
          (* Can't find the requested Task ID *)
          RUNNING_TASK_PTR^.ERROR_HANDLER[ILLEGAL_TASK_ID](TASK);
          asm popf end;
          exit;
        end (* if...then *)
      else
        begin
          (* Check to see if special control structures need to be removed. *)
          if DESTROYED_TASK_PTR^.WAITING_FOR <> nil then
            begin
              if (USER_KEYBOARD_SEM_PTR <> nil) and
                 (DESTROYED_TASK_PTR^.WAITING_FOR^.SEM_PTR =
                  USER_KEYBOARD_SEM_PTR)
                then
                  begin
                    dispose(USER_KEYBOARD_SEM_PTR);
                    USER_KEYBOARD_SEM_PTR := nil;
                  end (* if...then *)
                else
                  if (USER_MOUSE_EVENT_PTR <> nil) and
                     (DESTROYED_TASK_PTR^.WAITING_FOR^.SEM_PTR =
                      SEMAPHORE_PTR(USER_MOUSE_EVENT_PTR)) then
                    begin
                      dispose(USER_MOUSE_EVENT_PTR);
                      USER_MOUSE_EVENT_PTR := nil;
                    end; (* if...then *)

              (* Get rid of whatever the task was waiting for *)
              dispose(DESTROYED_TASK_PTR^.WAITING_FOR);
            end; (* if...then *)

          (* De-allocate the task's mailbox (if applicable) *)
          REMOVE_MBOX_OF(DESTROYED_TASK_PTR^.TASK_ID);

          (* Switch stacks if user stack is going to be destroyed *)
          if DESTROYED_TASK_WAS_RUNNING then
            begin
              RUNNING_TASK_PTR := REMOVE_FROM_HEAD(READY_QUEUE);
              VALUE := RUNNING_TASK_PTR^.CONTEXT.SS;
              asm
                push  VALUE
                pop   ss
              end;
              VALUE := RUNNING_TASK_PTR^.CONTEXT.SP;
              asm mov sp,VALUE end;
            end; (* if...then *)
```

```
         (* Remove task structures *)
         FREE_TASK_ID(DESTROYED_TASK_PTR^.TASK_ID);
         with DESTROYED_TASK_PTR^ do freemem(STACK_PTR, STACK_SIZE);
         dispose(DESTROYED_TASK_PTR);

         if DESTROYED_TASK_WAS_RUNNING then
           (* Cannot come back here because stack is gone! *)
           asm jmp DISPATCH_TASK end;
       end; (* if...then...else *)
   end; (* DESTROY *)

procedure SUSPEND
(*************************************************************************)
  (TASK : TASK_IDS);
(*-----------------------------------------------------------------------*)
(*   This routine causes a previously running or ready task  to  be  blocked *)
(* unconditionally.                                                      *)
(*************************************************************************)

  begin (* SUSPEND *)
    asm
        pushf
        cli
    end; (* asm *)
    REQUEUE_AND_RESCHEDULE(TASK, BLOCKED);
    asm popf end;
  end; (* SUSPEND *)

procedure RESUME
(*************************************************************************)
  (TASK : TASK_IDS);
(*-----------------------------------------------------------------------*)
(*   This routine causes a previously suspended task to be made ready.  The *)
(* task is not necessarily the next to run, that is based on the priorities *)
(* of the other ready tasks.                                             *)
(*************************************************************************)

  var   (*****************************************************************)
        (*  Pointer to the Task Control Block of the task that is being made *)
        (* READY.                                                        *)
        (*****************************************************************)
        RESUMED_TASK_PTR : TASK_CONTROL_BLOCK_PTR;

  begin (* RESUME *)
    asm
        pushf
        cli
    end; (* asm *)
    RESUMED_TASK_PTR := FIND_AND_REMOVE
                          (
                           TASK,
                           (From the) BLOCKED_QUEUE,
                           UNCONDITIONALLY,
                           THERE_IS_NO_BLOCK_TO_MATCH
                          );
    if RESUMED_TASK_PTR <> nil
      then
        begin
          INSERT_INTO(READY_QUEUE, RESUMED_TASK_PTR);
          REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID, READY);
        end (* if...then *)
      else
        if RUNNING_TASK_PTR^.TASK_ID <> NULL_TASK_ID then
          with RUNNING_TASK_PTR^ do
            ERROR_HANDLER[TASK_IS_NOT_ACTIVE](TASK_ID);
    asm popf end;
  end; (* RESUME *)

function GET_MILLISECOND_TICKS : longint;
(*************************************************************************)
(*   This routine retuns the number of milliseconds that have elapsed  since *)
(* the program started executing.                                       *)
(*************************************************************************)

  begin (* GET_MILLISECOND_TICKS *)
    asm
        pushf
        cli
    end; (* asm *)
    GET_MILLISECOND_TICKS := MILLISECOND_TICKS;
    asm popf end;
  end; (* GET_MILLISECOND_TICKS *)

procedure WAIT_FOR_DELAY
(*************************************************************************)
  (DURATION : TIME);
(*-----------------------------------------------------------------------*)
(*   This system call causes the currently running task  to  become  blocked *)
(* for the specified length of time.  There is no guarantee  that  the  task *)
(* will begin executing when the time expires, the only  guarantee  is  that *)
(* the task will become ready to execute at that time.                   *)
(*   No attempt is made to detect 'Global Ticks' rollover, as  a  result  of *)
(* this after the program has been running for a long time ((2^30-1)*1  msec *)
(* i.e. 12.4 days) there is a possibility of a delay  request  resulting  in *)
(* near immediate rescheduling.                                          *)
(*************************************************************************)

  var   (*****************************************************************)
        (*  Temporary variable used to hold the MILLISECOND_TICKS value when *)
        (* the task will be 'awakened'.                                  *)
        (*****************************************************************)
        ASSOLUTE_TIME : longint;

  begin (* WAIT_FOR_DELAY *)
    asm
        pushf
        cli
    end; (* asm *)
    with DURATION do
      ABSOLUTE_TIME := MILLISECOND_TICKS +
                       trunc(longint(DAYS)         * MILLISECONDS_PER_DAY +
                             longint(HOURS)        * MILLISECONDS_PER_HOUR +
                             longint(MINUTES)      * MILLISECONDS_PER_MINUTE +
                             longint(SECONDS)      * MILLISECONDS_PER_SECOND +
                             longint(MILLISECONDS) +
                             1 (* Ensure that we always 'round' up *)
                            );
    CHECK_AND_SETUP_BLOCK(ASSOLUTE_TIME, nil);
    asm popf end;
  end; (* WAIT_FOR_DELAY *)

procedure PREEMPTASLE_DELAY
(*************************************************************************)
  (DURATION : TIME);
(*-----------------------------------------------------------------------*)
(*   This system call causes the currently running task to be delayed by the *)
(* specified number of milliseconds.                                     *)
(*************************************************************************)

  var   (*****************************************************************)
        (*  Temporary variable used to hold the MILLISECOND_TICKS value when *)
        (* the task will be allowed to continue executing.               *)
        (*****************************************************************)
        DELAY_COMPLETE_TIME : longint;

  begin (* PREEMPTASLE_DELAY *)
    asm
        pushf
        cli
    end; (* asm *)
    with DURATION do
      DELAY_COMPLETE_TIME := MILLISECOND_TICKS +
                             trunc
                               (
                                longint(DAYS)     * MILLISECONDS_PER_DAY +
                                longint(HOURS)    * MILLISECONDS_PER_HOUR +
                                longint(MINUTES)  * MILLISECONDS_PER_MINUTE +
                                longint(SECONDS)  * MILLISECONDS_PER_SECOND +
                                longint(MILLISECONDS)
                               );

    asm popf end;

    repeat (* Nothing! *) until MILLISECOND_TICKS >= DELAY_COMPLETE_TIME;
  end; (* PREEMPTASLE_DELAY *)

procedure CHANGE_PRIORITY
(*************************************************************************)
  (PRIORITY : USER_PRIORITIES);
(*-----------------------------------------------------------------------*)
(*   This routine simply changes the priority of the currently running  task *)
(* and preempts the running task (causing a  dispatch  of  the  now  highest *)
(* priority task) if appropriate.                                        *)
(*************************************************************************)

  begin (* CHANGE_PRIORITY *)
    asm
        pushf
        cli
    end; (* asm *)
    RUNNING_TASK_PTR^.PRIORITY := PRIORITIES[PRIORITY];
    REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID, READY);
  end; )* CHANGE_PRIORITY *)

procedure SIGNAL_SEMAPHORE
(*************************************************************************)
  (var SEM : SEMAPHORE);
(*-----------------------------------------------------------------------*)
(*   This routine performs an Up() operation on the specified semaphore.  If *)
(* there are tasks waiting on the semaphore then one is awakened.  If  there *)
(* are none waiting then the semaphore is incremented.                   *)
(*************************************************************************)

  var   (*****************************************************************)
        (*  Pointer to Task Control Block which is waiting on the  semaphore *)
        (* (may very well be nil).                                       *)
        (*****************************************************************)
        WAITING_TASK_PTR : TASK_CONTROL_BLOCK_PTR;

  begin (* SIGNAL_SEMAPHORE *)
    asm
        pushf
        cli
    end; (* asm *)
    if TASKING_CONFIGURATION.PRIORITY_INHERITANCE_ENABLED then
      FIND_AND_REMOVE_LOCKED_RESOURCE(addr(SEM));
    WAITING_TASK_PTR := FIND_AND_REMOVE
                          (
                           ANY_TASK,
                           (From the) BLOCKED_QUEUE,
                           BECAUSE_THE_BLOCK_WAS_SIGNALLED,
                           (At) addr(SEM)
                          );
    if WAITING_TASK_PTR = nil
      then
        inc(SEM)
      else
        INSERT_INTO(READY_QUEUE, WAITING_TASK_PTR);
    REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID, READY);
    asm popf end;
  end; (* SIGNAL_SEMAPHORE *)

procedure WAIT_ON_SEMAPHORE
(*************************************************************************)
  (var SEM : SEMAPHORE);
(*-----------------------------------------------------------------------*)
(*   This routine performs a Down() operation on  the  specified  semaphore. *)
(* If the semaphore is zero then the running task is suspended.   If  it  is *)
(* non-zero then the semaphore is decremented.                           *)
(*************************************************************************)

  begin (* WAIT_ON_SEMAPHORE *)
    asm
        pushf
        cli
    end; (* asm *)
    if SEM > 0
      then
        begin
          dec(SEM);
          if TASKING_CONFIGURATION.PRIORITY_INHERITANCE_ENABLED and (SEM = 0)
            then
              CHECK_OR_ADD_RESOURCE_TO_LOCKED_LIST(addr(SEM));
          REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID, READY);
        end (* if...then *)
      else
        begin
          if TASKING_CONFIGURATION.PRIORITY_INHERITANCE_ENABLED then
            CHECK_OR_ADD_RESOURCE_TO_LOCKED_LIST(addr(SEM));
          CHECK_AND_SETUP_BLOCK(NO_TIME_DELAY, addr(SEM));
        end; (* if...then...else *)
    asm popf end;
  end; (* WAIT_ON_SEMAPHORE *)

procedure SIGNAL_EVENT
(*************************************************************************)
  (var THE_EVENT : EVENT);
(*-----------------------------------------------------------------------*)
(*   This routine signals the event specified, if there are tasks waiting on *)
(* the event then one task is made ready.  If there are no tasks waiting  on *)
(* the event then the signal is saved, only  one  signal  is  maintained  no *)
(* matter how many times the event is signaled.                          *)
(*************************************************************************)

  var   (*****************************************************************)
        (*  Pointer to Task Control Block which is waiting on the event (may *)
        (* very well be nil).                                            *)
        (*****************************************************************)
        WAITING_TASK_PTR : TASK_CONTROL_BLOCK_PTR;

  begin (* SIGNAL_EVENT *)
    asm
        pushf
        cli
    end; (* asm *)
    if TASKING_CONFIGURATION.PRIORITY_INHERITANCE_ENABLED then
      FIND_AND_REMOVE_LOCKED_RESOURCE(addr(THE_EVENT));
    WAITING_TASK_PTR := FIND_AND_REMOVE
                          (
                           ANY_TASK,
                           (From the) BLOCKED_QUEUE,
                           BECAUSE_THE_BLOCK_WAS_SIGNALLED,
                           (At) addr(THE_EVENT)
                          );
    if WAITING_TASK_PTR = nil
      then
        THE_EVENT := SIGNALED
      else
        INSERT_INTO(READY_QUEUE, WAITING_TASK_PTR);
```

```
    REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID, READY);
    asm popf end;
  end; (* SIGNAL_EVENT *)

procedure BROADCAST_EVENT
(******************************************************************************)
  (var THE_EVENT : EVENT);
(*----------------------------------------------------------------------------*)
(*   This routine signals the event specified, if there are tasks waiting on *)
(* the event then they are all made ready.  If there are no tasks waiting on *)
(* the event then the signal is saved, only  one  signal  is  maintained  no *)
(* matter how many times the event is signaled.                              *)
(******************************************************************************)

  var   (***********************************************************************)
        (*  Pointer to Task Control Block which is waiting on the event (may *)
        (* very well be nil).                                                *)
        (***********************************************************************)
        WAITING_TASK_PTR : TASK_CONTROL_BLOCK_PTR;

  begin (* BROADCAST_EVENT *)
    asm
        pushf
        cli
    end; (* asm *)
    if TASKING_CONFIGURATION.PRIDRITY_INHERITANCE_ENABLED then
      FIND_AND_REMOVE_LOCKED_RESOURCE(addr(THE_EVENT));
    repeat
      WAITING_TASK_PTR := FIND_AND_REMOVE
                          (
                           ANY_TASK,
                           (From the) BLOCKED_QUEUE,
                           BECAUSE_THE_BLOCK_WAS_SIGNALLED,
                           (At) addr(THE_EVENT)
                          );
      if WAITING_TASK_PTR <> nil then
        INSERT_INTO(READY_QUEUE, WAITING_TASK_PTR);
    until WAITING_TASK_PTR = nil;
    REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID, READY);
    asm popf end;
  end; (* BROADCAST_EVENT *)

procedure START_PERIODIC_EVENT
(******************************************************************************)
  (var THE_EVENT : EVENT; INTERVAL : TIME);
(*----------------------------------------------------------------------------*)
(*   This routine starts the periodic signalling (by TASKING) of  the  event *)
(* specified at the specified interval.  This routine cannot cause a task to *)
(* be suspended but a re-schedule may occur.                                 *)
(******************************************************************************)

  var (************************************************************************)
      (*  Pointer to the list of periodic events (possibly nil).            *)
      (************************************************************************)
      WALKING_PTR : PERIODIC_EVENTS_PTR;

      (************************************************************************)
      (*  Amount of time between successive signals on the event.           *)
      (************************************************************************)
      PERIOD_INTERVAL : longint;

  begin (* START_PERIODIC_EVENT *)
    asm
        pushf
        cli
    end; (* asm *)

    (* Compute period (in milliseconds) *)
    with INTERVAL do
      PERIOD_INTERVAL := trunc(longint(DAYS)    * MILLISECONDS_PER_DAY +
                               longint(HOURS)   * MILLISECONDS_PER_HOUR +
                               longint(MINUTES) * MILLISECONDS_PER_MINUTE +
                               longint(SECONDS) * MILLISECONDS_PER_SECOND +
                               longint(MILLISECONDS)
                              );

    (* Find the place in the list to insert the event *)
    if PERIODIC_EVENTS_LIST_PTR = nil
      then
        begin
          (* Insert element (i.e. create list) *)
          new(WALKING_PTR);
          with WALKING_PTR^ do
            begin
              EVENT_PTR := addr(THE_EVENT);
              PERIOD    := PERIOD_INTERVAL;
              NEXT      := nil;
            end; (* with...do *)
          PERIODIC_EVENTS_LIST_PTR := WALKING_PTR;
        end (* if...then *)
      else
        begin
          WALKING_PTR := PERIODIC_EVENTS_LIST_PTR;
          while (WALKING_PTR^.EVENT_PTR <> addr(THE_EVENT)) and
                (WALKING_PTR^.NEXT <> nil) do
            WALKING_PTR := WALKING_PTR^.NEXT;

          (* There can only be one period for each event *)
          if (WALKING_PTR^.EVENT_PTR = addr(THE_EVENT))
            then
              with RUNNING_TASK_PTR^ do
                ERROR_HANDLER[ILLEGAL_DPERRTION](TASK_ID)
            else
              begin
                (* Insert element (always at the end of the list!) *)
                new(WALKING_PTR^.NEXT);
                WALKING_PTR := WALKING_PTR^.NEXT;
                with WALKING_PTR^ do
                  begin
                    EVENT_PTR := addr(THE_EVENT);
                    PERIOD    := PERIOD_INTERVAL;
                    NEXT      := nil;
                  end; (* with...do *)
              end; (* if...then...else *)
        end; (* if...then...else *)

    REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID, READY);
    asm popf end;
  end; (* START_PERIODIC_EVENT *)

procedure STOP_PERIODIC_EVENT
(******************************************************************************)
  (var THE_EVENT : EVENT);
(*----------------------------------------------------------------------------*)
(*   This routine stops the periodic signalling (by TASKING)  of  the  event *)
(* specified.  This routine cannot cause  a  task  to  be  suspended  but  a *)
(* re-schedule may occur.                                                    *)
(******************************************************************************)

  var   (**********************************************************************)
        (*  Pointer to the list of periodic events (possibly nil)            *)
        (**********************************************************************)
        WALKING_PTR, PREVIOUS_PTR : PERIDDIC_EVENTS_PTR;

  begin (* STOP_PERIODIC_EVENT *)
    asm
        pushf
        cli
    end; (* asm *)

    (* Find the place in the list to delete the event *)
    PREVIOUS_PTR := nil;
    WALKING_PTR := PERIODIC_EVENTS_LIST_PTR;
    while (WALKING_PTR <> nil) and
          (WALKING_PTR^.EVENT_PTR <> addr(THE_EVENT)) do
      begin
        PREVIOUS_PTR := WALKING_PTR;
        WALKING_PTR := WALKING_PTR^.NEXT;
      end; (* while...do *)

    (* There must be one period for the event *)
    if (PREVIOUS_PTR = nil) or (PREVIOUS_PTR^.EVENT_PTR <> addr(THE_EVENT))
      then
        with RUNNING_TASK_PTR^ do
          ERROR_HANDLER[ILLEGAL_OPERATION](TASK_ID)
      else
        begin
          if PREVIOUS_PTR <> nil
            then
              PREVIOUS_PTR := WALKING_PTR^.NEXT
            else
              PERIODIC_EVENTS_LIST_PTR := nil;
          dispose(WALKING_PTR);
        end; (* if...then...else *)

    REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID, READY);
    asm popf end;
  end; (* STOP_PERIODIC_EVENT *)

procedure WAIT_ON_EVENT
(******************************************************************************)
  (var THE_EVENT : EVENT);
(*----------------------------------------------------------------------------*)
(*   This routine causes the calling task to be suspended  waiting  for  the *)
(* specified event to occur.  If the event has  already  occurred  then  the *)
(* calling task will immediately become ready, although  a  re-schedule  may *)
(* occur.                                                                    *)
(******************************************************************************)

  begin (* WAIT_ON_EVENT *)
    asm
        pushf
        cli
    end; (* asm *)
    if THE_EVENT = SIGNALED
      then
        begin
          THE_EVENT := UNSIGNALED;
          if TASKING_CONFIGURATION.PRIORITY_INHERITANCE_ENABLED then
            CHECK_OR_ADD_RESOURCE_TO_LOCKED_LIST(addr(THE_EVENT));
          REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID, READY);
        end (* if...then *)
      else
        begin
          if TASKING_CONFIGURATION.PRIORITY_INHERITANCE_ENABLED then
            CHECK_OR_ADD_RESDURCE_TO_LOCKED_LIST(addr(THE_EVENT));
          CHECK_AND_SETUP_BLOCK(NO_TIME_DELAY, addr(THE_EVENT));
        end; (* if...then...else *)
    asm popf end;
  end; (* WAIT_ON_EVENT *)

procedure SIGNAL_BINARY_SEMAPHORE
(******************************************************************************)
  (var SEM : BINARY_SEMAPHORE);
(*----------------------------------------------------------------------------*)
(*   This routine performs an Up() operation on the specified semaphore.  If *)
(* there are tasks waiting on the semaphore then one is awakened.  If  there *)
(* are none waiting then the semaphore is set.                               *)
(******************************************************************************)

  begin (* SIGNAL_BINARY_SEMAPHORE *)
    asm
        pushf
        cli
    end; (* asm *)
    if SEM = 1
      then
        begin
          with RUNNING_TASK_PTR^ do
            ERROR_HANDLER[ILLEGAL_OPERATION](TASK_ID);
          REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID, READY);
        end (* if...then *)
      else
        SIGNAL_EVENT(EVENT(SEM));
    asm popf end;
  end; (* SIGNAL_BINARY_SEMAPHORE *)

procedure WAIT_ON_BINARY_SEMAPHORE
(******************************************************************************)
  (var SEM : BINARY_SEMAPHORE);
(*----------------------------------------------------------------------------*)
(*   This routine performs a Down() operation on  the  specified  semaphore. *)
(* If the semaphore is zero then the running task is suspended.   If  it  is *)
(* non-zero then the semaphore is cleared.                                   *)
(******************************************************************************)

  begin (* WAIT_ON_BINARY_SEMAPHORE *)
    WAIT_ON_EVENT(EVENT(SEM));
  end; (* WAIT_ON_BINARY_SEMAPHORE *)

procedure SIGNAL_CONDITION_VARIABLE
(******************************************************************************)
  (var C_VAR : CONDITION_VARIABLE);
(*----------------------------------------------------------------------------*)
(*   This routine signals the condition variable specified, if  there  is  a *)
(* task waiting on the condition variable then it is made ready (or  one  of *)
(* the multiple waiting tasks is made ready.  If there are no tasks  waiting *)
(* on the condition variable then the signal is lost.                        *)
(******************************************************************************)

  var   (**********************************************************************)
        (*  Pointer to Task Control Block which is waiting on the  condition *)
        (* variable (may very well be nil).                                  *)
        (**********************************************************************)
        WAITING_TASK_PTR : TASK_CONTROL_BLOCK_PTR;

  begin (* SIGNAL_CONDITION_VARIABLE *)
    asm
        pushf
        cli
    end; (* asm *)
    WAITING_TASK_PTR := FIND_AND_REMOVE
                        (
                         ANY_TASK,
                         (From the) BLOCKED_QUEUE,
                         BECAUSE_THE_BLOCK_WAS_SIGNALLED,
                         (At) addr(C_VAR)
                        );
    if WAITING_TASK_PTR <> nil then
      INSERT_INTO(READY_QUEUE, WAITING_TASK_PTR);
    REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID, READY);
    asm popf end;
  end; (* SIGNAL_CONDITION_VARIABLE *)

procedure WAIT_ON_CONDITION_VARIABLE
(******************************************************************************)
  (var C_VAR : CONDITION_VARIABLE);
(*----------------------------------------------------------------------------*)
```

```
(*  This routine causes the calling task to become blocked.  Because *)
(* condition variables have no memory, the calling task is guaranteed to *)
(* become blocked.  As soon as another task signals the condition variable *)
(* the waiting task will be made ready (or one of the multiple waiting tasks *)
(* will be made ready).                                                    *)
(**************************************************************************)

  begin (* WAIT_ON_CONDITION_VARIABLE *)
    asm
        pushf
        cli
    end; (* asm *)
    CHECK_AND_SETUP_BLOCK(NO_TIME_DELAY, addr(C_VAR));
    asm popf end;
  end; (* WAIT_ON_CONDITION_VARIABLE *)

procedure SEND_MESSAGE_TO
(**************************************************************************)
  (RECIPIENT_TASK : TASK_IDS; XMIT_MESSAGE_PTR : pointer);
(*------------------------------------------------------------------------*)
(*  This routine sends the message (actually the pointer) to the task by *)
(* placing it into the task's mailbox (which is automatically created if *)
(* necessary).  Although the calling task cannot block while sending a the *)
(* message it is possible for a reschedule to occur.                      *)
(**************************************************************************)

  var   (********************************************************************)
        (*  Pointer to Task Control Block which is waiting on the message *)
        (* (may very well be nil).                                          *)
        (********************************************************************)
        WAITING_TASK_PTR : TASK_CONTROL_BLOCK_PTR;

        (********************************************************************)
        (*  Pointer to mailbox structure which is to receive the message.   *)
        (********************************************************************)
        RECIPIENT_MBOX_PTR : MAILBOX_PTR;

        (********************************************************************)
        (*  Pointer to message list elements, used to 'walk' through the *)
        (* message list until the end of the list is found (messages must be *)
        (* received in FIFO manner).                                        *)
        (********************************************************************)
        WALKING_PTR : MESSAGE_PTR;

        (********************************************************************)
        (*  Temporary message pointer used to insert the message into the *)
        (* message list.                                                    *)
        (********************************************************************)
        TEMP_PTR : MESSAGE_PTR;

  begin (* SEND_MESSAGE_TO *)
    asm
        pushf
        cli
    end; (* asm *)
    RECIPIENT_MBOX_PTR := FIND_OR_CREATE_MBOX_FOR(RECIPIENT_TASK);
    WALKING_PTR := RECIPIENT_MBOX_PTR^.CONTENTS;
    (* Find the end of the list, allocate storage and create link *)
    if WALKING_PTR = nil
      then
        begin
          new(TEMP_PTR);
          RECIPIENT_MBOX_PTR^.CONTENTS := TEMP_PTR;
        end (* if...then *)
      else
        begin
          while WALKING_PTR^.NEXT <> nil do WALKING_PTR := WALKING_PTR^.NEXT;
          new(WALKING_PTR^.NEXT);
          TEMP_PTR := WALKING_PTR^.NEXT;
        end; (* if...then...else *)
    (* Copy the mail into the recipient's mailbox *)
    TEMP_PTR^.INFO_ADDRESS := XMIT_MESSAGE_PTR;
    TEMP_PTR^.NEXT := nil;
    (* Tell the recipient that it is there, which enables interrupts *)
    SIGNAL_SEMAPHORE(RECIPIENT_MBOX_PTR^.SEM);
    asm popf end;
  end; (* SEND_MESSAGE_TO *)

procedure RECEIVE_MESSAGE
(**************************************************************************)
  (var RCV_MESSAGE_PTR : pointer);
(*------------------------------------------------------------------------*)
(*  This routine retrieves a message (actually a pointer to a message) from *)
(* the caller's mailbox (which is created if necessary), if there are no *)
(* messages in the mailbox then nil is returned.  The task cannot become *)
(* blocked by calling this routine but a reschedule is possible.          *)
(**************************************************************************)

  var   (********************************************************************)
        (*  Pointer to mailbox structure which has received the message (may *)
        (* very well point to empty mailbox).                               *)
        (********************************************************************)
        RECIPIENT_MBOX_PTR : MAILBOX_PTR;

        (********************************************************************)
        (*  Temporary pointer used to extract the mail from the message list.*)
        (********************************************************************)
        TEMP_PTR : MESSAGE_PTR;

  begin (* RECEIVE_MESSAGE *)
    asm
        pushf
        cli
    end; (* asm *)
    (* Setup return value in case the mailbox is empty *)
    RCV_MESSAGE_PTR := nil;
    RECIPIENT_MBOX_PTR := FIND_OR_CREATE_MBOX_FOR(RUNNING_TASK_PTR^.TASK_ID);
    (* Check to see if there is mail to receive *)
    if RECIPIENT_MBOX_PTR^.CONTENTS <> nil then
      begin
        (* Copy the mail into the recipients buffer *)
        TEMP_PTR := RECIPIENT_MBOX_PTR^.CONTENTS;
        RCV_MESSAGE_PTR := TEMP_PTR^.INFO_ADDRESS;
        (* Deallocate the mail storage *)
        RECIPIENT_MBOX_PTR^.CONTENTS := TEMP_PTR^.NEXT;
        dispose(TEMP_PTR);
      end; (* if...then *)
    REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID, READY);
    asm popf end;
  end; (* RECEIVE_MESSAGE *)

procedure WAIT_AND_RECEIVE_MESSAGE
(**************************************************************************)
  (var RCV_MESSAGE_PTR : pointer);
(*------------------------------------------------------------------------*)
(*  This routine retrieves a message (actually a pointer to a message) from *)
(* the caller's mailbox (which is created if necessary), if there are no *)
(* messages in the mailbox then the task is blocked waiting for the mailbox *)
(* to become non-empty, because of this the return pointer can never be nil. *)
(**************************************************************************)

  var   (********************************************************************)
        (*  Pointer to mailbox structure which has received the message (may *)
        (* very well point to empty mailbox).                               *)
        (********************************************************************)
        RECIPIENT_MBOX_PTR : MAILBOX_PTR;

        (********************************************************************)
        (*  Temporary pointer used to extract the mail from the message list.*)
```

```
        (********************************************************************)
        TEMP_PTR : MESSAGE_PTR;

  begin (* WAIT_AND_RECEIVE_MESSAGE *)
    asm
        pushf
        cli
    end; (* asm *)
    RECIPIENT_MBOX_PTR := FIND_OR_CREATE_MBOX_FOR(RUNNING_TASK_PTR^.TASK_ID);
    (* Check to see if there is mail to receive, which enables interrupts *)
    WAIT_ON_SEMAPHORE(RECIPIENT_MBOX_PTR^.SEM);
    (* Copy the mail into the recipients buffer *)
    TEMP_PTR := RECIPIENT_MBOX_PTR^.CONTENTS;
    RCV_MESSAGE_PTR := TEMP_PTR^.INFO_ADDRESS;
    (* Deallocate the mail storage *)
    RECIPIENT_MBOX_PTR^.CONTENTS := TEMP_PTR^.NEXT;
    dispose(TEMP_PTR);
    asm popf end;
  end; (* WAIT_AND_RECEIVE_MESSAGE *)

procedure ENABLE_MOUSE_ACTIONS
(**************************************************************************)
  (MOUSE_ACTIONS : word);
(*------------------------------------------------------------------------*)
(*  This routine enables the specified MS-Mouse events to be received by *)
(* the task which is (or will be) waiting to receive mouse actions.       *)
(**************************************************************************)

  begin (* ENABLE_MOUSE_ACTIONS *)
    asm
        pushf
        cli
    end; (* asm *)
    if not MOUSE_INSTALLED
      then
        with RUNNING_TASK_PTR^ do
          ERROR_HANDLER(INSUFFICIENT_RESOURCES)(TASK_ID)
      else
        begin
          if USER_MOUSE_EVENT_PTR = nil then
            begin
              new(USER_MOUSE_EVENT_PTR);
              USER_MOUSE_EVENT_PTR^ := UNSIGNALED;
            end; (* if...then *)
          PUSH_MOUSE_EVENT_HANDLER(MOUSE_ACTIONS, MOUSE_EVENT_HANDLER);
          ENABLE_MOUSE_DRIVER;
        end; (* if...then...else *)
    asm popf end;
  end; (* ENABLE_MOUSE_ACTIONS *)

procedure WAIT_AND_RECEIVE_MOUSE_ACTIONS
(**************************************************************************)
  (var MOUSE_INFO : MOUSE_PARAMETERS);
(*------------------------------------------------------------------------*)
(*  This routine receives MS-Mouse action parameters from the MS-Mouse *)
(* driver.  If none of the enabled mouse actions has occurred then the task *)
(* is blocked waiting for one (or more) action to occur.                  *)
(**************************************************************************)

  begin (* WAIT_AND_RECEIVE_MOUSE_MESSAGE *)
    asm
        pushf
        cli
    end; (* asm *)
    if USER_MOUSE_EVENT_PTR = nil
      then
        begin
          with RUNNING_TASK_PTR^ do
            ERROR_HANDLER[INSUFFICIENT_RESOURCES)(TASK_ID);
          REQUEUE_AND_RESCHEDULE(RUNNING_TASK_PTR^.TASK_ID, READY);
        end (* if...then *)
      else
        begin
          WAIT_ON_EVENT(USER_MOUSE_EVENT_PTR^);
          MOUSE_INFO := MOUSE_INFO_FROM_HANDLER;
        end; (* if...then...else *)
    asm popf end;
  end; (* WAIT_AND_RECEIVE_MOUSE_MESSAGE *)

function WAIT_ON_READKEY : char;
(**************************************************************************)
(*  This routine provides a means for an application task to perform I/O in *)
(* a 'blocking' manner.  This means that while there is no input from the PC *)
(* keyboard the calling task is suspended, as soon as input arrives from the *)
(* keyboard the caller is awakened and given the ASCII code of the key that *)
(* was pressed by the user (i.e., a blocking equivalent of 'readkey').    *)
(**************************************************************************)

  var   (********************************************************************)
        (*  Intel 80x86 register variables used to pass register values to *)
        (* software interrupt handler.                                      *)
        (********************************************************************)
        REGS : registers;

  begin (* WAIT_ON_READKEY *)
    asm
        pushf
        cli
    end; (* asm *)
    if USER_KEYBOAND_SEM_PTR = nil then
      begin
        new(USER_KEYBOAND_SEM_PTR);
        USER_KEYBOARD_SEM_PTR^ := 0;
        (* Flush keyboard buffer, throw away all characters in the buffer. *)
        REGS.AN := $0C;
        REGS.AL := $06;
        REGS.DX := $FFFF;
        intr($16, REGS);
        asm cli end;
      end; (* if...then *)
    WAIT_ON_SEMAPHORE(USER_KEYBOAND_SEM_PTR^);
    (* Using this service instead of 'readkey' allows F11 & F12 detection. *)
    REGS.AX := $0800;
    intr($21, REGS);
    asm cli end;
    (**********************************************************************)
    (*  To ensure that the application will get extended keys the null part *)
    (* of the extended key is returned and the controlling semaphore is *)
    (* signaled to allow the application to come back and get the extended *)
    (* part of the key without waiting.                                     *)
    (**********************************************************************)
    WAIT_ON_READKEY := chr(REGS.AL);
    if REGS.AL = ord(NULL_CHR) then
      SIGNAL_SEMAPHORE(USER_KEYBOARD_SEM_PTR^);
    asm popf end;
  end; (* WAIT_ON_READKEY *)

(**************************************************************************)
(*                     Scheduler Handling Operations                      *)
(**************************************************************************)

procedure LOCK_SCHEDULER;
(**************************************************************************)
(*  This routine prevents all task rescheduling from occurring.  Care must *)
(* be taken to ensure that a call to UNLOCK_SCHEDULER is made for each call *)
(* to LOCK_SCHEDULER.  In order to maximize system performance, execution *)
(* time within 'locked regions' should be an absolute minimum.            *)
(**************************************************************************)
```

```
  begin (* LOCK_SCHEDULER *)
    asm
        pushf
        cli
    end; (* asm *)
    inc(IN_PREEMPTABLE_REGION);
    asm popf end;
  end; (* LOCK_SCHEDULER *)

procedure UNLOCK_SCHEDULER;
(**************************************************************************)
(*   This routine allows task rescheduling to occur.  Care must be taken  to *)
(* ensure that this routine is only called after LOGK_SCHEDULER has  already *)
(* been called.                                                            *)
(**************************************************************************)

  begin (* UNLOCK_SCHEDULER *)
    asm
        pushf
        cli
    end; (* asm *)
    if IN_PREEMPTABLE_REGION > 0
      then
        dec(IN_PREEMPTABLE_REGION)
      else
        with RUNNING_TASK_PTR^ do ERROR_HANDLER[ILLEGAL_OPERRTION](TASK_ID);
    asm popf end;
  end; (* UNLOCK_SCHEDULER *)

(**************************************************************************)
(*                          End of user functions                          *)
(**************************************************************************)

(**************************************************************************)
(*                         Default Error Handlers                          *)
(*$F+  Handlers must be 'far' because they are used as procedural variables  *)
(**************************************************************************)

type (*********************************************************************)
     (*  String used to create time stamps for default error handlers.      *)
     (*********************************************************************)
     STAMP_STRING = string[27];

procedure OPEN_ERROR_LOG;
(**************************************************************************)
(*  Creates (or appends) the TASKING error log file in the same directory as *)
(* the program executable file.                                            *)
(**************************************************************************)

  var (*********************************************************************)
      (*  Used to determine the directory  of  the  application  executable, *)
      (* that is where the error log file will be placed.                 *)
      (*********************************************************************)
      DIR : dirstr;  NAME   namestr;  EXT : extstr;

  begin (* OPEN_ERROR_LOG *)
    (* Open Error Log file (with append if possible) *)
    fsplit(paramstr(0), DIR, NAME, EXT);
    assign(ERROR_LOG, DIR + 'TASKING.ERR');
    (*$I-*) reset(ERROR_LOG); (*$I+*)
    if ioresult = 0
      then
        append(ERROR_LOG)
      else
        rewrite(ERROR_LOG);
  end; (* OPEN_ERROR_LOG *)

function TIME_STAMP : STAMP_STRING; near;
(**************************************************************************)
(*  Returns a time stamp in the form : 'DD/MM/YY■HH:MM:SSam'.              *)
(**************************************************************************)

  var YEAR, MONTH, DAY, DAY_OF_WEEK : word;
      HOUR, MIN,   SEC, SEC100TH    : word;
      STAMP : STAMP_STRING;
      TEMP  : string[6];

  begin (* TIME_STAMP *)
    (* Date stamp *)
    getdate(YEAR, MONTH, DAY, DAY_OF_WEEK);
    STAMP := '';
    STAMP :=         chr(ord(MONTH div 10) + ord('0'));
    STAMP := STAMP + chr(ord(MONTH mod 10) + ord('0')) + '/';
    STAMP := STAMP + chr(ord(DAY   div 10) + ord('0'));
    STAMP := STAMP + chr(ord(DAY   mod 10) + ord('0')) + '/';
    STAMP := STAMP + chr(ord((YEAR mod 100) div 10) + ord('0'));
    STAMP := STAMP + chr(ord(YEAR  mod 10) + ord('0')) + '■';

    (* Time stamp *)
    gettime(HOUR, MIN, SEC, SEC100TH);
    str(HOUR mod 12, TEMP);
    if TEMP = '0'
      then
        TEMP := '12'
      else if HOUR < 10 then
        TEMP := TEMP + '0';
    STAMP := STAMP + TEMP + ':';
    if MIN < 10 then STAMP := STAMP + '0';
    str(MIN, TEMP);
    STAMP := STAMP + TEMP + ':';
    if SEC < 10 then STAMP := STAMP + '0';
    str(SEC, TEMP);
    STAMP := STAMP + TEMP;
    if (HOUR div 12) = 0
      then
        TIME_STAMP := STAMP + 'am'
      else
        TIME_STAMP := STAMP + 'pm';
  end; (* TIME_STAMP *)

procedure TASK_ALREADY_ACTIVE_ERROR_HANDLER(TASK_ID : TASK_IDS);

  begin (* TASK_ALREADY_ACTIVE_ERROR_HANDLER *)
    OPEN_ERROR_LOG;
    writeln(ERROR_LOG, TIME_STAMP, ': ',
      'Warning! Task is already active (', TASK_ID, '), request ignored.');
    close(ERROR_LOG);
  end; (* TASK_ALREADY_ACTIVE_ERROR_HANDLER *)

procedure INSUFFICIENT_RESOURCES_ERROR_HANDLER(TASK_ID : TASK_IDS);

  begin (* INSUFFICIENT_RESOURCES_ERROR_HANDLER *)
    OPEN_ERROR_LOG;
    writeln(ERROR_LOG, TIME_STAMP, ': ',
      'Warning! Insufficient resources (', TASK_ID, '), request ignored.');
    close(ERROR_LOG);
    halt($FF);
  end; (* INSUFFICIENT_RESOURCES_ERROR_HANDLER *)

procedure TASK_IS_NOT_ACTIVE_ERROR_HANDLER(TASK_ID : TASK_IDS);

  begin (* TASK_IS_NOT_ACTIVE_ERROR_HANDLER *)
    OPEN_ERROR_LOG;
    writeln(ERROR_LOG, TIME_STAMP, ': ',
      'Warning! Task is not active (', TASK_ID, '), request ignored.');
    close(ERROR_LOG);
  end; (* TASK_IS_NOT_ACTIVE_ERROR_HANDLER *)

procedure TASK_ALREADY_SUSPENDED_ERROR_HANDLER(TASK_ID : TASK_IDS);

  begin (* TASK_ALREADY_SUSPENDED_ERROR_HANDLER *)
    OPEN_ERROR_LOG;
    writeln(ERROR_LOG, TIME_STAMP, ': ',
      'Warning! Task already suspended (', TASK_ID, '), request ignored.');
    close(ERROR_LOG);
  end; (* TASK_ALREADY_SUSPENDED_ERROR_HANDLER *)

procedure ILLEGAL_TASK_ID_ERROR_HANDLER(TASK_ID : TASK_IDS);

  begin (* ILLEGAL_TASK_ID_ERROR_HANDLER *)
    OPEN_ERROR_LOG;
    writeln(ERROR_LOG, TIME_STAMP, ': ',
      'Warning! Illegal task identifier (', TASK_ID, '), request ignored.');
    close(ERROR_LOG);
  end; (* ILLEGAL_TASK_ID_ERROR_HANDLER *)

procedure ILLEGAL_OPERRTION_ERROR_HANDLER(TASK_ID : TASK_IDS);

  begin (* ILLEGAL_OPERRTION_ERROR_HANDLER *)
    OPEN_ERROR_LOG;
    writeln(ERROR_LOG, TIME_STAMP, ': ',
      'Error' Illegal operation on task (', TASK_ID, '), destroying task (',
      RUNNING_TASK_PTR^.TASK_ID, ')');
    DESTROY(RUNNING_TASK_PTR^.TASK_ID);
    close(ERROR_LOG);
  end; (* ILLEGAL_OPERRTION_ERROR_HANDLER *)

procedure DEADLOCK_DETECTED_ERROR_HANDLER;

  begin (* DEADLOCK_DETECTED_ERROR_HANDLER *)
    OPEN_ERROR_LOG;
    writeln(ERROR_LOG, TIME_STAMP, ': ',
      'Error' Deadlock detected'  Terminating application!');
    writeln(TIME_STAMP, ': ',
      'Error' Deadlock detected'  Terminating application!');
    close(ERROR_LOG);
    halt($FF);
  end; (* DEADLOCK_DETECTED_ERROR_HANDLER *)

procedure PROGRAM_CANNOT_TERMINATE_ERROR_HANDLER;

  begin (* PROGRAM_CANNOT_TERMINATE_ERROR_RAMDLER *)
    OPEN_ERROR_LOG;
    writeln(ERROR_LOG, TIME_STRMP, ': ',
      'Error' Termination delayed, (', NUMBER_OF_CHILD_PROGRAMS,
          '), child program(s) active.');
    close(ERROR_LOG);
  end; (* PROGRAM_CANNOT_TERMINATE_ERROR_HANDLER *)

function TASKING_HEAP_ERROR_HANDLER(SIZE : word) : integer;
(**************************************************************************)
(*   The run-time systems expects the heap error handler  to  return  status *)
(* information according to the following rules:                           *)
(*   0 indicates failure, and causes a run-time error to occur immediately.  *)
(*   1 indicates failure, and causes new()  or  getmem()  to  return  a  nil *)
(*     pointer.                                                            *)
(*   2 indicates success, and causes a retry (which could also cause another *)
(*     call to the heap error function).                                   *)
(**************************************************************************)

  begin (* TASKING_HEAP_ERROR_HANDLER *)
    if SIZE <> 0
      then
        begin
          OPEN_ERROR_LOG;
          writeln(ERROR_LOG, TIME_STAMP, ': ',
            'Error' Unable to allocate Heap storage for ', SIZE, ' bytes.');
          writeln(TIME_STAMP, ': ',
            'Error' Unable to allocate Heap storage for ', SIZE, ' bytes.');
          (* Cause run-time error *)
          TASKING_HEAP_ERROR_HANDLER := 0;
          close(ERROR_LOG);
        end (* if...then *)
      else
        (* False alarm *)
        TASKING_HEAP_ERROR_HANDLER := 2;
  end; (* TASKING_HEAP_ERROR_HANDLER *)

(**************************************************************************)
(*                       End of Default Error Handlers                     *)
(*$F-                                                                       *)
(**************************************************************************)

procedure REPORT_TASKING_STATISTICS; far;
(**************************************************************************)
(*  This routine is linked  into  the  exit  procedure  chain  if  TASKING *)
(* statistics are to be reported.                                          *)
(**************************************************************************)

  const (*********************************************************************)
        (*  Allows quick and easy conversion between hexadecimal digits and *)
        (* the corresponding ASCII character.                               *)
        (*********************************************************************)
        HEX_DIGIT : array [0..15] of char = ('0','1','2','3','4','5','6','7',
                                             '8','9','A','B','C','D','E','F');

        (*********************************************************************)
        (*  IRQ.RPT header for hardware IRQs at application termination.     *)
        (*********************************************************************)
        IRQ_HEADER =
         CR + LF +
         '  Vector     IRQ         Interrupts     Period (msec)' +
         '      Frequency (Hz)';

        (*********************************************************************)
        (*  SERVICES.RPT header for software service  calls  at  application *)
        (* termination.                                                     *)
        (*********************************************************************)
        SERVICE_HEADER =
         CR + LF +
         '  Vector(s)      Service Calls     Period (msec)'+
         '      Frequency (Hz)';

        (*********************************************************************)
        (*  TASKING.RPT header for tasks at application termination.        *)
        (*********************************************************************)
        TASKING_HEADER_1 = '    Tasks at Termination:';
        TASKING_HEADER_2 = '           Task State    Task            ' +
                           'Stack Size Stack Used Stack Used CPU Used';
        TASKING_HEADER_3 = '        (Delayed Until)  ID   Priority   ' +
                           '(words)    (words)      (%)        (%)   ';
        TASKING_HEADER_4 = '     _______________ _____ ________ ' +
                           '__________ __________ __________ ________';

  type  (*********************************************************************)
        (*  String used to convert long integers (double words) into strings *)
        (* with commas embedded, i.e., worst case is -2,147,483,647.          *)
        (*********************************************************************)
        LONGINT_STRING = string[14];

  var   (*********************************************************************)
        (*  Loop control variable of task IDs, allows all  task  IDs  to  be *)
        (* checked for activity so none are missed in task report.          *)
        (*********************************************************************)
        ACTIVE_TASK : TASK_IDS;

        (*********************************************************************)
        (*  Set used to search/access all TASK_IDs.                          *)
```

```
    (*******************************************************************)
    TASK_ID_SET : TASK_ID_SETS;

    (*******************************************************************)
    (*  Pointer to Task Control Blocks which are found  in  any  of  the *)
    (* TASKING queues.                                                  *)
    (*******************************************************************)
    ACTIVE_TASK_PTR : TASK_CONTROL_BLOCK_PTR;

    (*******************************************************************)
    (*  Total application execution time (as reported by TASKING).      *)
    (*******************************************************************)
    ELAPSED_EXECUTION_TIME : real;

    (*******************************************************************)
    (*  Used in computing ELAPSED_TIME.                                  *)
    (*******************************************************************)
    LCV, YEAR, MONTH, DAY, DAY_OF_WEEK, HOUR, MIN, SEC, SEC100 : word;

    (*******************************************************************)
    (*  Absolute stop time (as reported by TASKING).                     *)
    (*******************************************************************)
    STOP_TIME : real;

    (*******************************************************************)
    (*  Total application execution time (as reported by MS-DOS).       *)
    (*******************************************************************)
    ELAPSED_TIME : real;

    (*******************************************************************)
    (*  Amount of the CPU that the application used.                     *)
    (*******************************************************************)
    CPU_PERCENT, TOTAL_CPU_PERCENT : real;

    (*******************************************************************)
    (*  Loop control variables used  to  report  on  interrupt  handlers *)
    (* protected by TASKING (both hardware and software).               *)
    (*******************************************************************)
    IRQ, INTERRUPT_NUMBER : integer;

    (*******************************************************************)
    (*  Temporary variables used to detect ranges of interrupt with  the *)
    (* same report parameters (makes a less cluttered report).          *)
    (*******************************************************************)
    PREVIOUS_INTERRUPT_NUMBER, START, FINISH : integer;
    PREVIOUS_SERVICE_CALLS                   : longint;

    (*******************************************************************)
    (*  Counts the number of unprotected interrupt handlers so that  the *)
    (* output will not wrap to next output line.                        *)
    (*******************************************************************)
    NUM_UNPROTECTED : integer;

    (*******************************************************************)
    (*  Index into each tasks stack area, used to determine how much  of *)
    (* the stack was actually used by the task.                         *)
    (*******************************************************************)
    MAX_STACK_USED : longint;

    (*******************************************************************)
    (*  Percentage of task stack  space,  used  to  determine  how  much *)
    (* stack headroom exists.                                           *)
    (*******************************************************************)
    STACK_PERCENT, MAX_STACK_PERCENT : real;

    (*******************************************************************)
    (*  Total amount of memory used by the application tasks for  stacks *)
    (* (only counts tasks which are active at termination!).            *)
    (*******************************************************************)
    APPLICATION_STACK_SPACE : longint;

    (*******************************************************************)
    (*  Text filename used for all reports generated by TASKING.        *)
    (*******************************************************************)
    REPORT : text;

    function ADD_COMMAS
    (*******************************************************************)
      (NUMBER : longint) : LONGINT_STRING;
    (*-----------------------------------------------------------------*)
    (*  This procedure simply adds commas into the  NUMBER  and  returns *)
    (* the corresponding character string.                              *)
    (*******************************************************************)

      var TEMP_STR : LONGINT_STRING;
          CH       : integer;

      begin (* ADD_COMMAS *)
        str(NUMBER, TEMP_STR);
        CH := length(TEMP_STR) - 3;
        while CH > 0 do
          begin
            insert(',', TEMP_STR, CH + 1);
            CH := CH - 3;
          end; (* while..do *)
        ADD_COMMAS := TEMP_STR;
      end; (* ADD_COMMAS *)

begin (* REPORT_TASKING_STATISTICS *)
  (* Restore exit procedure *)
  exitproc := SAVE_EXIT;

  (* Determine program termination time *)
  getdate(YEAR, MONTH, DAY, DAY_OF_WEEK);
  gettime(HOUR, MIN, SEC, SEC100);
  STOP_TIME := 0.0;
  for LCV := 1 to MONTH-1 do
    STOP_TIME := STOP_TIME + DAYS_IN[LCV];
  STOP_TIME := (((STOP_TIME + DAY) * 24 + HOUR) * 60 + MIN) * 60 +
                SEC + SEC100 / 100;
  ELAPSED_TIME := STOP_TIME - TASKING_STATISTICS.START_TIME;
  if ELAPSED_TIME = 0 then ELAPSED_TIME := -1;

  (* Determine elapsed execution time *)
  ELAPSED_EXECUTION_TIME := MILLISECOND_TICKS / 1000 (Sec);

  if ACTUAL_TASKING_CONFIGURATION.STATISTICS in
     [HARDWARE_INTERRUPT_STATISTICS, ALL_STATISTICS] then
    begin
      (* Create Hardware Interrupt Report *)
      assign(REPORT, 'IRQ.RPT');
      (*$I-*) rewrite(REPORT); (* $I+*)
      writeln(REPORT, IRQ_HEADER);
      for IRQ := 0 to $F do
        with TASKING_STATISTICS do
          begin
            if IRQ < 8
              then
                write(REPORT, HEX_DIGIT[(IRQ + $08) shr 4] : 5,
                              HEX_DIGIT[(IRQ + $08) and $0F])
              else
                write(REPORT, HEX_DIGIT[(IRQ + $68) shr 4] : 5,
                              HEX_DIGIT[(IRQ + $68) and $0F]);
            case IRQ of
              $0 : write(REPORT, '     Timer 0     ');
              $1 : write(REPORT, '     Keyboard    ');
              $2 : write(REPORT, '    Slave 8259   ');
              $3 : write(REPORT, '      COM 1      ');
              $4 : write(REPORT, '      COM 2      ');
              $6 : write(REPORT, '   Floppy Disk   ');
              $7 : write(REPORT, '      LPT 1      ');
              $8 : write(REPORT, '  Real-Time Clock');
              $D : write(REPORT, '   Co-processor  ');
              $E : write(REPORT, '    Fixed Disk   ');
              else write(REPORT, '           ', IRQ, ' ' : 8 - (IRQ div 10));
            end; (* case...of *)
            write(REPORT, HARDWARE_INTERRUPTS[IRQ]   9);
            if HARDWARE_INTERRUPTS[IRQ] <> 0
              then
                writeln(REPORT, (ELAPSED_TIME * 1000 (mSec) /
                                HARDWARE_INTERRUPTS[IRQ]) : 16 : 2,
                                (HARDWARE_INTERRUPTS[IRQ] /
                                ELAPSED_TIME) : 20 : 2)
              else
                writeln(REPORT, 'N/A'   16, 'N/A' : 20);
          end; (* with...do *)
      close(REPORT);
    end; (* if...then *)

  if ACTUAL_TASKING_CONFIGURATION.STATISTICS in
     [MSDOS_SERVICES_STATISTICS, ALL_STATISTICS] then
    begin
      (* Create Software Interrupt Service Report *)
      assign(REPORT, 'SERVICES.RPT');
      (*$I-*) rewrite(REPORT); (* $I+*)
      NUM_UNPROTECTED := 0;
      FINISH := 0;
      START  := -1;
      PREVIOUS_INTERRUPT_NUMBER := -1;
      write(REPORT, CR, LF, ' Unprotected Interrupts:', CR, LF, '  ');
      for INTERRUPT_NUMBER := 0 to $FF do
        if not (INTERRUPT_NUMBER in PROTECTED_INTERRUPTS) then
          begin
            if START = -1
              then
                begin
                  START  := INTERRUPT_NUMBER,
                  FINISH := INTERRUPT_NUMBER;
                end (* if...then *)
              else
                if (INTERRUPT_NUMBER = PREVIOUS_INTERRUPT_NUMBER + 1)
                  then
                    FINISH := INTERRUPT_NUMBER
                  else
                    begin
                      if START = FINISH
                        then
                          begin
                            write(REPORT, ' ', HEX_DIGIT[START shr 4],
                                               HEX_DIGIT[START and $0F]);
                            inc(NUM_UNPROTECTED);
                            if (NUM_UNPROTECTED mod 25) = 0 then
                              write(REPORT, CR, LF, '  ');
                          end (* if...then *)
                        else
                          begin
                            write(REPORT, ' ', HEX_DIGIT[START shr 4],
                                               HEX_DIGIT[START and $0F],
                                          '..',HEX_DIGIT[FINISH shr 4],
                                               HEX_DIGIT[FINISH and $0F]);
                            inc(NUM_UNPROTECTED, 3);
                            if (NUM_UNPROTECTED mod 25) < 3 then
                              write(REPORT, CR, LF, '  ');
                          end; (* if...then...else *)
                      START := INTERRUPT_NUMBER;
                      FINISH := INTERRUPT_NUMBER;
                    end; (* if...then...else *)
            PREVIOUS_INTERRUPT_NUMBER := INTERRUPT_NUMBER;
          end; (* if...then *)
      writeln(REPORT, SERVICE_HEADER);
      FINISH := 0;
      START  := -1;
      PREVIOUS_SERVICE_CALLS := -1;
      PREVIOUS_INTERRUPT_NUMBER := -1;
      for INTERRUPT_NUMBER := 0 to $FF do
        if INTERRUPT_NUMBER in PROTECTED_INTERRUPTS then
          begin
            if START = -1 then START := INTERRUPT_NUMBER;
            if ((TASKING_STATISTICS.SERVICE_CALLS[INTERRUPT_NUMBER] =
                 PREVIOUS_SERVICE_CALLS) and
                (INTERRUPT_NUMBER = (PREVIOUS_INTERRUPT_NUMBER + 1))) or
                (PREVIOUS_SERVICE_CALLS = -1)
              then
                FINISH := INTERRUPT_NUMBER
              else
                begin
                  with TASKING_STATISTICS do
                    begin
                      if START = FINISH
                        then
                          write(REPORT, HEX_DIGIT[START shr 4] : 6,
                                        HEX_DIGIT[START and $0F],
                                        SERVICE_CALLS[INTERRUPT_NUMBER] . 18)
                        else
                          write(REPORT, HEX_DIGIT[START shr 4] : 4,
                                        HEX_DIGIT[START and $0F], '..',
                                        HEX_DIGIT[FINISH shr 4],
                                        HEX_DIGIT[FINISH and $0F],
                                        SERVICE_CALLS[INTERRUPT_NUMBER] :16);
                      if SERVICE_CALLS[INTERRUPT_NUMBER] <> 0
                        then
                          writeln(REPORT, (ELAPSED_TIME * 1000 (mSec) /
                                           SERVICE_CALLS]INTERRUPT_NUMBER])
                                           : 18 : 2,
                                          (SERVICE_CALLS[INTERRUPT_NUMBER] /
                                           ELAPSED_TIME) : 20 :2)
                        else
                          writeln(REPORT, 'N/A' : 18, 'N/A' : 20);
                    end; (* with...do *)
                  START  := INTERRUPT_NUMBER;
                  FINISH := INTERRUPT_NUMBER;
                end; (* if...then...else *)
            PREVIOUS_SERVICE_CALLS :=
                TASKING_STATISTICS.SERVICE_CALLS[INTERRUPT_NUMBER];
            PREVIOUS_INTERRUPT_NUMBER := INTERRUPT_NUMBER;
          end; (* if...then *)
      if START <> FINISH
        then
          with TASKING_STATISTICS do
            begin
              write(REPORT, HEX_DIGIT[START shr 4] : 4,
                            HEX_DIGIT[START and $0F], '..',
                            HEX_DIGIT[FINISH shr 4],
                            HEX_DIGIT[FINISH and $0F],
                            SERVICE_CALLS[INTERRUPT_NUMBER]   16);
              if SERVICE_CALLS]INTERRUPT_NUMBER] <> 0
                then
                  writeln(REPORT, (ELAPSED_TIME * 1000 (mSec) /
                                   SERVICE_CALLS[INTERRUPT_NUMBER]) : 18   2,
                                  (SERVICE_CALLS[INTERRUPT_NUMBER] /
                                   ELAPSED_TIME) : 20 :2)
                else
                  writeln(REPORT, 'N/A' : 18, 'N/A' : 20);
            end; (* with...do *)
      close(REPORT);
    end; (* if...then *)

  if ACTUAL_TASKING_CONFIGURATION.STATISTICS in
```

```
  (TASK_STATISTICS, ALL_STATISTICS) then
with TASKING_STATISTICS do
  begin
    (* Create Task Activity Report *)
    APPLICATION_STACK_SPACE := 0;
    MAX_STACK_PERCENT := 0;
    TOTAL_CPU_PERCENT := 0.0;
    assign(REPORT, 'TASKING.RPT');
    (*SI-*) rewrite(REPDRT); (* $I+*)
    if ACTUAL_TASKING_CONFIGURRTION.TASKING_MDDEL = COOPERRTIVE
      then
        write(REPORT, 'Cooperative Multi-tasking')
      else
        write(REPORT, 'Preemptive Multi-tasking');
    writeln(REPORT,   ' Statistical Information');
    write(REPORT, '  -Priority Inheritance ');
    if ACTUAL_TASKING_CONFIGURATIDN.PRIORITY_INHERITANCE_ENABLED
      then
        writeln(REPORT, 'Enabled')
      else
        writeln(REPORT, 'Disabled');
    case ACTUAL_TASKING_CONFIGURRTION.PRIORITY_SCHEDULING_PDLICY of
      STATIC_PRIORITIES   : writeln(REPORT, '  -Static Priorities');
      ROTATING_PRIORITIES : writeln(REPORT, '  -Rotating Priorities');
    end; (* case...of *)
    writeln(REPORT);
    writeln(REPORT,   '  Task Activity:');
    writeln(REPORT,   '     Context Switches = ', CONTEXT_SWITCHES : 12,
                      ' (-', round(CONTEXT_SWITCHES / ELAPSED_TIME),
                      ' per second)');
    writeln(REPORT,   '        Cooperative    = ',
                      CDOPERRTIVE_CONTEXT_SWITCHES : 12,
                      ' (-', round(COOPERRTIVE_CONTEXT_SWITCHES /
                            ELAPSED_TIME), ' per second)');
    writeln(REPORT,   '        Preemptive     = ',
                      CONTEXT_SWITCHES-COOPERRTIVE_CONTEXT_SWITCHES:12,
                      ' (-', round((CONTEXT_SWITCHES -
                               COOPERATIVE_CONTEXT_SWITCHES) /
                               ELAPSED_TIME), ' per second)');
    if ACTUAL_TASKING_CONFIGURRTION.TASKING_MODEL = PREEMPTIVE then
      writeln(REPORT, '     Target Time Slice        = ',
                      ACTUAL_TASKING_CONFIGURATION.TARGET_TIMESLICE,
                      ' usec');
    if CONTEXT_SWITCHES <> 0 then
      writeln(REPORT, '     Achieved Time Slice      = -',
                      round(ELAPSED_TIME / CONTEXT_SWITCHES * 1e6),
                      ' usec');
    if HARDWARE_INTERRUPTS(0) <> 0 then
      writeln(REPORT,   '     Available CPU Bandwidth = ',
                       (100.0 * NULL.CLOCK_TICKS /
                       HARDWARE_INTERRUPTS(0)) : 6 : 3, ' %');
    if NUMBER_DF_PERIDDIC_EVENTS <> 0 then
      writeln(REPORT, '     Periodic Event Faults    = ',
                      (100.0 * NUMBER_OF_PERIODIC_EVENTS_MISSED /
                      NUMBER_OF_PERIODIC_EVENTS) : 6 : 3, ' %');
    writeln(REPORT, TASKING_HEADER_1);
    writeln(REPORT, TASKING_HEADER_2);
    writeln(REPORT, TASKING_HEADER_3);
    writeln(REPORT, TASKING_HEADER_4);
    for TASK_ID_SET := low(TASK_ID_SETS) to high(TASK_ID_SETS) do
      for ACTIVE_TASK := (ord(TASK_ID_SET shl 8) +  low(byte)) to
                         (ord(TASK_ID_SET shl 8) + high(byte)) do
        if lo(ACTIVE_TASK) in ALLOCATED_TASK_IDS(TASK_ID_SET) then
          begin
            if RUNNING_TASK_PTR^.TASK_ID = ACTIVE_TASK
              then
                begin
                  ACTIVE_TASK_PTR := RUNNING_TASK_PTR;
                  write(REPDRT, '       Running ', ' ' : 8);
                end (* if...then *)
              else
                begin
                  ACTIVE_TASK_PTR := FIND_AND_REMOVE
                                      (
                                       ACTIVE_TASK,
                                       (From the) READY_QUEUE,
                                       UNCONDITIONALLY,
                                       THERE_IS_ND_BLOCK_TO_MATCH
                                      );
                  if ACTIVE_TASK_PTR = nil
                    then
                      begin
                        ACTIVE_TASK_PTR := FIND_AND_REMOVE
                                            (
                                             ACTIVE_TASK,
                                             (From the) BLOCKED_QUEUE,
                                             UNCONDITIONALLY,
                                             THERE_IS_NO_BLOCK_TO_MATCH
                                            );
                        if ACTIVE_TASK_PTR = nil
                          then
                            begin
                              ACTIVE_TASK_PTR := FIND_AND_REMOVE
                                                  (
                                                   ACTIVE_TASK,
                                                   (From the) DELAYED_QUEUE,
                                                   UNCONDITIONALLY,
                                                   THERE_IS_NO_BLOCK_TO_MATCH
                                                  );
                              write(REPORT, '       Delayed =',
                                    ACTIVE_TASK_PTR^.
                                    WAITING_FOR^.ABSOLUTE_TIME : 8);
                            end (* if...then *)
                          else
                            write(REPORT, '       Blocked ', ' '   8);
                      end (* if...then *)
                    else
                      write(REPORT, '       Ready   ', ' '   8);
                end; (* if...then...else *)
            with ACTIVE_TASK_PTR^ do
              begin
                MAX_STACK_USED := 0;
                while memw[seg(STACK_PTR^) : ofs(STACK_PTR^) +
                      MAX_STACK_USED] = DEFAULT_STACK_VALUE do
                  inc(MAX_STACK_USED, 2);
                dec(MAX_STACK_USED, 2);
                STACK_SIZE := STACK_SIZE div 2;
                MAX_STACK_USED := STACK_SIZE - (MAX_STACK_USED div 2);
                if PRIORITY > high(USER_PRIORITIES) then
                  PRIORITY := high(USER_PRIORITIES);
                if STACK_SIZE = 0
                  then
                    STACK_PERCENT := -1
                  else
                    STACK_PERCENT := MAX_STACK_USED / STACK_SIZE * 100;
                if STACK_PERCENT > MAX_STACK_PERCENT
                  then MAX_STACK_PERCENT := STACK_PERCENT;
                CPU_PERCENT := 100 * CLOCK_TICKS /
                               HARDWARE_INTERRUPTS(0);
                TOTAL_CPU_PERCENT := TOTAL_CPU_PERCENT + CPU_PERCENT;
                writeln(REPORT, TASK_ID : 4, PRIDRITY : 8,
                                STACK_SIZE : 11,
                                MAX_STACK_USED : 11,
                                STACK_PERCENT : 11 : 1,
                                CPU_PERCENT : 10 : 1);
                inc(APPLICATION_STACK_SPACE, STACK_SIZE * 2);
              end; (* with...do *)
          end; (* if...then *)
    writeln(REPDRT);
    writeln(REPORT, '  Maximum percent of stack used: ',
                     MAX_STACK_PERCENT : 3 : 1, '%');
    writeln(REPORT);
    writeln(REPORT, '  Heap Information (bytes):');
    writeln(REPORT, '     Total Available:             ',
                     ADD_COMMAS(TOTAL_AVAILABLE_HEAP) : 7);
    writeln(REPDRT, '     Used for Application Stacks: ',
                     ADD_COMMAS(APPLICATION_STACK_SPACE)   7);
    writeln(REPORT, '     Available to Application:    ',
                     ADD_COMMAS(TOTAL_AVAILABLE_HEAP -
                                APPLICATION_STACK_SPACE)   7);
    writeln(REPDRT);
    writeln(REPORT, '  Execution Time:');
    writeln(REPORT, '     Effective = ', ELAPSED_EXECUTION_TIME : 8 : 3,
                    ' seconds');
    writeln(REPORT, '     Absolute  = ', ELAPSED_TIME : 7   2,
                    ' seconds');
    close(REPORT);
  end; (* with...do *)
end; (* REPORT_TASKING_STATISTICS *)

procedure TASKING_EXIT; far;
(*************************************************************************)
(*   This routine is linked into the exit procedure chain to ensure that all *)
(* interrupt vectors that this unit hooks are unhooked  before  terminating. *)
(* The exit procedure chain is executed no matter what has happened to cause *)
(* the program to terminate.                                                 *)
(*   The first time this routine is executed it is because the 'main' of the *)
(* user application terminated, at that time all tasks  become  alive.   Any *)
(* subsequent execution of this routine actually terminates the program.     *)
(*   If the application 'main' (or any task) terminates abnormally then  the *)
(* program will terminate (whether tasks are alive or not).                  *)
(*************************************************************************)

  var   (****************************************************************)
        (*  Loop  control  variables  used  to  restore  interrupt  handlers *)
        (* protected by TASKING (both hardware and software).              *)
        (****************************************************************)
        IRQ, INTERRUPT_NUMBER : integer;

  begin (* TASKING_EXIT *)
    (* Application is trying to terminate, wait until it can if necessary... *)
    if NUMBER_OF_CHILD_PROGRAMS <> 0 then
      begin
        PROGRAM_CANNOT_TERMINATE_ERRDR_HANDLER;
        while NUMBER_OF_CHILD_PROGRAMS <> 0 do
          begin
            ACTUAL_TASKING_CONFIGURATIDN.TASKING_MDDEL := PREEMPTIVE;
            asm sti end;
          end; (* if...then *)
      end; (* if...then *)

    (* Make sure that no more context switches are performed *)
    LOCK_SCHEDULER;

    (* Unprogram the timer chip *)
    asm cli end;
    port(TIMER_CONTROL_PORT) := SYSTEM_TIMER_CONTROL_WORD,
    port(TIMER_O_DATA_PORT)  := lo(MSDOS_TIMER_0_VALUE);
    port(TIMER_0_DATA_PORT)  := hi(MSDOS_TIMER_0_VALUE);

    (* Restore revectored interrupt handler *)
    with OLD_REVECTDRED_INTERRUPT do
      begin
        memw($0000 :  REVECTORED_INTERRUPT_NUMBER shl 2)      := OFFSET;
        memw($0000 : (REVECTORED_INTERRUPT_NUMBER shl 2) + 2) := SEGMENT;
      end; (* with...do *)

    (* Restore H/W Interrupt Handler(s) *)
    for IRQ := 0 to 7 do
      with HARDWARE_HANDLER(IRQ) do
        begin
          memw($0000 : (($08 + IRQ) shl 2))     := OFFSET;
          memw($0000 : (($08 + IRQ) shl 2) + 2) := SEGMENT;
        end; (* with...do *)
    for IRQ := 8 to 15 do
      with HARDWARE_HANDLER(IRQ) do
        begin
          memw($0000 : (($68 + IRQ) shl 2))     := OFFSET;
          memw($0000 : (($68 + IRQ) shl 2) + 2) := SEGMENT;
        end; (* with...do *)

    (* Restore S/W Interrupt Handlers *)
    for INTERRUPT_NUMBER := 0 to $FF do
      if INTERRUPT_NUMBER in PROTECTED_INTERRUPTS then
        with INTERRUPT_VECTOR_HANDLER(INTERRUPT_NUMBER) do
          begin
            memw($0000 : (INTERRUPT_NUMBER shl 2))     := OFFSET;
            memw($0000 : (INTERRUPT_NUMBER shl 2) + 2) := SEGMENT;
          end; (* with...do *)

    (* Restore exit procedure? *)
    if ACTUAL_TASKING_CONFIGURRTION.STATISTICS <> NO_STATISTICS
      then
        exitproc := @REPORT_TASKING_STATISTICS
      else
        exitproc := SAVE_EXIT;
    asm sti end;
  end; (* TASKING_EXIT *)

procedure TASKING_START; far;
(*************************************************************************)
(*   This routine is linked into the exit procedure chain to ensure that all *)
(* interrupt vectors that this unit needs are hooked before initiating multi-*)
(* tasking.  The exit  procedure  chain  is  executed  no  matter  what  has *)
(* happened to cause the program to terminate, but only  normal  termination *)
(* initiates TASKING.                                                        *)
(*************************************************************************)

  const (****************************************************************)
        (*  Creates  a  convenient way  to  install  the  TASKING   hardware *)
        (* interrupt handlers.                                              *)
        (****************************************************************)
        HARDWARE_INTERCEPTORS : array(2..15) of pointer =
          (
           @IRQ_2_INTERCEPTOR, @IRQ_3_INTERCEPTOR, @IRQ_4_INTERCEPTOR,
           @IRQ_5_INTERCEPTOR, @IRQ_6_INTERCEPTDR, @IRQ_7_INTERCEPTOR,
           @IRQ_8_INTERCEPTOR, @IRQ_9_INTERCEPTOR, @IRQ_A_INTERCEPTDR,
           @IRQ_B_INTERCEPTOR, @IRQ_C_INTERCEPTOR, @IRQ_D_INTERCEPTOR,
           @IRQ_E_INTERCEPTOR, @IRQ_F_INTERCEPTOR
          );

  var   (****************************************************************)
        (*  Loop control variables used to save interrupt handlers protected *)
        (* by TASKING (both hardware and software).                         *)
        (****************************************************************)
        IRQ, INTERRUPT_NUMBER : integer;

        (****************************************************************)
        (*  80x86 registers user to make MS-DOS system calls.             *)
        (****************************************************************)
        REGS : registers;

        (****************************************************************)
        (*  Used to compute new value for the system clock timer.         *)
        (****************************************************************)
        TEMP_TIMER_0_VALUE : longint;

        (****************************************************************)
```

```
      (*  Used in computing ELAPSED_TIME.                                    *)
      (**********************************************************************)
      LCV, YEAR, MONTH, DAY, DAY_OF_WEEK, HOUR, MIN, SEC, SEC100 : word;

      (**********************************************************************)
      (*  Temporary INI file configuration.                                  *)
      (**********************************************************************)
      INI_FILE_CONFIGURATION : CONFIGURATION;

function PROCESS_INI_FILE(var CONFIG : CONFIGURATION) : boolean;
(**************************************************************************)
(*  Finds and reads the program INI file searching for a TASKING section, *)
(* if one is found then the TASKING parameters are taken from it.  The INI *)
(* file configuration can be over-ridden by parameters changed in the *)
(* program 'main'.                                                        *)
(**************************************************************************)

  const (*****************************************************************)
        (*   The following are used to parse the INI file lines of text.  *)
        (*****************************************************************)
        TASKING_LABEL                      = '(TASKING)';
        COMMENT_DELIMITER                  = ';';
        TRUE_FLAG                          = 'TRUE';
        FALSE_FLAG                         = 'FALSE';
        PRIORITY_SCHEDULING_POLICY_CONTROL = 'PRIORITYSCHEDULINGPOLICY';
        STATIC_FLAG                        = 'STATIC';
        ROTATING_FLAG                      = 'ROTATING';
        PRIORITY_INHERITANCE_CONTROL       = 'PRIORITYINHERITANCE';
        ENABLED_FLAG                       = 'ENABLED';
        DISABLED_FLAG                      = 'DISABLED';
        STATISTICS_CONTROL                 = 'STATISTICS';
        NO_STATISTICS_FLAG                 = 'NOSTATISTICS';
        TASK_STATISTICS_FLAG               = 'TASKSTATISTICS';
        MSDOS_SERVICES_STATISTICS_FLAG     = 'MSDOSSERVICESSTATISTICS';
        HARDWARE_INTERRUPT_STATISTICS_FLAG = 'HARDWAREINTERRUPTSTATISTICS';
        ALL_STATISTICS_FLAG                = 'ALLSTATISTICS';
        TASKING_MODEL_CONTROL              = 'TASKINGMODEL';
        COOPERATIVE_FLAG                   = 'COOPERATIVE';
        PREEMPTIVE_FLAG                    = 'PREEMPTIVE';
        TARGET_TIMESLICE_CONTROL           = 'TARGETTIMESLICE';

        (*****************************************************************)
        (*  State variable that defines the section of the INI file that *)
        (* is being parsed.                                               *)
        (*****************************************************************)
        PARSING :
          (
           UNDEFINED,
           TASKING_SECTION
          ) = UNDEFINED;

  var   (*****************************************************************)
        (*   INI file parsing variables.                                  *)
        (*****************************************************************)
        INI_FILENAME : string[sizeof(dirstr)+sizeof(namestr)+sizeof(extstr)];
        LINE         : string;
        INI_FILE     : text;
        POSITION     : integer;

  function UP_STRING(S : string) : string;
  (************************************************************************)
  (*   This routine returns a string corresponding to the upper case value *)
  (* of the string passed to it.                                          *)
  (************************************************************************)

    var LCV : integer;

    begin (* UP_STRING *)
      UP_STRING[0] := S[0];
      for LCV := 1 to length(S) do
        UP_STRING[LCV] := upcase(S[LCV]);
    end; (* UP_STRING *)

  procedure BOOLEAN_VALUE(CONTROL : string; var PARAMETER : boolean);
  (************************************************************************)
  (*   This procedure examines the global LINE searching for CONTROL, if *)
  (* it is found, it's boolean value is set based on the rest of the LINE. *)
  (************************************************************************)

    begin (* BOOLEAN_VALUE *)
      if UP_STRING(copy(LINE, 1, POSITION - 1)) = CONTROL then
        begin
          delete(LINE, 1, POSITION);
          if (UP_STRING(copy(LINE, 1, length(TRUE_FLAG))) = TRUE_FLAG) or
             (UP_STRING(copy(LINE, 1, length(ENABLED_FLAG))) = ENABLED_FLAG)
            then
              PARAMETER := true
            else
              if (UP_STRING(copy(LINE, 1, length(FALSE_FLAG))) = FALSE_FLAG)
                 or
                 (UP_STRING(copy(LINE, 1, length(DISABLED_FLAG))) =
                  DISABLED_FLAG) then
                PARAMETER := false;
        end; (* if...then *)
    end; (* BOOLEAN_VALUE *)

  procedure INTEGER_VALUE(CONTROL : string; var PARAMETER : longint);
  (************************************************************************)
  (*   This procedure examines the global LINE searching for CONTROL, if *)
  (* it is found, it's real value is set based on the rest of the LINE.   *)
  (************************************************************************)

    var (****************************************************************)
        (*   Used to convert text to integers.                          *)
        (****************************************************************)
        VALUE : longint; CODE : integer;

    begin (* REAL_VALUE *)
      if UP_STRING(copy(LINE, 1, POSITION - 1)) = CONTROL then
        begin
          delete(LINE, 1, POSITION);
          POSITION := 1;
          while LINE[POSITION] in ['0'..'9'] do inc(POSITION);
          val(copy(LINE, 1, POSITION - 1), VALUE, CODE);
          if CODE = 0 then PARAMETER := VALUE;
        end; (* if...then *)
    end; (* REAL_VALUE *)

  procedure POLICY_VALUE
  (************************************************************************)
    (CONTROL : string; var PARAMETER : PRIORITY_SCHEDULING_POLICIES);
  (*----------------------------------------------------------------------*)
  (*   This procedure examines the global LINE searching for CONTROL, if *)
  (* it is found then it's string value is examined to see if it matches *)
  (* one of the valid priority modes.                                     *)
  (************************************************************************)

    begin (* POLICY_VALUE *)
      if UP_STRING(copy(LINE, 1, POSITION - 1)) = CONTROL then
        begin
          delete(LINE, 1, POSITION);
          while LINE[length(LINE)] = ' ' do delete(LINE, length(LINE), 1);
          POSITION := 1;
          if UP_STRING(LINE) = STATIC_FLAG
            then
              PARAMETER := STATIC_PRIORITIES
          else if UP_STRING(LINE) = ROTATING_FLAG
            then
              PARAMETER := ROTATING_PRIORITIES;
        end; (* if...then *)
    end; (* POLICY_VALUE *)

  procedure STATISTICS_VALUE
  (************************************************************************)
    (CONTROL : string; var PARAMETER : STATISTIC_OPTIONS);
  (*----------------------------------------------------------------------*)
  (*   This procedure examines the global LINE searching for CONTROL, if *)
  (* it is found then it's string value is examined to see if it matches *)
  (* one of the valid statistic gathering modes.                          *)
  (************************************************************************)

    begin (* STATISTICS_VALUE *)
      if UP_STRING(copy(LINE, 1, POSITION - 1)) = CONTROL then
        begin
          delete(LINE, 1, POSITION);
          while LINE[length(LINE)] = ' ' do delete(LINE, length(LINE), 1);
          POSITION := 1;
          if UP_STRING(LINE) = NO_STATISTICS_FLAG
            then
              PARAMETER := NO_STATISTICS
          else if UP_STRING(LINE) = TASK_STATISTICS_FLAG
            then
              PARAMETER := TASK_STATISTICS
          else if UP_STRING(LINE) = MSDOS_SERVICES_STATISTICS_FLAG
            then
              PARAMETER := MSDOS_SERVICES_STATISTICS
          else if UP_STRING(LINE) = HARDWARE_INTERRUPT_STATISTICS_FLAG
            then
              PARAMETER := HARDWARE_INTERRUPT_STATISTICS
          else if UP_STRING(LINE) = ALL_STATISTICS_FLAG
            then
              PARAMETER := ALL_STATISTICS;
        end; (* if...then *)
    end; (* STATISTICS_VALUE *)

  procedure MODEL_VALUE(CONTROL : string; var PARAMETER : TASKING_MODELS);
  (************************************************************************)
  (*   This procedure examines the global LINE searching for CONTROL, if *)
  (* it is found then it's string value is examined to see if it matches *)
  (* one of the valid tasking models.                                     *)
  (************************************************************************)

    begin (* MODEL_VALUE *)
      if UP_STRING(copy(LINE, 1, POSITION - 1)) = CONTROL then
        begin
          delete(LINE, 1, POSITION);
          while LINE[length(LINE)] = ' ' do delete(LINE, length(LINE), 1);
          POSITION := 1;
          if UP_STRING(LINE) = COOPERATIVE_FLAG
            then
              PARAMETER := COOPERATIVE
          else if UP_STRING(LINE) = PREEMPTIVE_FLAG
            then
              PARAMETER := PREEMPTIVE;
        end; (* if...then *)
    end; (* MODEL_VALUE *)

  begin (* PROCESS_INI_FILE *)
    PROCESS_INI_FILE := false;
    INI_FILENAME := paramstr(0);
    delete(INI_FILENAME, length(INI_FILENAME) - 2, 3);
    INI_FILENAME := INI_FILENAME + 'INI';
    assign(INI_FILE, INI_FILENAME);
    {$I-} reset(INI_FILE); {$I+}
    if ioresult = 0 then
      begin
        PROCESS_INI_FILE := true;
        while not eof(INI_FILE) do
          begin
            readln(INI_FILE, LINE);
            POSITION := pos(COMMENT_DELIMITER, LINE);
            if POSITION <> 0 then
              delete(LINE, POSITION, length(LINE));
            if UP_STRING(copy(LINE, 1, length(TASKING_LABEL))) =
               TASKING_LABEL then
              begin
                PARSING := TASKING_SECTION;
                readln(INI_FILE, LINE);
                POSITION := pos(COMMENT_DELIMITER, LINE);
                if POSITION <> 0 then
                  delete(LINE, POSITION, length(LINE));
              end; (* if...then *)

            POSITION := pos('=', LINE);
            if (POSITION <> 0) and (PARSING = TASKING_SECTION) then
              with CONFIG do
                begin
                  POLICY_VALUE
                    (
                     PRIORITY_SCHEDULING_POLICY_CONTROL,
                     PRIORITY_SCHEDULING_POLICY
                    );
                  BOOLEAN_VALUE
                    (
                     PRIORITY_INHERITANCE_CONTROL,
                     PRIORITY_INHERITANCE_ENABLED
                    );
                  STATISTICS_VALUE
                    (
                     STATISTICS_CONTROL,
                     STATISTICS
                    );
                  MODEL_VALUE
                    (
                     TASKING_MODEL_CONTROL,
                     TASKING_MODEL
                    );
                  INTEGER_VALUE
                    (
                     TARGET_TIMESLICE_CONTROL,
                     TARGET_TIMESLICE
                    );
                end; (* Tasking Section *)
          end; (* while...do *)
        close(INI_FILE);
      end; (* if...then *)
  end; (* PROCESS_INI_FILE *)

begin (* TASKING_START *)
  if (exitcode = 0) and (erroraddr = nil)
    then
      begin
        INI_FILE_CONFIGURATION := TASKING_CONFIGURATION;
        if PROCESS_INI_FILE(INI_FILE_CONFIGURATION)
          then
            (* 'Take' INI file configuration parameters *)
            ACTUAL_TASKING_CONFIGURATION := INI_FILE_CONFIGURATION
          else
            (* 'Take' user configuration parameters *)
            ACTUAL_TASKING_CONFIGURATION := TASKING_CONFIGURATION;
        TASKING_CONFIGURATION := ACTUAL_TASKING_CONFIGURATION;

        (* Pass control to 'next' exit procedure *)
        exitproc  := @TASKING_EXIT;

        gettime(HOUR, MIN, SEC, SEC100);
        getdate(YEAR, MONTH, DAY, DAY_OF_WEEK);
        with TASKING_STATISTICS do
```

```
            begin
              START_TIME := 0.0;
              for LCV := 1 to MONTH-1 do
                START_TIME := START_TIME + DAYS_IN[LCV];
              START_TIME := (((START_TIME + DAY) * 24 +
                              HOUR) * 60 + MIN) * 60 + SEC + SEC100 / 100;
            end; (* with...do *)

          (* Save revectored interrupt handler *)
          getintvec(REVECTORED_INTERRUPT_NUMBER,
            OLD_REVECTORED_INTERRUPT.VECTOR);

          (* Chain into S/W Interrupt Handlers *)
          for INTERRUPT_NUMBER := 0 to $FF do
            if INTERRUPT_NUMBER in PROTECTED_INTERRUPTS then
              begin
                getintvec(INTERRUPT_NUMBER,
                  INTERRUPT_VECTOR_HANDLER[INTERRUPT_NUMBER].VECTOR);
                setintvec(INTERRUPT_NUMBER, @SW_INTERRUPT_INTERCEPTOR);
              end; (* if...then *)

          (* Chain into Keyboard handler *)
          getintvec(KEYBOARD_INTERRUPT_NUMBER,
                    HARDWARE_HANDLER[1].VECTOR);
          setintvec(KEYBOARD_INTERRUPT_NUMBER, @KEYBOARD_INTERCEPTOR);

          (* Chain into IRQ 2 through IRQ 7 handlers *)
          for IRQ := 2 to 7 do
            begin
              getintvec($08 + IRQ, HARDWARE_HANDLER[IRQ].VECTOR);
              setintvec($08 + IRQ, HARDWARE_INTERCEPTORS[IRQ]);
            end;
          (* Chain into IRQ 8 through IRQ 15 handlers *)
          for IRQ := 8 to 15 do
            begin
              getintvec($68 + IRQ, HARDWARE_HANDLER[IRQ].VECTOR);
              setintvec($68 + IRQ, HARDWARE_INTERCEPTORS[IRQ]);
            end;

          (* Enable re-shceduling when creating tasks *)
          ALLOW_RESCHEDULE_IN_CREATE := true;

          (* Chain into Clock handler *)
          asm cli end;
          with HARDWARE_HANDLER[0] do
            begin
              OFFSET  := memw[S0000: SYSTEM_TIMER_INTERRUPT_NUMBER shl 2];
              SEGMENT := memw[S0000:(SYSTEM_TIMER_INTERRUPT_NUMBER shl 2)+2];
            end; (* with...do *)
          CLOCK_VECTOR := HARDWARE_HANDLER[0].VECTOR;
          memw[$0000 : SYSTEM_TIMER_INTERRUPT_NUMBER shl 2]       :=
                ofs(CLOCK_INTERCEPTOR);
          memw[S0000 :(SYSTEM_TIMER_INTERRUPT_NUMBER shl 2) + 2] :=
                seg(CLOCK_INTERCEPTOR);

          (* Enforce target timeslice range restrictions (if necessary) *)
          with ACTUAL_TASKING_CONFIGURATION do
            if TARGET_TIMESLICE < MINIMUM_TIMESLICE
              then
                TARGET_TIMESLICE := MINIMUM_TIMESLICE
              else
                if TARGET_TIMESLICE > MAXIMUM_TIMESLICE then
                  TARGET_TIMESLICE := MAXIMUM_TIMESLICE;

          (* Compute new timer counter value *)
          TEMP_TIMER_0_VALUE := round(TIMER_CLOCK_FREQUENCY / 1e6 *
                                ACTUAL_TASKING_CONFIGURATION.TARGET_TIMESLICE);

          (* Verify that the timer will be at least as fast as TASKING needs *)
          if TEMP_TIMER_0_VALUE > ONE_MSEC_TIMER_0_VALUE
            then
              NEW_TIMER_0_VALUE := ONE_MSEC_TIMER_0_VALUE
            else
              NEW_TIMER_0_VALUE := TEMP_TIMER_0_VALUE;

          (* Establish system clock scaling factors *)
          TIMER_PERIOD := NEW_TIMER_0_VALUE /
                          TIMER_CLOCK_FREQUENCY (Sec) * 1e6;

          (* Reprogram the timer chip *)
          port[TIMER_CONTROL_PORT] := SYSTEM_TIMER_CONTROL_WORD;
          port[TIMER_0_DATA_PORT]  := lo(NEW_TIMER_0_VALUE);
          port[TIMER_0_DATA_PORT]  := hi(NEW_TIMER_0_VALUE);

          (* Talk about a 'goto'... *)
          asm jmp DISPATCH_TASK end;
        end (* if...then *)
      else
        exitproc  := SAVE_EXIT;
  end; (* TASKING_START *)

{***************************************************************************}
{*   This is the TASKING unit initialization, there is not much to do except *}
{* setup the system stack and  install  the  exit  procedure  and  interrupt *}
{* handlers.                                                                 *}
{***************************************************************************}
begin (* TASKING_INITIALIZATION *)
  TASKING_STATISTICS.TOTAL_AVAILABLE_HEAP := memavail;

  (* Clear the null task stack *)
  fillchar(NULL.STACK_PTR^, NULL.STACK_SIZE, byte(DEFAULT_STACK_VALUE));

  (* Initialize TASK_ID array *)
  for TASK_ID_SET := low(TASK_ID_SETS) to high(TASK_ID_SETS) do
    ALLOCATED_TASK_IDS[TASK_ID_SET] := [];

  (* Install exit procedure *)
  SAVE_EXIT := exitproc;
  exitproc  := @TASKING_START;

  heaperror := @TASKING_HEAP_ERROR_HARDLER;
end. (* TASKING_INITIALIZATION *)
```

# 9. Appendix B Genetic Algorithm Source Code

## 9.1 RTOS-APP.PAS

```
program REAL_TIME_APPLICATION;
{*******************************************************************}
{*  This program interprets an INI file and a CSV  (Comma separated text) *}
{* file to configure itself to implement a multitasking design.  After the *}
{* design has been executed a report is generated which indicates  how well *}
{* the design time restraints were met.                                   *}
{*******************************************************************}
{*                       Compiler Options (Ver. 7.0)                      *}
{*$A+                             Word Alignment                          *}
{*$B-                             Short Circuit Boolean Evaluation        *}
{*$D+                             Debug Code Generation ON (Sort of)      *}
{*                                 Requires /V option to TPC to activate  *}
{*$L+                             Local Debug symbols ON (Sort of)        *}
{*                                 Requires /V option to TPC to activate  *}
{*$F-                             Far calls only as needed always         *}
{*$G-                             Generic 80x86 code only                 *}
{*$I-                             I/O Checking OFF                        *}
{*$M $8000,50000,100000           Memory (Stack, Minheap, Maxheap)        *}
{*$N-                             Software Emulation of 80x87             *}
{*$E-                             No 80x87 run-time emulation             *}
{*$O-                             Overlays NOT allowed                    *}
{*$Q-                             Overflow Checking OFF                   *}
{*$R-                             Range Checking OFF                      *}
{*$S-                             Stack Checking OFF                      *}
{*$T+                             Force Typed '@' references              *}
{*$V-                             Var-string Checking OFF                 *}
{*$X+                             Enable Extended syntax                  *}
{*******************************************************************}

uses  TASKING;

const {*****************************************************************}
      {*  Limit on the number of tasks allowed, basically the internal data *}
      {* structure must fit within 64k, this limitation restricts the number *}
      {* of tasks.                                                         *}
      {*****************************************************************}
      MAX_TASKS = 50;

      {*****************************************************************}
      {*  The application will execute for a  length of time  equal  to  the *}
      {* longest task period times this factor.  This will ensure  that  the *}
      {* results gathered are statistically significant.                   *}
      {*****************************************************************}
      EXECUTION_DWELL_TIME_FACTOR = 100;

      {*****************************************************************}
      {*  The absolute execution of the application is limited so  that  the *}
      {* results can be gathered in a reasonable amount of time.           *}
      {*****************************************************************}
      MAXIMUM_DWELL_TIME = 5 * 60; {* Seconds *}

type  {*****************************************************************}
      {*  Intertask communication is accomplished via TASKING resources (e.g.*}
      {* events, semaphores, messages, condition variables),  the  resources *}
      {* can eitehr be recieved or signaled by the task.                   *}
      {*****************************************************************}
      RESOURCE_ACTIONS = (SIGNAL_RESOURCE, RECEIVE_RESOURCE);

      {*****************************************************************}
      {*  The TASKING types are used for all intertask communications,  this *}
      {* typemark defines those communication constructs.                  *}
      {*****************************************************************}
      RESOURCE_TYPES =
        (
         MESSAGE_RESOURCE,             SEMAPHORE_RESOURCE,
         BINARY_SEMAPHORE_RESOURCE,    EVENT_RESOURCE,
         CONDITION_VARIABLE_RESOURCE,  GENERIC_RESOURCE
        );

      {*****************************************************************}
      {*  The tasks are allowed to communicate with each  other,  this  data *}
      {* structure defines the manner in which that communication  can  take *}
      {* place.                                                            *}
      {*****************************************************************}
      COMMUNICATION_SPECIFICATION = record
        {***************************************************************}
        {*  Not all tasks use all (or any) of  the  intertask  communication *}
        {* resources, this element is used to determine which resources  are *}
        {* active for each task.                                         *}
        {***************************************************************}
        DEFINED : boolean;
        {***************************************************************}
        {*  The user defines the reource by a text string name, this is used *}
        {* to determine when the same resource is being used by more than  a *}
        {* single task.                                                  *}
        {***************************************************************}
        RESOURCE_NAME : string(25);
        {***************************************************************}
        {*  If a message is being used as the communication  mechanism  then *}
        {* the task ID of the task that is going to receive the  message  is *}
        {* needed, this element holds this information (if applicable).  *}
        {***************************************************************}
        RESOURCE_TASK_ID : TASK_IDS;
        {***************************************************************}
        {*  The most important specifcation parameter for  the  resource  is *}
        {* the absolute time from the beginning  of  the  period)  that  the *}
        {* intertask communication is to take plcae.                     *}
        {***************************************************************}
        RESOURCE_TIME : word;
        {***************************************************************}
        {*  The  actual  intertask  communication  resource  is  dynamically *}
        {* allocated by the task itself.                                 *}
        {***************************************************************}
        case RESOURCE_TYPE : RESOURCE_TYPES of
          MESSAGE_RESOURCE            : (MSG_PTR      : ^longint);
          SEMAPHORE_RESOURCE          : (SEM_PTR      : ^SEMAPHORE);
          BINARY_SEMAPHORE_RESOURCE   : (B_SEM_PTR    : ^BINARY_SEMAPHORE);
          EVENT_RESOURCE              : (EVENT_PTR    : ^EVENT);
          CONDITION_VARIABLE_RESOURCE : (C_VAR_PTR    : ^CONDITION_VARIABLE);
          GENERIC_RESOURCE            : (RESOURCE_PTR : pointer);
      end; {* COMMUNICATION_SPECIFICATION *}

type   {****************************************************************}
       {*   Depending on the timing requirements of the task,  the  task  is *}
       {* classified into one of the following categories.              *}
       {* Note: All of the following timing  descriptions  assume  that  the *}
       {*       workload time is less than (or equal to)  the  task  period. *}
       {*       The diagrams are *not*  to  scale,  only  relative  timing *}
       {*       information is intended to be conveyed.                  *}
       {****************************************************************}
       TASK_CATEGORIES = (
        {**************************************************************}
        {* Case #1 - Proactive task - deadline  occurs  during  the  task *}
        {*           execution, i.e., the task has a critical point in  its *}
        {*           execution that it must get to  before  the  'deadline' *}
        {*           amount of time has elapsed.                          *}
        {* Deadline |---------------|                                     *}
        {* Workload |--------------------------------|                    *}
        {* Period   |----------------------------------------------------|*}
        {* Time     |-------+-------+-------+-------+-------+-------+---| *}
        {**************************************************************}
        PROACTIVE,
        {**************************************************************}
        {* Case #2 - Reactive task - deadline occurs  after  the  task  has *}
        {*           finished its execution, i.e.  the  critical  event  is *}
        {*           outside the realm of this task.  It does not  actually *}
        {*           perform the critical operation but the timing of  that *}
        {*           operation is critical to the execution (actually  next *}
        {*           execution) of this task.                             *}
        {* Deadline |------------------------------------|                *}
        {* Workload |------------------------|                            *}
        {* Period   |----------------------------------------------------|*}
        {* Time     |-------+-------+-------+-------+-------+-------+---| *}
        {**************************************************************}
        REACTIVE,
        {**************************************************************}
        {* Case #3 - Passive task - no deadline.  The task merely  performs *}
        {*           some work but does not have  a  critical  deadline  to *}
        {*           meet (other than the implicit deadline of periodicity).*}
        {* Deadline |-----------------------------------------------------|*}
        {* Workload |------------------------|                            *}
        {* Period   |----------------------------------------------|      *}
        {* Time     |-------+-------+-------+-------+-------+-------+---| *}
        {**************************************************************}
        PASSIVE,
        {**************************************************************}
        {* Case #4 - Background task - no deadline, no  period.   The  task *}
        {*           performs continuous work, since it never  completes  a *}
        {*           cycle it *cannot* be late.  A task such as  this  will *}
        {*           consume resources and force the application to execute *}
        {*           at 100% CPU utilization.                             *}
        {* Deadline |-----------------------------------------------------|*}
        {* Workload |----------------------------------------------|      *}
        {* Period   |----------------------------------------------|      *}
        {* Time     |-------+-------+-------+-------+-------+-------+---| *}
        {**************************************************************}
        BACKGROUND);
       {****************************************************************}

{******************************************************************}
{*  All of the information about the task is maintained in this  data *}
{* structure.  The task design and intertask constructs are maintained *}
{* here as well as all performance monitoring parameters.           *}
{******************************************************************}
TASK_SPECIFICATION = record
  {****************************************************************}
  {*  Used to allow messages to be sent between the tasks.          *}
  {****************************************************************}
  TASK_ID : TASK_IDS;
  {****************************************************************}
  {*  Used to allow the task priorities to be reported at termination. *}
  {****************************************************************}
  TASK_PRIORITY : USER_PRIORITIES;
  {****************************************************************}
  {*  Used to determine which task ID the user wants the message  sent *}
  {* to (if messages are being used).  Also used when the report  file *}
  {* is generated to make it more user friendly (i.e.  the  report  is *}
  {* made in terms of the users task names).                       *}
  {****************************************************************}
  TASK_NAME : string(25);
  {****************************************************************}
  {*  Based on the characteristics of the task it  is  categorized  so *}
  {* that the correct statistics are computed for the task.   Not  all *}
  {* statistics are valid for all task characteristics.             *}
  {****************************************************************}
  TASK_CATEGORY : TASK_CATEGORIES;
  {****************************************************************}
  {*  Amount of time between successive executions of the task.     *}
  {****************************************************************}
  PERIOD : word;
  {****************************************************************}
  {*  Amount of time from the beginning of the tasks execution to  its *}
  {* 'critical time'.                                               *}
  {****************************************************************}
  DEADLINE : word;
  {****************************************************************}
  {*  How 'important' the deadline is: 10 = Hard Deadline (meeting the *}
  {* the deadline is critical), 1 = Soft (meeting the  deadline  would *}
  {* be 'nice'), linear scale for all other values.                 *}
  {****************************************************************}
  DEADLINE_HARDNESS : 1..10;
  {****************************************************************}
  {*  Amount of time that the task is busy working (i.e.,  the  amount *}
  {* of time that the task executes).                               *}
  {****************************************************************}
  WORKLOAD : word;
  {****************************************************************}
  {*  Each task is allowed to have up to two  intertask  communication *}
  {* resources (for obvious reasons, each must include both  types  of *}
  {* communication actions).                                        *}
  {****************************************************************}
  INTERTASK_COMMUNICATION :
    array (RESOURCE_ACTIONS, 1..2) of COMMUNICATION_SPECIFICATION;
  {****************************************************************}
  {*   As the various timing requirements of the tasks are  monitored, *}
  {* the application records how well the requirements are  met  using *}
  {* the following data structure.                                  *}
  {****************************************************************}
  PERIODIC_TIME_MONITOR, DEADLINE_TIME_MONITOR : record
    {**************************************************************}
    {*  Number of times that the event occurred at the correct time.  *}
    {**************************************************************}
    SUCCESSFUL : longint;
    {**************************************************************}
    {*  Number of times that the event occurred.                      *}
    {**************************************************************}
    TOTAL : longint;
  end; {* TIME_MONITOR *}
end; {* TASK_SPECIFICATION *}

var   {*****************************************************************}
      {*  Used for creating the supervisor task.                          *}
      {*****************************************************************}
      SUPERVISOR : TASK_IDS;
      TASK_ATTR : TASK_ATTRIBUTES;

      {*****************************************************************}
      {*  The application report file is generated if  the  program  command *}
      {* line is blank.  If anything is on the command line then *no* report *}
      {* report file is generated.  Whether the file is generated or not the *}
      {* result is returned via the MS-DOS errorlevel.                    *}
      {*****************************************************************}
      GENERATE_REPORT_FILE : boolean;
```

```
      (******************************************************************)
      (*  The main database for the application, all  tasks  are  completely *)
      (* specified by the information in this array.                      *)
      )******************************************************************)
      TASK : array[1..MAX_TASKS] of TASK_SPECIFICATION;

      (******************************************************************)
      )*  The number of tasks that the user has specified in the .CSV file.  *)
      (******************************************************************)
      NUMBER_OF_TASKS : integer;

      )******************************************************************)
      (*  The largest task period is used to determine the execution time of *)
      (* the application, this variable hold that value.                  *)
      (******************************************************************)
      MAXIMUM_MSEC_PERIOD : integer;

      (******************************************************************)
      (*  Name of this program .EXE file, used to determine the filenames of *)
      )* the .RPT, .INI and .CSV files.                                   *)
      (******************************************************************)
      PROGRAM_NAME : string;

procedure SUPERVISOR_TASK(TASK_ID : TASK_IDS; PRIORITY : USER_PRIORITIES); far;
(**********************************************************************)
(*  This task is *not* specified by the user (i.e. .CSV file).  Instead this *)
(* task is needed to bring the application to an  orderly  shutdown  and  to *)
(* produce the .RPT file.  In addition, this task sets the MS-DOS errorlevel *)
(* to the total application timeliness.                                 *)
(**********************************************************************)

  var (******************************************************************)
      (*  The total time that the application is  going  to  be  allowed  to *)
      (* execute.                                                         *)
      (******************************************************************)
      EXECUTION_TIME : TIME;

      (******************************************************************)
      (*  The number of seconds that the application is going to execute.  *)
      (******************************************************************)
      DWELL_SECONDS : integer;

      (******************************************************************)
      (*  The file handle for the application report.                     *)
      (******************************************************************)
      REPORT : text;

      )******************************************************************)
      )*  Loop  Control  Variable  for  accessing  all  tasks  in  the  task *)
      (* specification database.                                          *)
      (******************************************************************)
      TASK_LCV : integer;

      (******************************************************************)
      (*  The 'timeliness' of the application is computed for each of the *)
      (* following areas of execution.                                    *)
      (******************************************************************)
      PERIODICITY_RESULT, DEADLINE_RESULT, APPLICATION_RESULT : record
        TIMELINESS      : real;
        NUMBER_OF_TIMES : integer;
      end; (* TIMELINESS *)

  begin (* SUPERVISOR_TASK *)
    )* Calculate application dwell time *)
    DWELL_SECONDS := round(MAXIMUM_MSEC_PERIOD * EXECUTION_DWELL_TIME_FACTOR /
                           1000);
    if DWELL_SECONDS > MAXIMUM_DWELL_TIME then halt(1);

    (* Calculate a time delay to run the application for the dwell time *)
    with EXECUTION_TIME do
      begin
        DAYS          := 0;
        HOURS         := 0;
        MINUTES       := DWELL_SECONDS div 60;
        DWELL_SECONDS := DWELL_SECONDS mod 60;
        SECONDS       := DWELL_SECONDS;
        MILLISECONDS  := 0;
      end; (* with...do *)

    (* Wait for dwell time to elapse *)
    WAIT_FOR_DELAY)EXECUTION_TIME);

    (* Destroy all application tasks (makes results more accurate)... *)
    for TASK_LCV := 1 to NUMBER_OF_TASKS do
      DESTROY)TASK[TASK_LCV].TASK_ID);

    with PERIODICITY_RESULT do
      begin
        TIMELINESS      := 0.0;
        NUMBER_OF_TIMES := 0;
      end; )* with...do *)
    with DEADLINE_RESULT do
      begin
        TIMELINESS      := 0.0;
        NUMBER_OF_TIMES := 0;
      end; (* with...do *)
    with APPLICATION_RESULT do
      begin
        TIMELINESS      := 0.0;
        NUMBER_OF_TIMES := 0;
      end; (* with...do *)
    for TASK_LCV := 1 to NUMBER_OF_TASKS do
      with TASK[TASK_LCV] do
        begin
          with PERIODIC_TIME_MONITOR do
            begin
              if TOTAL <> 0 then
                PERIODICITY_RESULT.TIMELINESS := PERIODICITY_RESULT.TIMELINESS
                                                 +
                                                 SUCCESSFUL / TOTAL * 100;
              inc)PERIODICITY_RESULT.NUMBER_OF_TIMES);
            end; (* with...do *)
          case TASK_CATEGORY of
            PROACTIVE,
            REACTEVE :
              begin
                with DEADLINE_TIME_MONITOR do
                  begin
                    if TOTAL <> 0 then
                      DEADLINE_RESULT.TIMELINESS := DEADLINE_RESULT.TIMELINESS
                        +
                        SUCCESSFUL / TOTAL * 100 *
                        (1 - DEADLINE_HARDNESS / 10);
                    inc(DEADLINE_RESULT.NUMBER_OF_TIMES);
                  end; (* with...do *)
              end; (* Proactive ) Reactive Tasks *)
            PASSIVE    : begin (* Passive Task    *) end;
            BACKGROUND : begin (* Background Task *) end;
          end; (* case...of *)
        end; (* with...do *)
    if PERIODICITY_RESULT.NUMBER_OF_TIMES <> 0 then
      with APPLICATION_RESULT do
        begin
          TIMELINESS := PERIODICITY_RESULT.TIMELINESS + TIMELINESS;
          NUMBER_OF_TIMES := PERIODICITY_RESULT.NUMBER_OF_TIMES +
                             NUMBER_OF_TIMES;
        end; (* with...do *)
    if DEADLINE_RESULT.NUMBER_OF_TIMES <> 0 then
      with APPLICATION_RESULT do
        begin
          TIMELINESS := DEADLINE_RESULT.TIMELINESS + TIMELINESS;
          NUMBER_OF_TIMES := DEADLINE_RESULT.NUMBER_OF_TIMES +
                             NUMBER_OF_TIMES;
        end; (* with...do *)
    with APPLICATION_RESULT do
      if NUMBER_OF_TIMES <> 0 then
        TIMELINESS := TIMELINESS / NUMBER_OF_TIMES;

    if GENERATE_REPORT_FILE then
      begin
        (* Create Report File *)
        assign(REPORT, PROGRAM_NAME + '.RPT');
        (*$I-*) rewrite(REPORT); (*$I+ *)
        if ioresult <> 0 then halt(1);

        writeln(REPORT, '  Individual Task Timeliness');
        for TASK_LCV := 1 to NUMBER_OF_TASKS do
          with TASK[TASK_LCV] do
            begin
              writeln(REPORT, '    Task Name: ', TASK_NAME,
                              ', Priority = ', TASK_PRIORITY);
              with PERIODIC_TIME_MONITOR do
                begin
                  write(REPORT, '       Periodicity: (', PERIOD, ' mSec)',
                        ' = ', SUCCESSFUL, '/', TOTAL);
                  if TOTAL <> 0 then
                    writeln(REPORT,  ' (', SUCCESSFUL / TOTAL *
                                          100 : 2 : 2, '%)');
                end; (* with...do *)
              case TASK_CATEGORY of
                PROACTIVE,
                REACTIVE :
                  with DEADLINE_TIME_MONITOR do
                    begin
                      write(REPORT, '       Deadline:    (', OEADLINE, ' mSec)',
                            ' = ', SUCCESSFUL, '/', TOTAL);
                      if TOTAL <> 0 then
                        writeln(REPORT,  ' (', SUCCESSFUL / TOTAL *
                                              100 : 2 : 2, '%)');
                    end; (* with...do *)
                PASSIVE    : begin (* Passive Task    *) end;
                BACKGROUND : begin (* Background Task *) end;
              end; (* case...of *)
              writeln(REPORT);
            end; (* with...do *)
        writeln(REPORT);
        writeln(REPORT, '  Average Task Timeliness');
        with PERIODICITY_RESULT do
          if NUMBER_OF_TIMES <> 0 then
            writeln(REPORT, '      Periodicity: ', TIMELINESS /
                                                   NUMBER_OF_TIMES : 2 : 2,
                                                   '%');
        with DEADLINE_RESULT do
         if NUMBER_OF_TIMES <> 0 then
          writeln(REPORT, '      Deadline:    ', TIMELINESS /
                                                 NUMBER_OF_TIMES : 2   2, '%');
        writeln(REPORT);
        writeln(REPORT, '  Application Timeliness = ',
                        APPLICATION_RESULT.TIMELINESS : 2   2, '%');
        close(REPORT);
      end; )* if...then *)

    (* shut down the application in order to provide exit code... *)
    halt)round(APPLICATION_RESULT.TIMELINESS));
  end; (* SUPERVISOR_TASK *)

procedure GENERAL_TASK(TASK_ID : TASK_IDS; PRIORITY : USER_PRIORITIES); far;
(**********************************************************************)
(*  This task is used to emulate all user application tasks.  It  does  this *)
(* by assuming the characteristics of the user task specification.      *)
)**********************************************************************)

  var   (******************************************************************)
        (*  These are the absolute delays, they are used in  evaluating  how *)
        (* well the task performed.                                       *)
        (******************************************************************)
        PERIODIC_DELAY, DEADLINE_DELAY : TIME;

        (******************************************************************)
        (*  The periodic execution of the task is  handled  by  TASKING,  an *)
        (* event is used to accomplish this.                              *)
        (******************************************************************)
        PERIODIC_EVENT : EVENT;

        )******************************************************************)
        (*  'Stop watch' variables for evaluating performance.            *)
        (******************************************************************)
        START_TIME, STOP_TIME : longint;

        (******************************************************************)
        (*  Database of the  tasks  critical  events  (for  *each*  periodic *)
        (* execution).                                                    *)
        (******************************************************************)
        CRITICAL : array [1..6] of record
          (****************************************************************)
          (*  The relative delay for the specific event (from  the  previous *)
          (* event).                                                      *)
          (****************************************************************)
          DELAY : TIME;
          (****************************************************************)
          (*  The actual critical event classification.                   *)
          (****************************************************************)
          MILESTONE : (
            NOT_APPLICABLE,
            DEADLINE_EXPIRES,
            WORKLOAD_EXPIRES,
            WAIT_FOR_RESOURCE_1,
            SIGNAL_RESOURCE_1,
            WAIT_FOR_RESOURCE_2,
            SIGNAL_RESOURCE_2);
        end; (* CRITICAL *)

        (******************************************************************)
        (*  Loop Control Variables used to create the critical  event  list. *)
        (******************************************************************)
        TIME_LCV, INSERT : integer;

  procedure FIND_OR_CREATE_RESOURCE(ACTION : RESOURCE_ACTIONS;
                                    NUMBER : integer);
  (********************************************************************)
  (*  This routine searches the tasks' resources for the resource specified. *)
  (* If it is found, the tasks are made to point to the same resouce, if  it *)
  (* is not found, a new resource is created.                           *)
  (********************************************************************)

    var (******************************************************************)
        (*  Indicates the existance of this critical event within the events *)
        (* of the application (recall that resources can be shared  by  more *)
        (* than one task).                                                *)
        (******************************************************************)
        FOUND : boolean;

        (******************************************************************)
        (*  Loop Control Variable to search all existing task specifications.*)
        (******************************************************************)
        TASK_LCV : integer;
```

```
begin (* FIND_OR_CREATE_RESOURCE *)
  FOUND := false;

  (* Check for existing resource *)
  for TASK_LCV := 1 to MAX_TASKS do
    begin
      if not ((ACTION = RECEIVE_RESOURCE) and (NUMBER = 1)) then
        with TASK[TASK_LCV].INTERTASK_COMMUNICATION[RECEIVE_RESOURCE, 1] do
          if (RESOURCE_PTR <> nil)
            and
            (RESOURCE_NAME = TASK[TASK_ID].
                             INTERTASK_COMMUNICATION[ACTION, NUMBER].
                               RESOURCE_NAME) then
            begin
              TASK[TASK_ID].
                INTERTASK_COMMUNICATION[ACTION, NUMBER].RESOURCE_PTR :=
                  RESOURCE_PTR;
              FOUND := true;
            end; (* if...then *)
      if not ((ACTION = SIGNAL_RESOURCE) and (NUMBER = 1)) then
        with TASK[TASK_LCV].INTERTASK_COMMUNICATION[SIGNAL_RESOURCE, 1] do
          if (RESOURCE_PTR <> nil)
            and
            (RESOURCE_NAME = TASK[TASK_ID].
                             INTERTASK_COMMUNICATION[ACTION, NUMBER].
                               RESOURCE_NAME) then
            begin
              TASK[TASK_ID].
                INTERTASK_COMMUNICATION[ACTION, NUMBER].RESOURCE_PTR :=
                  RESOURCE_PTR;
              FOUND := true;
            end; (* if...then *)
      if not ((ACTION = RECEIVE_RESOURCE) and (NUMBER = 2)) then
        with TASK[TASK_LCV].INTERTASK_COMMUNICATION[RECEIVE_RESOURCE, 2] do
          if (RESOURCE_PTR <> nil)
            and
            (RESOURCE_NAME = TASK[TASK_ID].
                             INTERTASK_COMMUNICATION[ACTION, NUMBER].
                               RESOURCE_NAME) then
            begin
              TASK[TASK_ID].
                INTERTASK_COMMUNICATION[ACTION, NUMBER].RESOURCE_PTR :=
                  RESOURCE_PTR;
              FOUND := true;
            end; (* if...then *)
      if not ((ACTION = SIGNAL_RESOURCE) and (NUMBER = 2)) then
        with TASK[TASK_LCV].INTERTASK_COMMUNICATION[SIGNAL_RESOURCE, 2] do
          if (RESOURCE_PTR <> nil)
            and
            (RESOURCE_NAME = TASK[TASK_ID].
                             INTERTASK_COMMUNICATION[ACTION, NUMBER]
                               RESOURCE_NAME) then
            begin
              TASK[TASK_ID].
                INTERTASK_COMMUNICATION[ACTION, NUMBER].RESOURCE_PTR :=
                  RESOURCE_PTR;
              FOUND := true;
            end; (* if...then *)
    end; (* for...to...do *)

  if not FOUND then
    with TASK[TASK_ID].INTERTASK_COMMUNICATION[ACTION, NUMBER] do
      case RESOURCE_TYPE of
        MESSAGE_RESOURCE : getmem(pointer(MSG_PTR), sizeof(longint));
        SEMAPHORE_RESOURCE :
          begin
            getmem(pointer(SEM_PTR), sizeof(SEMAPHORE));
            SEM_PTR^ := 1;
          end; (* Semaphore *)
        BINARY_SEMAPHORE_RESOURCE
          begin
            getmem(pointer(B_SEM_PTR), sizeof(BINARY_SEMAPHORE));
            B_SEM_PTR^ := 1;
          end; (* Binary Semaphore *)
        EVENT_RESOURCE :
          begin
            getmem(pointer(EVENT_PTR), sizeof(EVENT));
            EVENT_PTR^ := SIGNALED;
          end; (* Event *)
        CONDITION_VARIABLE_RESOURCE
          getmem(pointer(C_VAR_PTR), sizeof(CONDITION_VARIABLE));
      end; (* case...of *)
end; (* FIND_OR_CREATE_RESOURCE *)

begin (* GENENAL_TASK *)
  with TASK[TASK_ID] do
    begin
      (* Initialize time sequences *)
      for TIME_LCV := 1 to 6 do
        with CRITICAL[TIME_LCV] do
          begin
            with DELAY do
              begin
                DAYS         := 0;
                HOURS        := 0;
                MINUTES      := 0;
                SECONDS      := 0;
                MILLISECONDS := 0;
              end; (* with...do *)
            MILESTONE := NOT_APPLICABLE;
          end; (* with...do *)

      (* Determine critial time sequence (absolute times only!) *)
      if (DEADLINE <= WORKLOAD) and (WORKLOAD < PERIOD) then
        begin
          TASK_CATEGORY := PROACTIVE;
          with CRITICAL[1] do
            begin
              DELAY.MILLISECONDS := DEADLINE;
              MILESTONE := DEADLINE_EXPIRES;
            end; (* with...do *)
          with CRITICAL[2] do
            begin
              DELAY.MILLISECONDS := WORKLOAD;
              MILESTONE := WORKLOAD_EXPIRES;
            end; (* with...do *)
        end (* if...then *)
      else if (WORKLOAD < DEADLINE) and (DEADLINE <= PERIOD) then
        begin
          TASK_CATEGORY := REACTIVE;
          with CRITICAL[1] do
            begin
              DELAY.MILLISECONDS := WORKLOAD;
              MILESTONE := WORKLOAD_EXPIRES;
            end; (* with...do *)
          with CRITICAL[2] do
            begin
              DELAY.MILLISECONDS := DEADLINE;
              MILESTONE := DEADLINE_EXPIRES;
            end; (* with...do *)
        end (* if...then *)
      else if (WORKLOAD < PERIOD) and (PERIOD <= DEADLINE) then
        begin
          TASK_CATEGORY := PASSIVE;
          with CRITICAL[1] do
            begin
              DELAY.MILLISECONDS := WORKLOAD;
              MILESTONE := WORKLOAD_EXPIRES;
            end; (* with...do *)
```

```
        end (* if...then *)
      else if (WORKLOAD = PERIOD) and (PERIOD <= DEADLINE) then
        begin
          TASK_CATEGORY := BACKGROUND;
          with CRITICAL[1] do
            begin
              DELAY.MILLISECONDS := WORKLOAD;
              MILESTONE := WORKLOAD_EXPIRES;
            end; (* with...do *)
        end (* if...then *)
      else
        halt(1);

      (* Add resources to critical milestone list *)
      with INTERTASK_COMMUNICATION[RECEIVE_RESOURCE, 1] do
        if DEFINED then
          begin
            FIND_OR_CREATE_RESOURCE(RECEIVE_RESOURCE, 1);
            TIME_LCV := 1;
            while (TIME_LCV < 6) and
                  (RESOURCE_TIME >= CRITICAL[TIME_LCV].DELAY.MILLISECONDS) do
              inc(TIME_LCV);

            for INSERT := 6 downto TIME_LCV+1 do
              CRITICAL[INSERT] := CRITICAL[INSERT-1];
            with CRITICAL[TIME_LCV] do
              begin
                DELAY.MILLISECONDS := RESOURCE_TIME;
                MILESTONE := WAIT_FOR_RESOURCE_1;
              end; (* with...do *)
          end; (* with...do *)
      with INTERTASK_COMMUNICATION[SIGNAL_RESOURCE, 1] do
        if DEFINED then
          begin
            FIND_OR_CREATE_RESOURCE(SIGNAL_RESOURCE, 1);
            TIME_LCV := 1;
            while (TIME_LCV < 6) and
                  (RESOURCE_TIME >= CRITICAL[TIME_LCV].DELAY.MILLISECONDS) do
              inc(TIME_LCV);

            for INSERT := 6 downto TIME_LCV+1 do
              CRITICAL[INSERT] := CRITICAL[INSERT-1];
            with CRITICAL[TIME_LCV] do
              begin
                DELAY.MILLISECONDS := RESOURCE_TIME;
                MILESTONE := SIGNAL_RESOURCE_1;
              end; (* with...do *)
          end; (* with...do *)
      with INTERTASK_COMMUNICATION[RECEIVE_RESOURCE, 2] do
        if DEFINED then
          begin
            FIND_OR_CREATE_RESOURCE(RECEIVE_RESOURCE, 2);
            TIME_LCV := 1;
            while (TIME_LCV < 6) and
                  (RESOURCE_TIME >= CRITICAL[TIME_LCV].DELAY.MILLISECONDS) do
              inc(TIME_LCV);

            for INSERT := 6 downto TIME_LCV+1 do
              CRITICAL[INSERT] := CRITICAL[INSERT-1];
            with CRITICAL[TIME_LCV] do
              begin
                DELAY.MILLISECONDS := RESOURCE_TIME;
                MILESTONE := WAIT_FOR_RESOURCE_2;
              end; (* with...do *)
          end; (* with...do *)
      with INTERTASK_COMMUNICATION[SIGNAL_RESOURCE, 2] do
        if DEFINED then
          begin
            FIND_OR_CREATE_RESOURCE(SIGNAL_RESOURCE, 2);
            TIME_LCV := 1;
            while (TIME_LCV < 6) and
                  (RESOURCE_TIME >= CRITICAL[TIME_LCV].DELAY.MILLISECONDS) do
              inc(TIME_LCV);

            for INSERT := 6 downto TIME_LCV+1 do
              CRITICAL[INSERT] := CRITICAL[INSERT-1];
            with CRITICAL[TIME_LCV] do
              begin
                DELAY.MILLISECONDS := RESOURCE_TIME;
                MILESTONE := SIGNAL_RESOURCE_2;
              end; (* with...do *)
          end; (* with...do *)

      (* Setup absolute deadline timer *)
      with DEADLINE_DELAY do
        begin
          DAYS         := 0;
          HOURS        := 0;
          MINUTES      := 0;
          SECONDS      := 0;
          MILLISECONDS := DEADLINE;
        end; (* with...do *)

      (* Setup Periodic Event *)
      PERIODIC_EVENT := UNSIGNALED;
      with PERIODIC_DELAY do
        begin
          DAYS         := 0;
          HOURS        := 0;
          MINUTES      := 0;
          SECONDS      := 0;
          MILLISECONDS := PERIOD;
        end; (* with...do *)

      (* Setup relative delays *)
      for TIME_LCV := 6 downto 2 do
        CRITICAL[TIME_LCV].DELAY.MILLISECONDS :=
              CRITICAL[TIME_LCV].DELAY.MILLISECONDS -
              CRITICAL[TIME_LCV-1].DELAY.MILLISECONDS;

      (* Start emulating the task behavior *)
      START_PERIODIC_EVENT(PERIODIC_EVENT, PERIODIC_DELAY);
      repeat
        (* Perform and evaluate delays (as appropriate) *)
        START_TIME := GET_MILLISECOND_TICKS;
        for TIME_LCV := 1 to 6 do
          with CRITICAL[TIME_LCV] do
            begin
              case MILESTONE of
                NOT_APPLICABLE      : (* Do nothing *);
                WORKLOAD_EXPIRES    : PREEMPTABLE_DELAY(DELAY);
                DEADLINE_EXPIRES    :
                  begin
                    PREEMPTABLE_DELAY(DELAY);
                    STOP_TIME := GET_MILLISECOND_TICKS;
                    if (STOP_TIME - START_TIME) <=
                       DEADLINE_DELAY.MILLISECONDS then
                      inc(DEADLINE_TIME_MONITOR.SUCCESSFUL);
                    inc(DEADLINE_TIME_MONITOR.TOTAL);
                  end; (* if...then *)
                WAIT_FOR_RESOURCE_1 :
                  with TASK[TASK_ID].
                       INTERTASK_COMMUNICATION[RECEIVE_RESOURCE, 1] do
                    case RESOURCE_TYPE of
                      MESSAGE_RESOURCE :
                        WAIT_AND_RECEIVE_MESSAGE(pointer(MSG_PTR));
                      SEMAPHORE_RESOURCE :
                        WAIT_ON_SEMAPHORE(SEM_PTR^);
                      BINARY_SEMAPHORE_RESOURCE :
```

```
                WAIT_ON_BINARY_SEMAPHORE(B_SEM_PTR^);
              EVENT_RESOURCE :
                WAIT_ON_EVENT(EVENT_PTR^);
              CONDITION_VARIABLE_RESOURCE :
                WAIT_ON_CONDITION_VARIABLE(C_VAR_PTR^);
            end; (* case...of *)
        SIGNAL_RESOURCE_1   :
          with TASK[TASK_ID].
               INTERTASK_COMMUNICATION[SIGNAL_RESOURCE, 1] do
            case RESOURCE_TYPE of
              MESSAGE_RESOURCE :
                SEND_MESSAGE_TO(RESOURCE_TASK_ID, pointer(MSG_PTR));
              SEMAPHORE_RESOURCE :
                SIGNAL_SEMAPHORE(SEM_PTR^);
              BINARY_SEMAPHORE_RESOURCE :
                SIGNAL_BINARY_SEMAPHORE(B_SEM_PTR^);
              EVENT_RESOURCE :
                SIGNAL_EVENT(EVENT_PTR^);
              CONDITION_VARIABLE_RESOURCE :
                SIGNAL_CONDITION_VARIABLE(C_VAR_PTR^);
            end; (* case...of *)
        WAIT_FOR_RESOURCE_2 :
          with TASK[TASK_ID].
               INTERTASK_COMMUNICATION[RECEIVE_RESOURCE, 2] do
            case RESOURCE_TYPE of
              MESSAGE_RESOURCE :
                WAIT_AND_RECEIVE_MESSAGE(pointer(MSG_PTR));
              SEMAPHORE_RESOURCE :
                WAIT_ON_SEMAPHORE(SEM_PTR^);
              BINARY_SEMAPHORE_RESOURCE :
                WAIT_ON_BINARY_SEMAPHORE(B_SEM_PTR^);
              EVENT_RESOURCE :
                WAIT_ON_EVENT(EVENT_PTR^);
              CONDITION_VARIABLE_RESOURCE :
                WAIT_ON_CONDITION_VARIABLE(C_VAR_PTR^);
            end; (* case...of *)
        SIGNAL_RESOURCE_2   :
          with TASK[TASK_ID].
               INTERTASK_COMMUNICATION[SIGNAL_RESOURCE, 2] do
            case RESOURCE_TYPE of
              MESSAGE_RESOURCE :
                SEND_MESSAGE_TO(RESOURCE_TASK_ID, pointer(MSG_PTR));
              SEMAPHORE_RESOURCE :
                SIGNAL_SEMAPHORE(SEM_PTR^);
              BINARY_SEMAPHORE_RESOURCE :
                SIGNAL_BINARY_SEMAPHORE(B_SEM_PTR^);
              EVENT_RESOURCE :
                SIGNAL_EVENT(EVENT_PTR^);
              CONDITION_VARIABLE_RESOURCE :
                SIGNAL_CONDITION_VARIABLE(C_VAR_PTR^);
            end; (* case...of *)
        else (* Do Nothing *);
      end; (* case...of *)
    end; (* for...to...do *)

      WAIT_ON_EVENT(PERIODIC_EVENT);
      STOP_TIME := GET_MILLISECOND_TICKS;
      if (STOP_TIME - START_TIME) <= PERIODIC_DELAY.MILLISECONDS then
        inc(PERIODIC_TIME_MONITOR.SUCCESSFUL);
      inc(PERIODIC_TIME_MONITOR.TOTAL);
    until false;
  end; (* with...do *)
end; (* GENERAL_TASK *)

procedure PROCESS_APPLICATION_TASK_SPECIFICATION_FILE;
(*********************************************************************)
(*  This reoutine parses the .CSV file and creates  the  task  specification *)
(* database and creats the application tasks.                        *)
(*********************************************************************)

  type (*************************************************************)
       (*  Definition of the initial priority assignment  gene  (other  gene *)
       (* definitions come directly from the TASKING unit).          *)
       (*************************************************************)
       PRIORITY_ASSIGNMENT_ALGORITNME = (
         (***********************************************************)
         (*  All tasks are initially assigned the same priority.    *)
         (***********************************************************)
         UNIFORM_ASSIGNMENT,
         (***********************************************************)
         (*  All tasks are initially assigned random priorities.    *)
         (***********************************************************)
         RANDOM_ASSIGNMENT,
         (***********************************************************)
         (*  Task priorities are assigned based on execution  rates:  higher *)
         (* execution rate implies higher priority.                 *)
         (***********************************************************)
         RATE_MONOTONIC_ASSIGNMENT,
         (***********************************************************)
         (*  Task priorities are  assigned  based  on  execution  deadlines: *)
         (* earlier execution deadline implies higher priority.     *)
         (***********************************************************)
         DEADLINE_MONOTONIC_ASSIGNMENT,
         (***********************************************************)
         (*  Task priorities are assigned based  on  workload  level: higher *)
         (* workload implies higher priority.                       *)
         (***********************************************************)
         WORKLOAD_MONOTONIC_ASSIGNMENT);
       (*************************************************************)

  var  (*************************************************************)
       (*  Used to create the application tasks.                    *)
       (*************************************************************)
       TASK_ID : TASK_IDS;
       TASK_ATTR : TASK_ATTRIBUTES;

       (*************************************************************)
       (*  Used when determining how the task priorities should be assigned. *)
       (*************************************************************)
       INITIAL_PRIORITY_ASSIGNMENT : PRIORITY_ASSIGNMENT_ALGORITNMS;

       (*************************************************************)
       (*  Used to get the random number generator seed so that the  results *)
       (* are reporducible.                                         *)
       (*************************************************************)
       RANDOM_NUMBER_SEED : longint;

       (*************************************************************)
       (*  Used to assign initial priorities  (i.e.  Workload  and  Deadline *)
       (* Monotonic).                                               *)
       (*************************************************************)
       MAXIMUM_MSEC_DEADLINE : integer;
       MAXIMUM_MSEC_WORKLOAD : integer;

       (*************************************************************)
       (*  Used to hold the actual priorities of the application tasks.    *)
       (*************************************************************)
       PRIORITIES : array [1..MAX_TASKS] of USER_PRIORITIES;

       (*************************************************************)
       (*  Loop Control Variable for creating all application tasks.   *)
       (*************************************************************)
       LCV : integer;

  function UP_STRING(S : string) : string;
  (*******************************************************************)
  (*   This routine returns a string corresponding to the upper  case  value *)
  (* of the string passed to it.                                     *)
  (*******************************************************************)

  var LCV : integer;

  begin (* UP_STRING *)
    UP_STRING[0] := S[0];
    for LCV := 1 to length(S) do
      UP_STRING[LCV] := upcase(S[LCV]);
  end; (* UP_STRING *)

procedure PROCESS_INI_FILE;
(*********************************************************************)
(*  This routine parses the .INI file to determine how the task priorities *)
(* should be assigned as they are created.                           *)
(*********************************************************************)

  const (************************************************************)
        (*   The following are used to parse the INI file lines of text.   *)
        (************************************************************)
        APPLICATION_LABEL                    = '[APPLICATION]';
        COMMENT_DELIMITER                    = ';';
        INITIAL_PRIORITY_ASSIGNMENT_CONTROL = 'INITIALPRIORITYASSIGNMENT';
        RANDOM_NUMBER_SEED_CONTROL           = 'RANDOMNUMBERSEED';
        UNIFORM_ASSIGNMENT_FLAG              = 'UNIFORMASSIGNMENT';
        RANDOM_ASSIGNMENT_FLAG               = 'RANDOMASSIGNMENT';
        RATE_MONOTONIC_ASSIGNMENT_FLAG       = 'RATEMONOTONICASSIGNMENT';
        DEADLINE_MONOTONIC_ASSIGNMENT_FLAG   = 'DEADLINEMONOTONICASSIGNMENT';
        WORKLOAD_MONOTONIC_ASSIGNMENT_FLAG   = 'WORKLOADMONOTONICASSIGNMENT';

        (************************************************************)
        (*  State variable that defines the section of the INI  file  that *)
        (* is being parsed.                                         *)
        (************************************************************)
        PARSING :
          (
           UNDEFINED,
           APPLICATION_SECTION
          ) = UNDEFINED;

  var   (************************************************************)
        (*   INI file parsing variables.                            *)
        (************************************************************)
        LINE       : string;
        INI_FILE   : text;
        POSITION   : integer;

  procedure ASSIGNMENT_VALUE(CONTROL : string;
                             var PARAMETER : PRIORITY_ASSIGNMENT_ALGORITNME);
  (*******************************************************************)
  (*   This procedure examines the global LINE searching for  CONTROL,  if *)
  (* it is found then it's string value is examined to see if  it  matches *)
  (* one of the valid priority modes.                                *)
  (*******************************************************************)

    begin (* ASSIGNMENT_VALUE *)
      if UP_STRING(copy(LINE, 1, POSITION - 1)) = CONTROL then
        begin
          delete(LINE, 1, POSITION);
          while LINE[length(LINE)] = ' ' do delete(LINE, length(LINE), 1);
          POSITION := 1;
          if UP_STRING(LINE) = UNIFORM_ASSIGNMENT_FLAG
            then
              PARAMETER := UNIFORM_ASSIGNMENT
          else if UP_STRING(LINE) = RANDOM_ASSIGNMENT_FLAG
            then
              PARAMETER := RANDOM_ASSIGNMENT
          else if UP_STRING(LINE) = RATE_MONOTONIC_ASSIGNMENT_FLAG
            then
              PARAMETER := RATE_MONOTONIC_ASSIGNMENT
          else if UP_STRING(LINE) = DEADLINE_MONOTONIC_ASSIGNMENT_FLAG
            then
              PARAMETER := DEADLINE_MONOTONIC_ASSIGNMENT
          else if UP_STRING(LINE) = WORKLOAD_MONOTONIC_ASSIGNMENT_FLAG
            then
              PARAMETER := WORKLOAD_MONOTONIC_ASSIGNMENT;
        end; (* if...then *)
    end; (* ASSIGNMENT_VALUE *)

  procedure INTEGER_VALUE(CONTROL : string; var PARAMETER : longint);
  (*******************************************************************)
  (*   This procedure examines the global LINE searching for  CONTROL,  if *)
  (* it is found, it's integer value is set based on the rest of the LINE. *)
  (*******************************************************************)

     var (**********************************************************)
         (*   Used to convert text to integers.                    *)
         (**********************************************************)
         VALUE : longint; CODE  : integer;

    begin (* INTEGER_VALUE *)
      if UP_STRING(copy(LINE, 1, POSITION - 1)) = CONTROL then
        begin
          delete(LINE, 1, POSITION);
          POSITION := 1;
          while LINE[POSITION] in ['0'..'9'] do inc(POSITION);
          val(copy(LINE, 1, POSITION - 1), VALUE, CODE);
          if CODE = 0 then PARAMETER := VALUE;
        end; (* if...then *)
     end; (* INTEGER_VALUE *)

  begin (* PROCESS_INI_FILE *)
    assign(INI_FILE, PROGRAN_NAME + '.INI');
    (*$I-*) reset(INI_FILE); (*$I+*)
    if ioresult = 0 then
      begin
        while not eof(INI_FILE) do
          begin
            readln(INI_FILE, LINE);
            POSITION := pos(COMMENT_DELIMITER, LINE);
            if POSITION <> 0 then
              delete(LINE, POSITION, length(LINE));
            if UP_STRING(copy(LINE, 1, length(APPLICATION_LABEL))) =
               APPLICATION_LABEL then
              begin
                PARSING := APPLICATION_SECTION;
                readln(INI_FILE, LINE);
                POSITION := pos(CONMENT_DELIMITER, LINE);
                if POSITION <> 0 then
                  delete(LINE, POSITION, length(LINE));
              end; (* if...then *)

            POSITION := pos('=', LINE);
            if (POSITION <> 0) and (PARSING = APPLICATION_SECTION) then
              begin
                ASSIGNMENT_VALUE(INITIAL_PRIORITY_ASSIGNMENT_CONTROL,
                                 INITIAL_PRIORITY_ASSIGNMENT);
                INTEGER_VALUE(RANDOM_NUMBER_SEED_CONTROL,
                              RANDOM_NUMBER_SEED);
              end; (* if...then *)
          end; (* while...do *)
        close(INI_FILE);
      end; (* if...then *)
  end; (* PROCESS_INI_FILE *)

procedure PROCESS_CSV_FILE;
(*********************************************************************)
(*  This routine parses the .CSV file to determine the characteristics  of *)
(* the application tasks.                                            *)
(*********************************************************************(
```

```
type  (*****************************************************************)
      (*  The .CSV file organization is very strict.  The  tasks  *must* *)
      (* be specified with the following characteristics and they *must* *)
      (* be in the order listed below.                                   *)
      (*****************************************************************)
      FIELDS =
        ((*    Field Name                       Type     Range   Size (Chars)   *)
         TASK_NAME_FIELD,                   (* Text                25         *)
         PERIOD_FIELD,                      (* mSec    0-1000       5         *)
         DEADLINE_FIELD,                    (* mSec    0-1000       5         *)
         HARDNESS_FIELD,                    (* mSec    0-1000       5         *)
         WORKLOAD_FIELD,                    (* mSec    0-1000       5         *)
         RECEIVE_RESOURCE_1_FIELD,          (* Text                25         *)
         RECEIVE_RESOURCE_TYPE_1_FIELD,     (* Text                10         *)
         RECEIVE_RESOURCE_TIME_1_FIELD,     (* mSec    0-1000       5         *)
         SIGNAL_RESOURCE_1_FIELD,           (* Text                25         *)
         SIGNAL_RESOURCE_TYPE_1_FIELD,      (* Text                10         *)
         SIGNAL_RESOURCE_TIME_1_FIELD,      (* mSec    0-1000       5         *)
         RECEIVE_RESOURCE_2_FIELD,          (* Text                25         *)
         RECEIVE_RESOURCE_TYPE_2_FIELD,     (* Text                10         *)
         RECEIVE_RESOURCE_TIME_2_FIELD,     (* mSec    0-1000       5         *)
         SIGNAL_RESOURCE_2_FIELD,           (* Text                25         *)
         SIGNAL_RESOURCE_TYPE_2_FIELD,      (* Text                10         *)
         SIGNAL_RESOURCE_TIME_2_FIELD       (* mSec    0-1000       5         *)
        );

const (*****************************************************************)
      (*  CSV file format is 'comma seperated text'.                     *)
      (*****************************************************************)
      COMMA = ',';

var   (*****************************************************************)
      (*  CSV file parsing variables.                                    *)
      (*****************************************************************)
      LINE        : string;
      CSV_FILE    : text;
      POSITION    : integer;

      (*****************************************************************)
      (*  These are used as the task specification is  parsed  (i.e.  as *)
      (* the fields are stripped from the line of text).                 *)
      (*****************************************************************)
      PARAMETER : string;
      PARAM_LCV : FIELDS;

function INTERPRET_FIELD(THE_FIELD : FIELDS; var FIELD_STR : string;
                         TASK_NUMBER : integer) : boolean;
(*****************************************************************)
(*  This routine examines the contents of the field and determines if it *)
(* is valid.  Note that very little validation is actually performed.    *)
(*****************************************************************)

  var (*****************************************************************)
      (*  Temporary variable to hold function return value.              *)
      (*****************************************************************)
      VALID_RESULT : boolean;

      (*****************************************************************)
      (*  Used to determine the validity of numeric fields.              *)
      (*****************************************************************)
      VALUE, CODE : integer;

  begin (* INTERPRET_FIELD *)
    VALID_RESULT := true;

    with TASK[TASK_NUMBER] do
      case THE_FIELD of
        TASK_NAME_FIELD :
          TASK_NAME := FIELD_STR;
        PERIOD_FIELD :
          begin
            val(FIELD_STR, VALUE, CODE);
            if (CODE = 0) and (VALUE > 0)
              then
                PERIOD := VALUE
              else
                VALID_RESULT := false;
          end; (* Period *)
        DEADLINE_FIELD :
          begin
            val(FIELD_STR, VALUE, CODE);
            if (CODE = 0) and (VALUE >= 0)
              then
                DEADLINE := VALUE
              else
                VALID_RESULT := false;
          end; (* Deadline *)
        HARDNESS_FIELD :
          begin
            val(FIELD_STR, VALUE, CODE);
            if (CODE = 0) and (1 <= VALUE) and )VALUE <= 10)
              then
                DEADLINE_HARDNESS := VALUE
              else
                VALID_RESULT := false;
          end; (* Hardness *)
        WORKLOAD_FIELD :
          begin
            val(FIELD_STR, VALUE, CODE);
            if (CODE = 0) and (VALUE > 0)
              then
                WORKLOAD := VALUE
              else
                VALID_RESULT := false;
          end; (* Workload *)
        RECEIVE_RESOURCE_1_FIELD :
          with INTERTASK_COMMUNICATION[RECEIVE_RESOURCE, 1] do
            RESOURCE_NAME := FIELD_STR;
        RECEIVE_RESOURCE_TYPE_1_FIELD :
          with INTERTASK_COMMUNICATION[RECEIVE_RESOURCE, 1] do
            begin
              if FIELD_STR = 'EVENT' then
                RESOURCE_TYPE := EVENT_RESOURCE
              else if FIELD_STR = 'SEMAPHORE' then
                RESOURCE_TYPE := SEMAPHORE_RESOURCE
              else if FIELD_STR = 'MESSAGE' then
                RESOURCE_TYPE := MESSAGE_RESOURCE
              else if FIELD_STR = 'BSEMAPHORE' then
                RESOURCE_TYPE := BINARY_SEMAPHORE_RESOURCE
              else if FIELD_STR = 'COND_VAR' then
                RESOURCE_TYPE := CONDITION_VARIABLE_RESOURCE
              else if FIELD_STR <> 'NONE' then
                VALID_RESULT := false;
              DEFINED := VALID_RESULT and (FIELD_STR <> 'NONE');
            end; (* Resource Type *)
        RECEIVE_RESOURCE_TIME_1_FIELD :
          begin
            val(FIELD_STR, VALUE, CODE);
            if (CODE = 0) and (VALUE >= 0)
              then
                INTERTASK_COMMUNICATION[RECEIVE_RESOURCE, 1].
                    RESOURCE_TIME := VALUE
              else
                VALID_RESULT := false;
           end; (* Communication Time *)
        SIGNAL_RESOURCE_1_FIELD :
          with INTERTASK_COMMUNICATION[SIGNAL_RESOURCE, 1] do
            RESOURCE_NAME := FIELD_STR;
        SIGNAL_RESOURCE_TYPE_1_FIELD :
          with INTERTASK_COMMUNICATION[SIGNAL_RESOURCE, 1] do
            begin
              if FIELD_STR = 'EVENT' then
                RESOURCE_TYPE := EVENT_RESOURCE
              else if FIELD_STR = 'SEMAPHORE' then
                RESOURCE_TYPE := SEMAPHORE_RESOURCE
              else if FIELD_STR = 'MESSAGE' then
                RESOURCE_TYPE := MESSAGE_RESOURCE
              else if FIELD_STR = 'BSEMAPHORE' then
                RESOURCE_TYPE := BINARY_SEMAPHORE_RESOURCE
              else if FIELD_STR = 'COND_VAR' then
                RESOURCE_TYPE := CONDITION_VARIABLE_RESOURCE
              else if FIELD_STR <> 'NONE' then
                VALID_RESULT := false;
              DEFINED := VALID_RESULT and (FIELD_STR <> 'NONE');
            end; (* Resource Type *)
        SIGNAL_RESOURCE_TIME_1_FIELD :
          begin
            val(FIELD_STR, VALUE, CODE);
            if (CODE = 0) and (VALUE >= 0)
              then
                INTERTASK_COMMUNICATION[SIGNAL_RESOURCE, 1].
                    RESOURCE_TIME := VALUE
              else
                VALID_RESULT := false;
           end; (* Communication Time *)
        RECEIVE_RESOURCE_2_FIELD :
          with INTERTASK_COMMUNICATION[RECEIVE_RESOURCE, 2] do
            RESOURCE_NAME := FIELD_STR;
        RECEIVE_RESOURCE_TYPE_2_FIELD :
          with INTERTASK_COMMUNICATION[RECEIVE_RESOURCE, 2] do
            begin
              if FIELD_STR = 'EVENT' then
                RESOURCE_TYPE := EVENT_RESOURCE
              else if FIELD_STR = 'SEMAPHORE' then
                RESOURCE_TYPE := SEMAPHORE_RESOURCE
              else if FIELD_STR = 'MESSAGE' then
                RESOURCE_TYPE := MESSAGE_RESOURCE
              else if FIELD_STR = 'BSEMAPHORE' then
                RESOURCE_TYPE := BINARY_SEMAPHORE_RESOURCE
              else if FIELD_STR = 'COND_VAR' then
                RESOURCE_TYPE := CONDITION_VARIABLE_RESOURCE
              else if FIELD_STR <> 'NONE' then
                VALID_RESULT := false;
              DEFINED := VALID_RESULT and (FIELD_STR <> 'NONE');
            end; (* Resource Type *)
        RECEIVE_RESOURCE_TIME_2_FIELD :
          begin
            val(FIELD_STR, VALUE, CODE);
            if (CODE = 0) and (VALUE >= 0)
              then
                INTERTASK_COMMUNICATION[RECEIVE_RESOURCE, 2].
                    RESOURCE_TIME := VALUE
              else
                VALID_RESULT := false;
           end; (* Communication Time *)
        SIGNAL_RESOURCE_2_FIELD :
          with INTERTASK_COMMUNICATION[SIGNAL_RESOURCE, 2] do
            RESOURCE_NAME := FIELD_STR;
        SIGNAL_RESOURCE_TYPE_2_FIELD :
          with INTERTASK_COMMUNICATION[SIGNAL_RESOURCE, 2] do
            begin
              if FIELD_STR = 'EVENT' then
                RESOURCE_TYPE := EVENT_RESOURCE
              else if FIELD_STR = 'SEMAPHORE' then
                RESOURCE_TYPE := SEMAPHORE_RESOURCE
              else if FIELD_STR = 'MESSAGE' then
                RESOURCE_TYPE := MESSAGE_RESOURCE
              else if FIELD_STR = 'BSEMAPHORE' then
                RESOURCE_TYPE := BINARY_SEMAPHORE_RESOURCE
              else if FIELD_STR = 'COND_VAR' then
                RESOURCE_TYPE := CONDITION_VARIABLE_RESOURCE
              else if FIELD_STR <> 'NONE' then
                VALID_RESULT := false;
              DEFINED := VALID_RESULT and (FIELD_STR <> 'NONE');
            end; (* Resource Type *)
        SIGNAL_RESOURCE_TIME_2_FIELD :
          begin
            val(FIELD_STR, VALUE, CODE);
            if )CODE = 0) and (VALUE >= 0)
              then
                INTERTASK_COMMUNICATION[SIGNAL_RESOURCE, 2].
                    RESOURCE_TIME := VALUE
              else
                VALID_RESULT := false;
           end; (* Communication Time *)
        else VALID_RESULT := true;
      end; (* case...of *)

    INTERPRET_FIELD := VALID_RESULT;
  end; (* INTERPRET_FIELD *)

begin (* PROCESS_CSV_FILE *)
  (* Remove report file from previous run (if it exists) *)
  assign(CSV_FILE, PROGRAM_NAME + '.RPT');
  (*$I-*) reset(CSV_FILE); (*$I+*)
  if ioresult = 0 then
    begin
      close(CSV_FILE);
      erase(CSV_FILE);
    end; )* if...then *)

  assign(CSV_FILE, PROGRAM_NAME + '.CSV');
  (*$I-*) reset(CSV_FILE); (*$I+*)
  if ioresult = 0 then
    begin
      NUMBER_OF_TASKS := 0;
      while not eof(CSV_FILE) do
        begin
          readln(CSV_FILE, LINE);
          (* Make sure there is a comma at the end of the line *)
          LINE := LINE + COMMA;
          while LINE[1] = ' ' do delete(LINE, 1, 1);
          if UP_STRING(copy(LIME, 1, pos(COMMA, LINE)-1)) = 'TASK=' then
            begin
              delete(LINE, 1, pos(COMMA, LINE));
              (* Parse valid task identification line *)
              inc(NUMBER_OF_TASKS);
              PARAM_LCV := low(FIELDS);
              repeat
                PARAMETER := UP_STRING(copy(LINE, 1, pos(COMMA, LINE)-1));
                if not INTERPRET_FIELD(PARAM_LCV, PARAMETER,
                                       NUMBER_OF_TASKS) then
                  halt(1);
                PARAM_LCV := succ(PARAM_LCV);
                delete(LINE, 1, pos(COMMA, LINE));
              until (length(LINE) = 0) or (PARAM_LCV > high(FIELDS));
            end; (* if...then *)
        end; (* while...do *)
      close(CSV_FILE);
    end; (* if...then *)
end; (* PROCESS_CSV_FILE *)

begin (* PROCESS_APPLICATION_TASK_SPECIFICATION_FILE *)
  PROCESS_INI_FILE;

  (* Setup default task parameters *)
  with TASK_ATTR do
    begin
```

```
      STACK_WORDS_NEEDED := 4000;
      (* Use default error handlers *)
      ERROR_HANDLERS[TASK_ALREADY_ACTIVE]      := nil;
      ERROR_HANDLERS[INSUFFICIENT_RESOURCES]   := nil;
      ERROR_HANDLERS[TASK_IS_NOT_ACTIVE]       := nil;
      ERROR_HANDLERS[TASK_ALREADY_SUSPENDED]   := nil;
      ERROR_HANDLERS[ILLEGAL_TASK_ID]          := nil;
      ERROR_HANDLERS[ILLEGAL_OPERATION]        := nil;
    end; (* with...do *)

  PROCESS_CSV_FILE;

  (* Determine maximum execution period, deadline and workload *)
  MAXIMUM_MSEC_PERIOD   := 1;
  MAXIMUM_MSEC_DEADLINE := 1;
  MAXIMUM_MSEC_WORKLOAD := 1;
  for LCV := 1 to NUMBER_OF_TASKS do
    begin
      if TASK[LCV].PERIOD > MAXIMUM_MSEC_PERIOD then
        MAXIMUM_MSEC_PERIOD := TASK[LCV].PERIOD;
      if TASK[LCV].DEADLINE > MAXIMUM_MSEC_DEADLINE then
        MAXIMUM_MSEC_DEADLINE := TASK[LCV].DEADLINE;
      if TASK[LCV].WORKLOAD > MAXIMUM_MSEC_WORKLOAD then
        MAXIMUM_MSEC_WORKLOAD := TASK[LCV].WORKLOAD;
    end; (* for...to...do *)

  (* Assign initial task priorities *)
  case INITIAL_PRIORITY_ASSIGNMENT of
    UNIFORM_ASSIGNMENT :
      for LCV := 1 to NUMBER_OF_TASKS do
        PRIORITIES[LCV] := [high(USER_PRIORITIES] + low(USER_PRIORITIES))
                           div 2;
    RANDOM_ASSIGNMENT :
      begin
        randseed := RANDOM_NUMBER_SEED;
        for LCV := 1 to NUMBER_OF_TASKS do
          PRIORITIES[LCV] := random(high(USER_PRIORITIES) -
                                    low(USER_PRIORITIES() +
                             low(USER_PRIORITIES);
      end; (* Random Assignment *)
    RATE_MONOTONIC_ASSIGNMENT :
      for LCV := 1 to MUMBER_OF_TASKS do
        PRIORITIES[LCV] := low(USER_PRIORITIES) + high(USER_PRIORITIES) -
                           round(high(USER_PRIORITIES( /
                           MAXIMUM_MSEC_PERIOD * TASK[LCV].PERIOD);
    DEADLINE_MONOTONIC_ASSIGNMENT :
      for LCV := 1 to MUMBER_OF_TASKS do
        PRIORITIES[LCV] := low(USER_PRIORITIES) + high(USER_PRIORITIES]
                           round[high(USER_PRIORITIES) /
                           MAXIMUM_MSEC_DEADLINE * TASK[LCV].DEADLINE);
    WORKLOAD_MONOTONIC_ASSIGNMENT :
      for LCV := 1 to NUMBER_OF_TASKS do
        PRIORITIES[LCV] := round((high(USER_PRIORITIES) - 1) /
                           MAXIMUM_MSEC_WORKLOAD * TASK[LCV].WORKLOAD);
  end; (* case...of *(

  for LCV := 1 to NUMBER_OF_TASKS do
    begin
      TASK_ATTR.PRIORITY := PRIORITIES[LCV];
      CREATE[TASK_ID, TASK_ATTR, GENERAL_TASK];
      TASK[LCV].TASK_ID       := TASK_ID;
      TASK[LCV].TASK_PRIORITY := TASK_ATTR.PRIORITY;
    end; (* for...to...do *)
end; (* PROCESS_APPLICATION_TASK_SPECIFICATION_FILE *)

begin (* REAL_TIME_APPLICATION *)
  (* Determine whether report file is required or not *)
  GENERATE_REPORT_FILE := (paramcount = 0);

  (* Determine base filename for input/output files *]
  PROGHAN_NAME := paramstr(0);
  delete(PROGRAM_NAME, length(PROGRAM_NAME] - 3, 4);

  (* Establish task database and create application tasks *)
  PROCESS_APPLICATION_TASK_SPECIFICATION_FILE;

  (* Setup the supervisor task *]
  with TASK_ATTR do
    begin
      PRIORITY           := high(USER_PRIORITIES);
      STACK_WORDS_NEEDED := 2000;
      (* Use default error handlers *)
      ERROR_HANDLERS[TASK_ALREADY_ACTIVE]      := nil;
      ERROR_HANDLERS[INSUFFICIENT_RESOURCES]   := nil;
      ERROR_HANDLERS[TASK_IS_NOT_ACTIVE]       := nil;
      ERROR_HANDLERS[TASK_ALREADY_SUSPENDED]   := nil;
      ERROR_HANDLERS[ILLEGAL_TASK_ID]          := nil;
      ERROR_HANDLERS[ILLEGAL_OPERATION[        := nil;
    end; (* with...do *)
  CREATE(SUPERVISOR, TASK_ATTR, SUPERVISOR_TASK);
end. (* REAL_TIME_APPLICATION *)
```

# 9.2 GRAPH-GA.PAS

```
program GENETIC_ALGORITHM_THESIS(input, output);
(**********************************************************************)
(*   This program uses a Genetic Algorithm (GA) to search the problem  space *)
(* for the 'best' solution to the problem specified.                          *)
(**********************************************************************)
(*                       Compiler Options (Ver. 7.0)                         *)
(*$A+                                  Word Alignment                        *)
(*$B-                                  Short Circuit Boolean Evaluation      *)
(*$D+                                  Debug Code Generation ON (Sort of)    *)
(*                                      Requires /V option to BPC to activate *)
(*$L+                                  Local Debug symbols ON (Sort of)      *)
(*                                      Requires /V option to BPC to activate *)
(*$F-                                  Far calls only as needed              *(
(*$I-                                  I/O Checking OFF                      *)
(*$M $FFF0,$FFF0,$FFF0                 Memory (Stack, Minheap, Maxheap)      *)
(*$N-                                  Software Emulation of 80x87           *)
(*$O-                                  Overlays NOT allowed                  *)
(*$P-                                  Standard 'string' parameters          *)
(*$Q-                                  Overflow Checking OFF                 *)
(*$R-                                  Range Checking OFF                    *)
(*$S-                                  Stack Checking OFF                    *)
(*$V-                                  Var-string Checking OFF               *)
(*$T+                                  Force Typed '@' references            *)
(*$X+                                  Enable Extended syntax                *)
(**********************************************************************)

uses  dos, crt, graph, GRAPHICS, BMP_UTIL,
      (*$IFDEF MATH_GA *)
          MATH_GA;
      (*$ELSE *)
          RTOS_GA;
      (*$ENDIF *)

const (**********************************************************************)
      (*  Used to allow a single routine to be used as 'paint screen' or  as *)
      (* 'update as needed' drawing routine.                                 *)
      (**********************************************************************)
      FORCE_UPDATE = true;
      DO_NOT_FORCE_UPDATE = false;

      (**********************************************************************)
      (*   Command line controllable parameters.                             *)
      (**********************************************************************)
        (*   Control flag used to specify  that  the  reverse  video  colors *)
        (* should be used.                                                   *)
        (**********************************************************************)
        REVERSE_VIDEO : boolean = false;
        (**********************************************************************)
        (*   Control flag used to specify that the display should be 640x480,*)
        (* 800x600 or 1024x768, 1280x1024.                                   *)
        (**********************************************************************)
type    VIDEO_MODES  = (NO_DISPLAY_MODE, VGA, SVGA, VESA, SVESA);
const   DISPLAY_MODE : VIDEO_MODES = NO_DISPLAY_MODE;
      (**********************************************************************)

type  (**********************************************************************)
      (*   When in graphics mode the follow colors are used for drawing  the *)
      (* screen.                                                             *)
      (**********************************************************************)
      GRAPHIC_COLORS = record
        BACKGROUND_COLOR       : word;
        AXIS_COLOR             : word;
        GRID_COLOR             : word;
        AXIS_LABEL_COLOR       : word;
        GRAPH_COLOR            : word;
        STATUS_COLOR           : word;
        STATUS_HIGHLIGHT_COLOR : word;
        MAIN_TITLE_COLOR       : word;
        BORDER_COLOR           : word;
      end; (* GRAPHIC_COLORS *)

const (**********************************************************************)
      (*   INI file only configurable parameters.                            *)
      (**********************************************************************)
        (* Directory where IMAGE_XX.BMP files will be stored.                *)
        (**********************************************************************)
        IMAGE_DIRECTORY : dirstr = '';
        (**********************************************************************)
        (*   The graphic display uses the following colors in normal video.  *)
        (**********************************************************************)
        NORMAL   GRAPHIC_COLORS =
          (
           BACKGROUND_COLOR       : black;
           AXIS_COLOR             : lightgray;
           GRID_COLOR             : darkgray;
           AXIS_LABEL_COLOR       : white;
           GRAPH_COLOR            : cyan;
           STATUS_COLOR           : yellow;
           STATUS_HIGHLIGHT_COLOR : white;
           MAIN_TITLE_COLOR       : lightcyan;
           BORDER_COLOR           : green
          );
        (**********************************************************************)
        (*   The graphic display uses the following colors in reverse video. *)
        (**********************************************************************)
        REVERSE : GRAPHIC_COLORS =
          (
           BACKGROUND_COLOR       : white;
           AXIS_COLOR             : darkgray;
           GRID_COLOR             : lightgray;
           AXIS_LABEL_COLOR       : black;
           GRAPH_COLOR            : blue;
           STATUS_COLOR           : red;
           STATUS_HIGHLIGHT_COLOR : lightred;
           MAIN_TITLE_COLOR       : lightblue;
           BORDER_COLOR           : green
          );
      (**********************************************************************)

      (**********************************************************************)
      (*   The maximum number of graphs on the screen at any one time.       *)
      (**********************************************************************)
      MAX_NUMBER_OF_GRAPHS = 3;

      (**********************************************************************)
      (*   User controllable parameters (run-time controls).                 *)
      (**********************************************************************)
        (*   Controls how many graphs are being displayed.                   *)
        (**********************************************************************)
        NUMBER_OF_GRAPHS_TO_DISPLAY : 0..MAX_NUMBER_OF_GRAPHS = 0;
        (**********************************************************************)
        (*   Allows the user to pause the display.                           *)
        (**********************************************************************)
        USER_PAUSED : boolean = false;
        (**********************************************************************)
        (*   Allows for an orderly shutdown of the application.              *)
        (**********************************************************************)
        USER_QUIT : boolean = false;
      (**********************************************************************)

      (**********************************************************************)
      (*   The following constants are used to control sizes/offsets of  the *)
      (* graphic screen.                                                     *)
      (**********************************************************************)
      POINT_SIZE  = 3;   TICK_LENGTH = 4;   EDGE_OFFSET = 4;   CURSOR_SIZE = 5;

      (**********************************************************************)
      (*   MS-DOS exit codes (i.e. 'errorlevel').                            *)
      (**********************************************************************)
      INVALID_PARAMETER_ERROR_LEVEL        = 20;
      NON_SUPPORTED_VIDEO_MODE_ERROR_LEVEL = 30;

      (**********************************************************************)
      (*   General purpose special characters.                               *)
      (**********************************************************************)
      BELL  = chr($07);   (* Bell character          *)
      SPACE = chr($20);   (* Blank character         *)
      CR    = chr($0D);   (* Carriage Return         *)
      LF    = chr($0A);   (* Line Feed               *)
      BS    = chr($08);   (* Backspace               *)
      ESC   = chr($1B);   (* Escape                  *)
      NULL  = chr($00);   (* Preceedes extended keys *)

type  (**********************************************************************)
      (*   The axis cross point can be placed anywhere on the graph.         *)
      (**********************************************************************)
      AXIS_TYPES = (AUTO_AXIS, FORCE_LL, FORCE_LR, FORCE_UL, FORCE_UR);

      (**********************************************************************)
      (*   The graph can be positioned at various places on the screen.      *)
      (**********************************************************************)
      GRAPH_POSITIONS = (FULL_SCREEN, TOP_HALF, BOTTOM_HALF, TOP_THIRD,
                         MIDDLE_THIRD, BOTTOM_THIRD);

      (**********************************************************************)
      (*   Type to specify parameters for the two axis.                      *)
      (**********************************************************************)
      AXIS = (X_AXIS, Y_AXIS);

      (**********************************************************************)
      (*   Type to specify how the graph should be drawn.                    *)
      (**********************************************************************)
      GRAPH_TYPES = (BAR_GRAPH, LINE_GRAPH);

      (**********************************************************************)
      (*   Graph lines are a sequence of points defined by this type.        *)
      (**********************************************************************)
      GRAPH_PTR = ^POINTS;
      POINTS = array[0..POPULATION_SIZE] of record
        POSITION : array[AXIS] of real;
      end; (* LINES *)

      (**********************************************************************)
      (*   The user controllable attributes (as well as some fixed paramters)*)
      (* of the graph are kept in this record type.                          *)
      (**********************************************************************)
      GRAPH_SCALE_INFO = record
        (**********************************************************************)
        (*   Main title to the graph.                                        *)
        (**********************************************************************)
        TITLE : string[25];
        (**********************************************************************)
        (*   Line or bar graph indicator.                                    *)
        (**********************************************************************)
        GRAPH_TYPE : GRAPH_TYPES;
        (**********************************************************************)
        (*   FIRST and LAST refer to the indices of the VALUES array.        *)
        (**********************************************************************)
        FIRST, LAST : integer;
        (**********************************************************************)
        (*   For each axis the maximum, minimum and  number  of  tick  marks *)
        (* must be specified.  The range is calculated as the graph is setup.*)
        (**********************************************************************)
        HIGH_LABEL, LOW_LABEL,
        RANGE,       NUMBER_OF_TICK_MARKS : array[AXIS] of longint;
        (**********************************************************************)
        (*   The label for the axis units.                                   *)
        (**********************************************************************)
        LABEL_NAME : array[AXIS] of string[10];
        (**********************************************************************)
        (*   Used to turn on/off peak detection for each of the axis.        *)
        (**********************************************************************)
        PEAK_DETECTION : array[AXIS] of boolean;
        (**********************************************************************)
        (*   Used to turn on/off average detection for the Y-Axis.           *)
        (**********************************************************************)
        AVERAGE_DETECTION : boolean;
      end; (* GRAPH_SCALE_INFO *)

      (**********************************************************************)
      (*   Data stucture that keeps all of the information about a graph  so *)
      (* that the graph display functions can access the correct graph  when *)
      (* more than one is being displayed at a time.                         *)
      (**********************************************************************)
      GRAPH_INFO = record
        (**********************************************************************)
        (*   Scaling for X and Y axis (see above).                           *)
        (**********************************************************************)
        GRAPH_SCALES : GRAPH_SCALE_INFO;
        (**********************************************************************)
        (*   X and Y values for the previous peaks/averages  (allows  peaks/ *)
        (* averages to be removed).                                          *)
        (**********************************************************************)
        PREVIOUS_PEAK, PREVIOUS_PEAK_VALUE, PRSVIOUS_AVERAGE_VALUE : real;
        (**********************************************************************)
        (*   Offset of the graph into the graphic window on the screen.      *)
        (**********************************************************************)
        HORIZONTAL_OFFSET, VERTICAL_OFFSET : real;
        (**********************************************************************)
        (*   Sizes of the graphic window on  the  screen  (function  of  the *)
        (* video mode selected).                                             *)
        (**********************************************************************)
        HORIZONTAL_SIZE, VERTICAL_SIZE : real;
        (**********************************************************************)
        (*   Offset of the graph axis within the graphic window.             *)
        (**********************************************************************)
        X_AXIS_OFFSET, Y_AXIS_OFFSET : real;
        (**********************************************************************)
        (*   The actual data that is being graphed, and the array limits.    *)
        (**********************************************************************)
        VALUES : record
          FIRST, LAST : integer;
          DATA : array[0..POPULATION_SIZE] of real;
        end; (* VALUES *)
        (**********************************************************************)
        (*   Pointers to graphic points that are  being  graphed  after  the *)
        (* appropriate conversion from value to pixel has been done.         *)
        (**********************************************************************)
        GRAPH_POINTS, PREVIOUS_GRAPH_POINTS : GRAPH_PTR;
        (**********************************************************************)
        (*   The color of the line being graphed.                            *)
        (**********************************************************************)
        GRAPH_COLOR : word;
      end; (* GRAPH_INFO *)

var   (**********************************************************************)
      (*   Maximum X and Y values for the graphics mode selected.            *)
      (**********************************************************************)
      MAX_X, MAX_Y : integer;

      (**********************************************************************)
```

```
      (*   The data structure for the graphs being displayed.              *)
      (********************************************************************)
      PLOT : array [1..MAX_NUMBER_OF_GRAPHS] of GRAPH_INFO;

      (********************************************************************)
      (*   The graphic display uses the following colors variables for  both *)
      (* normal and reversed color modes.                                  *)
      (********************************************************************)
      SCREEN : GRAPHIC_COLORS;

type  (********************************************************************)
      (*   The user is allowed to change the scales being  used,  this  data *)
      (* structure holds the user choices.                                 *)
      (********************************************************************)
      USER_GRAPH_SCALES = record
        (******************************************************************)
        (*   Contains all of the graph axis information.                  *)
        (******************************************************************)
        SCALES : GRAPH_SCALE_INFO;
        (******************************************************************)
        (*   Used to turn on/off peak detection for each of the axis.     *)
        (******************************************************************)
        PEAK_DETECTION : array[AXIS] of boolean;
        (******************************************************************)
        (*   Used to turn on/off average detection for the Y-Axis.        *)
        (******************************************************************)
        AVERAGE_DETECTION : boolean;
      end; (* USER_GRAPH_SCALES *)
var   USER_GRAPH : array [1..MAX_NUMBER_OF_GRAPHS] of USER_GRAPH_SCALES;
      (********************************************************************)

      (********************************************************************)
      (*   The offset from the edge of the screen to the graph is a function *)
      (* of the graphics mode selected.                                    *)
      (********************************************************************)
      GRAPH_OFFSET : integer;    MESSAGE_REGION : integer;

      (********************************************************************)
      (*   Graphic Video Driver mode parameter.                            *)
      (********************************************************************)
      GRAPHICS_MODE : integer;

      (********************************************************************)
      (*   Address of previous exit procedure in the exit call chain.      *)
      (********************************************************************)
      SAVE_EXIT : pointer;

function INT_STR(I: longint): string;
(**************************************************************************)
(*  Convert any integer type to a string.                                 *)
(**************************************************************************)

  var S : string[11];

  begin (* INT_STR *)
    str(I, S);
    INT_STR := S;
  end; (* INT_STR *)

function REAL_STR(R: real; F1, F2 : integer): string;
(**************************************************************************)
(*  Convert real type to a string (same format as write(R : F1 : F2)).    *)
(**************************************************************************)

  var S : string[11[;

  begin (* REAL_STR *)
    str(R : F1 : F2, S);
    REAL_STR := S;
  end; (* REAL_STR *)

function X_POSITION(var X : real; GRAPH_NUMBER : integer) : real;
(**************************************************************************)
(*   Determines the X position within the graphics window  that  corresponds *)
(* to the X value passed in.                                              *)
(**************************************************************************)

  var (*******************************************************************)
      (*   Temporary graphic screen position.                             *)
      (*******************************************************************)
      POSITION : real;

  begin (* X_POSITION *)
    with PLOT[GRAPH_NUMBER[ do
      with GRAPH_SCALES do
        begin
          X := X - LOW_LABEL[X_AXIS] - 1;
          POSITION := HORIZONTAL_OFFSET + 1 +
                      (X * (HORIZONTAL_SIZE)) / RANGE[X_AXIS];
          if (HORIZONTAL_OFFSET <= POSITION) and
             (POSITION <= (HORIZONTAL_OFFSET + HORIZONTAL_SIZE))
            then
              X := X + LOW_LABEL[X_AXIS] - 1
            else
              X := LOW_LABEL[X_AXIS] + 1;
          X_POSITION := POSITIOH;
        end; (* with...do *)
  end; (* X_POSITION *)

function X_VALUE(POSITION : integer; GRAPH_NUMBER : integer) : real;
(**************************************************************************)
(*   Determines the value of the function at the graphic position passed in, *)
(* this function assumes the entire screen is the viewport.               *)
(**************************************************************************)

  begin (* X_VALUE *)
    with PLOT[GRAPH_NUMBER] do
      with GRAPH_SCALES do
        begin
          X_VALUE := LOW_LABEL[X_AXIS] +
                     (POSITION - HORIZONTAL_OFFSET) /
                     HORIZONTAL_SIZE * RANGE[X_AXIS[;
        end; (* with...do *)
  end; (* X_VALUE *)

function Y_POSITION(var Y : real; GRAPH_NUMBER : integer) : real;
(**************************************************************************)
(*   Determines the Y position within the graphics window  that  corresponds *)
(* to the Y value passed in.                                              *)
(**************************************************************************)

  var (*******************************************************************)
      (*   Temporary graphic screen position.                             *)
      (*******************************************************************)
      POSITION : real;

  begin (* Y_POSITION *)
    with PLOT[GRAPH_NUMBER] do
      with GRAPH_SCALES do
        begin
          Y := Y - LOW_LABEL[Y_AXIS[ - 1;
          POSITIOH := VERTICAL_OFFSET + 1 +
                      (VERTICAL_SIZE - (Y * VERTICAL_SIZE) / RANGE[Y_AXIS[);
          if POSITION > (VERTICAL_OFFSET + VERTICAL_SIZE - 1) then
            Y_POSITION := VERTICAL_OFFSET + VERTICAL_SIZE - 1
          else if POSITION < (VERTICAL_OFFSET + 1) then
            Y_POSITION := VERTICAL_OFFSET + 1
          else
            Y_POSITION := POSITION;
          Y := Y + LOW_LABEL[Y_AXIS[ + 1;
        end; (* with...do *)
  end; (* Y_POSITION *)

function Y_VALUE(POSITION : integer; GRAPH_NUMBER : integer) : real;
(**************************************************************************)
(*   Determines the value of the function at the graphic position passed in, *)
(* this function assumes the entire screen is the viewport.               *)
(**************************************************************************)

  begin (* Y_VALUE *)
    with PLOT[GRAPH_NUMBER] do
      with GRAPH_SCALES do
        begin
          Y_VALUE := LOW_LABEL[Y_AXIS] +
                     (VERTICAL_SIZE + VERTICAL_OFFSET - POSITION) /
                     VERTICAL_SIZE * RANGE[Y_AXIS];
        end; (* with...do *)
  end; (* Y_VALUE *)

procedure UPDATE_STATUS_MESSAGES(FORCE_UPDATE : boolean);
(**************************************************************************)
(*   Updates the status messages with the present value of the status flags. *)
(**************************************************************************)

  type  (*****************************************************************)
        (*   This type allows smaller strings to be manipulated for the date *)
        (* status since the date is a fixed length.                      *)
        (*****************************************************************)
        DATE_STRINGS = string[8];

        (*****************************************************************)
        (*   Type used to contain strings for the present PC clock time.  *)
        (*****************************************************************)
        TIME_STRINGS = string[12];

  const (*****************************************************************)
        (*   The previous status parameters are maintained  static  so  that *)
        (* the update is flicker-free.                                    *)
        (*****************************************************************)
        PREVIOUS_USER_PAUSED : boolean      = false;
        PREVIOUS_DATE        : DATE_STRINGS = ' ';
        DATE                 : DATE_STRINGS = '';
        PREVIOUS_INDIVIDUAL  : integer      = POPULATION_SIZE;
        PREVIOUS_GEHERATION  : integer      = -1;

        (*****************************************************************)
        (*   Values of the present and previously displayed PC clock time. *)
        (*****************************************************************)
        PREVIOUS_TIME : TIME_STRINGS = ' ';
        TIME          : TIME_STRINGS = '';

  function TIME_NOW : TIME_STRINGS;
  (***********************************************************************)
  (*   This routine returns a string corresponding to the  present  TIME  in *)
  (* the PC.  The format is 'HH:MM:SS am/pm'.                            *)
  (***********************************************************************)

    var  (**************************************************************)
         (*  Used to determine the present time (as reported by the PC). *)
         (**************************************************************)
         HR, MIN, SEC, SEC100TH : word;

         (**************************************************************)
         (*  Temporary variables to hold the intermediate string results. *)
         (**************************************************************)
         S : TIME_STRINGS; TEMP . string[6];

    begin (* TIME_NOW *)
      gettime(HR, MIN, SEC, SEC100TH);
      str(HR mod 12, S);
      if S = '0' then S := '12';
      S := S + ':';
      if MIN < 10 then S := S + '0';
      str(MIN, TEMP);
      S := S + TEMP;
      S := S + ':';
      if SEC < 10 then S := S + '0';
      str[SEC, TEMP);
      S := S + TEMP;
      if (HR div 12) = 0
        then
          TIME_NOW := S + ' am'
        else
          TIME_NOW := S + ' pm';
    end; (* TIME_NOW *)

  function DATE_NOW : DATE_STRINGS;
  (***********************************************************************)
  (*   This routine returns a string corresponding to the  present  DATE  in *)
  (* the PC.  The string is of the form 'MM-DD-YY'.                      *)
  (***********************************************************************)

    var (***************************************************************)
        (*   Used to determine the preset date (as reported by the PC).  *)
        (***************************************************************)
        YEAR, MONTH, DAY, DAY_OF_WEEK : word;

        (***************************************************************)
        (*   Temporary variable to hold the intermediate string results. *)
        (***************************************************************)
        S : DATE_STRINGS;

    begin (* DATE_NOW *)
      getdate(YEAR, MONTH, DAY, DAY_OF_WEEK);
      S := '';
      S :=     chr(ord(MONTH div 10) + ord('0'));
      S := S + chr(ord(MONTH mod 10) + ord('0')) + '-';
      S .= S + chr(ord(DAY   div 10) + ord('0'));
      S := S + chr(ord(DAY   mod 10) + ord('0')) + '-';
      S := S + chr(ord((YEAR mod 100) div 10) + ord('0'));
      S := S + chr(ord(YEAR  mod 10) + ord('0'));
      DATE_NOW := S;
    end; (* DATE_NOW *)

  begin (* UPDATE_STATUS_MESSAGES *)
    (* Update Time string *)
    settextstyle(defaultfcnt, horizdir, 1);
    settextjustify(righttext, centertext);
    TIME := TIME_NOW;
    if FORCE_UPDATE or (TIME <> PREVIOUS_TIME) then
      begin
        setcolor(SCREEN.BACKGROUND_COLOR);
        outtextxy
          (
            MAX_X - 2 * EDGE_OFFSET,
            MAX_Y + 2 * MESSAGE_REGION div 3,
            PREVIOUS_TIME
          );
        PREVIOUS_TIME := TIME;
        setcolor(SCREEN.STATUS_COLOR);
        outtextxy
          (
            MAX_X - 2 * EDGE_OFFSET,
            MAX_Y + 2 * MESSAGE_REGION div 3,
            PREVIOUS_TIME
          );
      end; (* if...then *)
```

```
    (* Update Date string *)
    settextjustify(righttext, centertext);
    DATE := DATE_NOW;
    if FORCE_UPDATE or (DATE <> PREVIOUS_DATE) then
      begin
        setcolor(SCREEN.BACKGROUND_COLOR);
        outtextxy
          (
           MAX_X - 2 * EDGE_OFFSET,
           MAX_Y + MESSAGE_REGION div 3,
           PREVIOUS_DATE
          );
        setcolor(SCREEN.STATUS_COLOR);
        PREVIOUS_DATE := DATE;
        outtextxy
          (
           MAX_X - 2 * EDGE_OFFSET,
           MAX_Y + MESSAGE_REGION div 3,
           PREVIOUS_DATE
          );
      end; (* if...then *)

    (* Update Generation/Paused status *)
    settextjustify(lefttext, centertext);
    setcolor(SCREEN.BACKGROUND_COLOR);
    if PREVIOUS_USER_PAUSED
      then
        outtextxy
          (
           2 * EDGE_OFFSET,
           MAX_Y + 1 * MESSAGE_REGION div 3,
           'Paused (' + INT_STR(GENERATION) + ')'
          )
      else if FORCE_UPDATE or (PREVIOUS_GENERATION <> GENERATION) then
        outtextxy
          (
           2 * EDGE_OFFSET,
           MAX_Y + 1 * MESSAGE_REGION div 3,
           'Generation ' + INT_STR(GENERATION)
          );
    PREVIOUS_USER_PAUSED := USER_PAUSED;
    setcolor(SCREEN.STATUS_COLOR);
    if PREVIOUS_USER_PAUSED
      then
        outtextxy
          (
           2 * EDGE_OFFSET,
           MAX_Y + 1 * MESSAGE_REGION div 3,
           'Paused (' + INT_STR(GENERATION) + ')'
          )
      else if FORCE_UPDATE or (PREVIOUS_GENERATION <> GENERATION) then
        outtextxy
          (
           2 * EDGE_OFFSET,
           MAX_Y + 1 * MESSAGE_REGION div 3,
           'Generation ' + INT_STR(GENERATION)
          );
    PREVIOUS_GENERATION := GENERATION;

    if FORCE_UPDATE or (INDIVIDUAL <> PREVIOUS_INDIVIDUAL) then
      begin
        settextjustify(lefttext, centertext);
        setcolor(SCREEN.BACKGROUND_COLOR);
        outtextxy
          (
           2 * EDGE_OFFSET,
           MAX_Y + 2 * MESSAGE_REGION div 3,
           'Individual ' + INT_STR(PREVIOUS_INDIVIDUAL) + '/' +
                           INT_STR(POPULATION_SIZE)
          );
        PREVIOUS_INDIVIDUAL := INDIVIDUAL;
        setcolor(SCREEN.STATUS_COLOR);
        outtextxy
          (
           2 * EDGE_OFFSET,
           MAX_Y + 2 * MESSAGE_REGION div 3,
           'Individual ' + INT_STR(PREVIOUS_INDIVIDUAL) + '/' +
                           INT_STR(POPULATION_SIZE)
          );
      end; (* if...then *)
  end; (* UPDATE_STATUS_MESSAGES *)

procedure SET_GRAPH_SCALES(var USER : USER_GRAPH_SCALES; GRAPH_NUMBER:integer);
(***************************************************************************)
(*   Takes the user scales and moves them into the global graph scales.    *)
(***************************************************************************)

  begin (* SET_GRAPH_SCALES *)
    with USER_GRAPH(GRAPH_NUMBER).SCALES do
      begin
        case GRAPH_NUMBER of
          1 : begin
                FIRST := 1;
                LABT  := POPULATION_SIZE;
              end;
          2 : begin
                FIRET := 0;
                LABT  := 100;
              end;
          3 : begin
                FIRST := 0;
                LABT  := 100;
              end;
        end; (* case...of *)
        if USER.SCALES.HIGH_LABEL(X_AXIS) > USER.SCALES.LOW_LABEL(X_AXIS)
          then
            begin
              HIGH_LABEL(X_AXIS) := USER.SCALES.HIGH_LABEL(X_AXIS);
              LOW_LABEL(X_AXIS)  := USER.SCALES.LOW_LABEL(X_AXIS);
            end (* if...then *)
          else
            write(BELL);
        NUMBER_OF_TICK_MARKS(X_AXIS) :=
            USER.SCALES.NUMBER_OF_TICK_MARKS(X_AXIS);
        if USER.SCALES.HIGH_LABEL(Y_AXIS) > USER.SCALES.LOW_LABEL(Y_AXIS)
          then
            begin
              HIGH_LABEL(Y_AXIS) := USER.SCALES.HIGH_LABEL(Y_AXIS);
              LOW_LABEL(Y_AXIS)  := USER.SCALES.LOW_LAGEL(Y_AXIS);
            end (* if...then *)
          else
            write(BELL);
        NUMBER_OF_TICK_MARKS(Y_AXIS) :=
            USER.SCALES.NUMBER_OF_TICK_MARKS(Y_AXIS);
        with USER_GRAPH(GRAPH_NUMBER) do
          begin
            PEAK_DETECTION(X_AXIS)    := USER.PEAK_DETECTION(X_AXIS(;
            PEAK_DETECTION(Y_AXIS)    := USER.PEAK_DETECTION(Y_AXIS(;
          end; (* with...do *)
      end; (* with...do *)
  end; (* SET_GRAPH_SCALES *)

procedure INITIALIZE;
(***************************************************************************)
(*   This procedure parses  the  command-line  parameters,  initializes all *)
(* global data structures, reads the INI file.                              *)
(***************************************************************************)

  var (*********************************************************************)
      (*   Loop Control Variable for parsing the command-line parameters.  *)
      (*********************************************************************)
      LCV, GRAPH_NUMBER : integer;

      (*********************************************************************)
      (*   The command-line parameter begin parsed.                        *)
      (*********************************************************************)
      PARAM : string;

      (*********************************************************************)
      (*   Used to detect invalid command-line parameters so that the 'help' *)
      (* info can be displayed.                                             *)
      (*********************************************************************)
      INVALID_PARAM : boolean;

function UP_STRING(S : string) : string;
(***************************************************************************)
(*   This routine returns a string corresponding to the upper  case  value *)
(* of the string passed to it.                                              *)
(***************************************************************************)

  var LCV : integer;

  begin (* UP_STRING *)
    UP_STRING(0) := S(0);
    for LCV := 1 to length(S) do
      UP_STRING(LCV) := upcase(S(LCV));
  end; (* UP_STRING *)

procedure PROCESS_INI_FILE;
(***************************************************************************)
(*   This procedure finds and reads the INI file.  All  *valid*  INI  file *)
(* lines are processed (invalid lines are silently ignored).               *)
(***************************************************************************)

  const (*******************************************************************)
        (*   The following are used to parse the INI file lines of text.   *)
        (*******************************************************************)
        CONFIG_LABEL                     = '(CONFIG)';
        NORMAL_COLORS_LABEL              = '(NORMALCOLORS)';
        REVERSE_COLORS_LABEL             = '(REVERSECOLORS)';
        EVALUATIONS_LABEL                = '(EVALUATIONS)';
        FITNESS_HISTOGRAM_LABEL          = '(FITNESSHISTOGRAM)';
        DIVERSITY_HISTOGRAM_LABEL        = '(DIVERSITYHISTOGRAM)';
        COMMENT_DELIMITER                = ';';
        TRUE_FLAG                        = 'TRUE';
        FALSE_FLAG                       = 'FALSE';
        IMAGE_DIRECTORY_CONTROL          = 'IMAGEDIRECTORY';
        VIDEO_MODE_CONTROL               = 'VIDEOMODE';
        VGA_MODE_FLAG                    = 'VGA';
        SVGA_MODE_FLAG                   = 'SVGA';
        VESA_MODE_FLAG                   = 'VESA';
        SVESA_MODE_FLAG                  = 'SVESA';
        BACKGROUND_COLOR_CONTROL         = 'BACKGROUNDCOLOR';
        AXIS_COLOR_CONTROL               = 'AXISCOLOR';
        GRID_COLOR_CONTROL               = 'GRIDCOLOR';
        AXIS_LABEL_COLOR_CONTROL         = 'AXISLABELCOLOR';
        GRAPH_COLOR_CONTROL              = 'GRAPHCOLOR';
        STATUS_COLOR_CONTROL             = 'STATUSCOLOR';
        STATUS_HIGHLIGNT_COLOR_CONTROL   = 'STATUSHIGHLIGHTCOLOR';
        MAIN_TITLE_COLOR_CONTROL         = 'MAINTITLECOLOR';
        BORDER_COLOR_CONTROL             = 'BORDERCOLOR';
        MAX_X_CONTROL                    = 'MAXX';
        MIN_X_CONTROL                    = 'MINX';
        X_TICK_MARKS_CONTROL             = 'XTICKMARKS';
        X_PEAK_DETECTION_CONTROL         = 'XPEAKDETECTION';
        MAX_Y_CONTROL                    = 'MAXY';
        MIN_Y_CONTROL                    = 'MINY';
        Y_TICK_MARKS_CONTROL             = 'YTICKMARKS';
        Y_PEAK_DETECTION_CONTROL         = 'YPEAKDETECTION';
        Y_AVERAGE_DETECTION_CONTROL      = 'YAVERAGEDETECTION';

        (*******************************************************************)
        (*  State variable that defines the section of the INI  file  that *)
        (* is eing parsed.                                                  *)
        (*******************************************************************)
        PARSING :
          (
           UNDEFINED,
           CONFIG_SECTION,
           COLOR_SECTION,
           REVERSE_SECTION,
           EVALUATIONS_SECTION,
           FITNESS_HISTOGRAM_SECTION,
           DIVERSITY_HISTOGRAM_SECTION
          ) = UNDEFINED;

  var   (*******************************************************************)
        (*    INI file parsing variables.                                   *)
        (*******************************************************************)
        INI_FILENAME : string(sizeof(dirstr)+sizeof(namestr)+sizeof(extstr));
        LINE         : string;
        INI_FILE     : text;
        POSITION     : integer;
        CONTROL_CHAR : char;

  procedure BOOLEAN_VALUE(CONTROL : string; var PARAMETER : boolean);
  (*************************************************************************)
  (*   This procedure examines the global LINE searching for  CONTROL,  if *)
  (* it is found, it's boolean value is set based on the rest of the LINE. *)
  (*************************************************************************)

    begin (* BOOLEAN_VALUE *)
      if UP_STRING(copy(LINE, 1, POSITION - 1)) = CONTROL then
        begin
          delete(LINE, 1, POSITION);
          if UP_STRING(copy(LINE, 1, length(TRUE_FLAG))) = TRUE_FLAG then
            PARAMETER := true
          else if UP_STRING(copy(LINE, 1, length(FALSE_FLAG))) = FALSE_FLAG
            then
              PARAMETER := false;
        end; (* if...then *)
    end; (* BOOLEAN_VALUE *)

  procedure INTEGER_VALUE(CONTROL : string; var PARAMETER : longint);
  (*************************************************************************)
  (*   This procedure examines the global LINE searching for  CONTROL,  if *)
  (* it is found, it's integer value is set based on the rest of the LINE. *)
  (*************************************************************************)

    var (*****************************************************************)
        (*   Used to convert text to integers.                            *)
        (*****************************************************************)
        VALUE, CODE  : integer;

    begin (* INTEGER_VALUE *)
      if UP_STRING(copy(LINE, 1, POSITION - 1)) = CONTROL then
        begin
          delete(LINE, 1, POSITION);
          POSITION := 1;
          while LINE(POSITION) in ('0'..'9') do inc(POSITION);
          val(copy(LINE, 1, POSITION - 1), VALUE, CODE);
          if CODE = 0 then PARAMETER := VALUE;
        end; (* if...then *)
    end; (* INTEGER_VALUE *)

  procedure DIRECTORY_VALUE(CONTROL : string; var PARAMETER : dirstr);
  (*************************************************************************)
  (*   This procedure examines the global LINE searching for  CONTROL,  if *)
```

```
(* it is found then it's string value is set based on the  rest  of  the *)
(* LINE.                                                                  *)
(**************************************************************************)

  var (********************************************************************)
      (*  Temporary variables used to strip characters from the pathname.*)
      (********************************************************************)
      DIRECTORY   dirstr; FILENAME : namestr; EXTENSION : extstr;

  begin (* DIRECTORY_VALUE *)
    if UP_STRING(copy(LINE, 1, POSITION - 1)) = CONTROL then
      begin
        PARAMETER := copy(LINE, POSITION + 1, length(LINE));
        if pos(SPACE, PARAMETER) <> 0 then
          PARAMETER[pos(SPACE, PARAMETER)] := '\';
        if pos('\\', PARAMETER) <> 0 then
          PARAMETER[0] := chr(pos('\\', PARAMETER));
        fsplit(PARAMETER, DIRECTORY, FILENAME, EXTENSION);
        PARAMETER := UP_STRING(DIRECTORY);
      end; (* if...then *)
  end; (* DIRECTORY_VALUE *)

procedure VIDEO_VALUE(CONTROL : string; var PARAMETER : VIDEO_MODES);
(**************************************************************************)
(*  This procedure examines the global LINE searching for  CONTROL,  if *)
(* it is found then it's string value is examined to see if  it  matches *)
(* one of the valid video modes.                                          *)
(**************************************************************************)

  begin (* VIDEO_VALUE *)
    if UP_STRING(copy(LINE, 1, POSITION - 1)) = CONTROL then
      begin
        delete(LINE, 1, POSITION);
        while LINE[length(LINE)] = SPACE do delete(LINE, length(LINE), 1);
        POSITION := 1;
        if UP_STRING(LINE) = VGA_MODE_FLAG
          then
            PARAMETER := VGA
        else if UP_STRING(LINE) = SVGA_MODE_FLAG
          then
            PARAMETER := SVGA
        else if UP_STRING(LINE) = VESA_MODE_FLAG
          then
            PARAMETER := VESA
        else if UP_STRING(LINE) = SVESA_MODE_FLAG
          then
            PARAMETER := SVESA;
      end; (* if...then *)
  end; (* VIDEO_VALUE *)

procedure COLOR_VALUE(CONTROL : string; var PARAMETER : word);
(**************************************************************************)
(*  This procedure examines the global LINE searching for  CONTROL,  if *)
(* it is found, it's string value is examined to see if it  matches  one *)
(* of the valid colors.                                                   *)
(**************************************************************************)

  var (********************************************************************)
      (*  Temporary variable to hold the color value, needed because it *)
      (* may not be valid.                                                *)
      (********************************************************************)
      TEMP_COLOR : byte;

  function COLOR_MATCH(PARAMETER : string; var COLOR : byte) : boolean;
  (************************************************************************)
  (*  This routine performs an associative match on  the  PARAMETER  to *)
  (* the list of possible color choices.  Any enumeration of the letters *)
  (* of a color will match as long as enough  letters  are  provided  to *)
  (* determine which color is desired.   For example: matched to  'blue' *)
  (* include blu, ble, bue & be.                                          *)
  (************************************************************************)

    const (****************************************************************)
          (*   Strings to match the colors to.                            *)
          (****************************************************************)
          COLOR_STRING : array [0..15] of string =
            (
             'BLACK', 'BLUE', 'GREEN', 'CYAN', 'RED', 'MAGENTA', 'BROWN',
             'LIGHTGRAY', 'DARKGRAY', 'LIGHTBLUE', 'LIGHTGRSEN',
             'LIGHTCYAN', 'LIGNTRED', 'LIGHTMAGENTA', 'YELLOW', 'WHITE'
            );

    var   (****************************************************************)
          (*  Index into parameter characters.                            *)
          (****************************************************************)
          PARAM_LCV . integer;

          (****************************************************************)
          (*  Flag to indicate that the character being searched for was *)
          (* found or not.                                                *)
          (****************************************************************)
          FOUND : boolean;

          (****************************************************************)
          (*  Local copy of the string being searched.                    *)
          (****************************************************************)
          LOCAL_PARAMETER : string;

    begin (* COLOR_MATCH *)
      COLOR := 0;
      repeat
        LOCAL_PARAMETER := PARAMETER;
        FOUND := pos(LOCAL_PARAMETER[1], COLOR_STRING[COLOR]) = 1;
        PARAM_LCV := 1;
        while FOUND end (PARAM_LCV <= length(PARAMETER)) do
          begin
            FOUND := FOUND and (pos(LOCAL_PARAMETER[PARAM_LCV],
                                    COLOR_STRING[COLOR]) <> 0);
            inc(PARAM_LCV);
          end; (* while...do *)
        inc(COLOR);
      until FOUND or (COLOR > 15);
      if not FOUND
        then
          COLOR := $FF
        else
          dec(COLOR);
      COLOR_MATCH := FOUND;
    end; (* COLOR_MATCH *)

  begin (* COLOR_VALUE *)
    if UP_STRING(copy(LINE, 1, POSITION - 1)) = CONTROL then
      begin
        delete(LINE, 1, POSITION);
        while LINE[length(LINE)] = SPACE do delete(LINE, length(LINE), 1);
        POSITION := 1;
        if COLOR_MATCH(UP_STRING(LINE), TEMP_COLOR) then
          PARAMETER := TEMP_COLOR;
      end; (* if...then *)
  end; (* COLOR_VALUE *)

begin (* PROCESS_INI_FILE *)
  INI_FILENAME := paramstr(0);
  delete(INI_FILERAME, length(INI_FILENAME) - 2, 3);
  INI_FILENAME := INI_FILENAME + 'INI';
  assign(INI_FILE, INI_FILENAME);
  (*$I-*) reset(INI_FILE); (*$I+*)
  if ioresult = 0 then
    begin
      while not eof(INI_FILE) do
        begin
          readln(INI_FILE, LINE);
          POSITION := pos(COMMENT_DELIMITER, LINE);
          if POSITION <> 0 then
            delete(LINE, POSITION, length(LINE));
          if UP_STRING(copy(LINE, 1, length(NORMAL_COLORS_LABEL))) =
                                            NORMAL_COLORS_LABEL then
            begin
              PARSING := COLOR_SECTION;
              readln(INI_FILE, LINE);
              POSITION := pos(COMMENT_DELIMITER, LINE);
              if POSITION <> 0 then
                delete(LINE, POSITION, length(LINE));
            end (* if...then *)
          else if UP_STRING(copy(LINE, 1, length(REVERSE_COLORS_LABEL))) =
                                            REVERSE_COLORS_LABEL then
            begin
              PARSING := REVERSE_SECTION;
              readln(INI_FILE, LINE);
              POSITION := pos(COMMENT_DELIMITER, LINE);
              if POSITION <> 0 then
                delete(LINE, POSITION, length(LINE));
            end (* if...then *)
          else if UP_STRING(copy(LINE, 1, length(CONFIG_LABEL))) =
                                            CONFIG_LABEL then
            begin
              PARSING := CONFIG_SECTION;
              readln(INI_FILE, LINE);
              POSITION := pos(COMMENT_DELIMITER, LINE);
              if POSITION <> 0 then
                delete(LINE, POSITION, length(LINE));
            end (* if...then *)
          else if UP_STRING(copy(LINE, 1, length(EVALUATIONS_LABEL))) =
                                            EVALUATIONS_LABEL then
            begin
              PARSING := EVALUATIONS_SECTION;
              readln(INI_FILE, LINE);
              POSITION := pos(COMMENT_DELIMITER, LINE);
              if POSITION <> 0 then
                delete(LINE, POSITION, length(LINE));
            end (* if...then *)
          else if UP_STRING(copy(LINE, 1,
                          length(FITNESS_HISTOGRAM_LABEL))) =
                                  FITNESS_HISTOGRAM_LABEL then
            begin
              PARSING := FITNESS_HISTOGRAM_SECTION;
              readln(INI_FILE, LINE);
              POSITION := pos(COMMENT_DELIMITER, LINE);
              if POSITION <> 0 then
                delete(LINE, POSITION, length(LINE));
            end (* if...then *)
          else if UP_STRING(copy(LINE, 1,
                          length(DIVERSITY_HISTOGRAM_LABEL))) =
                                  DIVERSITY_HISTOGRAM_LABEL then
            begin
              PARSING := DIVERSITY_HISTOGRAM_SECTION;
              readln(INI_FILE, LINE);
              POSITION := pos(COMMENT_DELIMITER, LINE);
              if POSITION <> 0 then
                delete(LINE, POSITION, length(LINE));
            end; (* if...then *)

          POSITION := pos('=', LINE);
          if POSITION <> 0 then
            case PARSING of
              CONFIG_SECTION
                begin
                  DIRECTORY_VALUE(IMAGE_DIRECTORY_CONTROL, IMAGE_DIRSCTORY);
                  if DISPLAY_MODE = NO_DISPLAY_MODE then
                    VIDEO_VALUE(VIDEO_MODE_CONTROL,         DISPLAY_MODE);
                end; (* Configuration Section *)
              COLOR_SECTION :
                begin
                  COLOR_VALUE(BACKGROUND_COLOR_CONTROL,
                              NORMAL.BACKGROUND_COLOR);
                  COLOR_VALUE(AXIS_COLOR_CONTROL,
                              NORMAL.AXIS_COLOR);
                  COLOR_VALUE(GRID_COLOR_CONTROL,
                              NORMAL.GRID_COLOR);
                  COLOR_VALUE(AXIS_LABEL_COLOR_CONTROL,
                              NORMAL.AXIS_LABEL_COLOR);
                  COLOR_VALUE(GRAPH_COLOR_CONTROL,
                              NORMAL.GRAPH_COLOR);
                  COLOR_VALUE(STATUS_COLOR_CONTROL,
                              NORMAL.STATUS_COLOR);
                  COLOR_VALUE(STATUS_HIGHLIGHT_COLOR_CONTROL,
                              NORMAL.STATUS_HIGHLIGHT_COLOR);
                  COLOR_VALUE(MAIN_TITLE_COLOR_CONTROL,
                              NORMAL.MAIN_TITLE_COLOR);
                  COLOR_VALUE(BORDER_COLOR_CONTROL,
                              NORMAL.BORDER_COLOR);
                end; (* Color Section *)
              REVERSE_SECTION :
                begin
                  COLOR_VALUE(BACKGROUND_COLOR_CONTROL,
                              REVERSE.BACKGROUND_COLOR);
                  COLOR_VALUE(AXIS_COLOR_CONTROL,
                              REVERSE.AXIS_COLOR);
                  COLOR_VALUE(GRID_COLOR_CONTROL,
                              REVERSE.GRID_COLOR);
                  COLOR_VALUE(AXIS_LABEL_COLOR_CONTROL,
                              REVERSE.AXIS_LABEL_COLOR);
                  COLOR_VALUE(GRAPH_COLOR_CONTROL,
                              REVERSE.GRAPH_COLOR);
                  COLOR_VALUE(STATUS_COLOR_CONTROL,
                              REVERSE.STATUS_COLOR);
                  COLOR_VALUE(STATUS_HIGHLIGNT_COLOR_CONTROL,
                              REVERSE.STATUS_HIGHLIGHT_COLOR);
                  COLOR_VALUE(MAIN_TITLE_COLOR_CONTROL,
                              REVERSE.MAIN_TITLE_COLOR);
                  COLOR_VALUE(BORDER_COLOR_CONTROL,
                              REVERSE.BORDER_COLOR);
                end; (* Reverse Color Section *)
              EVALUATIONS_SECTION :
                with USER_GRAPH[1].SCALES do
                  begin
                    INTEGER_VALUE(MAX_X_CONTROL,
                                  HIGH_LABEL[X_AXIS]);
                    INTEGER_VALUE(MIN_X_CONTROL,
                                  LOW_LABEL[X_AXIS]);
                    INTEGER_VALUE(X_TICK_MARKS_CONTROL,
                                  NUMBER_OF_TICK_MARKS[X_AXIS]);
                    BOOLEAK_VALUE(X_PEAK_DETECTION_CONTROL,
                                  PEAK_DETECTION[X_AXIS]);
                    INTEGER_VALUE(MAX_Y_CONTROL,
                                  HIGH_LABEL[Y_AXIS]);
                    INTEGER_VALUE(MIN_Y_CONTROL,
                                  LOW_LABEL[Y_AXIS]);
                    INTEGER_VALUE(Y_TICK_MARKS_CONTROL,
                                  NUMBER_OF_TICK_MARKS[Y_AXIS]);
                    BOOLEAK_VALUE(Y_PEAK_DETECTION_CONTROL,
                                  PEAK_DETECTION[Y_AXIS]);
                    BOOLEAK_VALUE(Y_AVERAGE_DETECTION_CONTROL,
                                  AVERAGE_DETECTION);
                  end; (* with...do *)
              FITNESS_HISTOGRAM_SECTION :
                with USER_GRAPH[2].SCALES do
                  begin
```

```
                  INTEGER_VALUE(MAX_X_CONTROL,
                                HIGH_LABEL[X_AXIS]];
                  INTEGER_VALUE(MIN_X_CONTROL,
                                LOW_LABEL[X_AXIS]];
                  INTEGER_VALUE(X_TICK_MAXKS_CONTROL,
                                NUMBER_OF_TICK_MARKS[X_AXIS]);
                  BOOLEAN_VALUE(X_PEAK_DETECTION_CONTROL,
                                PEAK_DETECTION[X_AXIS]);
                  INTEGER_VALUE(MAX_Y_CONTROL,
                                HIGH_LABEL[Y_AXIS]);
                  INTEGER_VALUE(MIN_Y_CONTROL,
                                LOW_LABEL[Y_AXIS]);
                  INTEGER_VALUE(Y_TICK_MARKS_CONTROL,
                                NUMBER_OF_TICK_MARKS[Y_AXIS]);
                  BOOLEAN_VALUE(Y_PEAK_DETECTION_CONTROL,
                                PEAK_DETECTION[Y_AXIS]);
                  BOOLEAN_VALUE(Y_AVERAGE_DETECTION_CONTROL,
                                AVERAGE_DETECTION);
                end; (* with...do *)
            DIVERSITY_HISTOGRAM_SECTION :
              with USER_GRAPH[3].SCALES do
                begin
                  INTEGER_VALUE(MAX_X_CONTROL,
                                HIGH_LABEL[X_AXIS]);
                  INTEGER_VALUE(MIN_X_CONTROL,
                                LOW_LABEL[X_AXIS]);
                  INTEGER_VALUE(X_TICK_MARKS_CONTROL,
                                NUMBER_OF_TICK_MARKS[X_AXIS]);
                  BOOLEAN_VALUE(X_PEAK_DETECTION_CONTROL,
                                PEAK_DETECTION[X_AXIS]);
                  INTEGER_VALUE(MAX_Y_CONTROL,
                                HIGH_LABEL[Y_AXIS]);
                  INTEGER_VALUE(MIN_Y_CONTROL,
                                LOW_LABEL[Y_AXIS]);
                  INTEGER_VALUE(Y_TICK_MARKS_CONTROL,
                                NUMBER_OF_TICK_MARKS[Y_AXIS]);
                  BOOLEAK_VALUE(Y_PEAK_DETECTION_CONTROL,
                                PEAK_DETECTION(Y_AXIS));
                  BOOLEAN_VALUE(Y_AVERAGE_DETECTION_CONTROL,
                                AVERAGE_DETECTION);
                end; (* with...do *)
            else (* Do Nothing! *);
          end; (* case...of *)
      end; (* while...do *)
    close(INI_FILE];
  end; (* if...then *)
end; (* PROCESS_INI_FILE *)

begin (* INITIALIZE *)
  (* Initialize 'constants' *)
  directvideo := false;

  (*  Parse Command-Line parameters *)
  INVALID_PARAM := false;
  for LCV := 1 to paramcount do
    begin
      PARAM := UP_STRING(paramstr(LCV));
      case PARAM[2] of
        'R' : if PARAM = '-REVERSE'
                then
                  REVERSE_VIDEO := true
                else
                  INVALID_PARAM := true;
        'V' . if PARAM = '-VGA' then
                DISPLAY_MODE  := VGA
              else if PARAM = '-VESA' then
                DISPLAY_MODE  := VESA
              else
                INVALID_PARAM := true;
        'S' : if PARAM = '-SVGA' then
                DISPLAY_MODE  := SVGA
              else if PARAM = '-SVESA' then
                DISPLAY_MODE  := SVESA
              else
                INVALID_PARAM := true;
        else INVALID_PARAM := true;
      end; (* case...of *)
    end; (* for...to...do *)

  if INVALID_PARAM then
    begin
      writeln(LF,
              '  Exolution of Solutions to Real-Time Problems', CR, LF,
              LF,
              '  Usage:', CR, LF,
              LF,
              (*$IFDEF MATH_GA *)
              '      GMATH-GA [-VGA(-SVGA|-VESA(-SVESA] [-Reverse]', CR, LF,
              (*$ELSE *)
              '      GRTOS-GA [-VGA|-SVGA|-VESA(-SVESA] [-Reverse]', CR, LF,
              (*$ENDIF *]
              LF,
              '      Where.', CR, LF,
              '        -VGA     = 640x480 graphics mode', CR, LF,
              '        -SVGA    = 800x600 graphics mode', CR, LF,
              '        -VESA    = 1024x768 graphics mode (default)', CR, LF,
              '        -SVESA   = 1280x1024 graphics mode', CR, LF,
              '        -Reverse = Reverse Video Color Scheme', CR, LF
             );
      halt(INVALID_PARAMETER_ERROR_LEVEL);
    end; (* if...then *)

    GRAPH_OFFSET := 6 * EDOE_OFFSET;
    with USER_GRAPH[1].SCALES do
      begin
        TITLE                          := 'Fitness';
        GRAPH_TYPE                     := LINE_GRAPH;
        LABEL_NAME[X_AXIS]             := 'Individual';
        HIGH_LABEL[X_AXIS]             := (*Configuration #*) POPULATION_SIZE;
        LOW_LABEL[X_AXIS]              := (*Configuration #*] 0;
        NUMBER_OF_TICK_MARKS[X_AXIS]   := 10;
        PEAK_DETECTION[X_AXIS]         := false;
        LABEL_NAME[Y_AXIS]             := 'Fitness';
        HIGH_LABEL[Y_AXIS]             := (*Score*) 100;
        LOW_LABEL]Y_AXIS]              := (*Score*) 0;
        NUMBER_OF_TICK_MARKS[Y_AXIS]   := 10;
        PEAK_DETECTION[Y_AXIS]         := false;
        AVERAGE_DETECTION              := false;
      end; (* with...do *)
    with USER_GRAPH[2].SCALES do
      begin
        TITLE                          := 'Fitness Histogram';
        GRAPH_TYPE                     := BAR_GRAPH;
        LABEL_NAME[X_AXIS]             := 'Fitness';
        HIGH_LABEL[X_AXIS]             := (*Score*) 100;
        LOW_LABEL[X_AXIS]              := (*Score*) 0;
        NUMBER_OF_TICK_MARKS[X_AXIS]   := 20;
        PEAK_DETECTION[X_AXIS]         := false;
        LABEL_NAME[Y_AXIS]             := '%';
        HIGH_LABEL[Y_AXIS]             := (*Percent*] 100;
        LOW_LABEL[Y_AXIS]              := (*Percent*] 0;
        NUMBER_OF_TICK_MARKS]Y_AXIS]   := 10;
        PEAK_DETECTION[Y_AXIS]         := true;
        AVERRGE_DETECTION              := false;
      end; (* with...do *)
    with USER_GRAPH[3].SCALES do
      begin
        TITLE                          := 'Diversity Histogram';
        GRAPH_TYPE                     := BAR_GRAPH;
        LABEL_NAME[X_AXIS]             := 'Diversity';
        HIGH_LABEL[X_AXIS]             := (*Configuration #*) 100;
        LOW_LABEL[X_AXIS]              := (*Configuration #*) 0;
        NUMBER_OF_TICK_MARKS[X_AXIS]   := 20;
        PEAK_DETECTION[X_AXIS]         := false;
        LABEL_NAME[Y_AXIS]             := '%';
        HIGH_LABEL[Y_AXIS]             := (*Precent*) 100;
        LOW_LABEL[Y_AXIS]              := (*Percent*) 0;
        NUMBER_OF_TICK_MARKS[Y_AXIS]   := 10;
        PEAK_DETECTION[Y_AXIS]         := true;
        AVERAGE_DETECTION              := false;
      end; (* with...do *)

    PROCESS_INI_FILE;

    if REVERSE_VIDEO
      then
        SCREEN := REVERSE
      else
        SCREEN := NORMAL;

    (* Handle the case when no INI file (or parameter) was found *)
    if DISPLAY_MODE = NO_DISPLAY_MODE then
      DISPLAY_MODE := VESA;

    (* Allocate Plot data structures *)
    for GRAPH_NUMBER := 1 to MAX_NUMBER_OF_GRAPHS do
      begin
        new(PLOT[GRAPH_NUMBER].GRAPH_POINTS);
        for LCV := 0 to POPULATION_SIZE do
          with PLOT[GRAPH_NUMBER].GRAPH_POINTS^[LCV] do
            begin
              POSITION[X_AXIS] := 0.0;
              POSITION[Y_AXIS] := 0.0;
            end; (* for...to...do *)
        new(PLOT[GRAPH_NUMBER).PREVIOUS_GRAPH_POINTS);
        for LCV := 0 to POPULATION_SIZE do
          with PLOT[GRAPH_NUMBER].PREVIOUS_GRAPH_POINTS^[LCV] do
            begin
              POSITION[X_AXIS] := 0.0;
              POSITION]Y_AXIS] := 0.0;
            end; (* for...to...do *)
        SET_GRAPH_SCALES(USER_GRAPH[GRAPH_NUMBER], GRAPH_NUMBER);
      end; (* for...to...do *]
  end; (* INITIALIZE *]

procedure INIT_GRAPHICS;
(*************************************************************************)
(*   This procedure initializes the video graphics card based on  the  video *)
(* mode that has been selected.                                          *)
(*************************************************************************)

  var (*********************************************************************)
      (*   Graphic Video Driver parameters.                                *)
      (*********************************************************************)
      DRIVER : integer;

  begin (* INIT_GRAPHICS *)
    if DISPLAY_MODE = NO_DISPLAY_MODE then DISPLAY_MODE := VESA;
    case DISPLAY_MODE of
      VGA   : begin
                registerbgidriver(addr(EGAVGA_DRIVER));
                MESSAGE_REGION := 24;
              end; (* VGA *)
      SVGA  : begin
                installuserdriver('VESA16', @DETECT_VESA_16);
                registerbgidriver(addr(VESA16_DRIVER));
                MESSAGE_REGION := 36;
                VESA_16_MODE := 0;
              end; (* SVGA *)
      VESA  : begin
                installuserdriver('VESA16', @DETECT_VESA_16);
                registerbgidriver(addr(VESA16_DRIVER));
                MESSAGE_REGION := 36;
                VESA_16_MODE := 1;
              end; (* VESA *)
      SVESA . begin
                installuserdriver('VESA16', @DETECT_VESA_16];
                registerbgidriver(addr(VESA16_DRIVER)];
                MESSAGE_REGION := 36;
                VESA_16_MODE := 2;
              end; (* SVESA *]
    end; (* case...of *]

    DRIVER := detect;
    initgraph(DRIVER, GRAPHICS_MODE, '');
    if graphresult <> grOK then
      begin
        writeln(CR, LF, 'Fatal Error: Video mode not supported');
        halt(NON_SUPPORTED_VIDEO_MODE_ERROR_LEVEL);
      end; (* if...then *)
  end; (* INIT_GRAPHICS *)

procedure SHUTDOWN_GRAPHICS; far;
(*************************************************************************)
(*   Restores the video controller to standard DOS text mode.            *)
(*************************************************************************)

  begin (* SHUTDOWN_GRAPHICS *)
    closegraph;
  end; (* SHUTDOWN_GRAPHICS *)

procedure REDRAW_GRAPH(NUMBER_OF_GRAPHS : integer);
(*************************************************************************)
(*   Redraws the graph axis or displays the signon  logo  if  there  are  no *)
(* active graphs.                                                        *]
(*************************************************************************)

  var (*********************************************************************)
      (*   Graphic screen position of the next status text message.        *)
      (*********************************************************************)
      STATUS_POS : integer;

  procedure HIGHLIGNTED_TEXT(STR : string);
  (***********************************************************************)
  (*    Displays a text string in the status message  box  where  the  first *)
  (* letter of the string is highlighted.                                *]
  (***********************************************************************)

  var (*********************************************************************)
      (*   Loop Control Variable for indexing through string.              *)
      [*********************************************************************)
      LCV : integer;

    begin (* HIGHLIGHTED_TEXT *)
      setcolor(SCREEN.STATUS_HIGHLIGHT_COLOR];
      outtextxy
        (
         STATUS_POS,
         MAX_Y + MESSAGE_REGION div 3,
         STR[1]
        );
      STATUS_POS := STATUS_POS + textwidth(STR[1]);
      setcolor(SCREEN.STATUS_COLOR];
      outtextxy
        (
         STATUS_POS,
         MAX_Y + MESSAGE_REGION div 3,
         copy(STR, 2, length(STR))
```

```
    );
  for LCV := 2 to length(STR) do
    STATUS_POS := STATUS_POS + textwidth(STR[LCV]);

  STATUS_POS := STATUS_POS + 2 * textwidth('h');
end; (* HIGHLIGHTED_TEXT *)

procedure DRAW_RIT_LOGO(X, Y : integer);
(***********************************************************************)
(*  Draws the 'RIT' logo on the screen at the specified coordinates. *)
(***********************************************************************)

  const (**************************************************************)
        (*  Pointer to a buffer that contains the drawn logo (valid only *)
        (* after the first call). This allows the logo to be displayed *)
        (* more than once but only drawn once.                           *)
        (**************************************************************)
        LOGO : pointer = nil;

  var   (**************************************************************)
        (*  Work area of the screen coordinates.                         *)
        (**************************************************************)
        WORK_X, WORK_Y : integer;

  begin (* DRAW_RIT_LOGO *)
    X := X - 225;
    if LOGO = nil then
      begin
        (* Create Logo *)
        setfillstyle(solidfill, red);
        bar(X - 70, Y - 60, X + 520, Y + 60);
        setcolor(white);
        rectangle(X - 65, Y - 55, X + 515, Y + 55);
        setfillstyle(solidfill, white);
        floodfill(X, Y - 10, white);
        setcolor(red);
        settextstyle(TriplexFont, vertdir, 5);
        settextjustify(centertext, centertext);
        outtextxy(X -  40, Y - 1, 'R I T');
        setfillstyle(solidfill, red);
        circle(X -  32, Y + 17, 2); floodfill(X -  32, Y + 17, red);
        circle(X -  32, Y - 17, 2); floodfill(X -  32, Y - 17, red);
        setfillstyle(solidfill, lightgray);
        bar(X - 10, Y - 50, X + 5, Y + 50);
        settextstyle(SIMPLEX_FONT, horizdir, 0);
        settextjustify(lefttext, centertext);
        setusercharsize(7, 8, 3, 4);
        outtextxy(X +  40, Y +  5, 'ROCHESTER INSTITUTE OF TECHNOLOGY');
        setlinestyle(solidln, 0, thickwidth);
        setcolor(lightgray);
        line(X + 40, Y + 30, X + 470, Y + 30);

        (* Keep a copy of the logo so it doesn't need to redrawn *)
        getmem(LOGO, imagesize(X - 70, Y - 60, X + 520, Y + 60));
        getimage(X - 70, Y - 60, X + 520, Y + 60, LOGO^);
      end (* if...then *)
    else
      putimage(X - 70, Y - 60, LOGO^, normalput);
  end; (* DRAW_RIT_LOGO *)

procedure DRAW_COPYRIGHT;
(***********************************************************************)
(*  Draws the 'circle c' copyright notice.                              *)
(***********************************************************************)

  begin (* DRAW_COPYRIGHT *)
    outtext('c');
    circle
      (
       getx,
       gety + round(0.3 * textheight('c')),
       round(textheight('C') / 2.5)
      );
  end; (* DRAW_COPYRIGHT *)

procedure CREATE_GRAPH
(***********************************************************************)
  (
   AXIS_TYPE    : AXIS_TYPES;
   GRAPH_POS    : GRAPH_POSITIONS;
   GRAPH_NUMBER : integer;
   SCALE_INFO   : GRAPH_SCALE_INFO
  );
(*---------------------------------------------------------------------*)
(*  Creates a graph by allocating memory for the graph data structures *)
(* and initializing them as directed by the various parameters.        *)
(***********************************************************************)

  var (****************************************************************)
      (*  Loop Control Variable for initializing screen position values. *)
      (****************************************************************)
      LCV : integer;

  begin (* CREATE_GRAPH *)
    with PLOT[GRAPH_NUMBER] do
      begin
        GRAPH_SCALES := SCALE_INFO;
        VALUES.FIRST := GRAPH_SCALES.FIRST;
        VALUES.LABT  := GRAPH_SCALES.LABT;
        with GRAPH_SCALES do
          begin
            RANGE[X_AXIS] := abs(HIGH_LABEL[X_AXIS] - LOW_LABEL]X_AXIS]);
            RANGE[Y_AXIS] := abs(HIGH_LABEL[Y_AXIS] - LOW_LABEL]Y_AXIS]);
          end; (* with...do *)
        GRAPH_COLOR              := SCREEN.GRAPH_COLOR;
        PREVIOUS_PEAK            := getmaxx + 50;
        PREVIOUS_PEAK_VALUE      := getmaxy + 50;
        PREVIOUS_AVERAGE_VALUE   := getmaxy + 50;
        HORIZONTAL_OFFSET        := 2 * EDGE_OFFSET + GRAPH_OFFSET;
        HORIZONTAL_SIZE          := MAX_X - 2 * (GRAPH_OFFSET + EDGE_OFFSET);
        VERTICAL_OFFSET          := 2 * EDGE_OFFSET + GRAPH_OFFSET;
        VERTICAL_SIZE            := MAX_Y - 2 * (GRAPH_OFFSET + EDGE_OFFSET);

        case GRAPH_POS of
          FULL_SCREEN : (* Do Nothing *);
          TOP_HALF :
            begin
              VERTICAL_SIZE   := VERTICAL_SIZE / 2 - VERTICAL_OFFSET;
            end;
          BOTTOM_HALF :
            begin
              VERTICAL_SIZE   := VERTICAL_SIZE / 2 - VERTICAL_OFFSET;
              VERTICAL_OFFSET := VERTICAL_SIZE + 2 * VERTICAL_OFFSET;
            end;
          TOP_THIRD :
            begin
              VERTICAL_SIZE   := VERTICAL_SIZE / 3 - 1.5 * VERTICAL_OFFSET;
            end;
          MIDDLE_THIRD :
            begin
              VERTICAL_SIZE    := VERTICAL_SIZE / 3 - 1.5 * VERTICAL_OFFSET;
              VERTICAL_OFFSET  :=      VERTICAL_SIZE + 3 * VERTICAL_OFFSET;
            end;
          BOTTOM_THIRD :
            begin
              VERTICAL_SIZE   := VERTICAL_SIZE / 3 - 1.5 * VERTICAL_OFFSET;
              VERTICAL_OFFSET := 2 * VERTICAL_SIZE + 5 * VERTICAL_OFFSET;
            end;
        end; (* case...of *)

        with GRAPH_SCALES do
          begin
            (* Determine position for X Axis (Low, Centered, High) *)
            if ((LOW_LABEL[Y_AXIS] >= 0) and (HIGH_LABEL]Y_AXIS] >= 0) and
                (AXIS_TYPE = AXIS_AXIS)] or
               (AXIS_TYPE in [FORCE_LL, FORCE_LR])
              then
                X_AXIS_OFFSET := VERTICAL_OFFSET + VERTICAL_SIZE
              else if ((LOW_LABEL[Y_AXIS] <= 0) and (HIGH_LABEL]Y_AXIS] <= 0)
                      and (AXIS_TYPE = AUTO_AXIS)) or
                      (AXIS_TYPE in ]FORCE_UL, FORCE_UR]) then
                X_AXIS_OFFSET := VERTICAL_OFFSET
              else
                X_AXIS_OFFSET := VERTICAL_OFFSET + VERTICAL_SIZE / 2;

            (* Determine position for Y Axis (Left, Centered, Right) *)
            if ((LOW_LABEL[X_AXIS] >= 0) and (HIGH_LABEL]X_AXIS] >= 0) and
                (AXIS_TYPE = AUTO_AXIS)) or
               (AXIS_TYPE in [FORCE_LL, FORCE_UL])
              then
                Y_AXIS_OFFSET := HORIZONTAL_OFFSET
              else if ((LOW_LABEL]X_AXIS] <= 0) and (HIGH_LABEL[X_AXIS] <= 0]
                      and (AXIS_TYPE = AUTO_AXIS)) or
                      (AXIS_TYPE in ]FORCE_LR, FORCE_UR]) then
                Y_AXIS_OFFSET := HORIZONTAL_OFFSET + HORIZONTAL_SIZE
              else
                Y_AXIS_OFFSET := HORIZONTAL_OFFSET + HORIZONTAL_SIZE / 2;
          end; (* with...do *)

        for LCV := 0 to POPULATION_SIZE do
          with PLOT]GRAPH_NUMBER].PREVIOUS_GRAPH_POINTS^]LCV] do
            begin
              POSITION]X_AXIS] := HORIZONTAL_OFFSET;
              POSITION[Y_AXIS] := VERTICAL_OFFSET;
            end; (* with...do *)
      end; (* with...do *)
  end; (* CREATE_GRAPH *)

procedure DRAW_GRAPH_AXIS(GRAPH_NUMBER : integer);
(***********************************************************************)
(*   Draws the title, axis, grid lines, and tick marks for the graph.   *)
(***********************************************************************)

  var (****************************************************************)
      (*   Loop Control Variable for creating tick marks.              *)
      (****************************************************************)
      LCV : integer;

      (****************************************************************)
      (*   Value of axis tick mark graphic position.                   *)
      (****************************************************************)
      TEMP : longint;

      (****************************************************************)
      (*   Value of the tick mark label.                               *)
      (****************************************************************)
      TICK_LABEL : real;

  begin (* DRAW_GRAPH_AXIS *)
    with PLOT]GRAPH_NUMBER] do
      begin
        (* Add Graph Title *)
        setcolor(SCREEN.MAIN_TITLE_COLOR);
        settextjustify(centertext, centertext);
        case DISPLAY_MODE of
          VGA   : settextstyle(TRIPLEX_FOHT, horizdir, 1);
          SVGA  : settextstyle(TRIPLEX_FONT, horizdir, 2);
          VESA,
          SVESA : settextstyle(TRIPLEX_FONT, horizdir, 3);
        end; (* case...of *)
        outtextxy
          (
           MAX_X div 2,
           round(VERTICAL_OFFSET - GRAPH_OFFSET),
           GRAPH_SCALES.TITLE
          );

        (* Add X Tick Marks *)
        setusercharsize(1, 3, 1, 3);
        settextstyle(simplex_font, horizdir, 0);
        settextjustify(centertext, toptext);
        with GRAPH_SCALES do
          if NUMBER_OP_TICK_MARKS[X_AXIS] <> 0 then
            for LCV := 0 to NUMBER_OF_TICK_MARKS]X_AXIS] do
              begin
                TEMP := round(HORIZOHTAL_OPFSET +
                              HORIZONTAL_SIZE /
                              NUMBER_OF_TICK_MARKS]X_AXIS] * LCV);
                setcolor(SCREEN.AXIS_COLOR);
                setlinestyle(solidln, 0, normwidth);
                line
                  (
                   TEMP, round(X_AXIS_OFFSET),
                   TEMP, round(X_AXIS_OFFSET + TICK_LENGTH)
                  );
                setcolor(SCREEN.GRID_COLOR);
                setlinestyle(dottedln, 0, normwidth);
                line
                  (
                   TEMP, round(VERTICAL_OFFSET),
                   TEMP, round(X_AXIS_OFFSET)
                  );
                TICK_LABEL := RANGE[X_AXIS] /
                              NUMBER_OF_TICK_MARKS]X_AXIS] * LCV +
                              LOW_LABEL[X_AXIS];
                if TICK_LABEL = round(TICK_LABEL) then
                  begin
                    setcolor(SCREEN.AXIS_COLOR);
                    outtextxy
                      (
                       TEMP,
                       round(X_AXIS_OFFSET + TICK_LENGTH - 1),
                       INT_STR(round(TICK_LABEL))
                      );
                  end; (* if...then *)
              end; (* for...to...do *)

        (* Add Y Tick Marks *)
        settextjustify(righttext, centertext);
        with GRAPH_SCALES do
          if NUMBER_OF_TICK_MARKS[Y_AXIS] <> 0 then
            for LCV := 0 to NUMBER_OF_TICK_MARKS[Y_AXIS] do
              begin
                TEMP  := round(VERTICAL_OFFSET +
                               VERTICAL_SIZE /
                               NUMBER_OF_TICK_MARKS]Y_AXIS] * LCV);
                setcolor(SCREEN.AXIS_COLOR);
                setlinestyle(solidln, 0, normwidth);
                line
                  (
                   round(Y_AXIS_OFFSET - TICK_LENGTH), TEMP,
                   round(Y_AXIS_OFFSET), TEMP
                  );
                setcolor(SCREEN.GRID_COLOR);
                setlinestyle(dottedln, 0, normwidth);
                line
                  (
                   round(Y_AXIS_OFFSET), TEMP,
                   MAX_X - EDGE_OFFSET - GRAPH_OFFSET, TEMP
```

```
            );
          TICK_LABEL := RANGE[Y_AXIS] /
                        NUMBER_OF_TICK_MARKS(Y_AXIS) *
                        )NUMBER_OF_TICK_MARKS[Y_AXIS] - LCV) +
                        LOW_LABEL[Y_AXIS];
          if TICK_LABEL = round(TICK_LABEL) then
            begin
              setcolor(SCREEN.AXIS_COLOR);
              outtextxy
                (
                 round(Y_AXIS_OFFSET - TICK_LENGTH),
                 TEMP - 1,
                 INT_STR(round(TICK_LABEL))
                );
            end; (* if...then *)
        end; (* with...to...do *)

    )* Draw X Axis *)
    setcolor(SCREEN.AXIS_COLOR);
    setlinestyle(solidln, 0, normwidth);
    line
      )
       round(HORIZONTAL_OFFSET),
       round(X_AXIS_OFFSET),
       round(HORIZONTAL_OFFSET + HORIZONTAL_SIZE),
       round(X_AXIS_OFFSET)
      );
    (* Draw Y Axis *)
    setcolor(SCREEN.AXIS_COLOR);
    setlinestyle(solidln, 0, normwidth);
    line
      (
       round)Y_AXIS_OFFSET),
       round)VERTICAL_OFFSET),
       round)Y_AXIS_OFFSET),
       round(VERTICAL_OFFSET + VERTICAL_SIZE)
      );

    with GRAPH_SCALES do
      begin
        )* Label Axis *)
        setcolor(SCREEN.AXIS_LABEL_COLOR);
        setusercharsize)1, 2, 1, 2];
        settextjustify(righttext, centertext);
        outtextxy
          (
           round(HORIZONTAL_OFFSET + HORIZONTAL_SIZE),
           round(X_AXIS_OFFSET      + round(0.75 * textheight('M'))),
           LABEL_NAME[X_AXIS]
          );
        settextjustify(lefttext, centertext);
        outtextxy
          (
           round(Y_AXIS_OFFSET    - textwidth('M')),
           round(VERTICAL_OFFSET - round(0.75 * textheight('M'))),
           LABEL_NAME[Y_AXIS]
          );
      end; (* with...do *)
  end; (* with...do *)
  setlinestyle(solidln, 0, normwidth);
end; (* DRAW_GRAPH_AXIS *)

procedure DRAW_BOX(UL_X, UL_Y, LR_X, LR_Y : integer; COLOR, LINE_WIDTH:word);
(*****************************************************************************)
(*   Draws a colored, unfilled, solid line rectangle at the specified *)
(* coordinates.                                                        *)
(*****************************************************************************)

  begin (* DRAW_BOX *)
    setlinestyle(solidln, 0, LINE_WIDTH);
    setcolor(COLOR);
    rectangle(UL_X, UL_Y, LR_X, LR_Y);
  end; )* DRAW_BOX *)

begin (* REDRAW_GRAPH *)
  cleardevice;
  DRAW_BOX(0, 0, getmaxx, getmaxy, SCREEN.BACKGROUND_COLOR, normwidth);
  setfillstyle(solidfill, SCREEN.BACKGROUND_COLOR);
  floodfill(1, 1, SCREEN.BACKGROUND_COLOR);
  MAX_X := getmaxx;
  MAX_Y := getmaxy;

  DRAW_BOX
    (
     0, 0,
     MAX_X, MAX_Y,
     SCREEN.BORDER_COLOR,
     normwidth
    );
  MAX_Y := MAX_Y - MESSAGE_REGION;
  DRAW_BOX
    (
     0, MAX_Y,
     MAX_X, MAX_Y + MESSAGE_REGION,
     SCREEN.BORDER_COLOR,
     normwidth
    );
  settextjustify(lefttext, centertext);
  case DISPLAY_MODE of
    VGA   : begin
              setusercharsize(2, 5, 2, 5);
              settextstyle(SIMPLEX_FONT, horizdir, 0);
              STATUS_POS := MAX_X div 4;
            end; (* VGA *)
    SVGA  : begin
              setusercharsize(1, 2, 1, 2);
              settextstyle(SIMPLEX_FONT, horizdir, 0);
              STATUS_POS := MAX_X div 4;
            end; (* SVGA *)
    VESA,
    SVESA : begin
              settextstyle(SIMPLEX_FONT, horizdir, 2);
              STATUS_POS := MAX_X div 4;
            end; (* VESA)SVESA *)
  end; (* case...of *)
  HIGHLIGHTED_TEXT('Evolve');
  HIGHLIGHTED_TEXT('Pause');
  HIGHLIGHTED_TEXT('Grab image');
  HIGHLIGHTED_TEXT('Reverse colors');
  HIGHLIGHTED_TEXT('Make report');
  HIGHLIGHTED_TEXT('Quit');

  (* Always display two graphs *)
  CREATE_GRAPH(FORCE_LL,    TOP_THIRD, 1, USER_GRAPH[1].SCALES);
  CREATE_GRAPH(FORCE_LL, MIDDLE_THIRD, 2, USER_GRAPH[2].SCALES);
  CREATE_GRAPH)FORCE_LL, BOTTOM_THIRD, 3, USER_GRAPH)3).SCALES);

  case NUMBER_OF_GRAPHS of
    0 : begin
          setcolor(SCREEN.MAIN_TITLE_COLOR);
          case DISPLAY_MODE of
            VGA,
            SVGA  : settextstyle(TRIPLEX_FONT, horizdir, 4);
            VESA,
            SVESA : settextstyle(TRIPLEX_FONT, horizdir, 5);
          end; (* case...of *)
          settextjustify(centertext, centertext);
          moveto
            (
             MAX_X div 2,
             MAX_Y div 2 + 0*textheight('H')
            );
          outtext('Evolutution of Solutions to');
          moverel(0, round(1.5 * textheight('H')));
          outtext('Real-Time Problems');
          case DISPLAY_MODE of
            VGA,
            SVGA    settextstyle(TRIPLEX_FONT, horizdir, 3);
            VESA,
            SVESA : settextstyle(TRIPLEX_FONT, horizdir, 4);
          end; (* case...of *)
          moverel(0, 4 * textheight('H'));
          outtext('Greg P. Semeraro');
          moverel(0, textheight('H'));
          case DISPLAY_MODE of
            VGA,
            SVGA    settextstyle(SIMPLEX_FONT, horizdir, 2);
            VESA,
            SVESA : settextstyle(SIMPLEX_FONT, horizdir, 3);
          end; (* case...of *)
          moverel) 1 * textwidth('H'), 0);
          outtext('Copyright 1997');
          moverel(-8 * textwidth('H'), 0);
          DRAW_COPYRIGNT;
          DRAW_RIT_LOGO(MAX_X div 2, (MAX_Y div 4));
        end; (* No Graphs *)
    1 : begin
          DRAW_GRAPH_AXIS(1);
        end; (* One, full-screen graph *)
    2   begin
          DRAW_GRAPH_AXIS(1);
          DRAW_GRAPH_AXIS(2);
        end; (* Two graphs *)
    3   begin
          DRAW_GRAPH_AXIS(1);
          DRAW_GRAPH_AXIS(2);
          DRAW_GRAPH_AXIS(3);
        end; (* Three graphs *)
  end; (* case...of *)
  settextstyle(defaultfont, horizdir, 1);
  UPDATE_STATUS_MESSAGES(FORCE_UPDATE);
end; (* REDRAW_GRAPH *)

procedure UPDATE_GRAPHS(NUMBER_OF_GRAPHS : integer);
(*****************************************************************************)
(*   This procedures updates then graphs on the screen by getting the data *)
(* from the Sound Blaster DMA buffer and then drawing the graph.           *)
(*****************************************************************************)

  const (*********************************************************************)
        (*   Flag that indicates which half of the DMA buffer contains the *)
        (* present data samples.                                             *)
        (*********************************************************************)
        FRONT_HALF : boolean = false;

        (*********************************************************************)
        (*   Static data to allow previous values to be erased from the screen*)
        (*********************************************************************)
        PREVIOUS_GENERATION                        : longint = 0;
        PREVIOUS_STD_DEV                           : real    = 0.0;
        PREVIOUS_DIVERSITY_STD_DEV                 : real    = 0.0;
        PREVIOUS_BEST_INDIVIDUAL_FITNESS           : real    = 0.0;
        PREVIOUS_ALL_TIME_BEST_INDIVIDUAL_FITNESS : real    = 0.0;

  var   (*********************************************************************)
        (*   Index into the data values arrays.                              *)
        (*********************************************************************)
        X_LCV, LAST_ELEMENT : integer;

        (*********************************************************************)
        (*   Keeps track of the number of DMA interrupts that occurred since *)
        (* the last one was serviced.                                        *)
        (*********************************************************************)
        NUMBER_OF_INTERRUPTS : integer;

        (*********************************************************************)
        (*   Loop Control Variable to update all graphs being displayed.     *)
        (*********************************************************************)
        PLOT_NUMBER : integer;

  procedure DRAW_LINE_GRAPH(GRAPH_NUMBER : integer);
  (***************************************************************************)
  (*   Draws an entire line graph using small line segments. If previous line *)
  (* is erased if necessary (depending on the graph type selected).          *)
  (***************************************************************************)

    var   (*******************************************************************)
          (*   Used to index into the data value array.                      *)
          (*******************************************************************)
          X : integer;

          )*******************************************************************)
          (*   Used to hold the X-Axis value (converts integer to real).     *)
          (*******************************************************************)
          ACTUAL_X_VALUE   real;

          )*******************************************************************)
          (*   Used to calculate the position of the Y-Axis peak/average.    *)
          (*******************************************************************)
          PEAK_VALUE, AVERAGE_VALUE : real; PEAK_DETECTED : boolean;

          (*******************************************************************)
          (*   Used to hold the X-Axis position of the Y-Axis peak.          *)
          (*******************************************************************)
          PEAK : real;

          (*******************************************************************)
          (*   Used to swap the present and previous lines.                  *)
          (*******************************************************************)
          TEMP_PTR : pointer;

          (*******************************************************************)
          (*   Used to calculate the average value of each sweep of the graph. *)
          (*******************************************************************)
          NUMBER_IN_AVERAGE : integer;

          (*******************************************************************)
          (*   Used to hold the string value of the 'real' peak-to-average.  *)
          )*******************************************************************)
          TEMP_STR : string[20];

          (*******************************************************************)
          (*   Markers for the first and last X values that fit on the screen *)
          (* (needed when the X-AXis limits are changed from the defaults).  *)
          (*******************************************************************)
          FIRST_X, LAST_X : integer;
          X_VALUE_OFF_SCREEN : boolean;

    procedure PUT_X_BLOCK(COLOR : word; X_POS : real; GRAPH_NUMBER : integer);
    (*************************************************************************)
    (*   Draws a '■' along the X-Axis corresponding to the X position.       *)
    (*************************************************************************)

      var (*****************************************************************)
          (*   Graphic screen position of X-Axis.                          *)
          (*****************************************************************)
          Y_POS   real;
```

```
  begin (* PUT_X_BLOCK *)
    with PLOT(GRAPH_NUMBER) do
      with GRAPH_SCALES do
        begin
          Y_POS := VERTICAL_OFFSET + VERTICAL_SIZE - 1 - POINT_SIZE;
          setfillstyle(solidfill, COLOR);
          bar(round(X_POS - POINT_SIZE), round(Y_POS - POINT_SIZE),
              round(X_POS + POINT_SIZE), round(Y_POS + POINT_SIZE));
          setcolor(COLOR);
          settextjustify(centertext, centertext);
          outtextxy
            (
             round(X_POS),
             round(Y_POS - textheight('H')),
             REAL_STR(X_VALUE(round(X_POS), GRAPH_NUMBER), 1, 1)
            );
        end; (* with...do *)
  end; (* PUT_X_BLOCK *)

procedure PUT_Y_BLOCK(COLOR : word; Y_POS : real; GRAPH_NUMBER : integer);
(*************************************************************************)
(*   Draws a '■' along the Y-Axis corresponding to the Y position.       *)
(*************************************************************************)

  var (*******************************************************************)
      (*   Graphic screen position of Y-Axis.                            *)
      (*******************************************************************)
      X_POS : real;

  begin (* PUT_Y_BLOCK *)
    with PLOT(GRAPH_NUMBER) do
      with GRAPH_SCALES do
        begin
          X_POS := HORIZONTAL_OFFSET + 1 + POINT_SIZE;
          setfillstyle(solidfill, COLOR);
          bar(round(X_POS - POINT_SIZE), round(Y_POS - POINT_SIZE),
              round(X_POS + POINT_SIZE), round(Y_POS + POINT_SIZE));
          setcolor(COLOR);
          settextjustify(lefttext, centertext);
          outtextxy
            (
             round(X_POS + textwidth('H')),
             round(Y_POS),
             REAL_STR(Y_VALUE(round(Y_POS), GRAPH_NUMBER), 1, 1)
            );
        end; (* with...do *)
  end; (* PUT_Y_BLOCK *)

procedure GRAPH_BAR(X1, Y1, X2, Y2 : integer);
(*************************************************************************)
(*   Draws a graph bar with diagonal hash marks in it.                   *)
(*************************************************************************)

  const (*****************************************************************)
        (*   Distance between diagonal hash marks.                       *)
        (*****************************************************************)
        OFFSET = 7;

  var   (*****************************************************************)
        (*   Position of diagonal line within bar.                       *)
        (*****************************************************************)
        DIAGONAL : integer;

  begin (* GRAPH_BAR *)
    if (X1 <> X2) and (Y1 <> Y2) then
      begin
        rectangle(X1, Y1, X2, Y2);
        DIAGONAL := Y1;
        Y1       := Y1 + OFFSET;
        while (Y1 < Y2) do
          begin
            line(X1, Y1, X2, DIAGONAL);
            DIAGONAL := DIAGONAL + OFFSET;
            Y1       := Y1 + OFFSET;
          end; (* if...then *)
      end; (* if...then *)
  end; (* GRAPH_BAR *)

begin (* DRAW_LINE_GRAPH *)
  setwritemode(xorput);

  with PLOT(GRAPH_NUMBER) do
    with GRAPH_SCALES do
      begin
        X_VALUE_OFF_SCREEN := false;
        FIRST_X := -1;
        X := VALUES.FIRST;
        repeat
          with GRAPH_POINTS^(X) do
            begin
              ACTUAL_X_VALUE := X;
              POSITION(X_AXIS) := X_POSITION(ACTUAL_X_VALUE, GRAPH_NUMBER);

              (* Check for X value beyond graph boundaries *)
              if (ACTUAL_X_VALUE >= LOW_LABEL(X_AXIS)) and (FIRST_X = -1)
                then
                  FIRST_X := X - 1;
              if (ACTUAL_X_VALUE >= HIGH_LABEL(X_AXIS)) or
                 (X = VALUES.LAST) then
                begin
                  LAST_X := X;
                  X_VALUE_OFF_SCREEN := true;
                end; (* if...then *)

              POSITION(Y_AXIS) := Y_POSITION(VALUES.DATA(X), GRAPH_NUMBER);
            end; (* with...do *)
          inc(X);
        until X_VALUE_OFF_SCREEN;

        AVERAGE_VALUE := 0.0;
        PEAK_DETECTED := false;
        PEAK_VALUE := maxint;
        setcolor(GRAPH_COLOR xor SCREEN.BACKGROUND_COLOR);
        for X := FIRST_X to (LAST_X - 1) do
          with GRAPH_POINTS^(X) do
            begin
              AVERAGE_VALUE := AVERAGE_VALUE + POSITION(Y_AXIS);
              if POSITION(Y_AXIS) < PEAK_VALUE then
                begin
                  PEAK          := POSITION(X_AXIS);
                  PEAK_VALUE    := POSITION(Y_AXIS);
                  PEAK_DETECTED := true;
                end; (* if...then *)
              if GRAPH_TYPE = LINE_GRAPH
                then
                  begin
                    line
                      (
                       round(PREVIOUS_GRAPH_POINTS^(X    ).
                                POSITION(X_AXIS) + 0.5),
                       round(PREVIOUS_GRAPH_POINTS^(X    ).
                                POSITION(Y_AXIS) + 0.5),
                       round(PREVIOUS_GRAPH_POINTS^(X + 1).
                                POSITION(X_AXIS) - 0.5),
                       round(PREVIOUS_GRAPH_POINTS^(X + 1).
                                POSITION(Y_AXIS) - 0.5)
                      );
                    line
                      (
                       round(GRAPH_POINTS^(X    ).POSITION(X_AXIS) + 0.5),
                       round(GRAPH_POINTS^(X    ).POSITION(Y_AXIS) + 0.5),
                       round(GRAPH_POINTS^(X + 1).POSITION(X_AXIS) - 0.5),
                       round(GRAPH_POINTS^(X + 1).POSITION(Y_AXIS) - 0.5)
                      );
                  end (* if...then *)
                else
                  begin
                    GRAPH_BAR
                      (
                       round(PREVIOUS_GRAPH_POINTS^(X    )
                                POSITION(X_AXIS) + 0.5),
                       round(PREVIOUS_GRAPH_POINTS^(X    )
                                POSITION(Y_AXIS) + 1),
                       round(PREVIOUS_GRAPH_POINTS^(X + 1).
                                POSITION(X_AXIS) - 0.5),
                       round(X_AXIS_OFFSET) - 1
                      );
                    GRAPH_BAR
                      (
                       round(GRAPH_POINTS^(X    ).POSITION(X_AXIS) + 0.5),
                       round(GRAPH_POINTS^(X    ).POSITION(Y_AXIS) + 1),
                       round(GRAPH_POINTS^(X + 1).POSITION(X_AXIS) - 0.5),
                       round(X_AXIS_OFFSET) - 1
                      );
                  end; (* if...then...else *)
            end; (* for...to...do *)

        (* Draw last line segment ( bar *)
        if PLOT(GRAPH_NUMBER).GRAPH_SCALES.GRAPH_TYPE = BAR_GRAPH then
          begin
            GRAPH_BAR
              (
               round(PREVIOUS_GRAPH_POINTS^(LAST_X).POSITION(X_AXIS) + 0.5),
               round(PREVIOUS_GRAPH_POINTS^(LAST_X).POSITION(Y_AXIS) + 1),
               round(PREVIOUS_GRAPH_POINTS^(LAST_X).POSITION(X_AXIS) +
                     (
                      PREVIOUS_GRAPH_POINTS^(LAST_X).POSITION(X_AXIS) -
                      PREVIOUS_GRAPH_POINTS^(LAST_X-1(.POSITION(X_AXIS)
                     )
                    ),
               round(X_AXIS_OFFSET) - 1
              );
            GRAPH_BAR
              (
               round(GRAPH_POINTS^(LAST_X).POSITION(X_AXIS) + 0.5),
               round(GRAPH_POINTS^(LAST_X).POSITION(Y_AXIS) + 1),
               round(GRAPH_POINTS^(LAST_X).POSITION(X_AXIS) +
                     (
                      GRAPH_POINTS^(LAST_X).POSITION(X_AXIS) -
                      GRAPH_POINTS^(LAST_X-1).POSITION(X_AXIS)
                     )
                    ),
               round(X_AXIS_OFFSET) - 1
              );
          end; (* if...then...else *)
        with GRAPH_POINTS^(LAST_X) do
          if POSITION(Y_AXIS) < PEAK_VALUE then
            begin
              PEAK          := POSITION(X_AXIS);
              PEAK_VALUE    := POSITION(Y_AXIS);
              PEAK_DETECTED := true;
            end; (* if...then *)
        NUMBER_IN_AVERAGE := LAST_X - FIRST_X - 1;

        TEMP_PTR := GRAPH_POINTS;
        GRAPH_POINTS := PREVIOUS_GRAPH_POINTS;
        PREVIOUS_GRAPH_POINTS := TEMP_PTR;

        settextjustify(lefttext, centertext);

        if AVERAGE_DETECTION and (NUMBER_IN_AVERAGE <> 0) then
          begin
            AVERAGE_VALUE := AVERAGE_VALUE / NUMBER_IN_AVERAGE;
            PUT_Y_BLOCK(SCREEN.BACKGROUND_COLOR, PREVIOUS_AVERAGE_VALUE,
                        GRAPH_NUMBER);
            PUT_Y_BLOCK(SCREEN.AXIS_COLOR, AVERAGE_VALUE, GRAPH_NUMBER);
          end; (* if...then *)

        if PEAK_DETECTION(X_AXIS) and PEAK_DETECTED then
          begin
            PUT_X_BLOCK(SCREEN.BACKGROUND_COLOR, PREVIOUS_PEAK,
                        GRAPH_NUMBER);
            PUT_X_BLOCK(SCREEN.AXIS_COLOR, PEAK, GRAPH_NUMBER);
          end; (* if...then *)

        if PEAK_DETECTION(Y_AXIS) and PEAK_DETECTED then
          begin
            PUT_Y_BLOCK(SCREEN.BACKGROUND_COLOR, PREVIOUS_PEAK_VALUE,
                        GRAPH_NUMBER);
            PUT_Y_BLOCK(SCREEN.AXIS_COLOR, PEAK_VALUE, GRAPH_NUMBER);
          end; (* if...then *)

        PREVIOUS_PEAK          := PEAK;
        PREVIOUS_PEAK_VALUE    := PEAK_VALUE;
        PREVIOUS_AVERAGE_VALUE := AVERAGE_VALUE;
      end; (* with...do *)

  setwritemode(normalput);
end; (* DRAW_LINE_GRAPH *)

begin (* UPDATE_GRAPHS *)
  settextjustify(lefttext, centertext);
  (* Update Generation Counter *)
  if GENERATION <> PREVIOUS_GENERATION then
    begin
      setcolor(SCREEN.BACKGROUND_COLOR);
      outtextxy
        (
         2 * EDGE_OFFSET + textwidth('h') * 11,
         MAX_Y + 1 * MESSAGE_REGION div 3,
         INT_STR(PREVIOUS_GENERATION)
        );
      setcolor(SCREEN.STATUS_COLOR);
      PREVIOUS_GENERATION := GENERATION;
      outtextxy
        (
         2 * EDGE_OFFSET + textwidth('h') * 11,
         MAX_Y + 1 * MESSAGE_REGION div 3,
         INT_STR(PREVIOUS_GENERATION)
        );

      if NUMBER_OF_GRAPHS <> 0 then
        begin
          settextjustify(righttext, centertext);
          (* Update Statistics *)
          if LONG_TERM.STANDARD_DEVIATION <> PREVIOUS_STD_DEV then
            begin
              setcolor(SCREEN.BACKGROUND_COLOR);
              outtextxy
                (
                 MAX_X - 2 * EDGE_OFFSET,
                 round(PLOT(2).VERTICAL_OFFSET) - 2 * textheight('H'),
                 'Std. Dev.=' + REAL_STR(PREVIOUS_STD_DEV, 2, 2)
                );
              setcolor(SCREEN.AXIS_COLOR);
              PREVIOUS_STD_DEV := LONG_TERM.STANDARD_DEVIATION;
              outtextxy
                (
```

```
            MAX_X - 2 * EDGE_OFFSET,
            round(PLOT(2).VERTICAL_OFFSET) - 2 * textheight('H'),
            'Std. Dev.=' + REAL_STR(PREVIOUS_STD_DEV, 2, 2)
           );
        end; (* if...then *)
      if GENETIC_ALGORITHM_FITNESS_CONVERGED
        then
          setcolor(SCREEN.AXIS_COLOR)
        else
          setcolor(SCREEN.BACKGROUND_COLOR);
      outtextxy
        (
         MAX_X - 2 * EDGE_OFFSET - 3 * textwidth('H'),
         round(PLOT(2).VERTICAL_OFFSET) - 1 * textheight('H'),
         'Converged'
         );
      if LONG_TERM_DIVERSITY.STANDARD_DEVIATION <>
         PREVIOUS_DIVERSITY_STD_DEV then
        begin
          setcolor(SCREEN.BACKGROUND_COLOR);
          outtextxy
            (
             MAX_X - 2 * EDGE_OFFSET,
             round(PLOT(3).VERTICAL_OFFSET) - 2 * textheight('H'),
             'Std. Dev.=' + REAL_STR(PREVIOUS_DIVERSITY_STD_DEV, 2, 2)
             );
          setcolor(SCREEN.AXIS_COLOR);
          PREVIOUS_DIVERSITY_STD_DEV := LONG_TERM_DIVERSITY.
                                          STANDARD_DEVIATION;
          outtextxy
            (
             MAX_X - 2 * EDGE_OFFSET,
             round(PLOT(3).VERTICAL_OFFSET) - 2 * textheight('H'),
             'Std. Dev.=' + REAL_STR(PREVIOUS_DIVERSITY_STD_DEV, 2, 2)
            );
        end; (* if...then *)
      if GENETIC_ALGORITHM_DIVERSITY_CONVERGED
        then
          setcolor(SCREEN.AXIS_COLOR)
        else
          setcolor(SCREEN.BACKGROUND_COLOR);
      outtextxy
        (
         MAX_X - 2 * EDGE_OFFSET - 3 * textwidth('H'),
         round(PLOT(3).VERTICAL_OFFSET) - 1 * textheight('H'),
         'Converged'
         );
      (* Update Most Fit Individual *)
      if PRESENT_GENERATION.BEST <> PREVIOUS_BEST_INDIVIDUAL_FITNESS then
        begin
          setcolor(SCREEN.BACKGROUND_COLOR);
          outtextxy
            (
             MAX_X - 2 * EDGE_OFFSET,
             round(PLOT(1).VERTICAL_OFFSET) - 2 * textheight('H'),
             'Present Best=' +
             REAL_STR(PREVIOUS_BEST_INDIVIDUAL_FITNESS, 2, 2)
             );
          setcolor(SCREEN.AXIS_COLOR);
          PREVIOUS_BEST_INDIVIDUAL_FITNESS := PRESENT_GENERATION.BEST;
          outtextxy
            (
             MAX_X - 2 * EDGE_OFFSET,
             round(PLOT(1).VERTICAL_OFFSET) - 2 * textheight('H'),
             'Present Best=' +
             REAL_STR(PREVIOUS_BEST_INDIVIDUAL_FITNESS, 2, 2)
            );
        end; (* if...then *)
      (* Update Most Fit Individual *)
      if LONG_TERM.BEST <> PREVIOUS_ALL_TIME_BEST_INDIVIDUAL_FITNESS then
        begin
          setcolor(SCREEN.BACKGROUND_COLOR);
          outtextxy
            (
             MAX_X - 2 * EDGE_OFFSET,
             round(PLOT(1).VERTICAL_OFFSET)   1 * textheight('H'),
             'All-Time Best=' +
             REAL_STR(PREVIOUS_ALL_TIME_BEST_INDIVIDUAL_FITNESS, 2, 2)
             );
          setcolor(SCREEN.AXIS_COLOR);
          PREVIOUS_ALL_TIME_BEST_INDIVIDUAL_FITNESS := LONG_TERM.BEST;
          outtextxy
            (
             MAX_X - 2 * EDGE_OFFSET,
             round(PLOT(1).VERTICAL_OFFSET) - 1 * textheight('H'),
             'All-Time Best=' +
             REAL_STR(PREVIOUS_ALL_TIME_BEST_INDIVIDUAL_FITNESS, 2, 2)
            );
        end; (* if...then *)
    end; (* if...then *)
  end; (* if...then *)

  if NUMBER_OF_GRAPHS <> 0 then
    begin
      with PLOT[1] do
        for X_LCV := GRAPH_SCALES.FIRST to GRAPH_SCALES.LAST do
          VALUES.DATA(X_LCV) := POPULATION(X_LCV].FITNESS;

      with PLOT(2) do
        for X_LCV := GRAPH_SCALES.FIRST to GRAPH_SCALES.LAST do
          VALUES.DATA(X_LCV) := LONG_TERM.HISTOGRAM(X_LCV);

      with PLOT(3] do
        for X_LCV := GRAPH_SCALES.FIRST to GRAPH_SCALES.LAST do
          VALUES.DATA(X_LCV] := LONG_TEBM_DIVERSITY.HISTOGRAM(X_LCV);

      DRAW_LINE_GRAPH(1);
      DRAW_LINE_GRAPH(2);
      DRAW_LINE_GRAPH(3);
    end; (* if...then *)
end; (* UPDATE_GRAPHS *)

function IMAGE_FILENAME : string;
(*************************************************************************)
(*  This routine determines the next available IMAGE_xx.BMP filename.    *)
(*************************************************************************)

  var (******************************************************************)
      (*   Used to assign a unique filename to the image file.          *)
      (******************************************************************)
      IMAGE_NUMBER : integer;

      (******************************************************************)
      (*   Result of I/O operation, used to find first available filename. *)
      (******************************************************************)
      RESULT : integer;

      (******************************************************************)
      (*  Filename of file to get image.                                *)
      (******************************************************************)
      FILENAME : string(sizeof(namestr) + sizeof(dirstr) + 1);

      (******************************************************************)
      (*  File used to find first available file.                       *)
      (******************************************************************)
      IMAGE : file;

  begin (* IMAGE_FILENAME *)
    IMAGE_NUMBER := 0;
    repeat
      FILENAME := 'IMAGE_';
      if IMAGE_NUMBER < 10 then
        FILENAME := FILENAME + '0';
      FILENAME := FILENAME + INT_STR(IMAGE_NUMBER) + '.BMP';
      inc(IMAGE_NUMBER);
      assign(IMAGE, IMAGE_DIRECTORY + FILENAME);
      (*$I-*) reset(IMAGE); (*$I+*)
      RESULT := ioresult;
      if RESULT = 0 then
        close(IMAGE);
    until RESULT = 2;
    IMAGE_FILENAME := FILENAME;
  end; (* IMAGE_FILENAME *)

procedure RAMDLE_KEYBOARD_EVENT;
(*************************************************************************)
(*   This routine processes all user keyboard input by performing the action *)
(* requested (if possible).                                              *)
(*************************************************************************)

  var (******************************************************************)
      (*   Character to hold user input command.                        *)
      (******************************************************************)
      USER_INPUT : char;

  begin (* HANDLE_KEYBOARD_EVENT *)
    USER_INPUT := upcase(readkey);

    case USER_INPUT of
      'E' : begin
              if NUMBER_OF_GRAPRS_TO_DISPLAY = 0
                then
                  NUMBER_OF_GRAPHS_TO_DISPLAY := 3
                else
                  NUMBER_OF_GRAPHS_TO_DISPLAY := 0;
              REDRAW_GRAPH(NUMBER_OF_GRAPHS_TO_DISPLAY);
            end; (* Run Evolution *)
      'G' : begin
              UPDATE_STATUS_MESSAGES(FORCE_UPDATE);
              SAVE_IMAGE_AS_16_COLOR_BMP_FILE
                (
                 IMAGE_DIRECTORY + IMAGE_FILENAME,
                 0, 0, getmaxx, getmaxy
                );
              UPDATE_STATUS_MESSAGES(FORCE_UPDATE);
            end; (* Grab Image *)
      'Q' : USER_QUIT  := true;
      'P' : begin
              USER_PAUSED := not USER_PAUSED;
              UPDATE_STATUS_MESSAGES(FORCE_UPDATE);
            end; (* 'Pause *)
      'M' : begin
              USER_PAUSED := true;
              PRODUCE_GENETIC_ALGORITHM_REPORT;
              UPDATE_STATUS_MESSAGES(FORCE_UPDATE);
            end; (* 'Pause *)
      'R'   begin
              REVERSE_VIDEO := not REVERSE_VIDEO;
              if REVERSE_VIDEO
                then
                  SCREEN := REVERSE
                else
                  SCREEN := NORMAL;
              REDRAW_GRAPH(NUMBER_OF_GRAPHS_TO_DISPLAY);
            end; (* Reverse Video Colors *)
      else  write(BELL);
    end; (* case...of *)
  end; (* RAMDLE_KEYBOARD_EVENT *)

begin (* GENETIC_ALGORITHM_THESIS *)
  (* Install exit procedure *)
  SAVE_EXIT := exitproc;
  exitproc := @SHUTDOWN_GRAPHICS;

  (* Initialization *)
  INITIALIZE;
  INIT_GRAPHICS;

  REDRAW_GRAPH(NUMBER_OF_GRAPHS_TO_DISPLAY);
  (* Go until user wants to exit *)
  repeat
    if keypressed then HANDLE_KEYBOARD_EVENT;

    if not USER_QUIT and not USER_PAUSED and not keypressed and
       not (GENETIC_ALGORITHM_FITNESS_CONVERGED and
            GENETIC_ALGORITHM_DIVERSITY_CONVERGED) then
      begin
        if NUMBER_OF_GRAPHS_TO_DISPLAY <> 0 then
          repeat
            UPDATE_STATUS_MESSAGES(DO_NOT_FORCE_UPDATE);
          until (PROCESS_GENETIC_ALGORITHM_INDIVIDUAL = POPULATION_SIZE);
        UPDATE_GRAPHS(NUMBER_OF_GRAPHS_TO_DISPLAY);
        UPDATE_STATUS_MESSAGES(DO_NOT_FORCE_UPDATE);
      end; (* while...do *)
  until USER_QUIT or (GENERATION = maxlongint);

  if GENERATION <> 0 then
    PRODUCE_GENETIC_ALGORITHM_REPORT;
end. (* GENETIC_ALGORITHM_THESIS *)
```

# 9.3 RTOS_GA.PAS

```pascal
unit RTOS_GA;
(**********************************************************************)
(*   This unit implements a Genetic Algorithm (GA) which evolves a  solution *)
(* to a real-time operating system (RTOS) optimization problem.  This GA can *)
(* report five candidate solutions.  The optimal solution is not  guarentted *)
(* to among them but a reasonable solution will be among them.              *)
(**********************************************************************)
(*                      Compiler Options (Ver. 7.0)                         *)
(*$A+                                Word Alignment                          *)
(*$B-                                Short Circuit Boolean Evaluation        *)
(*$D+                                Debug Code Generation ON (Sort of(      *)
(*                                    Requires /V option to TPC to activate  *)
(*$L+                                Local Debug symbols ON (Sort of)        *)
(*                                    Requires /V option to TPC to activate  *)
(*$F-                                Far calls only as needed                *)
(*$I-                                I/O Checking OFF                        *)
(*$N-                                Software Emulation of 80x87             *)
(*$O-                                Overlays NOT allowed                    *)
(*$P-                                Standard 'string' parameters            *)
(*$Q-                                Overflow Checking OFF                   *)
(*$R-                                Range Checking OFF                      *)
(*$S-                                Stack Checking OFF                      *)
(*$T+                                Force Typed '@' references              *)
(*$V-                                Var-string Checking OFF                 *)
(*$X+                                Enable Extended syntax                  *)
(**********************************************************************)

interface

uses  TASKING;

const (**********************************************************************)
      (*  The following defines the number of individuals in the population. *)
      (**********************************************************************)
      POPULATION_SIZE = 200;

      (**********************************************************************)
      (*  Genetic operator probabilities; these define the speed that the GA *)
      (* converges as well as the probability of false convergance.          *)
      (**********************************************************************)
      PROBABILITY_OF_CROSSOVER = 0.30;
      PROBABILITY_OF_MUTATION  = 0.06;
      CONVERGENCE_THRESHOLD    = 5;

      (**********************************************************************)
      (*  The name of the TASKING program that is to be executed to evaluate *)
      (* the RTOS configuration.  This program  *must*  create  the  TASKING *)
      (* report file (TASKING.RPT) and a program specific report  file  with *)
      (* an extension of .RPT.                                               *)
      (**********************************************************************)
      PROGRAM_NAME = 'RTOS-APP';

      (**********************************************************************)
      (*  Each aspect of the evaluation (RTOS and application)  is  assigned *)
      (* a percentage (of course the total must be 1.0).                     *)
      (**********************************************************************)
      RTOS_EVALUATION_PERCENTAGE = 1.0;
      APPLICATION_EVALUATION_PERCENTAGE = 1.0 - RTOS_EVALUATION_PERCENTAGE;

      (**********************************************************************)
      (*  Long term statistic are computing using data  from  (up  to)  this *)
      (* many previous generations.                                          *)
      (**********************************************************************)
      FITNESS_MAX_AVERAGE_GENERATIONS   = 50;
      DIVERSITY_MAX_AVERAGE_GENERATIONS = 50;

      (**********************************************************************)
      (*  Number of 'best' individuals to maintain for reporting.            *)
      (**********************************************************************)
      NUMBER_OF_BEST_TO_REPORT = 5;

      (**********************************************************************)
      (*  Defines the bounds of the histogram, values should  be  guarenteed *)
      (* to be beyond the possible fitness  values  (this  is  verified  and *)
      (* reported any fitness is beyond this range).                         *)
      (**********************************************************************)
      MINIMUM_FITNESS = 0.0;      LOWEST_BUCKET  = round(MINIMUM_FITNESS);
      MAXIMUM_FITNESS = 100.0;    HIGHEST_BUCKET = round(MAXIMUM_FITNESS);

type  (**********************************************************************)
      (*  List of all genes that make up the genotype.                       *)
      (**********************************************************************)
      RTOS_GENES = (
        (**********************************************************************)
        (*  Tasking Model - Cooperative, Preemptive                          *)
        (**********************************************************************)
        TASKING_MODEL_GENE,
        (**********************************************************************)
        (*  Preemptive CPU Timeslice                                         *)
        (**********************************************************************)
        TARGET_TIMESLICE_GENE,
        (**********************************************************************)
        (*  Priority Inheritance Protocol - Enabled, Disabled                *)
        (**********************************************************************)
        PRIORITY_INHERITANCE_GENE,
        (**********************************************************************)
        (*  Priority Allocation Algorithm - Static, Dynamic (Rotating)       *)
        (**********************************************************************)
        PRIORITY_ALLOCATION_GENE,
        (**********************************************************************)
        (*  Initial Priority Assignment - Uniform,  Random,  Rate Monotonic, *)
        (* Deadline Monotonic, Workload Monotonic                            *)
        (**********************************************************************)
        INITIAL_PRIORITY_ASSIGNMENT_GENE);
      (**********************************************************************)

      (**********************************************************************)
      (*  Definition of the initial priority assignment gene (the other gene *)
      (* definitions come directly from the TASKING unit).                   *)
      (**********************************************************************)
      PRIORITY_ASSIGNMENT_ALGORITHMS = (
        (**********************************************************************)
        (*  All tasks are initially assigned the same priority.              *)
        (**********************************************************************)
        UNIFORM_ASSIGNMENT,
        (**********************************************************************)
        (*  All tasks are initially assigned random priorities.              *)
        (**********************************************************************)
        RAMDOM_ASSIGNMENT,
        (**********************************************************************)
        (*  Task priorities are assigned based on  execution  rates:  higher *)
        (* execution rate implies higher priority.                           *)
        (**********************************************************************)
        RATE_MONOTONIC_ASSIGNMENT,
        (**********************************************************************)
        (*  Task priorities  are  assigned  based  on  execution  deadlines: *)
        (* earlier execution deadline implies higher priority.               *)
        (**********************************************************************)
        DEADLINE_MONOTONIC_ASSIGNMENT,
        (**********************************************************************)
        (*  Task priorities are assigned  based  on  workload  level: higher *)
        (* workload implies higher priority.                                 *)
        (**********************************************************************)
        WORKLOAD_MONOTONIC_ASSIGNMENT);
      (**********************************************************************)

      (**********************************************************************)
      (*  Problem specific parameter.  This structure defines the 'genes' of *)
      (* the individuals that will make up the population.                   *)
      (**********************************************************************)
      GENOTYPE = record
        (**********************************************************************)
        (*  Cooperative or Preemptive Mulittasking.                          *)
        (**********************************************************************)
        TASKING_MODEL : TASKING_MODELS;
        (**********************************************************************)
        (*  Preemptive multitasking timeslice (microseconds).                *)
        (**********************************************************************)
        TARGET_TIMESLICE : longint;
        (**********************************************************************)
        (*  Enable/Disable Priority Inheritance.                             *)
        (**********************************************************************)
        PRIORITY_INHERITANCE_ENABLED : boolean;
        (**********************************************************************)
        (*  Static or Rotating priorities.                                   *)
        (**********************************************************************)
        PRIORITY_ALLOCATION : PRIORITY_SCHEDULING_POLICIES;
        (**********************************************************************)
        (*  Uniform, random, rate monatonic, deadline monatonic, or workload *)
        (* monatonic priority assignments.                                   *)
        (**********************************************************************)
        INITIAL_PRIORITY_ASSIGNMENT : PRIORITY_ASSIGNMENT_ALGORITHMS;
        (**********************************************************************)
        (*  Genetic Algorithm bookkeeping variables used to determine  which *)
        (* individuals are allowed to 'live' in the next generation.         *)
        (**********************************************************************)
          (*  The individual's fitness evaluation parameters.                *)
          (**********************************************************************)
          FITNESS, RELATIVE_FITNESS, CUMULATIVE_FITNESS : real;
          (**********************************************************************)
          (*  The individual's diversity parameter (there is a  1:1  mapping *)
          (* of genotype values and diversity value).                        *)
          (**********************************************************************)
          DIVERSITY : longint;
          (**********************************************************************)
          (*  Flag to indicate whether the individual  will  'live'  in  the *)
          (* next generation.                                                *)
          (**********************************************************************)
          SURVIVOR : boolean;
          (**********************************************************************)
          (*  Seed for random  number  generator  so  that  results  can  be *)
          (* reproduced.                                                     *)
          (**********************************************************************)
          RANDOM_NUMBER_SEED : longint;
      end; (* GENOTYPE *)

      (**********************************************************************)
      (*  This data structure is used to determine how well the GA is doing. *)
      (* The only criteria used is convergence to a single solution (or to a *)
      (* single 'fitness' value).  The GA is assumed to  be  done  when  the *)
      (* population has converged reguardless of the  actual  fitness  value *)
      (* achieved.                                                           *)
      (**********************************************************************)
      STATISTICS = record
        (**********************************************************************)
        (*  Basic values for the population...                              *)
        (**********************************************************************)
        BEST, MEAN, WORST : real;
        (**********************************************************************)
        (*  Statistics based on the entire population...                     *)
        (**********************************************************************)
        VARIANCE, STANDARD_DEVIATION : real;
        (**********************************************************************)
        (*  Normalized histogram buckets.                                    *)
        (**********************************************************************)
        HISTOGRAM : array [round(MINIMUM_FITNESS - 0.5) ..
                           round(MAXIMUM_FITNESS + 0.5)) of real;
      end; (* STATISTICS *)

const (**********************************************************************)
      (*  This is the current generation that has completed processing.      *)
      (**********************************************************************)
      GENERATION : longint = 0;

      (**********************************************************************)
      (*  This is the current individual that has complete processing.       *)
      (**********************************************************************)
      INDIVIDUAL : integer = POPULATION_SIZE;

      (**********************************************************************)
      (*  This is the maximum value for the genotype diversity, it  is  used *)
      (* to scale the diversity display.                                     *)
      (**********************************************************************)
      MAX_NORMALIZED_DIVERSITY : real = 0.0;

      (**********************************************************************)
      (*  Indicates the status of the GA, can be used to terminate the GA.   *)
      (**********************************************************************)
      GENETIC_ALGORITHM_FITNESS_CONVERGED   : boolean = false;
      GENETIC_ALGORITHM_DIVERSITY_CONVERGED : boolean = false;

var   (**********************************************************************)
      (*  Statistics are availaible for each generation that  is  processed, *)
      (* the data is independant from generation to generation.              *)
      (**********************************************************************)
      PRESENT_GENERATION : STATISTICS;

      (**********************************************************************)
      (*  Cumulative  statistics  are  availible   for  the   most   recent *)
      (* generations (see FITNESS_MAX_AVERAGE_GENERATIONS).                  *)
      (**********************************************************************)
      LONG_TERM : STATISTICS;

      (**********************************************************************)
      (*  Diversity statistics for the present generation.                   *)
      (**********************************************************************)
      PRESENT_DIVERSITY   STATISTICS;

      (**********************************************************************)
      (*  Cumumative diversity statistics for the  most  recent  generations *)
      (* (see DIVERSITY_MAX_AVERAGE_GENERATIONS).                            *)
      (**********************************************************************)
      LONG_TERM_DIVERSITY : STATISTICS;

      (**********************************************************************)
      (*  These are all of the genes of the population.                      *)
      (**********************************************************************)
      POPULATION : array [0..POPULATION_SIZE] of GENOTYPE;

      (**********************************************************************)
      (*  Used to verify the existance of the TASKING application program.   *)
      (**********************************************************************)
      EXECUTABLE_FILE : file;

function PROCESS_GENETIC_ALGORITHM_INDIVIDUAL : integer;
(**********************************************************************)
(*  This routine performs all processing on the individual (i.e., Selection, *)
(* Crossover, Mutation and Evaluation).  The individual that  was  processed *)
(* returned by this function.                                                *)
(**********************************************************************)
```

```
procedure PROCESS_GENETIC_ALGORITHM_GENERATION;
|**************************************************************************|
|*  This routine performs all processing on the  population  for  an  entire *|
|* generation |i.e., Selection/Crossover, Mutation and Evaluation|.          *|
|**************************************************************************|

procedure PROOUCE_GENETIC_ALGORITHM_REPORT;
|**************************************************************************|
|*  This routine produces a report on the 'best' individuals found.        *|
|**************************************************************************|

implementation

uses  dos;

var   |*******************************************************************|
      |*  List of all time best indivuals that will be rported.           *|
      |*******************************************************************|
      ALL_TIME_BEST : array |O..|NUMBER_OF_BEST_TO_REPORT-1|| of GENOTYPE;

      |*******************************************************************|
      |*  The genes are allowed to range within these bounds  so  that  each *|
      |* individual created is guarenteed to have genes that are  reasonable *|
      |* for the problem |i.e. all individuals are viable|.                  *|
      |*******************************************************************|
      UPPER_BOUND, LOWER_BOUND, RANGE : array |RTOS_GENES| of real;

const |*******************************************************************|
      |*  The index of the lowest in the list is maintained so that the list *|
      |* can be kept as a circular buffer very efficiently.                 *|
      |*******************************************************************|
      LOWEST_IN_LIST_OF_BEST_INOIVIOUALS : integer = O;

      |*******************************************************************|
      |*  Flag that indicates that at least one fitness value was outside of *|
      |* the allowable range |0.0 through 100.0|, used in report.          *|
      |*******************************************************************|
      FITNESS_OUT_OF_RANGE : boolean = false;

      |*******************************************************************|
      |*  Text string of gene names |for  INI  file  processing  and  report *|
      |* generating|.                                                       *|
      |*******************************************************************|
      GENE_NAME : array |RTOS_GENES| of string =
        |
         'TaskingModel',
         'TargetTimeslice',
         'PriorityInheritance',
         'PriorityAllocation',
         'InitialPriorityAssignment'
        |,

      |*******************************************************************|
      |*  Text strings for tasking model gene |for INI file  processing  and *|
      |* report generating|.                                                *|
      |*******************************************************************|
      TASKING_MOOEL_NAME : array |TASKING_MOOELS| of string =
        |
         'Cooperative',
         'Preemptive'
        |;

      |*******************************************************************|
      |*  Text strings for priority inheritance gene |for INI file processing*|
      |* and report generating|.                                            *|
      |*******************************************************************|
      PRIOITY_INHERITANCE_NAME   array |boolean| of string =
        |
         'Oisabled',
         'Enabled'
        |;

      |*******************************************************************|
      |*  Text strings for priority scheduling policy  gene  |for  INI  file *|
      |* processing and report generating|.                                 *|
      |*******************************************************************|
      PRIORITY_ALLOCATION_NAME :
        array |PRIORITY_SCHEOULING_POLICIES| of string =
        |
         'Static',
         'Rotating'
        |;

      |*******************************************************************|
      |*  Text strings for priority assignment algorithm gene |for INI  file *|
      |* processing and report generating|.                                 *|
      |*******************************************************************|
      INITIAL_PRIORITY_ASSIGNMENT_NAME :
        array |PRIORITY_ASSIGNMENT_ALGORITNME| of string =
        |
         'UniformAssignment',
         'RandomAssignment',
         'RateMonotonicAssignment',
         'OeadlineMonotonicAssignment',
         'WorkloadMonotonicAssignment'
        |;

function RANDOM_VALUE|LOWER, UPPER : real| : real;
|**************************************************************************|
|*  Produces a random floating point number between LOWER and UPPER.       *|
|**************************************************************************|

  begin |* RANDOM_VALUE *|
    RANDOM_VALUE := |random|maxint| / maxint| * |UPPER - LOWER| + LOWER;
  end; |* RANDOM_VALUE *|

procedure WRITE_TO_INI_FILE|var INI_FILE : text; var GENES : GENOTYPE|;
|**************************************************************************|
|*  This routine takes the individual's genotype and creates a  TASKING  INI *|
|* file that corresponds to it.  The TASKING program  will  then  use  these *|
|* parameters to evaluate the effectiveness of the RTOS for the application. *|
|**************************************************************************|

  begin |* WRITE_TO_INI_FILE *|
    writeln|INI_FILE, '|Tasking|'|;
    writeln|INI_FILE, 'Statistics=TaskStatistics'|;
    with GENES do
      begin
        RANDOM_NUMBER_SEEO := |longint|random|SFFFF|| shl 16| or
                              longint|random|$FFFF||;
        writeln|INI_FILE, GENE_NAME|TASKING_MOOEL_GENE|,              '=',
                          TASKING_MOOEL_NAME|TASKING_MOOEL||;
        writeln|INI_FILE, GENE_NAME|TARGET_TIMESLICE_GENE|,           '=',
                          TARGET_TIMESLICE|;
        writeln|INI_FILE,
             GENE_NAME|PRIORITY_INHERITANCE_GENE|,         '=',
             PRIOITY_INHERITANCE_NAME|PRIORITY_INHERITANCE_ENABLEO||;
        writeln|INI_FILE,
             GENE_NAME|PRIORITY_ALLOCATION_GENE|,          '=',
             PRIORITY_ALLOCATION_NAME|PRIORITY_ALLOCATION||;
        writeln|INI_FILE|;
        writeln|INI_FILE, '|Application|'|;
        writeln|INI_FILE,
             GENE_NAME|INITIAL_PRIORITY_ASSIGNMENT_GENE|, '=',
             INITIAL_PRIORITY_ASSIGNMENT_NAME|INITIAL_PRIORITY_ASSIGNMENT||;
        writeln|INI_FILE, 'RandomNumberSeed=', RANDOM_NUMBER_SEEO|;
      end; |* with...do *|
  end; |* WRITE_TO_INI_FILE *|
```

```
function PROCESS_GENETIC_ALGORITHM_INDIVIOUAL : integer;
|**************************************************************************|
|*  This routine performs all processing on the individual |i.e., Selection, *|
|* Crossover, Mutation and Evaluation|.  The individual that  was  processed *|
|* returned by this function.                                              *|
|**************************************************************************|

  procedure RANDOM_GENE|var GENES : GENOTYPE; GENE : RTOS_GENES|;
  |************************************************************************|
  |*  Replaces the gene of the genotype with a new, randomly selected value. *|
  |************************************************************************|

    begin |* RANDOM_GENE *|
      with GENES do
        case GENE of
          TASKING_MOOEL_GENE :
            TASKING_MOOEL :=
                 TASKING_MOOELS
                   |
                   round
                     |
                     RANDOM_VALUE
                       |
                       LOWER_BOUND|TASKING_MOOEL_GENE|,
                       UPPER_BOUND|TASKING_MOOEL_GENE|
                       |
                     |
                   |;
          TARGET_TIMESLICE_GENE :
            TARGET_TIMESLICE :=
                 round
                   |
                   RANDOM_VALUE
                     |
                     LOWER_BOUND|TARGET_TIMESLICE_GENE|,
                     UPPER_BOUND|TARGET_TIMESLICE_GENE|
                     |
                   |;
          PRIORITY_INHERITANCE_GENE :
            PRIORITY_INHERITANCE_ENABLEO :=
                 boolean
                   |
                   round
                     |
                     RANDOM_VALUE
                       |
                       LOWER_BOUND|PRIORITY_INHERITANCE_GENE|,
                       UPPER_BOUND|PRIORITY_INHERITANCE_GENE|
                       |
                     |
                   |;
          PRIORITY_ALLOCATION_GENE :
            PRIORITY_ALLOCATION :=
                 PRIORITY_SCHEOULING_POLICIES
                   |
                   round
                     |
                     RANDOM_VALUE
                       |
                       LOWER_BOUND|PRIORITY_ALLOCATION_GENE|,
                       UPPER_BOUND|PRIORITY_ALLOCATION_GENE|
                       |
                     |
                   |;
          INITIAL_PRIORITY_ASSIGNMENT_GENE :
            INITIAL_PRIORITY_ASSIGNHENT :=
                 PRIORITY_ASSIGNMENT_ALGORITHMS
                   |
                   round
                     |
                     RANDOM_VALUE
                       |
                       LOWER_BOUND|INITIAL_PRIORITY_ASSIGNMENT_GENE|,
                       UPPER_BOUNO|INITIAL_PRIORITY_ASSIGNMENT_GENE|
                       |
                     |
                   |;
      end; |* case...of *|
  end; |* RANDOM_GENE *|

procedure INITIALIZE_POPULATION;
|**************************************************************************|
|*  This routine initializes the GA by randomly initializing the genes  of *|
|* the population and initializes the long term statistics.                *|
|**************************************************************************|

  var |*******************************************************************|
      |*  Loop control variables.                                         *|
      |*******************************************************************|
      INDIVIOUAL_LCV, LCV : integer;
      GENE                : RTOS_GENES;

  begin |* INITIALIZE_POPULATION *|
    for INDIVIOUAL_LCV := 1 to POPULATION_SIZE do
      with POPULATION|INOIVIOUAL_LCV| do
        begin
          LOWER_BOUND|TASKING_MOOEL_GENE| := integer|low|TASKING_MOOEL||;
          UPPER_BOUNO|TASKING_MOOEL_GENE| := integer|high|TASKING_MOOEL||;
          RANGE|TASKING_MOOEL_GENE|       :=
               UPPER_BOUND|TASKING_MOOEL_GENE| -
               LOWER_BOUND|TASKING_MOOEL_GENE|;

          LOWER_BOUND|TARGET_TIMESLICE_GENE| := MINIMUM_TIMESLICE;
          UPPER_BOUND|TARGET_TIMESLICE_GENE| := $FFFF;
          RANGE|TARGET_TIMESLICE_GENE|       :=
               UPPER_BOUND|TARGET_TIMESLICE_GENE| -
               LOWER_BOUND|TARGET_TIMESLICE_GENE|;

          LOWER_BOUND|PRIORITY_INHERITANCE_GENE| :=
               integer|low|PRIORITY_INHERITANCE_ENABLEO||;
          UPPER_BOUND|PRIORITY_INHERITANCE_GENE| :=
               integer|high|PRIORITY_INHERITANCE_ENABLEO||;
          RANGE|PRIORITY_INHERITANCE_GENE|       :=
               UPPER_BOUND|FRIORITY_INHERITANCE_GENE| -
               LOWER_BOUND|PRIORITY_INHERITANCE_GENE|;

          LOWER_BOUND|PRIORITY_ALLOCATION_GENE| :=
               integer|low|PRIORITY_ALLOCATION||;
          UPPER_BOUNO|PRIORITY_ALLOCATION_GENE| :=
               integer|high|PRIORITY_ALLOCATION||;
          RANGE|PRIORITY_ALLOCATION_GENE|       :=
               UPPER_BOUND|PRIORITY_ALLOCATION_GENE| -
               LOWER_BOUND|PRIORITY_ALLOCATION_GENE|;

          LOWER_BOUND|INITIAL_PRIORITY_ASSIGNMENT_GENE| :=
               integer|low|INITIAL_PRIORITY_ASSIGNMENT||;
          UPPER_BOUND|INITIAL_PRIORITY_ASSIGNMENT_GENE| :=
               integer|high|INITIAL_PRIORITY_ASSIGNMENT||;
          RANGE|INITIAL_PRIORITY_ASSIGNMENT_GENE|       :=
               UPPER_BOUND|INITIAL_PRIORITY_ASSIGNMENT_GENE| -
               LOWER_BOUND|INITIAL_PRIORITY_ASSIGNHENT_GENE|;
        end; |* for...to...do *|

    |* Randomly initialize population *|
    for INDIVIOUAL_LCV := 1 to POPULATION_SIZE do
      for GENE := low|RTOS_GENES| to high|RTOS_GENES| do
        RANDOM_GENE|POPULATION|INDIVIOUAL_LCV|, GENE|;
```

```
    (* Initialize Statistics *)
    with PRESENT_GENERATION do
      begin
        BEST  := -maxlongint;
        WORET :=  maxlongint;
        MEAN  := 0.0;
        VARIANCE := 0.0;
        STANDARD_DEVIATION := 0.0;
        for LCV := LOWEST_BUCKET to HIGHEST_BUCKET do
          HISTOGRAM(LCV) := 0.0;
      end; (* with...do *)
    LONG_TERM           := PRESENT_GENERATION;
    PRESENT_DIVERSITY   := PRESENT_GENERATION;
    LONG_TERM_DIVERSITY := PRESENT_GENERATION;
    for LCV := 0 to (NUMBER_OF_BEST_TO_REPORT-1) do
      ALL_TIME_BEST(LCV).FITNESS := -maxlongint;
  end; (* INITIALIZE_POPULATION *)

procedure EVALUATE(THIS_INDIVIDUAL : integer);
(***********************************************************************)
(*  This routine evaluates the fitness for  a  single  individual  of  the *)
(* population.                                                         *)
(***********************************************************************)

  var (*******************************************************************)
      (*  Text file to contain the RTOS application configuration.       *)
      (*******************************************************************)
      INI_FILE : text;

  function TASKING_REPORT : real;
  (*********************************************************************)
  (*  This routine analyzes the TASKING report file and produces a floating*)
  (* point number from 0.0 to 100.0 that corresponds to how  well  TASKING *)
  (* thinks the RTOS parameters tune TASKING to the application.        *)
  (*********************************************************************)

    const (**************************************************************)
          (*  The  TASKING  parameters  that  are  measured  are  assigned *)
          (* proportions that  are  used  to  weight  each  parameter  (of *)
          (* course the total must sum to 1.0).                         *)
          (**************************************************************)
          PERIODIC_FAULTS_PERCENTAGE = 0.85;
          BANDWIDTH_PERCENTAGE        = 1.0 - PERIOOIC_FAULTS_PERCENTAGE;

    var   (**************************************************************)
          (*  The text file that corresponds to the TASKING report file.  *)
          (**************************************************************)
          REPORT : text;

          (**************************************************************)
          (*  Text string used to parse the TASKING report file.        *)
          [**************************************************************)
          LINE : string;

          (**************************************************************)
          [*  Parameters from the TASKING report used to evaluate the RTOS *)
          (* configuration.                                             *)
          (**************************************************************)
          EANDWIDTH, PERIODIC_FAULTS : real;

          (**************************************************************)
          [*  Used to convert strings to real numbers (indicates status).  *)
          (**************************************************************)
          CODE : integer;

    begin (* TASKING_REPORT *)
      assign(REPORT, 'TASKING.RPT');
      (*$I-*) reset(REPORT); (*$I+*)
      repeat
        readln(REPORT, LINE);
      until pos('Available CPU Bandwidth', LINE) <> 0;
      delete(LINE, 1, pos('=', LINE));
      while LINE(1) = ' ' do delete(LINE, 1, 1);
      delete[LINE, length(LINE) - 1, 2);
      val)LINE, BANDWIDTH, CODE);

      PERIODIC_FAULTS := 0.0;
      readln(REPORT, LINE);
      if pos('Periodic Event Faults', LINE) <> 0 then
        begin
          delete(LINE, 1, pos('=', LINE));
          while LINE[1] = ' ' do delete(LINE, 1, 1);
          delete(LINE, length(LINE) - 1, 2);
          val(LINE, PERIODIC_FAULTS, CODE);
        end; (* if...then *)

      TASKING_REPORT := PERIODIC_FAULTS_PERCENTAGE * (100 - PERIODIC_FAULTS)
                        +
                        EANDWIDTH_PERCENTAGE * BANDWIDTH;
      close(REPORT);
    end; (* TASKING_REPORT *)

  function APPLICATION_REPORT : real;
  (*********************************************************************)
  (*  This routine analyzes the TASKING report file and produces a floating*)
  (* point number from 0.0 to 100.0 that corresponds to how  well  TASKING *)
  (* thinks the RTOS parameters tune TASKING to the application.        *)
  (*********************************************************************)

    begin (* APPLICATION_REPORT *)
      APPLICATION_REPORT := dosexitcode;
    end; (* APPLICATION_REPORT *)

  begin (* EVALUATE *)
    assign(INI_FILE, PROGRAM_NAME + '.INI');
    [*$I-*) rewrite(INI_FILE); (*$I+*)
    WRITE_TO_INI_FILE(INI_FILE, POPULATION(THIS_INDIVIDUAL));
    close(INI_FILE);

    exec(PROGRAM_NAME + '.EXE', 'No Report File');
    with POPULATION(THIS_INDIVIDUAL) do
      begin
        FITNESS := RTOS_EVALUATION_PERCENTAGE        * TASKING_REPORT +
                   APPLICATION_EVALUATION_PERCENTAGE * APPLICATION_REPORT;
        DIVERSITY := longint(TARGET_TIMESLICE and $0000FFFF) +
                     longint(TASKING_MODEL) shl 16 +
                     longint(PRIORITY_INHERITANCE_ENABLED) shl 17 +
                     longint(PRIORITY_ALLOCATION) shl 18 +
                     longint(INITIAL_PRIORITY_ASSIGNMENT) shl 19;
      end; [* with...do *)
  end; (* EVALUATE *)

procedure COMPUTE_POPULATION_STATISTICS;
(***********************************************************************)
(*  This routine computes bothe  the  present  generation  and  long  term *)
(* statistics.                                                         *)
(***********************************************************************)

  var (*******************************************************************)
      (* Loop Control Variables.                                         *)
      (*******************************************************************)
      INDIVIDUAL_LCV, LCV : integer;

      (*******************************************************************)
      [*  For computing variance and correlation.                        *)
      (*******************************************************************)
      SUM_OF_SQUARES : real;

      [*******************************************************************)
      (*  For computing correlation.                                     *)
      (*******************************************************************)
      SUM : real;

      (*******************************************************************)
      (*  Weight used to average for long term statistics.               *)
      (*******************************************************************)
      AVERAGE_WEIGHT : real;

      (*******************************************************************)
      (*  For determining where to put the fitness value (in histogram).  *)
      (*******************************************************************)
      BUCKET : integer;

begin (* COMPUTE_POPULATION_STATISTICS *)
  with PRESENT_GENERATION do
    begin
      BEST  := -maxlongint;
      WORST :=  maxlongint;
    end, (* with...do *)

  with PRESENT_GENERATION do
    begin
      MEAN := 0.0;
      for INDIVIDUAL_LCV := 1 to POPULATION_SIZE do
        with POPULATION(INDIVIDUAL_LCV) do
          begin
            MEAN := MEAN + FITNESS;
            if FITNESS > PRESENT_GENERATION.BEST then
              PRESENT_GENERATION.BEST := FITNESS;
            if FITNESS > LONG_TERM.BEST then
              begin
                LONG_TERM.BEST := FITNESS;
                ALL_TIME_BEST(LOWEST_IN_LIST_OF_BEST_INDIVIDUALS] :=
                    POPULATION)INDIVIDUAL_LCV);
                LOWEST_IN_LIST_OF_BEST_INDIVIDUALS :=
                    (LOWEST_IN_LIST_OF_BEST_INDIVIDUALS + 1) mod
                    NUMBER_OF_BEST_TO_REPORT;
              end; (* if...then *)
            if FITNESS < PRESENT_GENERATION.WORST then
              PRESENT_GENERATION.WORST := FITNESS;
            if FITNESS < LONG_TERM.WORST then
              LONG_TERM.WORST := FITNESS;
          end;
      MEAN := MEAN / POPULATION_SIZE;

      SUM_OF_SQUARES := 0.0;
      for INDIVIDUAL_LCV := 1 to POPULATION_SIZE do
        SUM_OF_SQUARES := SUM_OF_SQUARES +
                          sqr[POPULATION(INDIVIDUAL_LCV).FITNESS - MEAN);
      VARIANCE := SUM_OF_SQUARES / POPULATION_SIZE;

      STANDARD_DEVIATION := sqrt(VARIANCE);

      for LCV := LOWEST_BUCKET to HIGHEST_BUCKET do HISTOGRAM(LCV) := 0.0;
      for INDIVIDUAL_LCV := 1 to POPULATION_SIZE do
        begin
          BUCKET := round(POPULATION(INDIVIDUAL_LCV).FITNESS);
          if (BUCKET < LOWEST_BUCKET) or (HIGHEST_BUCKET < BUCKET) then
            FITNESS_OUT_OF_RAMGE := true;
          HISTOGRAM[BUCKET] := HISTOGRAM(BUCKET) + 1;
        end; (* for...to...do *)
      SUM := 0.0;
      for LCV :=LOWEST_BUCKET to HIGHEST_BUCKET do
        SUM := SUM + HISTOGRAM(LCV);
      for LCV := LOWEST_BUCKET to HIGHEST_BUCKET do
        HISTOGRAM(LCV) := HISTOGRAM(LCV) /
                          (SUM / [HIGHEST_BUCKET - LOWEST_BUCKET + 1));
    end; (* with...do *)

  (* Compute long term statistics *)
  if GENERATION > FITNESS_MAX_AVERAGE_GENERATIONS
    then
      AVERAGE_WEIGNT := 1 / FITNESS_MAX_AVERAGE_GENERATIONS
     else
      AVERAGE_WEIGHT := 1 / GENERATION;

  with LONG_TERM do
    begin
      MEAN               := MEAN * (1 - AVERAGE_WEIGHT) +
                            PRESENT_GENERATION.MEAN * AVERAGE_WEIGHT;
      VARIANCE           := VARIANCE * (1 - AVERAGE_WEIGHT) +
                            PRESENT_GENERATION.VARIANCE * AVERAGE_WEIGHT;
      STANDARD_DEVIATION := STANDARD_DEVIATION * (1 - AVERAGE_WEIGHT) +
                            PRESENT_GENERATION.STANDARD_DEVIATION *
                            AVERAGE_WEIGHT;
      for LCV := LOWEST_BUCKET to HIGHEST_BUCKET do
        HISTOGRAM(LCV)   := HISTOGRAM(LCV) * (1 - AVERAGE_WEIGHT) +
                            PRESENT_GENERATION.HISTOGRAM(LCV) *
                            AVERAGE_WEIGNT;
    end; (* with...do *)

  (* Compute Diversity Statistics *)
  with PRSSENT_DIVERSITY do
    begin
      BEST  := -maxlongint;
      WORST :=  maxlongint;
      MEAN  :=  0.0;
      for INDIVIDUAL_LCV := 1 to POPULATION_SIZE do
        with POPULATION(INDIVIDUAL_LCV) do
          MEAN := MEAN + DIVERSITY / MAX_NORMALIZED_DIVERSITY * 100;
      MEAN := MEAN / POPULATION_SIZE;

      SUM_OF_SQUARES := 0.0;
      for INDIVIDUAL_LCV := 1 to POPULATION_SIZE do
        SUM_OF_SQUARES := SUM_OF_SQUARES +
                          sqr(POPULATION(INDIVIDUAL_LCV).DIVERSITY /
                              MAX_NORMALIZED_DIVERSITY * 100 - MEAN);
      VARIANCE := SUM_OF_SQUARES / POPULATION_SIZE;

      STANDARD_DEVIATION := sqrt(VARIANCE);
    end; (* with...do *)

  (* Compute long term statistics *)
  if GENERATION > DIVERSITY_MAX_AVERAGE_GENERATIONS
    then
      AVERAGE_WEIGHT := 1 / DIVERSITY_MAX_AVERAGE_GENERATIONS
     else
      AVERAGE_WEIGHT := 1 / GENERATION;

  with LONG_TERM_DIVERSITY do
    begin
      MEAN               := MEAN * (1 - AVERAGE_WEIGHT) +
                            PRESENT_DIVEREITY.MEAN * AVERAGE_WEIGHT;
      VARIANCE           := VARIANCE * [1 - AVERAGE_WEIGHT) +
                            PRESENT_DIVERSITY.VARIANCE * AVERAGE_WEIGHT;
      STANDARD_DEVIATION := STANDARD_DEVIATION * [1 - AVERAGE_WEIGHT) +
                            PRESENT_DIVERSITY.STANDARD_DEVIATION *
                            AVERAGE_WEIGHT;
      for LCV := LOWEST_BUCKET to HIGHEST_BUCKET do HISTOGRAM(LCV) := 0.0;
      for INDIVIDUAL := 1 to POPULATION_SIZE do
        begin
          BUCKET := round(POPULATION(INDIVIDUAL).DIVERSITY /
                          MAX_NORMALIZED_DIVERSITY * 100);
          if (BUCKET < LOWEST_BUCKET) or [HIGHEST_BUCKET < BUCKET) then
            FITNESS_OUT_OF_RAMGE := true;
          HISTOGRAM[BUCKET] := HISTOGRAM(BUCKET) + 1;
        end; [* for...to...do *)
      SUM := 0.0;
```

```
      for LCV :=LOWEST_BUCKET to HIGHEST_BUCKET do
        SUM := SUM + HISTOGRAM[LCV];
      for LCV := LOWEST_BUCKET to HIGHEST_BUCKET do
        HISTOGRAM[LCV] := HISTOGRAM[LCV] /
                          (SUM / (HIGHEST_BUCKET - LOWEST_BUCKET + 1((;
    end; (* with...do *(

    GENETIC_ALGORITHM_FITNESS_CONVERGED :=
      abs[LONG_TERM.STANDARD_DEVIATION( < CONVERGENCE_THRESHOLD;

    GENETIC_ALGORITHM_DIVERSITY_CONVERGED :=
      abs[LONG_TERM_DIVERSITY.STANDARD_DEVIATION( < CONVERGENCE_THRESHOLD;
  end; (* COMPUTE_POPULATION_STATISTICS *)

procedure EVALUATE_POPULATION;
(**************************************************************************(
(* This routine evaluates the fitness for each individual in the          *(
(* population.                                                            *(
(**************************************************************************(

  var (********************************************************************(
      (* Loop Control variables.                                          *(
      (********************************************************************(
      INDIVIDUAL, LCV : integer;

  begin (* EVALUATE_POPULATION *)
    for INDIVIDUAL := 1 to POPULATION_SIZE do
      EVALUATE(INDIVIDUAL(;
    COMPUTE_POPULATION_STATISTICS;
  end; (* EVALUATE_POPULATION *(

procedure CROSSOVER_POPULATION;
(**************************************************************************(
(* Determines which survivors will be used to reproduce offspring to take *(
(* the place of all non-survivors.                                        *(
(**************************************************************************(

  var (********************************************************************(
      (* Each generation, the survivors must be counted so that selection *(
      [* for reproduction can be done using only survivors.               *(
      [********************************************************************)
      NUMBER_OF_SURVIVORS : integer;

      (********************************************************************)
      (* Loop Control Variables.                                          *)
      (********************************************************************(
      INDIVIDUAL_LCV, LCV : integer;
      GENE                : RTOS_GENES;

      (********************************************************************(
      (* Indices of the parents that will be 'crossed' to produce the     *(
      (* offspring.                                                       *(
      (********************************************************************(
      FIRST_PARENT, SECOND_PARENT : integer;

      (********************************************************************(
      (* For each parent the Nth survivor is chosen, these variables keep *(
      (* track of the index to be chosen.                                 *(
      (********************************************************************)
      FIRST_INDEX, SECOND_INDEX, COUNT   integer;

      (********************************************************************(
      (* Genes 0..CROSSOVER_POINT-1 are taken from the first parent and   *(
      (* the rest are taken from the second parent.                       *(
      (********************************************************************(
      CROSSOVER_POINT : RTOS_GENES;

  procedure NATURAL_SELECTION;
  (************************************************************************)
  (* 'Culls' the population, retaining all individuals that are 'above   *)
  (* average' and also a percentage of the other individuals based on the *(
  (* PROBABILITY_OF_CROSSOVER.                                             *(
  (************************************************************************(

    var (******************************************************************(
        [* Loop Control Variable.                                         *(
        (******************************************************************(
        INDIVIDUAL_LCV : integer;

        (******************************************************************(
        (* Total of the fitness of all individuals in the population.     *(
        (******************************************************************(
        SUM : real;

    begin (* NATURAL_SELECTION *)
      SUM := 0.0;
      for INDIVIDUAL_LCV := 1 to POPULATION_SIZE do
        SUM := SUM + POPULATION[INDIVIDUAL_LCV).FITNESS;

      for INDIVIDUAL_LCV := 1 to POPULATION_SIZE do
        with POPULATION[INDIVIDUAL_LCV) do
          RELATIVE_FITNESS := FITNESS / SUM;

      POPULATION[1).CUMULATIVE_FITNESS := POPULATION[1).RELATIVE_FITNESS;
      for INDIVIDUAL_LCV := 2 to POPULATION_SIZE do
        with POPULATION[INDIVIDUAL_LCV) do
          CUMULATIVE_FITNESS :=
            POPULATION[INDIVIDUAL_LCV-1).CUMULATIVE_FITNESS +
            RELATIVE_FITNESS;

      NUMBER_OF_SURVIVORS := 0;
      for INDIVIDUAL_LCV := 1 to POPULATION_SIZE do
        if (POPULATION[INDIVIDUAL_LCV).FITNESS > PRESENT_GENERATION.MEAN( xor
           ((random(maxint( / maxint( <= PROBABILITY_OF_CROSSOVER(
          then
            begin
              POPULATION[INDIVIDUAL_LCV).SURVIVOR := true;
              inc(NUMBER_OF_SURVIVORS(;
            end (* if...then *(
          else
            POPULATION[INDIVIDUAL_LCV).SURVIVOR := false;
    end; (* NATURAL_SELECTION *)

  begin (* CROSSOVER_POPULATION *(
    NATURAL_SELECTION;
    for INDIVIDUAL_LCV := 1 to POPULATION_SIZE do
      with POPULATION[INDIVIDUAL_LCV) do
        if not SURVIVOR then
          begin
            FIRST_INDEX  := random(NUMBER_OF_SURVIVORS( + 1;
            SECOND_INDEX := random(NUMBER_OF_SURVIVORS( + 1;

            COUNT := 0;
            FIRST_PARENT := 1;
            SECOND_PARENT := 1;
            for LCV := 1 to POPULATION_SIZE do
              if POPULATION[LCV).SURVIVOR then
                begin
                  inc(COUNT);
                  if COUNT = FIRST_INDEX  then FIRST_PARENT  := LCV;
                  if COUNT = SECOND_INDEX then SECOND_PARENT := LCV;
                end; (* if...then *)

            CROSSOVER_POINT := RTOS_GENES(random[integer(high[RTOS_GENES())));

            if CROSSOVER_POINT <> low[RTOS_GENES) then
            for GENE := low(RTOS_GENES) to pred(CROSSOVER_POINT( do
              case GENE of
                TASKING_MODEL_GENE :
                  TASKING_MODEL := POPULATION[FIRST_PARENT).TRSKING_MODEL;
                TARGET_TIMESLICE_GENE :
                  TARGET_TIMESLICE :=
                      POPULATION[FIRST_PARENT).TARGET_TIMESLICE;
                PRIORITY_INHERITANCE_GENE :
                  PRIORITY_INHERITANCE_ENABLED :=
                      POPULATION[FIRST_PARENT).PRIORITY_INHERITANCE_ENABLED;
                PRIORITY_ALLOCATION_GENE :
                  PRIORITY_ALLOCATION :=
                      POPULATION[FIRST_PARENT).PRIORITY_ALLOCATION;
                INITIAL_PRIORITY_ASSIGNMENT_GENE :
                  INITIAL_PRIORITY_ASSIGNMENT :=
                      POPULATION[FIRST_PARENT).INITIAL_PRIORITY_ASSIGNMENT;
              end; (* case...of *(
            for GENE := CROSSOVER_POINT to high(RTOS_GENES( do
              case GENE of
                TASKING_MODEL_GENE :
                  TASKING_MODEL := POPULATION[FIRST_PARENT).TASKING_MODEL;
                TARGET_TIMESLICE_GENE :
                  TARGET_TIMESLICE :=
                      POPULATION[FIRST_PARENT).TARGET_TIMESLICE;
                PRIORITY_INHERITANCE_GENE :
                  PRIORITY_INHERITANCE_ENABLED :=
                      POPULATION[FIRST_PARENT(.PRIORITY_INHERITANCE_ENABLED;
                PRIORITY_ALLOCATION_GENE :
                  PRIORITY_ALLOCATION :=
                      POPULATION[FIRST_PARENT).PRIORITY_ALLOCATION;
                INITIAL_PRIORITY_ASSIGNMENT_GENE :
                  INITIAL_PRIORITY_ASSIGNMENT :=
                      POPULATION[FIRST_PARENT).INITIAL_PRIORITY_ASSIGHMENT;
              end; (* case...of *(
          end (* if...then *(
  end; (* CROSSOVER_POPULATION *)

procedure MUTATE_POPULATION;
(**************************************************************************(
(* This routine performs the population wide mutation on individuals.     *(
(**************************************************************************(

  var (********************************************************************(
      (* Maximum number of mutations that will be performed for this      *(
      (* generation (directly derived from probability of mutation(.      *(
      (********************************************************************)
      MAX_MUTATIONS : integer;

      (********************************************************************(
      (* The individual in the population being mutated.                  *(
      (********************************************************************)
      MUTANT_INDIVIDUAL : integer;

      (********************************************************************(
      (* The gene in the individual being mutated.                        *(
      (********************************************************************(
      MUTANT_GENE : RTOS_GENES;

      (********************************************************************(
      (* Loop Control variable.                                           *(
      (********************************************************************(
      LCV : integer;

  begin (* MUTATE_POPULATION *)
    MAX_MUTATIONS := round[POPULATION_SIZE * (integer(high(RTOS_GENES)) -
                                   integer(low[RTOS_GENES)( + 1) *
                        PROBABILITY_OF_MUTATION(;

    for LCV := 0 to random(MAX_MUTATIONS( do
      begin
        MUTANT_INDIVIDUAL := random(POPULATION_SIZE) + 1;
        MUTANT_GENE       := RTOS_GENES(random[integer(high[RTOS_GENES(((;

        RANDOM_GENE(POPULATION[MUTANT_INDIVIDUAL), MUTANT_GENE);
      end; (* for...to...do *(
  end; (* MUTATE_POPULATION *(

begin (* PROCESS_GENETIC_ALGORITNH_INDIVIDUAL *(
  if INDIVIDUAL = POPULATION_SIZE then
    begin
      INDIVIDUAL := 0;
      if GENERATION = 0
        then
          INITIALIZE_POPULATION
        else
          begin
            CROSSOVER_POPULATION;
            MUTATE_POPULATION;
          end; [* if...then...else *)
    end; (* if...then *)

  inc(INDIVIDUAL);
  EVALUATE(INDIVIDUAL);

  PROCESS_GENETIC_ALGORITHM_INDIVIDUAL := INDIVIDUAL;
  if INDIVIDUAL = POPULATION_SIZE then
    begin
      inc(GENERATION(;
      COMPUTE_POPULATION_STATISTICS;
    end; (* if...then *(
end; (* PROCESS_GENETIC_ALGORITHM_INDIVIDUAL *(

procedure PROCESS_GENETIC_ALGORITHM_GENERATION;
(**************************************************************************(
(* This routine performs all processing on the population for an entire   *(
(* generation (i.e., Selection/Crossover, Mutation and Evaluation(.       *)
[**************************************************************************)

  var INDIVIDUAL_LCV : integer;

  begin [* PROCESS_GENETIC_ALGORITHM_GENERATION *(
    INDIVIDUAL_LCV := 1;
    repeat
      PROCESS_GENETIC_ALGORITHM_INDIVIDUAL;
      inc(INDIVIDUAL_LCV);
    until (INDIVIDUAL_LCV > POPULATION_SIZE(;
  end; [* PROCESS_GENETIC_ALGORITHM_GENERATION *(

procedure PRODUCE_GENETIC_ALGORITHM_REPORT;
(**************************************************************************(
(* This routine produces a report on the 'best' individuals found.        *(
(**************************************************************************(

  const (******************************************************************(
        (* General purpose special characters.                            *)
        (******************************************************************(
        CR = chr($0D);   (* Carriage Return *)
        LF = chr($0A);   (* Line Feed       *)

  var   (******************************************************************(
        (* Loop Control Variables.                                        *)
        (******************************************************************(
        LCV, COUNTER : integer;
        GENE         : RTOS_GENES;

        (******************************************************************)
        (* Text file for report.                                          *)
        (******************************************************************(
        REPORT : text;

        (******************************************************************)
        (* Temporary variable to hold the fitness of the individuals in the *)
```

```
      (* report.                                                           *)
      (*********************************************************************)
      THMP : real;

      (*********************************************************************)
      (*  Used to extract the genotype from  the  most  popular  diversity *)
      (* value within the entire population.                               *)
      (*********************************************************************)
      CHOICE : longint;

  function REPORT_FILENAME : string;
  (*************************************************************************)
  (*   This routine determines the next available GA_xx.RPT filename.     *)
  (*************************************************************************)

    const (*****************************************************************)
          (*   Used to assign a unique filename to the report file.        *)
          (*****************************************************************)
          REPORT_NUMBER . integer = 0;

    var   (*****************************************************************)
          (*   Result  of  I/O  operation,  used  to  find  first  available *)
          (* filename.                                                     *)
          (*****************************************************************)
          RESULT   integer;

          (*****************************************************************)
          (*  Filename of file to get report.                              *)
          (*****************************************************************)
          FILENAME : string(sizeof(namestr) + sizeof(dirstr) + 1);

          (*****************************************************************)
          (*  File used to find first available file.                      *)
          (*****************************************************************)
          REPORT   file;

    begin (* REPORT_FILENAME *)
      repeat
        FILENAME := 'GA_';
        FILENAME := FILENAME + chr(ord('0') + REPORT_NUMBER div 10) +
                              chr(ord('0') + REPORT_NUMBER mod 10);
        FILENAME := FILENAME + '.RPT';
        inc(REPORT_NUMBER);
        assign(REPORT, FILENAME);
        (*$I-*) reset(REPORT); (*$I+*)
        RESULT := ioresult;
        if RESULT = 0 then
          close(REPORT);
      until RESULT = 2;
      REPORT_FILENAME := FILENAME;
    end; (* REPORT_FILENAME *)

  begin (* PRODUCE_GENETIC_ALGORITHM_REPORT *)
    assign(REPORT, REPORT_FILENAME);
    (*$I-*) rewrite(REPORT); (*$I+*)
    if FITNESS_OUT_OF_RAMGE then
      writeln(REPORT, ';   Fatal Error! Fitness out of range [0..100)!');
    if GENETIC_ALGORITHM_FITNESS_CONVERGED then
      writeln(REPORT, ';    Fitness converged, ');
    if GENETIC_ALGORITHM_DIVERSITY_CONVERGED then
      writeln(REPORT, ';    Diversity converged, ');
    writeln(REPORT, ';    ', GENERATION, ' Generations');
    for COUNTER := 0 to (NUMBER_OF_BEST_TO_REPORT-1) do
      begin
        THMP := ALL_TIME_BEST[(LOWEST_IN_LIST_OF_BEST_INDIVIDUALS + COUNTER)
                              mod
                             NUMBER_OF_BEST_TO_REPORT).FITNESS;
        if THMP <> -maxlongint then
          begin
            writeln(REPORT, ';');
            writeln(REPORT, ',====================================',
                           '====================================');
            writeln(REPORT, ';');
            writeln(REPORT, '; Rank: ', NUMBER_OF_BEST_TO_REPORT - COUNTER);
            writeln(REPORT, '; Fitness: ', THMP : 3 : 3);
            WRITE_TO_INI_FILE(REPORT, ALL_TIME_BEST(
               (LOWEST_IN_LIST_OF_BEST_INDIVIDUALS + COUNTER) mod
               NUMBER_OF_BEST_TO_REPORT));
              writeln(REPORT, ';');
          end; (* if...then *)
      end; (* for...to...do *)
    close(REPORT);
  end; (* PRODUCE_GENETIC_ALGORITHM_REPORT *)

begin (* GENETIC_ALGORITHM *)
  randomize;
  MAX_NORMALIZED_DIVERSITY := $003FFFFF; (* 22 bits are used *)

  (* make sure application executable exists in this directory! *)
  assign(EXECUTABLE_FILE, PROGRAM_NAME + '.EXE');
  (*$I+*) reset(EXECUTABLE_FILE); (*$I-*)
  if ioresult <> 0 then
    begin
      writeln;
      writeln('RTOS_GA: TRSKING application does not exist, (',
                        PROGRAM_NAME, '.EXE)');
      halt(1);
    end; (* if...then *)
end. (* GENETIC_ALGORITHM *)
```

# 9.4 MATH_GA.PAS

```
unit MATH_GA;
(*************************************************************************)
(*   Genetic Algorithm (GA) specification to determine the maximum value for *)
(* a multi-dimensional function.  This specification is merely used to  test *)
(* the basic operation of the GA.                                         *)
(*************************************************************************)
(*                          Compiler Options (Ver. 7.0)                  *)
(*$A+                                   Word Alignment                    *)
(*$B-                                   Short Circuit Boolean Evaluation  *)
(*$D+                                   Debug Code Generation ON (Sort of) *)
(*                                       Requires /V option to TPC to activate *)
(*$L+                                   Local Debug symbols ON (Sort of)  *)
(*                                       Requires /V option to TPC to activate *)
(*$F-                                   Far calls only as needed          *)
(*$I-                                   I/O Checking OFF                  *)
(*$N+                                   Hardware 80x87 Used (if available) *)
(*$O-                                   Overlays NOT allowed              *)
(*$P-                                   Standard 'string' parameters      *)
(*$Q-                                   Overflow Checking OFF             *)
(*$R-                                   Range Checking OFF                *)
(*$S-                                   Stack Checking OFF                *)
(*$V-                                   Var-string Checking OFF           *)
(*ST+                                   Force Typed '@' references        *)
(*$X+                                   Enable Extended syntax            *)
(*************************************************************************)

interface

const (*******************************************************************)
      (*  The following defines the number of individuals in the population. *)
      (*******************************************************************)
      POPULATION_SIZE = 175;

      (*******************************************************************)
      (*  Genetic operator probabilities; these define the speed that the GA *)
      (* converges as well as the probability of false convergance.      *)
      (*******************************************************************)
      PROBABILITY_OF_CROSSOVER = 0.25;
      PROBABILITY_OF_MUTATION  = 0.03;
      CONVERGENCE_THRESHOLD    = 6;

      (*******************************************************************)
      (*  Long term statistic are computing using data  from  (up  to)  this *)
      (* many previous generations.                                      *)
      (*******************************************************************)
      FITNESS_MAX_AVERAGE_GENERATIONS   = 175;
      OIVERSITY_MAX_AVERAGE_GENERATIONS = 100;

      (*******************************************************************)
      (*  Number of 'best' individuals to maintain for reporting.        *)
      (*******************************************************************)
      NUMBER_OF_BEST_TO_REPORT = 5;

      (*******************************************************************)
      (*  Defines the bounds of the histogram, values should  be  guarenteed *)
      (* to be beyond the possible fitness  values  (this  is  verified  and *)
      (* reported any fitness is beyond this range).                     *)
      (*******************************************************************)
      MINIMUM_FITNESS = 0.0;      LOWEST_BUCKET  = round(MINIMUM_FITNESS);
      MAXIMUM_FITNESS = 100.0;    HIGHEST_BUCKET = round(MAXIMUM_FITNESS);

const (*******************************************************************)
      (*  Problem specific parameter.  This is the number of  dimensions  of *)
      (* the function to be searched.                                    *)
      (*******************************************************************)
      NUMBER_OF_VARIABLES = 2;

type  (*******************************************************************)
      (*  Problem specific parameter.  This structure defines the 'genes' of *)
      (* the individuals that will make up the population.               *)
      (*******************************************************************)
      GENOTYPE = record
        (*****************************************************************)
        (*  The actual problem 'genes', for this problem they  are  all  the *)
        (* same.  In general this will not be the case.                  *)
        (*****************************************************************)
        GENE : array [0..NUMBER_OF_VARIABLES-1] of real;
        (*****************************************************************)
        (*  The genes are allowed to range within these bounds so that  each *)
        (* individual created  is  guarenteed  to  produce  genes  that  are *)
        (* reasonable for the problem (i.e. all individuals are viable). *)
        (*****************************************************************)
        UPPER_BOUND, LOWER_BOUND : array [0..NUMBER_OF_VARIABLES-1] of real;
        (*****************************************************************)
        (*  Genetic Algorithm bookkeeping variables used to determine  which *)
        (* individuals are allowed to 'live' in the next generation.     *)
        (*****************************************************************)
          (*  The individual's fitness evaluation parameters.            *)
          (***************************************************************)
          FITNESS, RELATIVE_FITNESS, CUMULATIVE_FITNESS : real;
          (***************************************************************)
          (*  The individual's diversity parameter (there is a  1:1  mapping *)
          (* of genotype values and diversity value).                    *)
          (***************************************************************)
          DIVERSITY : real;
          (***************************************************************)
          (*  Flag to indicate whether the individual  will  'live'  in  the *)
          (* next generation.                                            *)
          (***************************************************************)
          SURVIVOR : boolean;
      end; (* GENOTYPE *)

      (*******************************************************************)
      (*  This data structure is used to determine how well the GA is doing. *)
      (* The only criteria used is convergence to a single solution (or to a *)
      (* single 'fitness' value).  The GA is assumed to  be  done  when  the *)
      (* population has converged reguardless of the  actual  fitness  value *)
      (* achieved.                                                       *)
      (*******************************************************************)
      STATISTICS = record
        (*****************************************************************)
        (*  Basic values for the population...                           *)
        (*****************************************************************)
        BEST, MEAN, WORST : real;
        (*****************************************************************)
        (*  Statistics based on the entire population...                 *)
        (*****************************************************************)
        VARIANCE, STANDARD_DEVIATION : real;
        (*****************************************************************)
        (*  Normalized histogram buckets.                                *)
        (*****************************************************************)
        HISTOGRAM : array [round(MINIMUM_FITNESS - 0.5) ..
                           round(MAXIMUM_FITNESS + 0.5)) of real;
      end; (* STATISTICS *)

const (*******************************************************************)
      (*  This is the current generation that has completed processing.  *)
      (*******************************************************************)
      GENERATION : longint = 0;

      (*******************************************************************)
```

```
      [*  This is the current individual that has complete processing.        *[
      [***********************************************************************[
      INDIVIDUAL : integer = POPULATION_SIZE;

      [***********************************************************************[
      [*  This is the maximum value for the genotype diversity, it  is  used *[
      [* to scale the diversity display.                                     *[
      [***********************************************************************[
      MAX_NORMALIZED_DIVERSITY : real = 0.0;

      [***********************************************************************)
      [*  Indicates the status of the GA, can be used to terminate the GA.   *[
      [***********************************************************************[
      GENETIC_ALGORITHM_FITNESS_CONVERGED : boolean = false;
      GENETIC_ALGORITHM_DIVERSITY_CONVERGED : boolean = false;

var   [***********************************************************************)
      [*  Statistics are avaialible for each generation that  is  processed, *[
      [* the data is independant from generation to generation.              *[
      [***********************************************************************[
      PRESENT_GENERATION : STATISTICS;

      [***********************************************************************[
      [*  Cumulative  statistics  are  availible   for   the   most   recent *[
      [* generations [see FITNESS_MAX_AVERAGE_GENERATIONS].                  *[
      [***********************************************************************[
      LONG_TERM : STATISTICS;

      [***********************************************************************[
      [*  Diversity statistics for the present generation.                   *[
      [***********************************************************************)
      PRESENT_DIVERSITY : STATISTICS;

      [***********************************************************************[
      [*  Cumumative diversity statistics for the  most  recent  generations *[
      [* [see DIVERSITY_MAX_AVERAGE_GENERATIONS).                            *[
      [***********************************************************************[
      LONG_TERM_DIVERSITY : STATISTICS;

      [***********************************************************************[
      [*  These are all of the genes of the population.                      *[
      [***********************************************************************[
      POPULATION : array [0..POPULATION_SIZE] of GENOTYPE;

function PROCESS_GENETIC_ALGORITHM_INDIVIDUAL : integer;
[*****************************************************************************]
[*  This routine performs all processing on the individual [i.e., Selection, *[
[* Crossover, Mutation and Evaluation[.  The individual that  was  processed *[
[* returned by this function.                                                *[
[*****************************************************************************[

procedure PROCESS_GENETIC_ALGORITHM_GENERATION;
[*****************************************************************************[
[*  This routine performs all processing on the  population  for  an  entire *[
[* generation [i.e., Selection, Crossover, Mutation and Evaluation[.         *)
[*****************************************************************************)

procedure PRODUCE_GENETIC_ALGORITHM_REPORT;
[*****************************************************************************)
[*  This routine produces a report on the 'best' individuals found.          *[
[*****************************************************************************[

implementation

uses  dos;

var   [***********************************************************************[
      [*  List of all time best indivuals that will be rported.              *[
      [***********************************************************************]
      ALL_TIME_BEST : array [0..[NUMBER_OF_BEST_TO_REPORT-1[] of GENOTYPE;

const [***********************************************************************[
      [*  The index of the lowest in the list is maintained so that the list *[
      [* can be kept as a circular buffer very efficiently.                  *[
      [***********************************************************************[
      LOWEST_IN_LIST_OF_BEST_INDIVIDUALS : integer = 0;

      [***********************************************************************[
      [*  Flag that indicates that at least one fitness value was outside of *[
      [* the allowable range [0.0 through 100.0[, used in report.            *[
      [***********************************************************************[
      FITNESS_OUT_OF_RAMGE : boolean = false;

var   [***********************************************************************)
      [*  INI file that contains the bounds for each variable.               *[
      [***********************************************************************[
      INI_FILE : text;

      [***********************************************************************[
      [*  Loop control variables.                                            *[
      [***********************************************************************[
      VARIABLE : integer;

      [***********************************************************************[
      [*  Bounds for each variable [from INI file[.                          *[
      [***********************************************************************[
      LOWER, UPPER : real;

function RANDOM_VALUE[LOWER, UPPER : real[ : real;
[*****************************************************************************[
[*  Produces a random floating point number between LOWER and UPPER.         *[
[*****************************************************************************[

  begin [* RANDOM_VALUE *]
    RAMDOM_VALUE := [random[maxint[ / maxint[ * [UPPER - LOWER[ + LOWER;
  end; [* RANDOM_VALUE *[

function PROCESS_GENETIC_ALGORITHM_INDIVIDUAL : integer;
[*****************************************************************************[
[*  This routine performs all processing on the individual [i.e., Selection, *]
[* Crossover, Mutation and Evaluation].  The individual that  was  processed *[
[* returned by this function.                                                *[
[*****************************************************************************[

  procedure INITIALIZE_POPULATION;
  [***************************************************************************[
  [*  This routine initializes the GA by determining the bounds of the  math *[
  [* problem to be solved [via MATH_GA.INI[.  It also  randomly  initializes *[
  [* the population and initializes the long term statistics.               *[
  [***************************************************************************[

    var [*********************************************************************)
        [*  Loop control variables.                                          *[
        [*********************************************************************[
        INDIVIDUAL, LCV : integer;

    begin [* INITIALIZE_POPULATION *[
      [* Randomly initialize population *[
      for INDIVIDUAL := 1 to POPULATION_SIZE do
        for VARIABLE := 0 to NUMBER_OF_VARIABLES-1 do
          with POPULATION[INDIVIDUAL] do
            GENE[VARIABLE] := RANDOM_VALUE[LOWER_BOUND[VARIABLE],
                                           UPPER_BOUND[VARIABLE][;

      [* Initialize Statistics *]
      with PRESENT_GENERATION do
        begin
          BEST  := -maxlongint;
          WORST :=  maxlongint;
          MEAN  := 0.0;
          VARIANCE := 0.0;
          STANDARD_DEVIATION := 0.0;
          for LCV := LOWEST_BUCKET to HIGHEST_BUCKET do
            HISTOGRAM[LCV] := 0.0;
        end; [* with...do *[
      LONG_TERM           := PRESENT_GENERATION;
      PRESENT_DIVERSITY   := PRESENT_GENERATION;
      LONG_TERM_DIVERSITY := PRESENT_GENERATION;
      for LCV := 0 to [NUMBER_OF_BEST_TO_REPORT-1[ do
        ALL_TIME_BEST[LCV].FITNESS := -maxlongint;
    end; [* INITIALIZE_POPULATION *[

procedure EVALUATE[THIS_INDIVIDUAL : integer[;
[*****************************************************************************)
[*  This routine evaluates the fitness for  a  single  individual  of  the *[
[* population.                                                               *[
[*****************************************************************************[

  begin [* EVALUATE *[
    with POPULATION[THIS_INDIVIDUAL] do
      begin
        FITNESS := 21.5 +
                   GENE[0] * sin[ 4 * pi * GENE[0][ +
                   GENE[1] * sin[10 * pi * GENE[1][;
      end; [* with...do *[
  end; [* EVALUATE *[

procedure COMPUTE_POPULATION_STATISTICS;
[*****************************************************************************[
[*  This routine computes  both  the  present  generation  and  long  term *[
[* statistics.                                                               *[
[*****************************************************************************[

  var [***********************************************************************[
      [* Loop Control Variables.                                             *[
      [***********************************************************************[
      INDIVIDUAL, LCV : integer;

      [***********************************************************************)
      [*  For computing variance and correlation.                            *)
      [***********************************************************************[
      SUM_OF_SQUARES : real;

      [***********************************************************************[
      [*  For computing correlation.                                         *[
      [***********************************************************************)
      SUM : real;

      [***********************************************************************[
      [*  Weight used to average for long term statistics.                   *[
      [***********************************************************************[
      AVERAGE_WEIGNT : real;

      [***********************************************************************)
      [*  For determining where to put the fitness value [in histogram).     *)
      [***********************************************************************[
      BUCKET : integer,

      [***********************************************************************[
      [* Loop Control Variable used to compute diversity  value  for  each *[
      [* genotype value [i.e., individual[.                                  *[
      [***********************************************************************[
      GENE_LCV : integer;

  begin [* COMPUTE_POPULATION_STATISTICS *[
    with PRESENT_GENERATION do
      begin
        BEST  := -maxlongint;
        WORST :=  maxlongint;
      end; [* with...do *[

    with PRESENT_GENERATION do
      begin
        MEAN := 0.0;
        for INDIVIDUAL := 1 to POPULATION_SIZE do
          with POPULATION[INDIVIDUAL] do
            begin
              MEAN := MEAN + FITNESS;
              if FITNESS > PRESENT_GENERATION.BEST then
                PRESENT_GENERATION.BEST := FITNESS;
              if FITNESS > LONG_TERM.BEST then
                begin
                  LONG_TERM.BEST := FITNESS;
                  ALL_TIME_BEST[LOWEST_IN_LIST_OF_BEST_INDIVIDUALS] :=
                      POPULATION[INDIVIDUAL];
                  LOWEST_IN_LIST_OF_BEST_INDIVIDUALS :=
                      [LOWEST_IN_LIST_OF_BEST_INDIVIDUALS + 1[ mod
                      NUMBER_OF_BEST_TO_REPORT;
                end; [* if...then *[
              if FITNESS < PRESENT_GENERATION.WORST then
                PRSSENT_GENERATION.WORST := FITNESS;
              if FITNESS < LONG_TERM.WORST then
                LONG_TERM.WORST := FITNESS;
            end;
        MEAN := MEAN / POPULATION_SIZE;

        SUM_OF_SQUARES := 0.0;
        for INDIVIDUAL := 1 to POPULATION_SIZE do
          SUM_OF_SQUARES := SUM_OF_SQUARES +
                            sqr[POPULATION[INDIVIDUAL].FITNESS - MEAN[;
        VARIANCE := SUM_OF_SQUARES / POPULATION_SIZE;

        STANDARD_DEVIATION := sqrt[VARIANCE];

        for LCV := LOWEST_BUCKET to HIGHEST_BUCKET do HISTOGRAM[LCV] := 0.0;
        for INDIVIDUAL := 1 to POPULATION_SIZE do
          begin
            BUCKET := round[POPULATION[INDIVIDUAL].FITNESS[;
            if [BUCKET < LOWEST_BUCKET[ or [HIGHEST_BUCKET < BUCKET[ then
              FITNESS_OUT_OF_RAMGE := true;
            HISTOGRAM[BUCKET] := HISTOGRAM[BUCKET] + 1;
          end; [* for...to...do *[
        SUM := 0.0;
        for LCV :=LOWEST_BUCKET to HIGHEST_BUCKET do
          SUM := SUM + HISTOGRAM[LCV];
        for LCV := LOWEST_BUCKET to HIGHEST_BUCKET do
          HISTOGRAM[LCV] := HISTOGRAM[LCV] /
                            [SUM / [HIGHEST_BUCKET - LOWEST_BUCKET + 1[[;
      end; [* with...do *)

    [* Compute long term statistics *[
    if GENERATION > FITNESS_MAX_AVERAGE_GENERATIONS
      then
        AVERAGE_WEIGHT := 1 / FITNESS_MAX_AVERAGE_GENERATIONS
      else
        AVERAGE_WEIGHT := 1 / GENERATION;

    with LONG_TERM do
      begin
        MEAN               := MEAN * [1 - AVERAGE_WEIGHT[ +
                              PRESENT_GENERATION.MEAN * AVERAGE_WEIGNT;
        VARIANCE           := VARIANCE * [1 - AVERAGE_WEIGNT[ +
                              PRESENT_GENERATION.VARIANCE * AVERAGE_WEIGNT;
        STANDARD_DEVIATION := STANDARD_DEVIATION * [1 - AVERAGE_WEIGNT] +
                              PRESENT_GENERATION.STANDARD_DEVIATION *
                              AVERAGE_WEIGNT;
        for LCV := LOWEST_BUCKET to HIGHEST_BUCKET do
          HISTOGRAM[LCV]   := HISTOGRAM[LCV] * [1 - AVERAGE_WEIGNT] +
```

```
                                PRSSENT_GENERATION.HISTOGRAM[LCV] *
                                AVERAGE_WEIGHT;
      end; (* with...do *)

    (* Compute Diversity statistics *)
    with PRSSENT_DIVERSITY do
      begin
        BEST   := -maxlongint;
        WORST  :=  maxlongint;
        MEAN   :=  0.0;
        for INDIVIDUAL := 1 to POPULATION_SIZE do
          with POPULATION[INDIVIDUAL] do
            begin
              DIVERSITY := 0.0;
              for GENE_LCV := 0 to NUMBER_OF_VARIABLES-1 do
                DIVERSITY := DIVERSITY + GENE[GENE_LCV];
              DIVERSITY := 100 * DIVERSITY / MAX_NORMALIZED_DIVERSITY;
              MEAN := MEAN + DIVERSITY;
            end;
        MEAN := MEAN / POPULATION_SIZE;

        SUM_OF_SQUARES := 0.0;
        for INDIVIDUAL := 1 to POPULATION_SIZE do
          SUM_OF_SQUARES := SUM_OF_SQUARES +
                            sqr(POPULATION[INDIVIDUAL].DIVERSITY - MEAN);
        VARIANCE := SUN_OF_SQUARES / POPULATION_SIZE;

        STANDARD_DEVIATION := sqrt(VARIANCE);
      end; (* with...do *)

    (* Compute long term statistics *)
    if GENERATION > DIVERSITY_MAX_AVERAGE_GENERATIONS
      then
        AVERAGE_WEIGHT := 1 / DIVERSITY_MAX_AVERAGE_GENERATIONS
      else
        AVERAGE_WEIGHT := 1 / GENERATION;

    with LONG_TERM_DIVERSITY do
      begin
        MEAN               := MEAN * (1 - AVERAGE_WEIGHT) +
                              PRSSENT_DIVERSITY.MEAN * AVERAGE_WEIGHT;
        VARIANCE           := VARIANCE * (1 - AVERAGE_WEIGHT) +
                              PRESENT_DIVERSITY.VARIANCE * AVERAGE_WEIGHT;
        STANDARD_DEVIATION := STANDARD_DEVIATION * (1 - AVERAGE_WEIGHT) +
                              PRSSENT_DIVERSITY.STANDARD_DEVIATION *
                              AVERAGE_WEIGHT;
        for LCV := LOWEST_BUCKET to HIGHEST_BUCKET do HISTOGRAM[LCV] := 0.0;
        for INDIVIDUAL := 1 to POPULATION_SIZE do
          begin
            BUCKET := round(POPULATION[INDIVIDUAL].DIVERSITY);
            if (BUCKET < LOWEST_BUCKET) or (HIGHEST_BUCKET < BUCKET) then
              FITNESS_OUT_OF_RAMGE := true;
            HISTOGRAM[BUCKET] := HISTOGRAM[BUCKET] + 1;
          end; (* for...to...do *)
        SUM := 0.0;
        for LCV :=LOWEST_BUCKET to HIGHEST_BUCKET do
          SUM := SUM + HISTOGRAM[LCV];
        for LCV := LOWEST_BUCKET to HIGHEST_BUCKET do
          HISTOGRAM[LCV] := HISTOGRAM[LCV] /
                            (SUM / (HIGHEST_BUCKET - LOWEST_BUCKET + 1));
      end; (* with...do *)

    GENETIC_ALGORITHM_FITNESS_CONVERGED :=
      abs(LONG_TERM.STANDARD_DEVIATION) < CONVERGENCE_THRESHOLD;

    GENETIC_ALGORITHM_DIVERSITY_CONVERGED :=
      abs(LONG_TERM_DIVERSITY.STANDARD_DEVIATION) < CONVERGENCE_THRSSHOLD;
  end; (* COMPUTE_POPULATION_STATISTICS *)

procedure EVALUATE_POPULATION;
(**************************************************************************)
(*  This routine evaluates the fitness for each individual in the         *)
(* population.                                                            *)
(**************************************************************************)

  var (*******************************************************************)
      (*  Loop Control variables.                                        *)
      (*******************************************************************)
      INDIVIDUAL, LCV : integer;

  begin (* EVALUATE_POPULATION *)
    for INDIVIDUAL := 1 to POPULATION_SIZE do
      EVALUATE(INDIVIDUAL);
    COMPUTE_POPULATION_STATISTICS;
  end; (* EVALUATE_POPULATION *)

procedure CROSSOVER_POPULATION;
(**************************************************************************)
(*  Determines which survivors will be used to reproduce offspring to take *)
(* the place of all non-survivors.                                        *)
(**************************************************************************)

  var (*******************************************************************)
      (*  Each generation, the survivors must be counted so that selection *)
      (* for reproduction can be done using only survivors.              *)
      (*******************************************************************)
      NUMBER_OF_SURVIVORS : integer;

      (*******************************************************************)
      (*  Loop Control Variables.                                        *)
      (*******************************************************************)
      INDIVIDUAL, LCV : integer;

      (*******************************************************************)
      (*  Indices of the parents that will be 'crossed' to produce the  *)
      (* offspring.                                                      *)
      (*******************************************************************)
      FIRST_PARENT, SECOND_PARENT   integer;

      (*******************************************************************)
      (*  For each parent the Nth survivor is chosen, these variables keep *)
      (* track of the index to be chosen.                                *)
      (*******************************************************************)
      FIRST_INDEX, SECOND_INDEX, COUNT : integer;

      (*******************************************************************)
      (*  Genes 0..CROSSOVER_POINT-1 are taken from the first parent and *)
      (* the rest are taken from the second parent.                      *)
      (*******************************************************************)
      CROSSOVER_POINT   integer;

  procedure NATURAL_SELECTION;
  (************************************************************************)
  (*  'Culls' the population, retaining all individuals that are 'above *)
  (* average' and also a percentage of the other individuals based on the *)
  (* PROBABILITY_OF_CROSSOVER.                                            *)
  (************************************************************************)

    var (*****************************************************************)
        (*  Loop Control Variable.                                       *)
        (*****************************************************************)
        INDIVIDUAL : integer;

        (*****************************************************************)
        (*  Total of the fitness of all individuals in the population.  *)
        (*****************************************************************)
        SUM : real;

    begin (* NATURAL_SELECTION *)
      SUM := 0.0;
      for INDIVIDUAL := 1 to POPULATION_SIZE do
        SUM := SUM + POPULATION[INDIVIDUAL].FITNESS;

      for INDIVIDUAL := 1 to POPULATION_SIZE do
        with POPULATION[INDIVIDUAL] do
          RSLATIVE_FITNESS := FITNESS / SUM;

      POPULATION[1].CUMULATIVE_FITNESS := POPULATION[1].RSLATIVE_FITNESS;
      for INDIVIDUAL := 2 to POPULATION_SIZE do
        with POPULATION[INDIVIDUAL] do
          CUMULATIVE_FITNESS := POPULATION[INDIVIDUAL-1].CUMULATIVE_FITNESS +
                                RSLATIVE_FITNESS;

      NUMBER_OF_SURVIVORS := 0;
      for INDIVIDUAL := 1 to POPULATION_SIZE do
        if (POPULATION[INDIVIDUAL].FITNESS > PRSSENT_GENERATION.MEAN) xor
           ((random(maxint) / maxint) <= PROBABILITY_OF_CROSSOVER)
          then
            begin
              POPULATION[INDIVIDUAL].SURVIVOR := true;
              inc(NUMBER_OF_SURVIVORS);
            end (* if...then *)
          else
            POPULATION[INDIVIDUAL].SURVIVOR := false;
    end; (* NATURAL_SELECTION *)

  begin (* CROSSOVER_POPULATION *)
    NATURAL_SELECTION;
    for INDIVIDUAL := 1 to POPULATION_SIZE do
      with POPULATION[INDIVIDUAL] do
        if not SURVIVOR then
          begin
            FIRST_INDEX  := random(NUMBER_OF_SURVIVORS) + 1;
            SECOND_INDEX := random(NUMBER_OF_SURVIVORS) + 1;

            COUNT := 0;
            FIRST_PARENT := 1;
            SECOND_PARENT := 1;
            for LCV := 1 to POPULATION_SIZE do
              if POPULATION[LCV].SURVIVOR then
                begin
                  inc(COUNT);
                  if COUNT = FIRST_INDEX  then FIRST_PARENT  := LCV;
                  if COUNT = SECOND_INDEX then SECOND_PARENT := LCV;
                end; (* if...then *)

            CROSSOVER_POINT := random(NUMBER_OF_VARIABLES);

            for LCV := 0 to CROSSOVER_POINT-1 do
              GENE[LCV] := POPULATION[FIRST_PARENT].GENE[LCV];
            for LCV := CROSSOVER_POINT to NUMBER_OF_VARIABLES-1 do
              GENE[LCV] := POPULATION[SECOND_PARENT].GENE[LCV];
          end (* if...then *)
  end; (* CROSSOVER_POPULATION *)

procedure MUTATE_POPULATION;
(**************************************************************************)
(*  This routine performs the population wide mutation on individuals.    *)
(**************************************************************************)

  var (*******************************************************************)
      (*  Maximum number of mutations that will be performed for this *)
      (* generation (directly derived from probability of mutation).     *)
      (*******************************************************************)
      MAX_MUTATIONS : integer;

      (*******************************************************************)
      (*  The individual in the population being mutated.               *)
      (*******************************************************************)
      MUTANT_INDIVIDUAL : integer;

      (*******************************************************************)
      (*  The gene in the individual being mutated.                      *)
      (*******************************************************************)
      MUTANT_GENE : integer;

      (*******************************************************************)
      (* Loop Control variable.                                          *)
      (*******************************************************************)
      LCV : integer;

  begin (* MUTATE_POPULATION *)
    MAX_MUTATIONS := round(POPULATION_SIZE * NUMBER_OF_VARIABLES *
                           PROBABILITY_OF_MUTATION);

    for LCV := 0 to random(MAX_MUTATIONS) do
      begin
        MUTANT_INDIVIDUAL := random(POPULATION_SIZE) + 1;
        MUTANT_GENE       := random(NUMBER_OF_VARIABLES);

        with POPULATION[MUTANT_INDIVIDUAL] do
          GENE[MUTANT_GENE] := RANDOM_VALUE
                               (
                                LOWER_BOUND[MUTANT_GENE],
                                UPPER_BOUND[MUTANT_GENE]
                               );
      end; (* for...to...do *)
  end; (* MUTATE_POPULATION *)

begin (* PROCESS_GENETIC_ALGORITHM_INDIVIDUAL *)
  if INDIVIDUAL = POPULATION_SIZE then
    begin
      INDIVIDUAL := 0;
      if GENERATION = 0
        then
          INITIALIZE_POPULATION
        else
          begin
            CROSSOVER_POPULATION;
            MUTATE_POPULATION;
          end; (* if...then...else *)
    end; (* if...then *)

  inc(INDIVIDUAL);
  EVALUATE(INDIVIDUAL);

  PROCESS_GENETIC_ALGORITHM_INDIVIDUAL := INDIVIDUAL;
  if INDIVIDUAL = POPULATION_SIZE then
    begin
      inc(GENERATION);
      COMPUTE_POPULATION_STATISTICS;
    end; (* if...then *)
end; (* PROCESS_GENETIC_ALGORITHM_INDIVIDUAL *)

procedure PROCESS_GENETIC_ALGORITHM_GENERATION;
(**************************************************************************)
(*  This routine performs all processing on the population for an entire *)
(* generation (i.e., Selection/Crossover, Mutation and Evaluation).      *)
(**************************************************************************)

  var INDIVIDUAL_LCV : integer;

  begin (* PROCESS_GENETIC_ALGORITHM_GENERATION *)
    INDIVIDUAL_LCV := 1;
    repeat
      PROCESS_GENETIC_ALGORITHM_INDIVIDUAL;
      inc(INDIVIDUAL_LCV);
    until (INDIVIDUAL_LCV > POPULATION_SIZE);
```

```
  end; (* PROCESS_GENETIC_ALGORITHM_GENERATION *)

procedure PRODUCE_GENETIC_ALGORITHM_REPORT;
(***************************************************************************)
(*  This routine produces a report on the 'best' individuals found.        *)
(***************************************************************************)

  const (*******************************************************************)
        (*   General purpose special characters.                           *)
        (*******************************************************************)
        CR = chr($0D);    (* Carriage Return *)
        LF = chr($0A);    (* Line Feed       *)

  var   (*******************************************************************)
        (*  Loop Control Variables.                                        *)
        (*******************************************************************)
        LCV, COUNTER : integer;

        (*******************************************************************)
        (*  Text file for report.                                          *)
        (*******************************************************************)
        REPORT : text;

  function REPORT_FILENAME : string;
  (*************************************************************************)
  (*  This routine determines the next available GA_xx.RPT filename.       *)
  (*************************************************************************)

    const (*****************************************************************)
          (*   Used to assign a unique filename to the report file.        *)
          (*****************************************************************)
          REPORT_NUMBER : integer = 0;

    var   (*****************************************************************)
          (*   Result  of  I/O  operation,  used  to  find  first  available *)
          (* filename.                                                     *)
          (*****************************************************************)
          RESULT : integer;

          (*****************************************************************)
          (*  Filename of file to get report.                              *)
          (*****************************************************************)
          FILENAME : string[sizeof(namestr) + sizeof(dirstr) + 1];

          (*****************************************************************)
          (*  File used to find first available file.                      *)
          (*****************************************************************)
          REPORT : file;

    begin (* REPORT_FILENAME *)
      repeat
        FILENAME := 'GA_';
        FILENAME  = FILENAME + chr(ord('0') + REPORT_NUMBER div 10) +
                               chr(ord('0') + REPORT_NUMBER mod 10);
        FILENAME := FILENAME + '.RPT';
        inc(REPORT_NUMBER);
        assign(REPORT, FILENAME);
        (*$I-*) reset(REPORT); (*$I+*)
        RESULT := ioresult;
        if RESULT = 0 then
          close(REPORT);
      until RESULT = 2;
      REPORT_FILENAME := FILENAME;
    end; (* REPORT_FILENAME *)

  begin (* PRODUCE_GENETIC_ALGORITHM_REPORT *)
    assign(REPORT, REPORT_FILENAME);
    (*$I-*) rewrite(REPORT); (*$I+*)
    if FITNESS_OUT_OF_RANGE then
      writeln(REPORT, 'Fatal Error! Fitness out of range [0..100]!');
    writeln;
    if GENETIC_ALGORITHM_FITNESS_CONVERGED then
      write(REPORT, 'Algorithm converged, ');
    writeln(REPORT, GENERATION, ' Generations');
    write(REPORT, '   Rank  Fitness');
    for LCV := 0 to NUMBER_OF_VARIABLES-1 do
      write(REPORT, '    Var[', LCV, ']');
    for COUNTER := (NUMBER_OF_BEST_TO_REPORT-1) downto 0 do
      with ALL_TIME_BEST[(LOWEST_IN_LIST_OF_BEST_INDIVIDUALS + COUNTER) mod
           NUMBER_OF_BEST_TO_REPORT] do
        if FITNESS <> -maxlongint then
          begin
            writeln(REPORT);
            write(REPORT, NUMBER_OF_BEST_TO_REPORT - COUNTER : 6,
                          FITNESS : 10 : 3);
            for LCV := 0 to NUMBER_OF_VARIABLES-1 do
              write(REPORT, GENE[LCV] : 10 : 3);
          end; (* with...do *)
    close(REPORT);
  end; (* PRODUCE_GENETIC_ALGORITHM_REPORT *)

begin (* GENETIC_ALGORITHM *)
  (* Process INI File *)
  assign(INI_FILE, 'MATH_GA.INI');
  (*$I+*) reset(INI_FILE); (*$I-*)
  if ioresult <> 0 then
    begin
      writeln;
      writeln('MATH_GA: Unable to open INI file [MATH_GA.INI]');
      halt(1);
    end; (* if...then *)
  readln(INI_FILE, randseed);
  for VARIABLE := 0 to NUMBER_OF_VARIABLES-1 do
    begin
      readln(INI_FILE, LOWER, UPPER);
      for INDIVIDUAL := 1 to POPULATION_SIZE do
        with POPULATION[INDIVIDUAL] do
          begin
            LOWER_BOUND[VARIABLE] := LOWER;
            UPPER_BOUND[VARIABLE] := UPPER;
          end; (* for...to...do *)
      MAX_NORMALIZED_DIVERSITY := MAX_NORMALIZED_DIVERSITY + UPPER;
    end; (* for...to...do *)
  close(INI_FILE);
end. (* GENETIC_ALGORITHM *)
```

# 10. Appendix C Support Software Source Code

## *10.1 DINING.PAS*

```
program DINING_PHILOSHERS(input, output(;
(******************************************************************)
(* This is a solution to the 'classic' multi-tasking problem called *(
(* 'Dining Philosophers'. This solution was taken from "Modern Operating *)
(* Systems" by Andrew Tanenbaum.                                     *)
(******************************************************************)
(*                    Compiler Options (Ver. 7.0)                   *)
(*$A+                            Word Alignment                     *)
(*$B-                            Short Circuit Boolean Evaluation   *)
(*$D+                            Debug Code Generation ON (Sort of) *)
(*                                Requires /V option to TPC to activate *)
(*$L+                            Local Debug symbols ON (Sort of)   *(
(*                                Requires /V option to TPC to activate *)
(*$F-                            Far calls only as needed           *)
(*$G-                            Generic 80x86 code only            *)
(*$I-                            I/O Checking OFF                   *)
(*$M 1024,150000,655360          Memory (Stack, Minbeap, Maxheap)   *)
(*$N-                            Software Emulation of 80x87        *)
(*$E-                            No 80x87 run-time emulation        *)
(*$O-                            Overlays NOT allowed               *)
(*$P-                            Standard 'string' parameters       *)
(*$Q-                            Overflow Checking OFF              *)
(*$R-                            Range Checking OFF                 *)
(*$S-                            Stack Checking OFF                 *)
(*$T-                            Force Typed '@' references         *)
(*$V+                            Var-string Checking ON             *)
(*$X-                            Disable Extended syntax            *)
(******************************************************************)

uses crt, TASKING;

const (******************************************************************)
      (* In order to have the program run consistantly across computers *)
      (* widely varying processing power, the philosophers must include time *)
      (* to 'delay' when eating and thinking.                           *)
      (******************************************************************)
      DELAY : TIME =
      (
       DAYS         : 0;
       HOURS        : 0;
       MINUTES      : 0;
       SECONDS      : 1;
       MILLISECONDS : 0
      );

      (******************************************************************)
      (*  General purpose special characters.                           *(
      (******************************************************************)
      CR    = chr($0D);   (* Carriage Return         *(
      LF    = chr($0A);   (* Line Feed               *)

      (******************************************************************)
      (* This is used to allow the philosophers to all eat and think for *)
      (* the same amount of time.                                       *)
      (******************************************************************(
      DELAY_COUNT = 100000;

      (******************************************************************)
      (* These are used to maintain/limit the number of philosophers that *)
      (* the user has indicated are present.                            *)
      (******************************************************************)
      NUMBER_OF_PHILOSOPHERS : integer = 20;              (* Default value *)
      MAX_NUMBER_OF_PHILOSOPHERS = MAXIMUM_NUMBER_OF_TASKS - 1;

      (******************************************************************(
      (* The size of the activity percent bar graph.                    *(
      (******************************************************************)
      BAR_SIZE = 60;

      (******************************************************************)
      (* This software event is broadcasted to all philosophers so that all *)
      (* are ready to eat at the same instant.                          *)
      (******************************************************************)
      DINNER_IS_SERVED : EVENT = UNSIGNALED;

      (******************************************************************(
      (* Controls mutually exclusive access to the 'forks' on the table. *)
      (******************************************************************)
      NUTEX : BINARY_SEMAPHORE = 1;

type  (******************************************************************)
      (* Philosophers only want to think and eat, being hungry is necessary *)
      (* before eating is possible.                                     *)
      (******************************************************************)
      ACTIVITIES = (THINKING, HUNGRY, EATING);

var   (******************************************************************)
      (* Table of philosopher activities (one for each guest).          *)
      (******************************************************************)
      ACTIVITY   array[0..(MAX_NUMBER_OF_PHILOSOPHERS-1)] of ACTIVITIES;

      (******************************************************************)
      (* Table of semaphores for philosophers to wait on if they need but *)
      (* cannot get forks.                                              *)
      (******************************************************************)
      NEED_FORKS : array[0..MAX_NUMBER_OF_PHILOSOPHERS-1] of BINARY_SEMAPHORE;

      (******************************************************************)
      (* Statistics for bar graph display of philosopher activities.    *)
      (******************************************************************(
      NUM_THINKING, NUM_EATING : integer;

      (******************************************************************)
      (* These are the task identifiers used in the application.        *)
      (******************************************************************)
      CONTROL, GUEST : TASK_IDS;

      (******************************************************************)
      (* Generic task attribute variable used to create all tasks.      *)
      (******************************************************************)
      TASK_ATTR : TASK_ATTRIBUTES;

      (******************************************************************)
      (* For command line timeslice string conversion.                  *)
      (******************************************************************)
      CODE, TEMP : integer; VALUE : longint;

      (******************************************************************)
      (* For user guest number prompt.                                  *)
      (******************************************************************(
      CH : char;

      (******************************************************************)
      (* Loop Control Variable for philosopher table initialization.    *)
      (******************************************************************)
      LCV : integer;

procedure UPDATE_DISPLAY(NUM_THINKING, NUM_HUNGRY, NUM_EATING : integer);
(******************************************************************)
(* This routine updates all of the bar graphs and averages, it is called *)
(* whenever the philosopher has changed activities.                 *)
(******************************************************************)

  const (******************************************************************)
        (* Maintains the averages of philosopher states over all updates. *)
        (******************************************************************)
        AVG_THINKING : real = 0;
        AVG_HUNGRY   : real = 0;
        AVG_EATING   : real = 0;

  procedure BAR_GRAPH(NUM : integer; var AVERAGE : real);
  (******************************************************************)
  (* This routine draws a bar graph which is proportional to the NUM *)
  (* parameter passed in. In addition the average bar position is updated *(
  (* to reflect the new bar graph.                                   *)
  (******************************************************************)

    const (******************************************************************(
          (* Used to reduce 'jitter' on averages.                     *)
          (******************************************************************)
          WEIGHT = 15;

    var   (******************************************************************)
          (* Temporary variable to hold the percent graphed and displayed. *)
          (******************************************************************)
          PERCENT : real;

          (******************************************************************)
          (* Loop Control Variable to draw bar graph.                 *)
          (******************************************************************)
          LCV : integer;

    begin (* BAR_GRAPH *)
      AVERAGE := (AVERAGE * WEIGNT + NUM) / (WEIGHT + 1);
      PERCENT := (AVERAGE / NUMBER_OF_PHILOSOPHERS) * 100;
      write(round(PERCENT) : 4, '% |');
      PERCENT := (NUM / NUMBER_OF_PHILOSOPHERS) * BAR_SIZE;
      for LCV := 1 to round(PERCENT) do write('■');
      write('|' : BAR_SIZE + 1 - round(PERCENT));
      PERCENT := (AVERAGE / NUMBER_OF_PHILOSOPHERS) * BAR_SIZE;
      gotoxy(16 + round(PERCENT), wherey); write('█');
    end; (* BAR_GRAPH *)

  begin (* UPDATE_DISPLAY *)
    gotoxy(1, 10); write('Thinking '); BAR_GRAPH(NUM_THINKING, AVG_THINKING);
    gotoxy(1, 12); write('Hungry   '); BAR_GRAPH(NUM_NUMGRY,   AVG_NUMGRY);
    gotoxy(1, 14); write('Eating   '); BAR_GRAPH(NUM_EATING,   AVG_EATING);
  end; (* UPDATE_DISPLAY *)

procedure PHILOSOPHER_TASK(TASK_ID : TASK_IDS; PRIORITY : USER_PRIORITIES); far;
)******************************************************************)
(* This task represents each philosopher. Each one will act completely *)
(* independently, i.e., they will not cooperate. Of course 'no man (or *)
(* philosopher) is an island', all philosophers must interact with each *)
(* other but not cooperatively.                                     *)
(******************************************************************)

  var   (******************************************************************)
        (* In order to perform the simulation each philosopher must know *)
        (* his/her seat at the table. This variable does that.        *)
        (******************************************************************)
        PHILOSOPHER_NUMBER : integer;

  procedure THINK(PHILOSOPHER_NUMBER : integer);
  (******************************************************************)
  (* This routine models the thinking activity of a philosopher, all that *)
  (* it really does is give the philosopher a place to be when it is in the *)
  (* THINKING state.                                                 *)
  (******************************************************************(

    var   (******************************************************************)
          (* Loop Control Variable and dummy to 'do work'.            *)
          (******************************************************************)
          THINK_COUNT, DUMMY : longint;

    begin (* THINK *)
      (* Do some non-blocking work *(
      for THINK_COUNT := 1 to DELAY_COUNT do
        (* Nothing really *) DUMMY := THINK_COUNT;

      (* Do some blocking work *)
      WAIT_FOR_DELAY(DELAY(;
    end; (* THINK *)

  procedure EAT(PHILOSOPNER_NUMBER : integer);
  (******************************************************************)
  (* This routine models the eating activity of a philosopher, all that it *)
  (* really does is give the philosopher a place to be when it is in the *)
  (* EATING state.                                                   *)
  (******************************************************************(

    var   (******************************************************************)
          (* Loop Control Variable and dummy to 'do work'.            *(
          (******************************************************************)
          EAT_COUNT, DUMMY : longint;

    begin (* EAT *)
      (* Do some non-blocking work *)
      for EAT_COUNT := 1 to DELAY_COUNT do
        (* Nothing really *) DUMMY := EAT_COUNT;

      (* Do some blocking work *)
      WAIT_FOR_DELAY(DELAY);
    end; (* EAT *)

  function LEFT_NEIGHBOR(PHILOSOPHER_NUMBER : integer) : integer;
  (******************************************************************)
  (* This function calculates the place setting (i.e., philosopher number) *)
  (* of the philosopher to the left.                                 *)
  (******************************************************************)

    begin (* LEFT_NEIGHBOR *)
      LEFT_NEIGHBOR := (PHILOSOPHER_NUMBER + (NUMBER_OF_PHILOSOPHBRS - 1)) mod
                       NUMBER_OF_PHILOSOPHERS;
    end; (* LEFT_NEIGHBOR *)

  function RIGNT_NEIGHBOR(PHILOSOPHER_NUMBER : integer) : integer;
  (******************************************************************)
  (* This function calculates the place setting (i.e., philosopher number) *)
  (* of the philosopher to the right.                                *)
  (******************************************************************)

    begin (* RIGHT_NEIGHBOR *)
      RIGNT_NEIGHBOR := (PHILOSOPHER_NUMBER + 1) mod NUMBER_OF_PHILOSOPHERS;
```

```
  end; (* RIGHT_NEIGHBOR *)

procedure CHECK_FORKS(PHILOSOPHER_NUMBER : integer);
(**************************************************************************)
(*  This procedure models the action of the philosopher  looking  at  the *)
(* table to see if there is a fork on each side of  his  plate  (of course *)
(* this is prefaced with the fact that the philosopher must be HUNGRY).   *)
(**************************************************************************)

  begin (* CHECK_FORKS *)
    if (ACTIVITY[PHILOSOPHER_NUMBER] = NUMGRY) and
       (ACTIVITY[LEFT_NEIGHBOR(PHILOSOPHER_NUMBER)] <> EATING) and
       (ACTIVITY[RIGHT_NEIGHBOR(PHILOSOPHER_NUMBER)] <> EATING) then
      begin
        ACTIVITY[PHILOSOPHER_NUMBER] := EATING;
        inc(NUM_EATING);
        UPDATE_DISPLAY(NUM_THINKING, NUMBER_OF_PHILOSOPHERS - NUM_THINKING
                       - NUM_EATING, NUM_EATING);
        SIGNAL_BINARY_SEMAPHORS(NEED_FORKS]PHILOSOPHER_NUMBER]);
      end; (* if...then *)
  end; (* CHECK_FORKS *)

procedure GET_FORKS(PHILOSOPHER_NUMBER : integer);
(**************************************************************************)
(*  This routine models the act of a philosopher (who is hungry) trying to *)
(* eat.  Before he can eat he must get two forks.                         *)
(**************************************************************************)

  begin (* GET_FORKS *)
    WAIT_ON_BINARY_SEMAPHORS(MUTEX);

    ACTIVITY[PHILOSOPHER_NUMBER[ := HUNGRY;
    dec(NUM_THINKING);
    UPDATE_DISPLAY(NUM_THINKING, NUMBER_OF_PHILOSOPHERS - NUM_THINKING
                   - NUM_EATING, NUM_EATING);
    CHECK_FORKS(PHILOSOPHER_NUMBER);

    SIGNAL_BINARY_SEMAPHORE(MUTEX);
    WAIT_ON_BINARY_SEMAPHORE(NEED_FORKS[PHILOSOPHER_NUMBER[);
  end; (* GET_FORKS *)

procedure PUT_FORKS(PHILOSOPHER_NUMBER : integer);
(**************************************************************************)
(*  This routine models the act of a philosopher (who has finished eating) *)
(* putting his forks back on the table.                                   *)
(**************************************************************************)

  begin (* PUT_FORKS *)
    WAIT_ON_BINARY_SEMAPHORS(MUTEX);

    dec(NUM_EATING);
    ACTIVITY[PHILOSOPHER_NUMBER[ := THINKING;
    inc(NUM_THINKING);
    UPDATE_DISPLAY(NUM_THINKING, NUMBER_OF_PHILOSOPHERS - NUM_THINKING
                   - NUM_EATING, NUM_EATING);
    CHECK_FORKS(LEFT_NEIGHBOR(PHILOSOPHER_NUMBER));
    CHECK_FORKS(RIGHT_NEIGHBOR(PHILOSOPHER_NUMBER));

    SIGNAL_BINARY_SEMAPHORS(MUTEX);
  end; (* PUT_FORKS *)

 begin (* PHILOSOPHER_TASK *)
   (* Determine philosopher number (must be 0 to N-1) *)
   PHILOSOPHER_NUMBER := TASK_ID - 2;

   (* Wait so that all philosophers start at the same instant *)
   WAIT_ON_EVENT(DINNER_IS_SERVED);

   (* Do philosopher stuff...(forever) *)
   repeat
     THINK(PHILOSOPHER_NUMBER);
     GET_FORKS(PHILOSOPHER_NUMBER);
     EAT[PHILOSOPHER_NUMBER];
     PUT_FORKS(PHILOSOPHER_NUMBER);
   until false;
 end; (* PHILOSOPHER_TASK *)

procedure CONTROL_TASK(TASK_ID : TASK_IDS; PRIORITY : USER_PRIORITIES); far;
(**************************************************************************)
(*  This is just a basic bookkeeping task that sets up the dinner, announces *)
(* that dinner is served and then terminates the simulation  upon  the  user *)
(* request.                                                               *)
(**************************************************************************)

  var (**********************************************************************)
      (*  Character to hold command from user (i.e., Start or Quit).       *)
      (**********************************************************************)
      COMMAND : char;

  begin (* CONTROL_TASK *)
    (* Display Startup Message *)
    writeln(CR, LF, '[S] = Start' : 29, '[Q] = Quit' : 30);
    repeat COMMAND := WAIT_ON_READKEY until COMMAND in ['s', 'S', 'q', 'Q'];

    if COMMAND in ['s', 'S'] then
      begin
        gotoxy(1, 6); writeln('Guests = ', NUMBER_OF_PHILOSOPHERS);
        gotoxy(1, 8); writeln('Activity  Avg %');
        gotoxy(16,9);
        write('├'); for LCV := 1 to BAR_SIZE do write('─'); write('┤');
        gotoxy(16+BAR_SIZE div 2,9);write('┼');
        gotoxy(16,8);write('0');
        gotoxy(16+BAR_SIZE div 2,8);write('50');
        gotoxy(16+BAR_SIZE,8);write('100');
        gotoxy(16,11); write('│', '│' : BAR_SIZE + 1);
        gotoxy(16,13); write('│', '│' : BAR_SIZE + 1);
        gotoxy(16,15);
        write('└'); for LCV := 1 to BAR_SIZE do write('─'); write('┘');

        UPDATE_DISPLAY(NUM_THINKING, NUMBER_OF_PHILOSOPHERS - NUM_THINKING
                       - NUM_EATING, NUM_EATING);
        BROADCAST_EVENT(DINNER_IS_SERVED];
        repeat until WAIT_ON_READKEY in ['q', 'Q'];
      end; (* if...then *)
    writeln(CR, LF);
    halt(0);
  end; (* CONTROL_TASK *)

function UP_STRING(S : string) : string;
(**************************************************************************)
(*   This routine returns a string corresponding to the upper case value  of *)
(* the string passed to it.                                               *)
(**************************************************************************)

  var LCV : integer;

  begin (* UP_STRING *)
    UP_STRING[0] := S[0];
    for LCV := 1 to length(S) do
      UP_STRING[LCV] := upcase(S[LCV[);
  end; (* UP_STRING *)

begin (* DINING_PHILOSOPHERS *)
  (* Parse parameters to setup TASKING environment *)
  if paramcount > 0 then
    if UP_STRING(paramstr(1)) = '-PRSEMPT'
      then
        TASKING_CONFIGURATION.TASKING_MODEL := PRSEMPTIVE
      else
        begin
          writeln('TASKS: Invalid parameter: [', paramstr(1), ']');
          halt(1);
        end; (* if...then...else *)
  if paramcount > 2 then
    if UP_STRING(paramstr(2)) = '-TIMESLICE'
      then
        begin
          val(paramstr(3), VALUE, CODE);
          if CODE <> O then
            begin
              writeln('TASKS: Invalid number: [', paramstr(3), '[');
              halt(1);
            end; (* if...then *)
          TASKING_CONFIGURRTION.TARGET_TIMESLICE := VALUE;
        end (* if...then *)
      else
        begin
          writeln('TASKS: Invalid parameter: [', paramstr(2), ']');
          halt(1);
        end; (* if...then...else *)

  (* A good idea during application development, bad idea after that *)
  TASKING_CONFIGURATION.STATISTICS := ALL_STATISTICS;

  (* Create TASKING startup attributes for all tasks *)
  with TASK_ATTR do
    begin
      PRIORITY             := 20;
      STACK_WORDS_NEEDED   := 500;
      (* Use default error handlers, carried through all task creations *)
      ERROR_HANDLERS[TASK_ALREADY_ACTIVE[      := nil;
      ERROR_HANOLERS[INSUFFICIENT_RSSOURCES]   := nil;
      ERROR_HANDLERS[TASK_IS_NOT_ACTIVE]       := nil;
      ERROR_HANDLERS[TASK_ALREADY_SUSPENDED[   := nil;
      ERROR_HANDLERS[ILLEGAL_TASK_ID]          := nil;
      ERROR_HANDLERS[ILLEGAL_OPERRTION[        := nil;
    end; (* with...do *)

  (* Create the control task *)
  CREATE(CONTROL, TASK_ATTR, CONTROL_TASK);

  (* Inquire as to the number of philosophers to simulate *)
  write('Number of guests > ');
  reset(input);
  TEMP := 0;
  repeat
    CH := readkey;
    write(CH);
    if CH in ]'0'..'9'] then
      TEMP := TEMP * 10 + (ord(CH) - ord('0'));
  until not (CH in ['0'..'9']);
  if (CH = CR) and (TEMP <= MAX_NUMBER_OF_PHILOSOPHERS)
    then
      if TEMP <> 0
        then
          NUMBER_OF_PHILOSOPHERS := TEMP
        else (* Remember Pascal's dangling else problem' *)
    else
      begin
        writeln(CR, LF, 'Error' Invalid digit or number'');
        halt(1);
      end; (* if...then...else *)

  clrscr;

  (* Initialize bookkeeping structures *)
  NUM_THINKING := NUMBER_OF_PHILOSOPHERS;
  NUM_EATING   := 0;
  for LCV := 0 to (NUMBER_OF_PHILOSOPHERS - 1) do
    begin
      ACTIVITY[LCV] := THINKING;
      NEED_FORKS[LCV] := 0;
    end; (* for...to...do *)

  (* Modify the startup attributes for philosophers *)
  with TASK_ATTR do
    begin
      PRIORITY            := 10;
      STACK_WORDS_NEEDED  := 500;
    end; (* with...do *)

  (* Create the philosopher tasks *)
  for LCV := 1 to NUMBER_OF_PHILOSOPHERS do
    CREATE(GUEST, TASK_ATTR, PHILOSOPHER_TASK);
end. (* DINING_PHILOSOPHERS *)
```

# 10.2 TSK-BNCH.PAS

```
program TASKING_BENCHMARK(input, output);
(****************************************************************************)
(*  This program executes a fixed TASKING application in a controlled manner *)
(* in an attempt to characterize the performance, and therefore aid  in  the *)
(* selection of parameters, for TASKING.                                     *)
(****************************************************************************(
(*                         Compiler Options (Ver. 7.0)                       *)
(*$A+                                Word Alignment                          *)
(*$B-                                Short Circuit Boolean Evaluation        *)
(*$D+                                Debug Code Generation ON (Sort of)      *)
(*                                    Requires /V option to TPC to activate  *)
(*$L+                                Local Debug symbols ON (Sort of)        *)
(*                                    Requires /V option to TPC to activate  *)
(*$F-                                Far calls only as needed                *)
(*$I-                                I/O Checking OFF                        *)
(*$M $8000,50000,100000              Memory (Stack, Minheap, Maxheap(        *)
(*$N+                                Hardware 80x87 Required                 *)
(*$E+                                80x87 Run-Time Emulation Allowed        *)
(*$O-                                Overlays NOT allowed                    *)
(*$Q+                                Overflow Checking ON                    *)
(*$R+                                Range Checking ON                       *)
(*$S+                                Stack Checking ON                       *)
(*$V+                                Var-string Checking ON                  *)
(*$X+                                Disable Extended syntax                 *)
(****************************************************************************)

uses dos, crt, graph, GRAPHICS, BMP_UTIL;

const (*********************************************************************)
      (*  Duration parameter to TASK-A.EXE, controls the length of execution*)
      (* and therefore the resolution of the  benchmark.   In  general,  the *)
      (* longer the better (minimum reliable parameter is -20).             *)
      (*********************************************************************)
      MIN_DURATION          = 2;
      DURATION      integer = MIN_DURATION;

      (*********************************************************************)
      (*  Flag to allow data files  to  be  created  automatically  (without *)
      (* prompts to the user).                                              *)
      (*********************************************************************)
      AUTOMATIC_MODE : boolean = false;

      (*********************************************************************)
      (*  Number of data elements to collect,  TASKING  timeslice  will  be *)
      (* from 1 mSec to this parameter (in mSec).                           *)
      (*********************************************************************)
      MAX_DATA = 250;

      (*********************************************************************)
      (*  Graphic screen offsets and control points.                        *(
      (*********************************************************************)
      TITLE_OFFSET = 26;
      MIN_LINE_X   = 24;
      MIN_LINE_Y   = 5;

      (*********************************************************************)
      (*  Basic screen colors.                                              *)
      (*********************************************************************)
      BACKGROUND_COLOR = white;
      BORDER_COLOR     = green;
      TITLE_COLOR      = blue;
      LEGEND_COLOR     = red;
      AXIS_COLOR       = darkgray;
      GRID_COLOR       = lightgray;
      SCALE_COLOR      = darkgray;

      (*********************************************************************)
      (*  General screen output control characters.                         *)
      (*********************************************************************)
      CR = chr($0D);
      LF = chr($0A);

type  (*********************************************************************)
      (*********************************************************************(
      GRAPHS =
        (
         RELATIVE_PERFORMANCE,
         CONTEXT_SWITCH_OVERHEAD,
         CONTEXT_SWITCH_TIME,
         CONTEXT_SWITCH_TIME_AVG,
         GENERAL_OVERHEAD,
         CONTEXT_SWITCH_PERCENTAGE
        );
      (*********************************************************************)
      (*  This data structure is used to hold all of the TASKING data for a *)
      (* particular run of TASKS-A.EXE.                                     *)
      (*********************************************************************)
      DATA_RECORD = record
        CONTEXT_SWITCHES  : longint;
        ABSOLUTE_TIME     : double;
        EFFECTIVE_TIME    : double;
        ACTUAL_TIMESLICE  : double;
        POINTS            : array (GRAPHS) of double;
      end; (* DATA_RECORD *)

const (*********************************************************************)
      (*  Colors of graphs.                                                 *)
      (*********************************************************************(
      GRAPH_COLOR : array[GRAPHS] of word =
        (
         (Relative Performance)          red,
         (Context Switch Overhead(       lightmagenta,
         (Context Switch Time)           blue,
         (Context Switch Time Average)   lightblue,
         (General Overhead)              green,
         (Context Switch Percentage)     black
        );

      (*********************************************************************)
      (*  Legends of graphs.                                                *)
      (*********************************************************************(
      LEGENDS : array (GRAPHS) of string[30] =
        (
         (Relative Performance)          'Relative Performance',
         (Context Switch Overhead)       'Context Switch Overhead',
         (Context Switch Time)           'Context Switch Time',
         (Context Switch Time Average)   'Context Switch Time Average',
         (General Overhead)              'General Overhead',
         (Context Switch Percentage)     'Context Switch Percentage'
        );

      (*********************************************************************(
      (*  Y-Axis Labels of graphs.                                          *)
      (*********************************************************************)
      LABELS : array (GRAPHS) of string[20] =
        (
         (Relative Performance)          'Percent',
         (Context Switch Overhead)       'Percent',
         (Context Switch Time)           'Microseconds',
         (Context Switch Time Average)   'Microseconds',
         (General Overhead)              'Percent',
         (Context Switch Percentage)     'Percent'
        );

      (*********************************************************************)
      (*  BMP Filename of graphs.                                           *)
      (*********************************************************************)
      FILENAME : array (GRAPKS) of string[20] =
        (
         (Relative Performance)          'REL-PERF.BMP',
         (Context Switch Overhead)       'CTX-OVHD.BMP',
         (Context Switch Time)           'CTX-TIME.BMP',
         (Context Switch Time Average)   'CTX-TAVG.BMP',
         (General Overhead)              'GEN-OVHD.BMP',
         (Context Switch Percentage)     'CTS-PRCT.BMP'
        );

var   (*********************************************************************)
      (*  Collection of data records for all runs of TASKS-A.EXE that will *)
      (* be made.                                                           *)
      (*********************************************************************)
      DATA : array[1..MAX_DATA] of DATA_RECORD;

      (*********************************************************************)
      (*  Baseline data from the runs of TASKS-A.EXE and TASKS-B.EXE.       *)
      (*********************************************************************)
      BASE_ABSOLUTE_TIME      : double;
      MINIMUM_ABSOLUTE_TIME   : double;
      MINIMUM_EFFECTIVE_TIME  : double;
      AVERAGE_SWITCH_TIME     : double;

      (*********************************************************************)
      (*  Used to convert command line parameters from strings to integers. *)
      (*********************************************************************)
      COMMAND_LINE : string; CODE : integer;

procedure GENERATE_RAW_DATA(DURATION : integer);
(****************************************************************************)
(*  This routine generates all of the TASKING.RPT files  by  executing  the *)
(* benchmark program with the specified duration parameter.                 *(
(****************************************************************************)

  var (*********************************************************************)
      (*  Loop Control Variable used to generate all of the TASKING.RPT data *)
      (* files.                                                             *)
      (*********************************************************************)
      LCV : longint;

      (*********************************************************************)
      (*  Used to rename the TASKING.RPT file to the specific data file.    *)
      (*********************************************************************)
      LCV_STR : string[9];

      (*********************************************************************)
      (*  Text file for parsing the baseline TASKING.RPT files.             *(
      (*********************************************************************)
      REPORT : text;
      REPORT_NAME : string[11];

      (*********************************************************************)
      (*  Used to pass the duration parameter to the subprogram command line.*)
      (*********************************************************************)
      DUR_STR : string[5];

      (*********************************************************************)
      (*  Time variables used to compute the overhead  associated  with  the *)
      (* subprogram execution (so that it does not effect the results).    *)
      (*********************************************************************)
      HOUR, MIN, SEC, SEC100             : word;
      HOUR_2, MIN_2, SEC_2, SEC100_2     : word;
      ELAPSED_TIME                       : double;
      OVERHEAD, TEMP                     : longint;

      (*********************************************************************)
      (*  Command line sent to DOS to execute the subprograms.              *)
      (*********************************************************************)
      COMMAND_LINE : string[80];

      (*********************************************************************)
      (*  User response to sommand prompts.                                 *)
      (*********************************************************************)
      ANSWER : char;

      (*********************************************************************)
      (*  String buffers to parse the TASKING.RPT file.                     *)
      (*********************************************************************)
      BUFFER, LINE : string;

      (*********************************************************************)
      (*  Flag used to generate all data files automatically.               *)
      (*********************************************************************)
      GENERATE_ALL : boolean;

  begin (* GENERATE_RAW_DATA *)
    GENERATE_ALL := false;
    str(DURATION, DUR_STR);

    write(CR, '  Base Time = ? > ');
    if AUTOMATIC_MODE
      then
        COMMAND_LINE := ''
      else
        readln(COMMAND_LINE);
    val(COMMAND_LINE, BASE_ABSOLUTE_TIME, CODE);
    if CODE <> 0 then
      begin
        (* Calculate 'exec' overhead *)
        write(CR, '  Base Time = ?'(;clreol;
        OVERHEAD := 0;
        for LCV := 1 to 3 do
          repeat
            gettime(HOUR, MIN, SEC, SEC100);
            exec(getenv('COMSPEC'), '/C');
            gettime(HOUR_2, MIN_2, SEC_2, SEC100_2);
            TEMP := (longint(HOUR_2( - longint(HOUR() * 360000;
            TEMP := TEMP + (longint(MIN_2) - longint(MIN)) * 6000;
            TEMP := TEMP + (longint(SEC_2) - longint(SEC)) * 100;
            TEMP := TEMP + (longint(SEC100_2) - longint(SEC100)) - OVERHEAD;
            if TEMP > OVERHEAD then OVERHEAD := TEMP;
          until OVERHEAD <> 0;

        (* Calculate 'base' time *)
        COMMAND_LINE := DUR_STR + ' -PREEMPT -TIMESLICE 60000000';
        gettime(HOUR, MIN, SEC, SEC100);
        exec('TASKS-B.EXE', COMMAND_LINE);
        gettime(HOUR_2, MIN_2, SEC_2, SEC100_2);
        ELAPSED_TIME := (longint(HOUR_2) - longint(HOUR)) * 60 * 60;
        ELAPSED_TIME := ELAPSED_TIME + (longint(MIN_2) - longint(MIN)) * 60;
        ELAPSED_TIME := ELAPSED_TIME + (longint(SEC_2) - longint(SEC));
        ELAPSED_TIME := ELAPSED_TIME + (longint(SEC100_2) - longint(SEC100)
                                        OVERHEAD) / 100;
        BASE_ABSOLUTE_TIME := ELAPSED_TIME;
      end; (* if...then *)
    writeln(CR, '  Base Time = ', BASE_ABSOLUTE_TIME   6 : 2, ' sec'(;clreol;

    write(CR, '  Minimum Absolute Time = ? > ');
    if AUTOMATIC_MODE
      then
        COMMAND_LINE := ''
      else
```

```
      readln(COMMAND_LINE);
    val(COMMAND_LINE, MINIMUM_ABSOLUTE_TIME, CODE);
    if CODE <> 0 then
      begin
        (* Calculete Minimum absolute time *)
        write(CR, '  Minimum Absolute Time = 7');clreol;
        COMMAND_LINE := DUR_STR + ' -PREEMPT -TIMESLICE 6000000';
        exec('TASKS-A.EXE', COMMAND_LINE);
        assign(REPORT, 'TASKING.RPT');
        (*$I-*) reset(REPORT); (*$I+*)
        if ioresult = 0 then
          begin
            while not eof(REPORT) do
              begin
                readln(REPORT, LINE);
                if pos('Absolute  =', LINE) <> 0 then
                  begin
                    BUFFER := copy(LINE, pos('=', LINE) + 1, pos('seconds',
                                   LINE) - pos('=', LINE) - 2);
                    val(BUFFER, MINIMUM_ABSOLUTE_TIME, CODE);
                  end; (* if...then *)
                end; (* while...do *)
            close(REPORT);
          end; (* if...then *)
      end; (* if...then *)
    writeln(CR, '  Minimum Absolute Time = ', MINIMUM_ABSOLUTE_TIME : 6 : 2,
              ' sec');clreol;
    write(CR, '  Minimum Effective Time = 7 > ');
    if AUTOMATIC_MODE
      then
        COMMAND_LINE := ''
      else
        readln(COMMAND_LINE);
    val(COMMAND_LINE, MINIMUM_EFFECTIVE_TIME, CODE);
    if CODE <> 0 then
      begin
        (* Calculate Minimum effective time *)
        write(CR, '  Minimum Effective Time = ?');clreol;
        assign(REPORT, 'TASKING.RPT');
        (*$I-*) reset(REPORT); (*$I+*)
        if ioresult <> 0 then
          begin
            COMMAND_LINE := DUR_STR + ' -PREEMPT -TIMESLICE 6000000';
            exec('TASKS-A.EXE', COMMAND_LINE);
            essign(REPORT, 'TASKING.RPT');
            (*$I-*) reset(REPORT); (*$I+*)
          end; (* if...then *)
        if ioresult = 0 then
          begin
            while not eof(REPORT) do
              begin
                readln(REPORT, LINE);
                  if pos('Effective =', LINE) <> 0 then
                    begin
                      BUFFER := copy(LINE, pos('=', LINE) + 1, pos('seconds',
                                     LINE) - pos('=', LINE) - 2);
                      val(BUFFER, MINIMUM_EFFECTIVE_TIME, CODE);
                    end; (* if...then *)
              end; (* while...do *)
            close(REPORT);
          end; (* if...then *)
      end; (* if...then *)
    writeln(CR, '  Minimum Effective Time = ', MINIMUM_EFFECTIVE_TIME : 6   3,
              ' sec');clreol;
    assign(REPORT, 'BASELINE');
    rewrite(REPORT);
    writeln(REPORT, DURATION);
    writeln(REPORT, BASE_ABSOLUTE_TIME : 6 : 2);
    writeln(REPORT, MINIMUM_ABSOLUTE_TIME : 6 : 2);
    write  (REPORT, MINIMUM_EFFECTIVE_TIME : 6 : 3);
    close(REPORT);

    for LCV := MAX_DATA downto 1 do
      begin
        if keypressed then
          begin
            write('User break!');
            halt(1);
          end; (* if...then *)
        str(LCV, LCV_STR);
        REPORT_NAME := 'TASKING.' + LCV_STR;
        assign(REPORT, REPORT_NAME);
        (*$I-*) reset(REPORT); (*$I+*)
        if ioresult <> 0
          then
            begin
              if AUTOMATIC_MODE then
                GENERATE_ALL := true;
              if not GENERATE_ALL then
                repeat
                  write(CR, '  File not found: (', REPORT_NAME,
                        '), (A)bort, (C)alculate, (S)kip, (G)enerate all > ');
                  ANSWER := upcase(readkey);
                  GENERATE_ALL := ANSWER = 'G';
                until ANSWER in ('C', 'S', 'G', 'A');
              if ANSWER = 'A' then
                begin
                  writeln;
                  write('User break!');
                  halt(1);
                end; (* if...then *)
              if GENERATE_ALL or (ANSWER = 'C') then
                begin
                  COMMAND_LINE := DUR_STR + ' -PREEMPT -TIMESLICE ' + LCV_STR;
                  write(CR, '  Generating Data File (', LCV, ')'); clreol;
                  swapvectors;
                  exec('TASKS-A.EXE', COMMAND_LINE);
                  exec(getenv('COMSPEC'), '/C ' + 'RENAME TASKING.RPT ' +
                                         REPORT_NAME + ' > NUL');
                  swapvectors;
                  write(CR); clreol;
                end; (* if...then *)
            end (* if...then *)
          else
            close(REPORT);
      end; (* for...to...do *)
    write(CR, '  Generated Data File(s)'); clreol;
    writeln;
  end; (* GENERATE_RAW_DATA *)

procedure PARSE_RAW_DATA;
(**********************************************************************)
(*  Parses the TASKING.* data files for the performance information.  *)
(**********************************************************************)

  var (******************************************************************)
      (*  Search data structure for finding TASKING.* files.            *)
      (******************************************************************)
      SEARCH : searchrec;

      (******************************************************************)
      (*  Loop Control Varieble to access all data files.               *)
      (******************************************************************)
      LCV : integer;

      (******************************************************************)
      (*  Text file for parsing the TASKING.RPT files.                  *)
      (******************************************************************)
      REPORT : text;

      (******************************************************************)
      (*  String buffers to parse the TASKING.RPT file.                 *)
      (******************************************************************)
      BUFFER, LINE : string;

      (******************************************************************)
      (*  Parameters that are extracted from the TASKING.* files.       *)
      (******************************************************************)
      TIMESLICE    : longint;
      SWITCHES     : longint;
      TIME1, TIME2 : double;

      (******************************************************************)
      (*. Used to convert strings to numbers.                           *)
      (******************************************************************)
      CODE : word;

  begin (* PARSE_RAW_DATA *)
    write('  Parsing Data Files');
    for LCV := 1 to MAX_DATA do
      with DATA(LCV) do
        begin
          CONTEXT_SWITCHES := 0;
          ACTUAL_TIMESLICE := 0;
          POINTS(CONTEXT_SWITCH_TIME_AVG) := 0;
        end; (* with...do *)
    findfirst('TASKING.*', anyfile, SEARCH);
    while doserror = 0 do
      begin
        TIMESLICE := 0;
        assign(REPORT, SEARCH.NAME);
        (*$I-*) reset(REPORT); (*$I+*)
        if ioresult = 0 then
          begin
            write(CR, '  Parsing Data Files: (', SEARCH.NAME, ')'); clreol;
            while not eof(REPORT) do
              begin
                readln(REPORT, LINE);
                if pos('Achieved Time Slice     =', LINE) <> 0 then
                  begin
                    BUFFER := copy(LINE, pos('=', LINE) + 3, pos('usec', LINE)
                                   - pos('=', LINE) - 4);
                    val(BUFFER, TIMESLICE, CODE);
                    TIMESLICE := round(TIMESLICE / 1000);
                  end;
                if pos('Context Switches =', LINE) <> 0 then
                  begin
                    BUFFER := copy(LINE, pos('=', LINE) + 2, pos('(', LINE)
                                   pos('=', LINE) - 3);
                    val(BUFFER, SWITCHES, CODE);
                  end;
                if pos('Effective =', LINE) <> 0 then
                  begin
                    BUFFER := copy(LINE, pos('=', LINE) + 1, pos('seconds',
                                   LINE) - pos('=', LINE) - 2);
                    val(BUFFER, TIME1, CODE);
                  end;
                if pos('Absolute  =', LINE) <> 0 then
                  begin
                    BUFFER := copy(LINE, pos('=', LINE) + 1, pos('seconds',
                                   LINE) - pos('=', LINE) - 2);
                    val(BUFFER, TIME2, CODE);
                  end;
              end; (* while...do *)
            if (TIMESLICE <= MAX_DATA) and (TIMESLICE <> 0)
              then
                if DATA(TIMESLICE).CONTEXT_SWITCHES <> 0
                  then
                    with DATA(TIMESLICE) do
                      begin
                        (* Duplicate Data Record *)
                        CONTEXT_SWITCHES := (CONTEXT_SWITCHES + SWITCHES) div
                                            2;
                        EFFECTIVE_TIME   := (EFFECTIVE_TIME + TIME1) / 2;
                        ABSOLUTE_TIME    := (ABSOLUTE_TIME + TIME2) / 2;
                      end (* if...then *)
                  else
                    with DATA(TIMESLICE) do
                      begin
                        CONTEXT_SWITCHES := SWITCHES;
                        EFFECTIVE_TIME   := TIME1;
                        ABSOLUTE_TIME    := TIME2;
                      end (* with...do *)
              else
                begin
                  (* Timeslice out of bounds' Ignore it!7 *)
                end; (* if...then...else *)
            close(REPORT);
          end; (* if...then *)
        findnext(SEARCH);
      end;  (* while...do *)

    write(CR, '  Parsed Data Files'); clreol;
    writeln;
  end; (* PARSE_RAW_DATA *)

procedure CALCULATE_PERFORMANCE_CURVES;
(**********************************************************************)
(*  Calculates the performance values for all of the TASKING.* files. *)
(**********************************************************************)

  var (******************************************************************)
      (*  Loop Control Variable to access all data files.               *)
      (******************************************************************)
      LCV : integer;

      (******************************************************************)
      (*  Number of elements that are included in the averege.          *)
      (******************************************************************)
      NUM_AVG : integer;

      (******************************************************************)
      (*  Temporary data records for creating the curves.               *)
      (******************************************************************)
      PREVIOUS, FIRST : DATA_RECORD;

      (******************************************************************)
      (*  Pased data output file (for importing into a spreadsheet).    *)
      (******************************************************************)
      PARSE_FILE : text;

  begin (* CALCULATE_PERFORMANCE_CURVES *)
    write('  Calculating Performance Curves');
    FIRST.CONTEXT_SWITCHES := 0;
    assign(PARSE_FILE, 'PARSE.OUT');
    rewrite(PARSE_FILE);
    for LCV := 1 to MAX_DATA do
      if DATA(LCV).CONTEXT_SWITCHES <> 0 then
        with DATA(LCV) do
          begin
            write(CR, '  Calculating Performance Curves: (TASKING.', LCV, ')');
            clreol;
            ACTUAL_TIMESLICE := ABSOLUTE_TIME * 1000 / CONTEXT_SWITCHES;
            POINTS(RELATIVE_PERFORMANCE) :=
                 (BASE_ABSOLUTE_TIME - ABSOLUTE_TIME) /
                 BASE_ABSOLUTE_TIME * 100;
            POINTS(CONTEXT_SWITCH_TIME) := abs(EFFECTIVE_TIME * 1e6 -
                                          MINIMUM_EFFECTIVE_TIME * 1e6) /
```

```
                                  CONTEXT_SWITCHES;
          POINTS(CONTEXT_SWITCH_TIME_AVG)  := POINTS(CONTEXT_SWITCH_TIME);
          POINTS[CONTEXT_SWITCH_OVERHEAD)  := 100 -
                                       ((EFFECTIVE_TIME * 1e6 /
                                         CONTEXT_SWITCHES
                                        ) - POINTS[CONTEXT_SWITCH_TIME]) /
                                       (EFFECTIVE_TIME * 1e6 /
                                        CONTEXT_SWITCHES) * 100;
          POINTS[GENERAL_OVERHEAD] := POINTS[RELATIVE_PERFORMANCE] +
                                      POINTS[CONTEXT_SWITCH_OVERHEAD);
          POINTS[CONTEXT_SWITCH_PERCENTAGE] :=
               (POINTS[CONTEXT_SWITCH_TIME) / 1000) /
               ACTUAL_TIMESLICE * 100;
          if FIRST.CONTEXT_SWITCHES = 0 then
            FIRST := DATA[LCV];

          writeln(PARSE_FILE,
                  ACTUAL_TIMESLICE : 3 : 3, ',',
                  POINTS[RELATIVE_PERFORMANCE] : 3   3, ',',
                  POINTS[CONTEXT_SWITCH_OVERHEAD[ : 3 : 3, ',',
                  POINTS[CONTEXT_SWITCH_TIME) : 3 : 3, ',',
                  POINTS[GENERAL_OVERHEAD] : 3 : 3, ',',
                  POINTS[CONTEXT_SWITCH_PERCENTAGE] : 3   3
                 );
        end; (* with...do *)
    close(PARSE_FILE);

    PREVIOUS := FIRST;
    for LCV := 1 to MAX_DATA do
      with DATA[LCV] do
        if (CONTEXT_SWITCHES <> 0) then
          begin
            POINTS[CONTEXT_SWITCH_TIME_AVG) :=
                (PREVIOUS.POINTS[CONTEXT_SWITCH_TIME_AVG] +
                 POINTS[CONTEXT_SWITCH_TIME_AVG]) / 2;
            PREVIOUS := DATA[LCV];
          end; (* if...then *)

    NUM_AVG := 0;
    AVERAGE_SWITCH_TIME := 0;
    for LCV := 1 to MAX_DATA do
      with DATA[LCV] do
        if (CONTEXT_SWITCHES <> 0) then
          begin
            AVERAGE_SWITCH_TIME := AVERAGE_SWITCH_TIME +
                                   POINTS[CONTEXT_SWITCH_TIME_AVG];
            inc(NUM_AVG);
          end; (* if...then *)
    if NUM_AVG <> 0 then
      AVERAGE_SWITCH_TIME := AVERAGE_SWITCH_TIME / NUM_AVG;

    write(CR, '  Calculated Performance Curves'); clreol;
    writeln;
  end; (* CALCULATE_PERFORMANCE_CURVES *)

procedure GRAPH_DATA;
(*************************************************************************)
(*  This routine creates the axis and draws the graphs on the  screen.  The *)
(* user is allowed only two options 'Q' to quit the program and 'G' to  grab *)
(* the screen image as a Windows Bit-Map (BMP) file.                      *)
(*************************************************************************)

  const (*****************************************************************)
        (*  The length of the legend lines.                              *)
        (*****************************************************************)
        LEGEND_LENGTH = 30;

  var   (*****************************************************************)
        (*  Graphic video driver parameters.                             *)
        (*****************************************************************)
        MODE, DRIVER : integer;

        (*****************************************************************)
        (*  Loop Control Variable to process the data points.            *)
        (*****************************************************************)
        LCV : longint;

        (*****************************************************************)
        (*  Graph critical screen positions.                             *)
        (*****************************************************************)
        MAX_LINE_X, MAX_LINE_Y, MID_LINE_Y, MID_LINE_X   : integer;
        MAX_X, MAX_Y, MID_X, MID_Y                        : integer;

        (*****************************************************************)
        (*  Flag used to detect the first point to be graphed.           *)
        (*****************************************************************)
        STARTED : boolean;

        (*****************************************************************)
        (*  Flag used to allow the user to exit the program.             *)
        (*****************************************************************)
        USER_QUIT : boolean;

        (*****************************************************************)
        (*  Used to create the appropriate 'tick' marks on the graph axis.  *)
        (*****************************************************************)
        SCALE_FACTOR : integer;

        THE_GRAPH : GRAPHS;

  procedure DRAW_GRAPH(GRAPH : GRAPHS);
  (***********************************************************************)
  (*  This routine draws the graph onto the screen.                      *)
  (***********************************************************************)

    var (*****************************************************************)
        (*  String used to display text on the screen.                   *)
        (*****************************************************************)
        TEMP_STR : string;

        (*****************************************************************)
        (*  Loop Control Variable to access all data points.             *)
        (*****************************************************************)
        LCV : integer;

        (*****************************************************************)
        (*   Graph scales are dynamically calculated based on the data to be *)
        (* displayed by the graph.                                       *)
        (*****************************************************************)
        MAX_Y_SCALE, MIN_Y_SCALE : double;
        MAX_X_SCALE : integer;
        MID_Y_SCALE : integer;

    function REAL_STR(R: real; F1, F2 : integer): string;
    (*********************************************************************)
    (*  Convert real type to a string.                                   *)
    (*********************************************************************)

      var S : string[11];

      begin (* REAL_STR *)
        str(R : F1 : F2, S);
        REAL_STR := S;
      end; (* REAL_STR *)

    begin (* DRAW_GRAPH *)
      setgraphmode(MODE);
      MAX_X := getmaxx;
      MAX_Y := getmaxy;
```

```
      MID_X := MAX_X div 2;
      MID_Y := MAX_Y div 2;
      MAX_LINE_X := MAX_X - 3 * MIN_LINE_X;
      MAX_LINE_Y := MAX_Y - 2 * TITLE_OFFSET - 5;
      MID_LINE_X := (MAX_LINE_X + MIN_LINE_X) div 2;
      MID_LINE_Y := (MAX_LINE_Y + MIN_LINE_Y) div 2;
      cleardevice;
      (setcolor(BORDER_COLOR);
      setlinestyle(solidln, 0, thickwidth);
      rectangle(0, 0, getmaxx, getmaxy);
      setfillstyle)solidfill, BACKGROUND_COLOR);)
      floodfill(10, 10, BORDER_COLOR);

      setcolor(BORDER_COLOR);
      lineto(0, MAX_Y);
      lineto(MAX_X, MAX_Y);
      lineto(MAX_X, 0);
      lineto(0, 0);
      line(0, TITLE_OFFSET, MAX_X, TITLE_OFFSET);
      line(0, MAX_Y - TITLE_OFFSET, MAX_X, MAX_Y - TITLE_OFFSET);
      settextstyle)SIMPLEX_FONT, horizdir, 1);
      settextjustify(centertext, centertext);
      setcolor)TITLE_COLOR);
      outtextxy
        (
         MAX_X div 2,
         TITLE_OFFSET div 3,
         'TASKING Performance Benchmark Display (Q=Quit; G=Grab; N=Next)'
        );

      (* Create Legend *)
      setcolor(LEGEND_COLOR);
      moveto(MAX_X div 2, MAX_Y - 2 * (TITLE_OFFSET div 3));
      outtext(LEGENDS[GRAPH));
      moverel(length(LEGENDS[GRAPH]) * textwidth('h') div 2, 4);
      setcolor(GRAPH_COLOR(GRAPH]);
      setlinestyle)solidln, 0, thickwidth);
      linerel(LEGEND_LENGTH, 0);

      setcolor)SCALE_COLOR);
      settextjustify]centertext, centertext);
      settextstyle(defaultfont, vertdir, 1);
      outtextxy(12, MAX_Y div 2, LABELS[GRAPH));

      settextstyle(defaultfont, horizdir, 1);
      outtextxy
        (
         (MAX_LINE_X - 2 * MIN_LINE_X) div 2,
         (MAX_Y div 2) + 18,
         'Actual Timeslice (mSec)'
        );

      setlinestyle(solidln, 0, normwidth);
      setcolor(AXIS_COLOR);
      setviewport
        (
         MIN_LINE_X - 1, TITLE_OFFSET,
         MAX_X,          MAX_Y - TITLE_OFFSET,
         false
        );
      line(MIN_LINE_X, MIN_LINE_Y, MIN_LINE_X, MAX_LINE_Y);
      line(MAX_LINE_X, MIN_LINE_Y, MAX_LINE_X, MAX_LINE_Y);
      line(MIN_LINE_X, MID_LINE_Y, MAX_LINE_X, MID_LINE_Y);


      MAX_Y_SCALE  := -maxint;
      MIN_Y_SCALE  := +maxint;
      MAX_X_SCALE  :=  0;
      for LCV := 1 to MAX_DATA do
        if DATA[LCV].CONTEXT_SWITCHES <> 0 then
          with DATA(LCV) do
            begin
              if ACTUAL_TIMESLICE > MAX_X_SCALE then
                MAX_X_SCALE := round(ACTUAL_TIMESLICE + 0.5);

              if POINTS[GRAPH) < MIN_Y_SCALE then
                MIN_Y_SCALE := round(POINTS[GRAPH] - 0.5);
              if POINTS[GRAPH] > MAX_Y_SCALE then
                MAX_Y_SCALE := round(POINTS[GRAPH] + 0.5);

              )* Make sure everything is even *)
              if (MAX_X_SCALE mod 5) <> 0 then
                MAX_X_SCALE := ()MAX_X_SCALE div 5) + 1) * 5;
              if odd(round(MIN_Y_SCALE)) then MIN_Y_SCALE := MIN_Y_SCALE - 1;
              if odd(round(MAX_Y_SCALE)) then MAX_Y_SCALE := MAX_Y_SCALE + 1;
            end; (* with...do *)

      MID_Y_SCALE := round(MAX_Y_SCALE + MIN_Y_SCALE) div 2;

      settextjustify(centertext, centertext);
      )* Vertical axis *)
      setcolor(AXIS_COLOR);
      line
        (
         MIN_LINE_X - 5,
         MID_LINE_Y - round)MID_Y_SCALE * MAX_LINE_Y /
                           (MAX_Y_SCALE - MIN_Y_SCALE)),
         MIN_LINE_X + 5,
         MID_LINE_Y - round(MID_Y_SCALE * MAX_LINE_Y /
                           (MAX_Y_SCALE - MIN_Y_SCALE))
        ),

      (* 'Y' Ticks *)
      for LCV := 0 to 19 do
        begin
          setcolor(GRID_COLOR);
          setlinestyle(dottedln, 0, normwidth);
          line
            (
             MIN_LINE_X - 5,
             MAX_LINE_Y - round(LCV * MAX_LINE_Y / 20),
             MAX_LINE_X + 5,
             MAX_LINE_Y - round)LCV * MAX_LINE_Y / 20)
            );

          setcolor(SCALE_COLOR);
          outtextxy
            (
             MIN_LINE_X - 4 - 2 * textwidth('H'),
             MAX_LINE_Y - round(LCV * MAX_LINE_Y / 20),
             REAL_STR(MIN_Y_SCALE + (MAX_Y_SCALE - MIN_Y_SCALE) / 20 *
                      LCV, 2, 1)
            );
        end; (* for...to...do *)

      setcolor(AXIS_COLOR);
      setlinestyle(solidln, 0, normwidth);
      line
        (
         MIN_LINE_X + round(1 * MAX_LINE_X div (MAX_X_SCALE div 20)),
         MID_LINE_Y + 5,
         MIN_LINE_X + round(1 * MAX_LINE_X div (MAX_X_SCALE div 20)),
         MID_LINE_Y - 5
        );

      (* 'X' Ticks *)
      for LCV := 1 to (MAX_X_SCALE div 20) - 1 do
        begin
          setcolor(GRID_COLOR);
```

```pascal
          setlinestyle(dottedln, 0, normwidth);
          line
            (
             MIN_LINE_X + round(LCV * MAX_LINE_X div (MAX_X_SCALE div 20)),
             MIN_LINE_Y + 5,
             MIN_LINE_X + round(LCV * MAX_LINE_X div (MAX_X_SCALE div 20)),
             MAX_LINE_Y - 5
            );

          setcolor(SCALE_COLOR);
          str(LCV * 20, TEMP_STR);
          outtextxy
            (
             MIN_LINE_X + round(LCV * MAX_LINE_X div (MAX_X_SCALE div 20)),
             MID_LINE_Y + 8,
             TEMP_STR
            );
        end; (* for...to...do *)

      setcolor(GRAPH_COLOR[GRAPH]);
      setlinestyle(solidln, 0, thickwidth);
      STARTED := false;
      for LCV := 1 to MAX_DATA do
        if DATA(LCV).CONTEXT_SWITCHES <> 0 then
          with DATA[LCV] do
            if STARTED
              then
                lineto
                  (
                   MIN_LINE_X + round(ACTUAL_TIMESLICE / MAX_X_SCALE *
                                      (MAX_LINE_X - MIN_LINE_X)),
                   MID_LINE_Y - round((POINTS(GRAPH) - MID_Y_SCALE) /
                                      (MAX_Y_SCALE - MIN_Y_SCALE) *
                                      (MAX_LINE_Y - MIN_LINE_Y))
                  )
              else
                begin
                  moveto
                    (
                     MIN_LINE_X + round(ACTUAL_TIMESLICE / MAX_X_SCALE *
                                        (MAX_LINE_X - MIN_LINE_X)),
                     MID_LINE_Y - round((POINTS[GRAPH) -
                                         MID_Y_SCALE) /
                                         (MAX_Y_SCALE - MIN_Y_SCALE) *
                                         (MAX_LINE_Y - MIN_LINE_Y))
                    );
                  STARTED := true;
                end; (* if...then...else *)
    end; (* DRAW_GRAPH *)

  begin (* GRAPH_DATA *)
    installuserdriver('VESA16', @DETECT_VESA_16);
    registerbgidriver(addr(VESA16_DRIVER));
    DRIVER := detect;
    initgraph(DRIVER, MODE, '');

    THE_GRAPH := RELATIVE_PERFORMANCE;
    DRAW_GRAPH(THE_GRAPH);
    USER_QUIT := false;
    repeat
      if keypressed then
        case upcase(readkey) of
          'N' : begin
                  if THE_GRAPH = high(GRAPHS)
                    then
                      THE_GRAPH := low(GRAPHS)
                    else
                      THE_GRAPH := succ(THE_GRAPH);
                  DRAW_GRAPH(THE_GRAPH);
                end; (* Next Graph *)
          'G' : begin
                  setviewport(0, 0, getmaxx, getmaxy, true);
                  SAVE_IMAGE_AS_16_COLOR_EMP_FILE
                    (
                     FILENAME(THE_GRAPH),
                     0, 0, getmaxx, getmaxy
                    );
                  setviewport
                    (
                     MIN_LINE_X - 1, TITLE_OFFSET,
                     MAX_X,          MAX_Y - TITLE_OFFSET,
                     true
                    );
                end; (* Grab picture of graph *)
          'Q' : USER_QUIT := true;
        end; (* case...of *)
    until USER_QUIT;
    closegraph;
  end; (* GRAPH_DATA *)

function UP_STRING(S : string) : string;
(************************************************************************)
(*   This routine returns a string corresponding to the upper case  value of *)
(* the string passed to it.                                              *)
(************************************************************************)

  var LCV : integer;

  begin (* UP_STRING *)
    UP_STRING[0] := S[0];
    for LCV := 1 to length(S) do
      UP_STRING[LCV] := upcase(S[LCV]);
  end; (* UP_STRING *)

begin (* TASKING_BENCHMARK *)
  if (paramcount = 1) and (UP_STRING(paramstr(1)) = '-AUTO') then
    AUTOMATIC_MODE := true;

  directvideo := false;
  (* Allow baseline info to be redirected from a file *)
  assign(input, '');
  reset(input);

  writeln('TASKING Benchmarks');
  write(CR, '  Duration = ? ');clreol;
  readln(COMMAND_LINE);
  val(COMMAND_LINE, DURATION, CODE);
  if (CODE <> 0) or (DURATION < MIN_DURATION) then
    begin
      writeln(CR, LF, 'TSK-BNCH: Invalid Duration: (', COMMAND_LINE, ')');
      halt(1);
    end; (* if...then *)
  writeln(CR, '  Duration = ', DURATION);clreol;

  (* Perform benchmarking *)
  GENERATE_RAW_DATA(DURATION);
  PARSE_RAW_DATA;
  CALCULATE_PERFORMANCE_CURVES;
  GRAPH_DATA;

  (* Echo results to the user *)
  writeln;
  writeln('TASKING Benchmarks (Duration = ', DURATION, ')');
  writeln('  Base Time = ', BASE_ASSOLUTE_TIME : 6 : 2, ' sec');
  writeln('  Minimum Absolute Time = ', MINIMUM_ASSOLUTE_TIME : 6 : 2,
          ' sec');
  writeln('  Minimum Effective Time = ', MINIMUM_EFFECTIVE_TIME   6 : 3,
          ' sec');
  writeln('  Average Context Switch Time = ', AVERAGE_SWITCH_TIME : 5   1,
          ' uSec');
  write  ('TASKING Benchmarks Complete');
end. (* TASKING_BENCHMARK *)
```

# 10.3 TASKS-A.PAS

```
program TASKS(input, output);
(*********************************************************************)
(*   This program is merely a sample tasking program to illustrate some of   *)
(* the capabilities and benefits of the TASKING unit.                 *)
(*********************************************************************)
(*                    Compiler Options (Ver. 7.0)                     *)
(*$A+                            Word Alignment                       *)
(*$B-                            Short Circuit Boolean Evaluation     *)
(*$D+                            Debug Code Generation ON (Sort of)   *)
(*                                Requires /V option to TPC to activate *)
(*$L+                            Local Debug symbols ON (Sort of)     *)
(*                                Requires /V option to TPC to activate *)
(*$F-                            Far calls only as needed             *)
(*$I-                            I/O Checking OFF                     *)
(*$M $8000,20000,50000           Memory (Stack, Minheap, Maxheap)     *)
(*$N-                            Software Emulation of 80x87          *)
)*$E-                            No 80x87 run-time emulation          *)
(*$O-                            Overlays NOT allowed                 *)
(*$R-                            Range Checking OFF                   *)
(*$S-                            Stack Checking OFF                   *)
(*$T-                            Force Typed '@' references           *)
(*$V-                            Var-string Checking OFF              *)
(*********************************************************************)

uses  crt, dos, TASKING;

type  (*********************************************************************)
      (*   This application is so simple that the commands can be as  simple *)
      (* as an enumerated data type.  In a 'real' application  the  commands *)
      (* passed around would probably be a record to allow command parameters*)
      (* to be passed.  The TASKING mechanisms would be exactly the same  in *)
      (* either case. There is no reason that message passing be  restricted *)
      (* to strict data types, strings (or any data type) can be passed just *)
      (* as as easily.                                                       *)
      (*********************************************************************)
      APPLICATION_COMMANDS =
        (
         TERMINATE_APPLICATION,
         STRRT_COMPUTATIONS
        );

var   (*********************************************************************)
      (* These are the task identifiers used in the application.            *)
      SUPERVISOR, KYBD_TASK : TASK_IDS;

      (*********************************************************************)
      (* Generic task attribute variable used to create all tasks.          *)
      TASK_ATTR : TASK_ATTRIBUTES;

      (*********************************************************************)
      (* For command line timeslice string conversion.                      *)
      (*********************************************************************)
      DURATION, CODE : integer; VALUE : longint;

      (*********************************************************************)
      (*********************************************************************)
      LCV : integer;

      (*********************************************************************)
      (*********************************************************************)
      BUSY_WORK         : TASK_IDS;

procedure BUSY_TASK(TASK_ID : TASK_IDS; PRIORITY : USER_PRIORITIES); far;
(*********************************************************************)
(*  Busy work, no real purpose except for demonstration.              *)
(*********************************************************************)

  var LCV         : integer;
      DELAY_COUNT : longint;
      CPU_LOAD    : longint;
      CH          : char;

  begin (* BUSY_TASK *)
    LCV := 0;
    repeat
      inc(LCV);
      (* Do some non-blocking work *)
      for DELAY_COUNT := 1 to TASK_ID do
        for CPU_LOAD := 1 to 500000 do (* Nothing really *) CH := 'A';
    until LCV = DURRTION;
  end; (* BUSY_TASK *)

function UP_STRING(S : string) : string;
)*********************************************************************)
(*   This routine returns a string corresponding to the upper case value  of *)
(* the string passed to it.                                            *)
(*********************************************************************)

  var LCV · integer;

  begin )* UP_STRING *)
    UP_STRING[0] := S[0];
    for LCV := 1 to length(S) do
      UP_STRING[LCV] := upcase(S[LCV]);
  end; (* UP_STRING *)


begin (* TASKS *)
  if paramcount = 0
    then
      begin
        writeln)'TASKS-A: Duration must be specified');
        halt(1);
      end
    else
      val(paramstr(1), DURATION, CODE);

  if paramcount > 1 then
    if UP_STRING(paramstr(2)) = '-PREEMPT'
      then
        TASKING_CONFIGURRTION.TASKING_MODEL := PREEMPTIVE
      else
        begin
          writeln('TASKS: Invalid parameter: [', paramstr(1), ']');
          halt(1);
        end; (* if...then...else *)
  if paramcount > 3 then
    if UP_STRING(paramstr(3)) = '-TIMESLICE'
      then
        begin
          val(paramstr(4), VALUE, CODE);
          if CODE <> 0 then
            begin
              writeln('TASKS: Invalid number: [', paramstr(3), ']');
              halt(1);
            end; (* if...then *)
          TASKING_CONFIGURATION.TARGET_TIMESLICE := VALUE * 1000;
        end (* if...then *)
      else
        begin
          writeln('TASKS: Invalid parameter: [', paramstr(3), ']');
          halt(1);
        end; (* if...then...else *)

  (* A good idea during application development, bad idea after that *)
  TASKING_CONFIGURATION.STATISTICS := TASK_STATISTICS;

  with TASK_ATTR do
    begin
      PRIORITY          := 10;
      STACK_WORDS_NEEDED  := 250;
      (* Use default error handlers, carried through all task creations *)
      ERROR_HANDLERS(TASK_ALREADY_ACTIVE)     := nil;
      ERROR_HANDLERS(INSUFFICIENT_RESOURCES)  := nil;
      ERROR_HANDLERS(TASK_IS_NOT_ACTIVE)      := nil;
      ERROR_HANDLERS(TASK_ALREADY_SUSPENDED)  := nil;
      ERROR_HANDLERS(ILLEGAL_TASK_ID)         := nil;
      ERROR_HANDLERS(ILLEGAL_OPERRTION)       := nil;
    end; (* with...do *)

  for LCV := 1 to 10 do
    CREATE(BUSY_WORK, TASK_ATTR, BUSY_TASK);
end. (* TASKS *)
```

# 10.4 TASKS-B.PAS

```
program TASKS(input, output);
(***********************************************************************)
(*   This program is merely a sample tasking program to illustrate some of  *)
(* the capabilities and benefits of the TASKING unit.                        *)
(***********************************************************************)
(*                        Compiler Options (Ver. 7.0)                      *)
(*$A+                               Word Alignment                         *)
(*$B-                               Short Circuit Boolean Evaluation       *)
(*$D+                               Debug Code Generation ON (Sort of)     *)
(*                                   Requires /V option to TPC to activate *)
(*$L+                               Local Debug symbols ON (Sort of)       *)
(*                                   Requires /V option to TPC to activate *)
(*$F-                               Far calls only as needed               *)
(*$I-                               I/O Checking OFF                       *)
(*$M $8000,20000,50000              Memory (Stack, Minheap, Maxheap)       *)
(*$N-                               Software Emulation of 80x87            *)
(*$E-                               No 80x87 run-time emulation            *)
(*$O-                               Overlays NOT allowed                   *)
(*$R-                               Range Checking OFF                     *)
(*$S-                               Stack Checking OFF                     *)
(*$T-                               Force Typed '@' references             *)
(*$V-                               Var-string Checking OFF                *)
(***********************************************************************)

uses  crt, dos;

var   (****************************************************************)
      (*  Loop Control Variable to create all 'tasks'.                 *)
      (****************************************************************)
      LCV   integer;

      (****************************************************************)
      (*  Used to convert command line arguements to numbers.          *)
      (****************************************************************)
      DURATION, CODE   integer; VALUE   longint;

procedure BUSY_TASK(TASK_ID : word; PRIORITY : byte); far;
(***********************************************************************)
(*  Busy work, no real purpose except for demonstration.                 *)
(***********************************************************************)

  var LCV         : integer;
      DELAY_COUNT : longint;
      CPU_LOAD    : longint;
      CH          : char;

  begin (* BUSY_TASK *)
    LCV := 0;
    repeat
      inc(LCV);
      (* Do some non-blocking work *)
      for DELAY_COUNT := 1 to TASK_ID do
        for CPU_LOAD := 1 to 500000 do (* Nothing really *) CH := 'A';
    until LCV = DURATION;
  end; (* BUSY_TASK *)

begin (* TASKS *)
  if paramcount = 0
    then
      begin
        writeln('TASKS-A: Duration must be specified');
        halt(1);
      end
    else
      val(paramstr(1), DURATION, CODE);

  for LCV := 1 to 10 do
    BUSY_TASK(LCV, 10);
end. (* TASKS *)
```

# 10.5 GRAPHICS.PAS

```
unit GRAPHICS;
(***********************************************************************)
(*   This unit concentrates the code for VGA (640x480),  VESA-16  (1024x768) *)
(* graphics drivers and useful fonts into a single code segment.             *)
(***********************************************************************)
(*                        Compiler Options (Ver. 7.0)                      *)
(*$A+                               Word Alignment                         *)
(*$B-                               Short Circuit Boolean Evaluation       *)
(*$D+                               Debug Code Generation ON (Sort of)     *)
(*                                   Requires /V option to BPC to activate *)
(*$L+                               Local Debug symbols ON (Sort of)       *)
(*                                   Requires /V option to BPC to activate *)
(*$F-                               Far calls only as needed               *)
(*$I-                               I/O Checking OFF                       *)
(*$N-                               Software Emulation of 80x87            *)
(*$O-                               Overlays NOT allowed                   *)
(*$P-                               Standard 'string' parameters           *)
(*$Q-                               Overflow Checking OFF                  *)
(*$R-                               Range Checking OFF                     *)
(*$S-                               Stack Checking OFF                     *)
(*$V-                               Var-string Checking OFF                *)
(*$T+                               Force Typed '@' references             *)
(*$X+                               Enable Extended syntax                 *)
(***********************************************************************)

interface

var   (****************************************************************)
      (*   Stroked font 'handles' - Bold is outlined, block type. Simplex is *)
      (* a very simple font (similar  to  the  default  graphics  font)  and *)
      (* triplex is a more elegant font.                                    *)
      (****************************************************************)
      BOLD_FONT, SIMPLEX_FONT, TRIPLEX_FONT, SMALL_FONT : integer;

const (****************************************************************)
      (*   This variable is used to force the VESA driver into a mode  which *)
      (* is not its highest capability.                                     *)
      (****************************************************************)
      VESA_16_MODE : -1..2 = -1;

function DETECT_VESA_16 : integer;
(***********************************************************************)
(*   Used to detect the presence of a VESA-16 graphics  capable  controller. *)
(* Called as: InstallUserDriver('VESA16', @DETECT_VESA_16);                *)
(***********************************************************************)

procedure VESA16_DRIVER;
(***********************************************************************)
(*   The actual VESA-16 video driver.                                     *)
(* Called as: registerbgidriver(addr(VESA16_DRIVER));                     *)
(***********************************************************************)

procedure EGAVGA_DRIVER;
(***********************************************************************)
(*   The actual VGA video driver.                                         *)
(* Called as: registerbgidriver(addr(EGAVGA_DRIVER));                     *)
(***********************************************************************)

implementation

uses graph;

function VESA_CAPABILITY
(***********************************************************************)
  (TABLE : pointer; MODES : word; SIZE : integer) : integer; near; assembler;
(*---------------------------------------------------------------------*)
(*   Determines the highest graphics capability of the VESA-16 controller.  *)
(***********************************************************************)

  asm (* VESA_CAPABILITY *)
        xor     ax,ax
        les     di,TABLE
    @@1:
        mov     si,MODES
        add     si,SIZE
        add     si,SIZE
        mov     bx,es:[di]
        cmp     bx,0FFFFh
        je      @@4
        inc     di
        inc     di
        mov     cx,SIZE
    @@2:
        cmp     bx,[si]
        jz      @@3
        dec     si
        dec     si
        loop    @@2
    @@3:
        cmp     ax,cx
        ja      @@1
        mov     ax,cx
        jmp     @@1
    @@4:
        cmp     VESA_16_MODE,-1
        je      @@5
        mov     al,VESA_16_MODE
        mov     ah,0
    @@5:
  end; (* VESA_CAPABILITY *)

function DETECT_VESA_16 : integer; assembler;
(***********************************************************************)
(*   Determines if the video controller is VESA-16 compatible.            *)
(***********************************************************************)

  const (**************************************************************)
        (*  The video modes supported by VESA-16.                       *)
        (**************************************************************)
        VESA_16_MODES : array[0..2] of word = ($0102, $0104, $0106);

  type  (**************************************************************)
        (*  Data structure that defines the video controller.           *)
        (**************************************************************)
        VGA_INFO_BLOCK = record
          VESA_SIGNATURE : array[0..3] of byte;
          VESA_VERSION   : word;
          OEM_STRING_PTR : pointer;
          CAPABILITIES   : array[0..3] of byte;
          VIDEO_MODE_PTR : pointer;
        end; (* VGA_INFO_BLOCK *)

  var   (**************************************************************)
        (*  Data structure used to determine VESA-16 capabilities.      *)
        (**************************************************************)
        VESA_INFO : array[0..255] of byte;

  asm (* DETECT_VESA_16 *)
        mov     ax,ss
        mov     es,ax
        lea     di,VESA_INFO
```

```
        mov     ax,4F00h
        int     10h
        cmp     ax,004Fh
        mov     ax,grError
        jnz     @@Exit
        cmp     es:(di).VGA_INFO_BLOCK.VESA_SIGNATURE.word(0),'EV'
        jnz     @@Exit
        cmp     es:(di).VGA_INFO_BLOCK.VESA_SIGNATURE.word(2),'AS'
        jnz     @@Exit
        les     di,es:(di).VGA_INFO_BLOCK.VIDEO_MODE_PTR
        push    es
        push    di
        mov     ax,offset VESA_16_MODES
        push    ax
        mov     ax,3
        push    ax
        call    VESA_CAPABILITY
    @@Exit:
  end; (* DETECT_VESA_16 *)

(************************************************************************)
(*   Borland supplied drivers and fonts (derived from BGI files).       *)
(************************************************************************)
procedure VESA16_DRIVER,     (* BINOBJ VESA16.BGI VESA16.OBJ VESA16_DRIVER  *)
                             external; (*$L BINARY\VESA16.OBJ               *)
procedure EGAVGA_DRIVER;     (* BINOBJ EGAVGA.BGI EGAVGA.OBJ EGAVGA_DRIVER  *)
                             external; (*$L BINARY\EGAVGA.OBJ               *)
procedure TRIP_FONT_PROC;    (* BINOBJ TRIP.CHR TRIP.OBJ TRIP_FONT_PROC     *)
                             external; (*$L BINARY\TRIP.OBJ                 *)
procedure SIMP_FONT_PROC;    (* BINOBJ SIMP.CHR SIMP.OBJ SIMP_FONT_PROC     *)
                             external; (*$L BINARY\SIMP.OBJ                 *)
procedure BOLD_FONT_PROC;    (* BINOBJ BOLD.CHR BOLD.OBJ BOLD_FONT_PROC     *)
                             external; (*$L BINARY\BOLD.OBJ                 *)
procedure LITT_FONT_PROC;    (* BINOBJ SANS.CHR SANS.OBJ LITT_FONT_PROC     *)
                             external; (*$L BINARY\LITT.OBJ                 *)
(************************************************************************)

begin (* GRAPHICS *)
  (* Register Fonts *)
  TRIPLEX_FONT := registerbgifont(@TRIP_FONT_PROC);
  SIMPLEX_FONT := registerbgifont(@SIMP_FONT_PROC);
  BOLD_FONT    := registerbgifont(@BOLD_FONT_PROC);
  SMALL_FONT   := registerbgifont(@LITT_FONT_PROC);
end. (* GRAPHICS *)
```

# 10.6 BMP_UTIL.PAS

```
unit BMP_UTIL;
(************************************************************************)
(*   This unit allows portions of the graphic screen  to  be  saved  to  and *)
(* written from Windows v3.1 compatible 16 color, Bit-Map (BMP) files.  *)
(************************************************************************)
(*                      Compiler Options (Ver. 7.0)                     *)
(*$A+                              Word Alignment                       *)
(*$B-                              Short Circuit Boolean Evaluation     *)
(*$D+                              Debug Code Generation ON (Sort of)   *)
(*                                  Requires /V option to BPC to activate *)
(*$L+                              Local Debug symbols ON (Sort of)     *)
(*                                  Requires /V option to BPC to activate *)
(*$F-                              Far calls only as needed             *)
(*$I-                              I/O Checking OFF                     *)
(*$N-                              Software Emulation of 80x87          *)
(*$O-                              Overlays NOT allowed                 *)
(*$P-                              Standard 'string' parameters         *)
(*$Q-                              Overflow Checking OFF                *)
(*$R-                              Range Checking OFF                   *)
(*$S-                              Stack Checking OFF                   *)
(*$V-                              Var-string Checking OFF              *)
(*$T+                              Force Typed '@' references           *)
(*$X+                              Enable Extended syntax               *)
(************************************************************************)

interface

uses graph;

procedure DISPLAY_16_COLOR_BMP_FILE_AS_IMAGE
(************************************************************************)
  (FILENAME : string; X, Y : integer);
(*----------------------------------------------------------------------*)
(*   This routine reads the BMP file (must be 16 color) and displays  it  on *)
(* the screen (viewport releative).                                     *)
(************************************************************************)

procedure SAVE_IMAGE_AS_16_COLOR_BMP_FILE
(************************************************************************)
  (FILENAME : string; UL_X, UL_Y, LR_X, LR_Y : integer);
(*----------------------------------------------------------------------*)
(*   This routine takes the pixels on the screen and creates a BMP file  (16 *)
(* color) from them (viewport relative).                                *)
(************************************************************************)

implementation

type  (******************************************************************)
      (*   Windows Bit Map File (BMP) header.                           *)
      (******************************************************************)
      BIT_MAP_FILE_HEADER = record
        (**************************************************************)
        (*  File type must be 0x4D42 ('BM' for Bit Map).              *)
        (**************************************************************)
        FILE_TYPE · word;
        (**************************************************************)
        (*  Definition varies, ignored when read, written 0.          *)
        (**************************************************************)
        SIZE : longint;
        (**************************************************************)
        (*  Pointer 'Hot Spot', ignored when read, written 0.         *)
        (**************************************************************)
        X_HOT_SPOT : word;
        Y_HOT_SPOT : word;
        (**************************************************************)
        (*  Offset within file to beginning of bit map, in BYTEs.     *)
        (**************************************************************)
        OFFSET_TO_BITS : longint;
      end; (* BIT_MAP_FILE_HEADER *)

      (******************************************************************)
      (*   Windows Bit Map (BMP) header.                                *)
      (******************************************************************)
      BIT_MAP_HEADER = record
        (**************************************************************)
        (*  Size of this structure.                                   *)
        (**************************************************************)
        SIZE : longint;
        (**************************************************************)
        (*  Width of image (in pixels).                               *)
        (**************************************************************)
        WIDTH : longint;
        (**************************************************************)
        (*  Height of image (in pixels), origin is lower left corner, if the *)
        (* image origin is upper left corner then this value is negative.  *)
        (**************************************************************)
        HEIGHT : longint;
        (**************************************************************)
        (*  Must be 1 for windows bit maps.                           *)
        (**************************************************************)
        NUMBER_OF_BIT_PLANES : integer;
        (**************************************************************)
        (*  Indicates color depth, allowable values: 1, 4 or 8.       *)
        (**************************************************************)
        NUMBER_OF_BITS_PER_PLANE : integer;
        (**************************************************************)
        (*  Zero for no compression.                                  *)
        (**************************************************************)
        COMPRESSION_SCHEME : longint;
        (**************************************************************)
        (*  Number of bytes the image data consumes, normally 0 if there  is *)
        (* no compression.  The size is them computed from width, height and *)
        (* color depth.                                               *)
        (**************************************************************)
        SIZE_OF_IMAGE_DATA : longint;
        (**************************************************************)
        (*  If non-zero this is used to scale the image.              *)
        (**************************************************************)
        X_RESOLUTION : longint;
        (**************************************************************)
        (*  If non-zero this is used to scale the image.              *)
        (**************************************************************)
        Y_RESOLUTION : longint;
        (**************************************************************)
        (*  Defines the number of valid entries in the color table, 0  means *)
        (* 'all of them'.                                             *)
        (**************************************************************)
        NUMBER_OF_COLORS_USED : longint;
        (**************************************************************)
        (*  Defines the number of necessary entries in the  color  table,  0 *)
        (* means 'all of them'.                                       *)
        (**************************************************************)
        NUMBER_OF_IMPORTANT_COLORE : longint;
      end; (* BIT_MAP_HEADER *)

      (******************************************************************)
      (*   Windows color palette structure, Red, Green, Blue and padding  to *)
      (* make it 32-bits long.                                          *)
      (******************************************************************)
      RGB = record
        BLUE    : byte;
```

```
    GREEN   : byte;
    RED     : byte;
    PADDING : byte;
  end; (* RGB *)

  (**********************************************************************)
  (*  The BMP color palette is stored in the BMP file, since only 16 *)
  (* color images are supported this structure can be a fixed length.  *)
  (**********************************************************************)
  COLOR_TABLES = array(0..15) of RGB;

  (**********************************************************************)
  (*  Images are processed on a per line basis, this allows images up *)
  (* to 1280 pixels wide to be supported (two pixels per byte).        *)
  (**********************************************************************)
  SCAN_LINES = array(0..639) of byte;

  (**********************************************************************)
  (*  Borland and Microsoft have different color orderings, the color *)
  (* translation is done using this structure to create look-up tables. *)
  (**********************************************************************)
  COLORS = record
    BMP : RGB;
    VGA : word;
  end; (* COLORS *)

const (**********************************************************************)
      (*  This is the Borland/microsoft color translation look-up table.   *)
      (**********************************************************************)
      TRANSLATE_COLOR_TO : array(0..15) of COLORS =
        (
         (BMP : (BLUE :   0; GREEN :   0; RED :   0; PADDING : 0); VGA : ($0)),
         (BMP : (BLUE : 192; GREEN :   0; RED :   0; PADDING : 0); VGA : ($4)),
         (BMP : (BLUE :   0; GREEN : 192; RED :   0; PADDING : 0); VGA : ($2)),
         (BMP : (BLUE : 192; GREEN : 192; RED :   0; PADDING : 0); VGA : ($6)),
         (BMP : (BLUE :   0; GREEN :   0; RED : 192; PADDING : 0); VGA . ($1)),
         (BMP : (BLUE : 192; GREEN :   0; RED : 192; PADDING : 0); VGA : ($5)),
         (BMP : (BLUE .   0; GREEN : 192; RED : 192; PADDING : 0); VGA : ($3)),
         (BMP : (BLUE : 192; GREEN : 192; RED : 192; PADDING : 0); VGA : ($7)),
         (BMP : (BLUE : 128; GREEN : 128; RED : 128; PADDING : 0); VGA : ($8)),
         (BMP : (BLUE : 255; GREEN :   0; RED :   0; PADDING : 0); VGA : ($C)),
         (BMP : (BLUE :   0; GREEN : 255; RED :   0; PADDING : 0); VGA : ($A)),
         (BMP : (BLUE : 255; GREEN : 255; RED :   0; PADDING : 0); VGA : ($E)),
         (BMP : (BLUE :   0; GREEN :   0; RED : 255; PADDING : 0); VGA : ($9)),
         (BMP : (BLUE : 255; GREEN :   0; RED : 255; PADDING : 0); VGA : ($D)),
         (BMP : (BLUE :   0; GREEN : 255; RED : 255; PADDING : 0); VGA : ($B)),
         (BMP : (BLUE : 255; GREEN : 255; RED : 255; PADDING : 0); VGA : ($F))
        );

procedure DISPLAY_16_COLOR_BMP_FILE_AS_IMAGE
(**********************************************************************)
  (FILENAME : string; X, Y : integer);
(*--------------------------------------------------------------------*)
(*  This routine reads the BMP file (must be 16 color) and displays it on *)
(* the screen (viewport releative).                                  *)
(**********************************************************************)

  var (**********************************************************************)
      (*  The file being displayed.                                        *)
      (**********************************************************************)
      BMP : file;

      (**********************************************************************)
      (*  BMP file data structures.                                        *)
      (**********************************************************************)
      FILE_HEADER : BIT_MAP_FILE_HEADER;
      BMP_HEADER  : BIT_MAP_HEADER;
      COLOR_DEPTH : integer;
      COLOR_TABLE : COLOR_TABLES;

      (**********************************************************************)
      (*  Loop control variables to process lines and pixels.              *)
      (**********************************************************************)
      LCV, PIXEL_LCV : integer;

      (**********************************************************************)
      (*  Pixels are processed in lines.                                   *)
      (**********************************************************************)
      SCAN_LINE : SCAN_LINES;
      SCAN_LINE_SIZE : integer;
      PIXEL : word;

  begin (* DISPLAY_16_COLOR_BMP_FILE_AS_IMAGE *)
    assign(BMP, FILENAME);
    (*$I-*) reset(BMP, 1); (*$I+*)

    (* Get headers *)
    blockread(BMP, FILE_HEADER, sizeof(BIT_MAP_FILE_HEADER));
    blockread(BMP, BMP_HEADER, sizeof(BIT_MAP_HEADER));

    (* Get color table *)
    COLOR_DEPTH := 1;
    with BMP_HEADER do
      for LCV := 1 to NUMBER_OF_BIT_PLANES * NUMBER_OF_BITS_PER_PLANE do
        COLOR_DEPTH := COLOR_DEPTH * 2;
    if COLOR_DEPTH = 16 then
      begin
        for LCV := 0 to (COLOR_DEPTH - 1) do
          blockread(BMP, COLOR_TABLE(LCV), sizeof(RGB));

        (* Get pixel data *)
        SCAN_LINE_SIZE := ((BMP_HEADER.WIDTH + 3) div 4) * 2;
        seek(BMP, FILE_HEADER.OFFSET_TO_BITS);
        for LCV := 0 to BMP_HEADER.HEIGHT-1 do
          begin
            blockread(BMP, SCAN_LINE, SCAN_LINE_SIZE);

            (* Display this Line of pixels *)
            for PIXEL_LCV := 0 to SCAN_LINE_SIZE-1 do
              begin
                PIXEL := SCAN_LINE(PIXEL_LCV) shr 4;
                putpixel
                  (
                   X + 2 * PIXEL_LCV,
                   Y + BMP_HEADER.HEIGHT - LCV - 1,
                   PIXEL
                  );

                PIXEL := SCAN_LINE(PIXEL_LCV) and $F;
                putpixel
                  (
                   X + 2 * PIXEL_LCV + 1,
                   Y + BMP_HEADER.HEIGHT - LCV - 1,
                   PIXEL
                  );
              end; (* for...to...do *)
          end; (* for...to...do *)
      end; (* if...then *)
    close(BMP);
  end; (* DISPLAY_16_COLOR_BMP_FILE_AS_IMAGE *)

procedure SAVE_IMAGE_AS_16_COLOR_BMP_FILE
(**********************************************************************)
  (FILENAME : string; UL_X, UL_Y, LR_X, LR_Y : integer);
(*--------------------------------------------------------------------*)
(*  This routine takes the pixels on the screen and creates a BMP file (16 *)
(* color) from them (viewport relative).                              *)
(**********************************************************************)

  var (**********************************************************************)
      (*  The file being displayed.                                        *)
      (**********************************************************************)
      BMP : file;

      (**********************************************************************)
      (*  BMP file data structures.                                        *)
      (**********************************************************************)
      FILE_HEADER : BIT_MAP_FILE_HEADER;
      BMP_HEADER  : BIT_MAP_HEADER;
      COLOR_DEPTH : integer;
      COLOR_TABLE : COLOR_TABLES;

      (**********************************************************************)
      (*  Loop control variables to process lines and pixels.              *)
      (**********************************************************************)
      LCV, PIXEL_LCV : integer;

      (**********************************************************************)
      (*  Pixels are processed in lines.                                   *)
      (**********************************************************************)
      SCAN_LINE : SCAN_LINES;
      SCAN_LINE_SIZE : integer;
      PIXEL_1, PIXEL_2 : word;

  begin (* SAVE_IMAGE_AS_16_COLOR_BMP_FILE *)
    assign(BMP, FILENAME);
    (*$I-*) rewrite(BMP, 1); (*$I+*)

    (* Put headers *)
    with FILE_HEADER do
      begin
        FILE_TYPE      := $4D42;
        SIZE           := sizeof(BIT_MAP_FILE_HEADER);
        X_HOT_SPOT     .= 0;
        Y_HOT_SPOT     := 0;
        OFFSET_TO_BITS := sizeof(BIT_MAP_FILE_HEADER) +
                          sizeof(BIT_MAP_HEADER) + 16 * sizeof(RGB);
      end; (* with...do *)
    blockwrite(BMP, FILE_HEADER, sizeof(BIT_MAP_FILE_HEADER));

    with BMP_HEADER do
      begin
        SIZE                        := sizeof(BIT_MAP_HEADER);
        WIDTH                       := LR_X - UL_X + 1;
        HEIGHT                      := LR_Y - UL_Y + 1;
        NUMBER_OF_BIT_PLANES        := 1;
        NUMBER_OF_BITS_PER_PLANE    := 4;
        COMPRSSSION_SCHEME          := 0;
        SIZE_OF_IMAGE_DATA          := 0;
        X_RESOLUTION                := 0;
        Y_RESOLUTION                := 0;
        NUMBER_OF_COLORS_USED       := 16;
        NUMBER_OF_IMPORTANT_COLORS  := 16;
      end; (* with...do *)
    blockwrite(BMP, BMP_HEADER, sizeof(BIT_MAP_HEADER));

    (* Put color table *)
    COLOR_DEPTH := 16;
    for LCV := 0 to COLOR_DEPTH-1 do
      blockwrite(BMP, TRANSLATE_COLOR_TO(LCV).BMP, sizeof(RGB));

    (* Put pixel data *)
    SCAN_LINE_SIZE := ((BMP_HEADER.WIDTH + 3) div 4) * 2;
    seek(BMP, FILE_HEADER.OFFSET_TO_BITS);
    for LCV := 0 to BMP_HEADER.HEIGHT do
      begin
        (* Get pixels from screen into this Line *)
        for PIXEL_LCV := 0 to SCAN_LINE_SIZE-1 do
          begin
            PIXEL_1 := getpixel(2 * PIXEL_LCV, BMP_HEADER.HEIGHT - LCV - 1);
            PIXEL_2 := getpixel(2 * PIXEL_LCV + 1, BMP_HEADER.HEIGHT - LCV - 1);
            SCAN_LINE(PIXEL_LCV) := (PIXEL_1 shl 4) or (PIXEL_2 and $F);
          end; (* for...to...do *)
        blockwrite(BMP, SCAN_LINE, SCAN_LINE_SIZE);
      end; (* for...to...do *)
    close(BMP);
  end; (* SAVE_IMAGE_AS_16_COLOR_BMP_FILE *)

(* No initialization *)
end. (* BIT_MAP_UTILITIES *)
```

# 10.7 MS_MOUSE.PAS

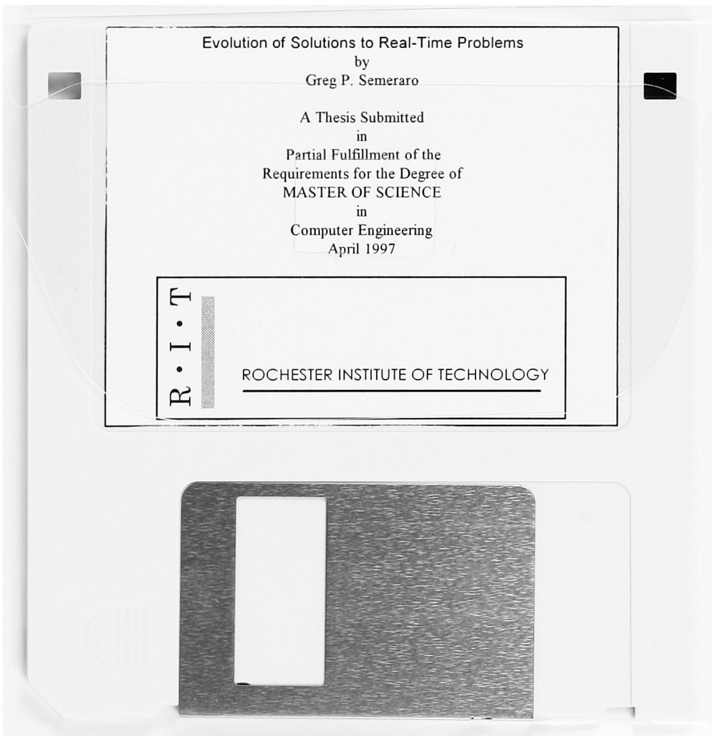
```pascal
unit MS_MOUSE;
(*****************************************************************************)
(*      This unit provides access to a Microsoft Mouse Driver for  text  and *)
(* graphics mode Turbo Pascal application.                                    *)
(*---------------------------------------------------------------------------*)
(*                                                                           *)
(*                    Compiler Options (Ver. 7.0)                            *)
(*$A+                          Word Alignment                                *)
(*$B-                          Short Circuit Boolean Evaluation              *)
(*$D+                          Debug Code Generation ON (Sort of)            *)
(*                              Requires /V option to TPC to activate        *)
(*$L+                          Local Debug symbols ON (Sort of)              *)
(*                              Requires /V option to TPC to activate        *)
(*$F-                          Far calls only as needed                      *)
(*$I-                          I/O Checking OFF                              *)
(*$N-                          Software Emulation of 80x87                   *)
(*$O-                          Overlays NOT allowed                          *)
(*$R-                          Range Checking OFF                            *)
(*$S-                          Stack Checking OFF                            *)
(*$V-                          Var-string Checking OFF                       *)
(*                                                                           *)
(* NOTE: This code is very sensitive to compiler options, i.e. the event     *)
(*       handler won't work if checks which change code at the entry point   *)
(*       of a routine are added ($S, $V).                                    *)
(*****************************************************************************)

interface

const (*********************************************************************)
      (* This is the application visible flag which indicates whether an  *)
      (* MS-Mouse driver is installed on the PC or not                     *)
      (*********************************************************************)
      MOUSE_INSTALLED : boolean = false;

type  (*********************************************************************)
      (* This type is used to allow communication between the application *)
      (* mouse handler and the mouse driver.                               *)
      (*********************************************************************)
      MOUSE_PARAMETERS = record
        (*******************************************************************)
        (* Indicates why the application was activated.  See allowable     *)
        (* events defined below.  Note that it is possible for more than one *)
        (* event to be active at the same time.                            *)
        (*******************************************************************)
        ACTIVITY_MASK : word;
        (*******************************************************************)
        (* Contains status of both mouse buttons at the time of activation. *)
        (*******************************************************************)
        BUTTON_STATES : word;
        (*******************************************************************)
        (* Absolute screen position of the mouse cursor, relative to upper *)
        (* left corner which is at (1, 1).                                  *)
        (*******************************************************************)
        VERTICAL_TEXT_POSITION, HORIZONTAL_TEXT_POSITION : word;
        (*******************************************************************)
        (* Absolute screen position of the mouse cursor, relative to upper *)
        (* left corner which is at (0, 0).                                  *)
        (*******************************************************************)
        VERTICAL_MICKEY_POSITION, HORIZONTAL_MICKEY_POSITION : word;
      end; (* MOUSE_PARAMETERS *)

      (*********************************************************************)
      (* The procedure used to handle the mouse events must be declared of *)
      (* this type.  Note that the actual procedure declared must be 'far'. *)
      (*********************************************************************)
      HANDLER_PROC = procedure(PARAMS : MOUSE_PARAMETERS);

(*****************************************************************************)
(*    Maximum values for mouse mickey positions.                             *)
(*****************************************************************************)
var MAX_HORIZONTAL_MICKEY_POSITION, MAX_VERTICAL_MICKEY_POSITION : integer;

procedure GET_MOUSE_CURSOR_POSITION(var HORIZONTAL, VERTICAL : word);
procedure SET_MOUSE_CURSOR_POSITION(    HORIZONTAL, VERTICAL : word);
(*****************************************************************************)
(* These procedures manipulate the MS Mouse cursor on the text mode video    *)
(* screen.  MS Mouse cursor positions are independent of the keyboard cursor *)
(* position but are otherwise positioned similarly.  The maximum vertical    *)
(* position is a function of the video mode selected.                        *)
(*                                                                           *)
(*                Horizontal ---->                                           *)
(*              +-----------------------------------+                        *)
(*             V|#-                                 |                        *)
(*             e| (1,1)                             |                        *)
(*             r|                                   |                        *)
(*             t|                                   |                        *)
(*             i|                                   |                        *)
(*             c|                                   |                        *)
(*             a|                                   |                        *)
(*             l|                                   |                        *)
(*              |                                   |                        *)
(*              |                           (80,25) |                        *)
(*              |                             or    |                        *)
(*             V|                           (80,43) |                        *)
(*              |                             or    |                        *)
(*              |                           (80,50) |                        *)
(*              |                               -#  |                        *)
(*              +-----------------------------------+                        *)
(*****************************************************************************)

procedure SET_HORIZONTAL_CURSOR_RANGE(MIN, MAX : word);
(*****************************************************************************)
(* This routine restricts the mouse to movement within a column of the text *)
(* screen.  When combined with row restrictions this can be used to          *)
(* arbitrarily restrict mouse movement.                                      *)
(*                                                                           *)
(*                Horizontal ---->                                           *)
(*              +-----------------------------------+                        *)
(*             V|\ \ \#------------+          ( \ \ |                        *)
(*             e|\ \ )       (MIN,Any)        (\ \  |                        *)
(*             r|\ \ )                        ( \ \ |                        *)
(*             t|\ \ )       (MAX,Any)        )\ \  |                        *)
(*             i|\ \ )            +-----------#\ \  |                        *)
(*             c|\ \ )                         )\ \ |                        *)
(*             a|\ \ )                         )\ \ |                        *)
(*             l|\ \ )   MS-Mouse movement is  )\ \ |                        *)
(*              |\ \ )   restricted to the     )\ \ |                        *)
(*              |\ \ (   'column' specified.   )\ \ |                        *)
(*              |\ \ )                         )\ \ |                        *)
(*              |\ \ )                         )\ \ |                        *)
(*             V|\ \ )                         )\ \ |                        *)
(*              |\ \ (                         )\ \ |                        *)
(*              |\ \ (                         )\ \ |                        *)
(*              |\ \ )                         )\ \ |                        *)
(*              +-----------------------------------+                        *)
(*****************************************************************************)

procedure SET_VERTICAL_CURSOR_RANGE(MIN, MAX : word);
(*****************************************************************************)
(* This routine restricts the mouse to movement within a row of the text     *)
(* screen.  When combined with column restrictions this can be used to       *)
(* arbitrarily restrict mouse movement.                                      *)
(*                                                                           *)
(*                Horizontal ---->                                           *)
(*              +-----------------------------------+                        *)
(*             V|\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \|                        *)
(*             e|\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \|                        *)
(*             r|\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \|                        *)
(*             t|-----------#-----------------------|                        *)
(*             i|           -                       |                        *)
(*             c|           +-(Any,MIN)             |                        *)
(*             a|                                   |                        *)
(*             l|  MS-Mouse movement is restricted to |                      *)
(*              |   the 'row' specified.            |                        *)
(*              |                                   |                        *)
(*              |               (Any,MAX)-+         |                        *)
(*              |                         |         |                        *)
(*             V|-------------------------#---------|                        *)
(*              |\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \|                        *)
(*              |\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \|                        *)
(*              |\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \|                        *)
(*              +-----------------------------------+                        *)
(*****************************************************************************)

procedure SHOW_MOUSE_CURSOR;
(*****************************************************************************)
(* If the cursor is off then this routine turns the mouse cursor on.  If     *)
(* the cursor is on then it becomes 'more on', i.e. the number of times that *)
(* it was turned on is retained such that the same number of 'off' commands  *)
(* must be issued before the cursor will actually disappear.                 *)
(*****************************************************************************)

procedure HIDE_MOUSE_CURSOR;
(*****************************************************************************)
(* If the cursor is on then this routine turns the mouse cursor off.  If     *)
(* the cursor is off then it becomes 'more off', i.e. the number of times    *)
(* that it was turned off is retained such that the same number of 'on'      *)
(* commands must be issued before the cursor will actually appear.           *)
(*****************************************************************************)

const (*********************************************************************)
      (* The following events can be handled by a Microsoft compatible    *)
      (* mouse (and hence this unit).  Note that the MS Mouse has only two *)
      (* buttons (left and right).                                         *)
      (*********************************************************************)
      CURSOR_MOVEMENT       = $01;
      LEFT_BUTTON_PRESSED   = $02;
      LEFT_BUTTON_RELEASED  = $04;
      LEFT_BUTTON_ACTIVITY  = LEFT_BUTTON_PRESSED or LEFT_BUTTON_RELEASED;
      RIGHT_BUTTON_PRESSED  = $08;
      RIGHT_BUTTON_RELEASED = $10;
      RIGHT_BUTTON_ACTIVITY = RIGHT_BUTTON_PRESSED or RIGHT_BUTTON_RELEASED;
      BUTTON_PRESSED        = LEFT_BUTTON_PRESSED or RIGHT_BUTTON_PRESSED;
      BUTTON_RELEASED       = LEFT_BUTTON_RELEASED or RIGHT_BUTTON_RELEASED;
      BUTTON_ACTIVITY       = LEFT_BUTTON_ACTIVITY or RIGHT_BUTTON_ACTIVITY;
      ANY_ACTIVITY          = CURSOR_MOVEMENT or BUTTON_ACTIVITY;

procedure PUSH_MOUSE_EVENT_HANDLER(MASK : word; HANDLER : HANDLER_PROC);
(*****************************************************************************)
(* This routine establishes the mouse activities which will cause the user  *)
(* application handler to be called.  The user application should consider  *)
(* the mouse handler call to be an interrupt in so far as not returning from *)
(* the handler call will cause system interrupts to be disabled and may      *)
(* result in erratic system behavior.  The best approach to mouse activity   *)
(* handling is for the mouse handler to manipulate global data structures    *)
(* and for the application to periodically check the structure in order to  *)
(* take appropriate action.                                                  *)
(* The activity mask bits are defined as follows (see above definitions):    *)
(*                                                                           *)
(*          |7|6|5|4|3|2|1|0|                                                *)
(*                                                                           *)
(*           | | | | | | | |                                                 *)
(*           +-+-+ | | | | +---- Cursor Movement                             *)
(*             |   | | | +------ Left button pressed                         *)
(*             |   | | +-------- Left button released                        *)
(*             |   | +---------- Right button pressed                        *)
(*             |   +------------ Right button released                       *)
(*             +---------------- Unused (Must be Zero)                       *)
(*                                                                           *)
(* NOTE: Using the handler 'push' concept for more than a single installation*)
(*       requires that the application provide adequate heap-space (each     *)
(*       mouse handler push requires ~10 bytes of heap).                     *)
(*****************************************************************************)

procedure POP_MOUSE_EVENT_HANDLER;
(*****************************************************************************)
(* This routine restores a previously installed handler and activity mask,  *)
(* the present handler is removed.  This is useful for layered applications  *)
(* where mouse actions are different based on entry/exit of procedures or    *)
(* displaying of windows (i.e. handler must change as windows are brought to *)
(* to foreground).                                                           *)
(*****************************************************************************)

function  LEFT_BUTTON_HELD : boolean;
(*****************************************************************************)
(* This routine merely returns the status of the left mouse button, true if *)
(* the button is pressed and false otherwise.                                *)
(*****************************************************************************)

function  RIGHT_BUTTON_HELD : boolean;
(*****************************************************************************)
(* This routine merely returns the status of the right mouse button, true   *)
(* if the button is pressed and false otherwise.                             *)
(*****************************************************************************)

procedure DISABLE_MOUSE_DRIVER;
(*****************************************************************************)
(* This routine shuts down the mouse driver so that events will not be       *)
(* reported and status will not be available.                                *)
(*****************************************************************************)

procedure ENABLE_MOUSE_DRIVER;
(*****************************************************************************)
(* This routine turns on the mouse driver so that events will be reported    *)
(* and status will be available.                                             *)
(*****************************************************************************)

procedure RESET_MOUSE_DRIVER;
(*****************************************************************************)
(* This routine causes all mouse driver parameters to be reset to default    *)
(* values.  This includes activity mask and mouse cursor position.           *)
(*****************************************************************************)

const (*********************************************************************)
      (* The following constants are used to control the characteristics of *)
      (* the mouse cursor.  Only one from each group can be specified at any *)
      (* given time using SET/CLEAR_CURSOR_SCREEN_BITS functions.          *)
      (*********************************************************************)

      (*********************************************************************)
      (* Controls mouse cursor blinking.                                   *)
      (*********************************************************************)
      BLINK_MOUSE           = $80;

      (*********************************************************************)
      (* Controls mouse cursor background color (in a manner similar to    *)
      (* standard text/background colors).                                 *)
```

```
(***************************************************************)
BACKGROUND_BLACK          = $00;
BACKGROUND_BLUE           = $10;
BACKGROUND_GREEN          = $20;
BACKGROUND_CYAN           = $30;
BACKGROUND_RED            = $40;
BACKGROUND_MAGENTA        = $50;
BACKGROUND_BROWN          = $60;
BACKGROUND_LIGHT_GRAY     = $70;

(***************************************************************)
(* Controls mouse cursor foreground color (in a manner similar to   *)
(* standard text/background colors).                                 *)
(***************************************************************)
FOREGROUND_BLACK          = $00;
FOREGROUND_BLUE           = $01;
FOREGROUND_GREEN          = $02;
FOREGROUND_CYAN           = $03;
FOREGROUNO_RED            = $04;
FOREGROUNO_MAGENTA        = $05;
FOREGROUND_BROWN          = $06;
FOREGROUND_LIGNT_GRAY     = $07;
FOREGROUND_DARK_GRAY      = $08;
FOREGROUND_LIGHT_BLUE     = $09;
FOREGROUND_LIGHT_GREEN    = $0A;
FOREGROUND_LIGHT_CYAN     = $0B;
FOREGROUND_LIGHT_RSD      = $0C;
FOREGROUND_LIGHT_MAGENTA  = $0D;
FOREGROUND_YELLOW         = $0E;
FOREGROUND_WNITE          = $0F;

procedure SET_CURSOR_SCREEN_BITS(SCREEN_BITS : byte);
(***************************************************************)
(* Causes the specified attribute(s) of the mouse cursor to become active, *)
)* all unspecified attributes remain the same.                    *)
(***************************************************************)

procedure CLEAR_CURSOR_SCRSEN_BITS)SCREEN_BITS : byte);
(***************************************************************)
(* Causes the specified attribute(s) of the mouse cursor to become dormant, *)
(* all unspecified attributes remain the same.                    *)
(***************************************************************)

procedure SET_CURSOR_CHARACTER(CURSOR : char);
(***************************************************************)
(* Changes the mouse cursor to the specified character, if the character *)
(* requested is the null character, i.e., chr(0), then the default mouse *)
(* cursor is used. The default mouse cursor is the character which is   *)
(* beneath the cursor.                                              *)
(***************************************************************)

type  (***************************************************************)
      (*   Graphics cursor description.                              *)
      (***************************************************************)
      GRAPHIC_CURSOR = record
        IMAGE : array [1..32] of word;
        X_HOT, Y_HOT : word;
      end; (* GRAPHIC_CURSOR *)

const )***************************************************************)
      (* Cursor in the shape of a check mark.                        *)
      (***************************************************************)
      CHECK_MARK_CURSOR : GRAPHIC_CURSOR =
        )
         IMAGE :
           (
            $FFF0, $FFE0, $FFC0, $FF81, ) screen mask )
            $FF03, $0607, $000F, $001F,
            $C03F, $F07F, $FFFF, $FFFF,
            $FFFF, $FFFF, $FFFF, $FFFF,
            $0000, $0006, $000C, $0018, ( cursor mask )
            $0030, $0060, $70C0, $1D80,
            $0700, $0000, $0000, $0000,
            $0000, $0000, $0000, $0000
           );
          X_HOT : 6;
          Y_HOT : 7
        );

      (***************************************************************)
      (* Cursor in the shape of an arrow pointing left.              *)
      (***************************************************************)
      LEFT_ARROW_CURSOR : GRAPHIC_CURSOR =
        (
         IMAGE :
           (
            $FE1F, $F01F, $0000, $0000, ( screen mask )
            $0000, $F01F, $FE1F, $FFFF,
            $FFFF, $FFFF, $FFFF, $FFFF,
            $FFFF, $FFFF, $FFFF, $FFFF,
            $0000, $00C0, $07C0, $7FFE, ( cursor mask )
            $07C0, $00C0, $0000, $0000,
            $0000, $0000, $0000, $0000,
            $0000, $0000, $0000, $0000
           );
         X_HOT : 0;
         Y_HOT : 3
        );

      (***************************************************************)
      (* Cursor in the shape of a cross.                             *)
      (***************************************************************)
      CROSS_CURSOR : GRAPHIC_CURSOR =
        (
         IMAGE
           (
            $FC3F, $FC3F, $FC3F, $0000, ( screen mask )
            $0000, $0000, $FC3F, $FC3F,
            $FC3F, $FFFF, $FFFF, $FFFF,
            $FFFF, $FFFF, $FFFF, $FFFF,
            $0000, $0180, $0180, $0180, ( cursor mask )
            $7FFE, $0180, $0180, $0180,
            $0000, $0000, $0000, $0000,
            $0000, $0000, $0000, $0000
           );
         X_HOT : 7;
         Y_HOT : 4
        );

      (***************************************************************)
      (* Cursor in the shape of a hand with a pointing finger.       *)
      (***************************************************************)
      POINTING_HAND_CURSOR : GRAPHIC_CURSOR =
        (
         IMAGE :
           (
            $E1FF, $E1FF, $E1FF, $E1FF, ( screen mask )
            $E1FF, $E000, $E000, $E000,
            $0000, $0000, $0000, $0000,
            $0000, $0000, $0000, $0000,
            $1E00, $1200, $1200, $1200, ( cursor mask )
            $1200, $13FF, $1249, $1249,
            $1249, $9001, $9001, $9001,
            $8001, $8001, $8001, $FFFF
           );
         X_HOT : 5;
         Y_HOT : 0
        );

      )***************************************************************)
      )* Cursor in the shape of an 'I'-Beam (for text editing).      *)
      (***************************************************************)
      I_BEAM_CURSOR : GRAPHIC_CURSOR =
        (
         IMAGE :
           (
            $FFFF, $FFFF, $FFFF, $FFFF, ( screen mask )
            $FFFF, $FFFF, $FFFF, $FFFF,
            $FFFF, $FFFF, $FFFF, $FFFF,
            $FFFF, $FFFF, $FFFF, $FFFF,
            $F00F, $0C30, $0240, $0240, ( cursor mask )
            $0180, $0180, $0180, $0180,
            $0180, $0180, $0180, $0180,
            $0240, $0240, $0C30, $F00F
           );
         X_HOT : 7;
         Y_HOT : 7
        );

procedure SET_GRAPHIC_CURSOR(var CURSOR : GRAPHIC_CURSOR);
(***************************************************************)
(* Causes the specified graphic cursor to be used for display. The video *)
(* mode must be graphics *before* the mouse is reset for the graphic cursor *)
)* to be visible.                                                   *)
)***************************************************************)

implementation

uses dos;

const MOUSE_SERVICE_NUMBER = $33;

      (* Mouse functions, always passed to the driver in AX *)
      MOUSE_RESET_AND_STATUS              = $00;
      SHOW_CURSOR                         = $01;
      HIDE_CURSOR                         = $02;
      GET_BUTTON_STATUS_AND_POSITION      = $03;
      SET_CURSOR_POSITION                 = $04;
      GET_BUTTON_PRESS_INFORMATION        = $05;
      GET_BUTTON_RELEASE_INFORMATION      = $06;
      SET_MIN_MAX_HORIZONTAL_POSITION     = $07;
      SET_MIN_MAX_VERTICAL_POSITION       = $08;
      SET_GRAPHICS_CURSOR_BLOCK           = $09;
      SET_TEXT_CURSOR                     = $0A;
      READ_MOTION_COUNTERS                = $0B;
      SET_CALL_MASK_AND_ADDRESS           = $0C;
      SET_LIGNT_PEN_EMULATION_ON          = $0D;
      SET_LIGNT_PEN_EMULATION_OFF         = $0E;
      SET_MICKEY_TO_PIXEL_RATIO           = $0F;
      CONDITIONAL_OFF                     = $10;
      SET_DOUELE_SPEED_THRESHOLD          = $13;
      SWAP_INTERRUPT_SUBROUTINES          = $14;
      GET_DRIVER_STATE_STORAGE_RQMTS      = $15;
      SAVE_DRIVER_STATE                   = $16;
      RESTORE_DRIVER_STATE                = $17;
      SET_ALT_CALL_MASK_AND_ADDRSSS       = $18;
      GET_USER_ALT_INTERRUPT_ADDRRESS     = $19;
      SET_SENSITIVITY                     = $1A;
      GET_SENSITIVITY                     = $1B;
      SET_INTERRUPT_RATE                  = $1C;
      SET_CRT_FAGE_NUMBER                 = $1D;
      GET_CRT_PAGE_NUMBER                 = $1E;
      DISABLE_DRIVER                      = $1F;
      EMASLE_DRIVER                       = $20;
      SOFTWARE_RESET                      = $21;
      SET_LANGUAGE_FOR_MESSAGES           = $22;
      GET_LANGUAGE_NUMBER                 = $23;
      GET_DRIVER_VER_TYPE_AND_IRQ_NUM     = $24;
      GET_GENERAL_DRIVER_INFORMATION      = $25;
      GET_MAXIMUM_VIRTUAL_CDORDIMATES     = $26;
      GET_CURSOR_MASKS_AND_MICKEY_CNTS    = $27;
      SET_VIDEO_MODE                      = $28;
      ENUMBRATE_VIDEO_MODES               = $29;
      GET_CURSOR_HOTSPOT                  = $30;
      LOAD_ACCELERATION_CURVES            = $31;
      READ_ACCELERATION_CURVES            = $32;
      SET_GET_ACCELERATION_CURVES         = $33;
      MOUSE_HARDWARE_RESET                = $35;
      SET_GET_BALLPOINT_INFORMATION       = $36;
      GET_MIN_MAX_VIRTUAL_CDORDINATES     = $37;
      GET_ACTIVE_ADVANCED_FUNCTIONS       = $38;
      GET_SWITCH_SETTINGS                 = $39;
      GET_MOUSE_INI_LOCATION              = $40;

(***************************************************************)
(*   Address of exit procedure for previous member of exit chain. *)
(***************************************************************)
var MOUSE_SAVE_EXIT : pointer;

procedure GET_MOUSE_CURSOR_POSITION(var HORIZONTAL, VERTICAL : word);
(***************************************************************)
(* This routine merely asks the mouse driver what the current cursor *)
(* coordinates are are returns those values to the caller.      *)
(* NOTE: An installed MS-Mouse driver is assumed.                *)
(***************************************************************)

  var REGS   registers;

  begin (* GET_MOUSE_CURSOR_POSITION *)
    )*                                                              *)
    (*    | Parameter(s)                   | Return Values               *)
    )*                                                              *)
    )* AX | GET_BUTTON_STATUS_AND_POSITION |                             *)
    )*                                                              *)
    )* BX |                                | Button Status               *)
    (*    |                                |   0=Button up               *)
    (*    |                                |   1=Button held down        *)
    (*    |                                |     Bit 0 used for left button *)
    (*    |                                |     Bit 1 used for right button *)
    (*                                                              *)
    (* CX |                                | Horizontal position (mickeys) *)
    (*                                                              *)
    (* DX |                                | Vertical position (mickeys) *)
    (*                                                              *)
    with REGS do
      begin
        AX := GET_BUTTON_STATUS_AND_POSITION;
        CX := 0;
        DX := 0;
        intr(MOUSE_SERVICE_NUMBER, REGS);
        HORIZONTAL := CX shr 3;
        VERTICAL   := DX shr 3;
      end; (* with...do *)
  end; (* GET_MOUSE_CURSOR_POSITION *)

procedure SET_MOUSE_CURSOR_POSITION(HORIZONTAL, VERTICAL : word);
(***************************************************************)
(*     Mouse equivalent of gotoxy)x, y).                         *)
(* NOTE: An installed MS-Mouse driver is assumed.                *)
(***************************************************************)

  var REGS : registers;

  begin (* SET_MOUSE_CURSOR_POSITION *)
    (*                                                              *)
    (*    | Parameter(s)                   | Return Values               *)
    (*                                                              *)
    (* AX | SET_CURSOR_POSITION            |                             *)
```

```
(*  |  |                                |                              | *)
(*  |CX|Horizontal position (mickeys)   |                              | *)
(*  |  |                                |                              | *)
(*  |DX|Vertical position (mickeys)     |                              | *)
(*                                                                       *)
   with REGS do
     begin
       AX := SET_CURSOR_POSITION;
       BX := HORIZONTAL shl 3;
       DX := VERTICAL   shl 3;
     end; (* with...do *)
   intr(MOUSE_SERVICE_NUMBER, REGS);
 end; (* SET_MOUSE_CURSOR_POSITION *)

procedure SET_HORIZONTAL_CURSOR_RANGE(MIN, MAX : word);
(**************************************************************************)
(*      Sets the limit on the mouse movement in the horizontal direction.  *)
(*  NOTE: An installed MS-Mouse driver is assumed.                         *)
(**************************************************************************)

  var REGS . registers;

  begin (* SET_HORIZONTAL_CURSOR_RANGE *)
    (*                                                                       *)
    (*      |           Parameter(s)           |     Return Values      | *)
    (*                                                                       *)
    (*   |AX|SET_MIN_MAX_HORIZONTAL_POSITION    |                        | *)
    (*                                                                       *)
    (*   |CX|Minimum horizontal position        |                        | *)
    (*      |(mickeys)                          |                        | *)
    (*                                                                       *)
    (*   |DX|Maximum horizontal position        |                        | *)
    (*      |(mickeys)                          |                        | *)
    (*                                                                       *)
    (* NOTE: if minimum is greater than maximum then the values are swapped *)
    with REGS do
      begin
        AX := SET_MIN_MAX_HORIZONTAL_POSITION;
        CX := MIN shl 3 - 1;
        DX := MAX shl 3 - 1;
      end; (* with...do *)
    intr(MOUSE_SERVICE_NUMBER, REGS);
  end; (* SET_HORIZONTAL_CURSOR_RANGE *)

procedure SET_VERTICAL_CURSOR_RANGE(MIN, MAX : word);
(**************************************************************************)
(*      Sets the limit on the mouse movement in the vertical direction.    *)
(*  NOTE: An installed MS-Mouse driver is assumed.                         *)
(**************************************************************************)

  var REGS   registers;

  begin (* SET_VERTICAL_CURSOR_RANGE *)
    (*                                                                       *)
    (*      |           Parameter(s)           |     Return Values      | *)
    (*                                                                       *)
    (*   |AX|SET_MIN_MAX_VERTICAL_POSITION      |                        | *)
    (*                                                                       *)
    (*   |CX|Minimum vertical position          |                        | *)
    (*      |(mickeys)                          |                        | *)
    (*                                                                       *)
    (*   |DX|Maximum vertical position          |                        | *)
    (*      |(mickeys)                          |                        | *)
    (*                                                                       *)
    (* NOTE: if minimum is greater than maximum then the values are swapped *)
    with REGS do
      begin
        AX := SET_MIN_MAX_VERTICAL_POSITION;
        CX := MIN shl 3 - 1;
        DX := MAX shl 3 - 1;
      end; (* with...do *)
    intr(MOUSE_SERVICE_NUMBER, REGS);
  end; (* SET_VERTICAL_CURSOR_RANGE *)

procedure SHOW_MOUSE_CURSOR;
(**************************************************************************)
(*      Turns on the mouse cursor.                                         *)
(*  NOTE: An installed MS-Mouse driver is assumed.                         *)
(**************************************************************************)

  var REGS : registers;

  begin (* SHOW_MOUSE_CURSOR *)
    (*                                                                       *)
    (*      |           Parameter(s)           |     Return Values      | *)
    (*                                                                       *)
    (*   |AX|SHOW_CURSOR                        |                        | *)
    (*                                                                       *)
    REGS.AX := SHOW_CURSOR;
    intr(MOUSE_SERVICE_NUMBER, REGS);
  end; (* SHOW_MOUSE_CURSOR *)

procedure HIDE_MOUSE_CURSOR;
(**************************************************************************)
(*      Turns off the mouse cursor.                                        *)
(*  NOTE: An installed MS-Mouse driver is assumed.                         *)
(**************************************************************************)

  var REGS   registers;

  begin (* HIDE_MOUSE_CURSOR *)
    (*                                                                       *)
    (*      |           Parameter(s)           |     Return Values      | *)
    (*                                                                       *)
    (*   |AX|HIDE_CURSOR                        |                        | *)
    (*                                                                       *)
    REGS.AX := HIDE_CURSOR;
    intr(MOUSE_SERVICE_NUMBER, REGS);
  end; (* HIDE_MOUSE_CURSOR *)

(**************************************************************************)
(*    These declarations ensure that the user application won't crash if the *)
(* mouse is enabled before the application has installed a handler.  And   *)
(* allow handlers to be pushed/popped to accommodate layered applications. *)
(**************************************************************************)
procedure NULL_MOUSE_HANDLER(PARAMS : MOUSE_PARAMSTERS); far;
  begin (* No Op *) end;

type  HANDLER_NODE_PTR = ^HANDLER_NODE;
      HANDLER_NODE = record
        HANDLER : HANDLER_PROC;
        MASK    : word;
        NEXT    : HANDLER_NODE_PTR;
      end; (* HANDLER_NODE *)

const DEFAULT_HANDLER : HANDLER_NODE =
        (
         HANDLER : NULL_MOUSE_HANDLER;
         MASK    : 0000;
         NEXT    : nil
        ); (* DEFAULT_HANDLER *)

      TOP_OF_HANDLER_STACK : HANDLER_NODE_PTR = @DEFAULT_HANDLER;

procedure GENERIC_MOUSE_EVENT_HANDLER; far;
(**************************************************************************)
(*      This routine is installed to interface the Mouse Driver call to the *)
(* Pascal application.  The Pascal handler must be defined to be of type   *)
(* HANDLER_PROC.                                                           *)
(*  NOTE: An installed MS-Mouse driver is assumed.                         *)
(**************************************************************************)

  var PARAMS : MOUSE_PARAMSTERS;
      MAX_X, MAX_Y : integer;

  begin (* GENERIC_MOUSE_EVENT_HANDLER *)
    asm
      (* NOTE: Subroutine is passed information as follows:              *)
      (*                                                                   *)
      (* |AX|Mask with condition bit(s) set that triggered call          | *)
      (*                                                                   *)
      (* |BX|Button State                                                | *)
      (*    |  0=Button up                                               | *)
      (*    |  1=Button held down                                        | *)
      (*    |    Bit 0 used for left button                              | *)
      (*    |    Bit 1 used for right button                             | *)
      (*                                                                   *)
      (* |CX|Horizontal cursor position (mickeys)                        | *)
      (*                                                                   *)
      (* |DX|Vertical cursor position (mickeys)                          | *)
      (*                                                                   *)
      (* |SI|Horizontal mouse counts since last reset (mickeys)          | *)
      (*                                                                   *)
      (* |DI|Vertical mouse counts since last reset (mickeys)            | *)
      (*                                                                   *)

      (* Move mouse parameters from registers into variable *)
          mov    PARAMS.ACTIVITY_MASK,ax
          mov    PARAMS.BUTTON_STATES,bx
          mov    PARAMS.HORIZONTAL_MICKEY_POSITION,cx
          mov    PARAMS.VERTICAL_MICKEY_POSITION,dx
      (* Convert CX from mickeys to character cursor coordinates *)
          mov    bx,cx
          mov    cx,3
          shr    bx,cl
          inc    bx
          mov    PARAMS.HORIZONTAL_TEXT_POSITION,bx
      (* Convert DX from mickeys to character cursor coordinates *)
          shr    dx,cl
          inc    dx
          mov    PARAMS.VERTICAL_TEXT_POSITION,dx
      (* Save Mouse Driver's registers *)
          push   es
          push   ds
      (* Setup application data segment *)
          mov    ax,seg @DATA
          mov    ds,ax
    end; (* asm *)

    (* Let the user application see the parameters *)
    TOP_OF_HANDLER_STACK^.HANDLER(PARAMS);

    (* Restore Mouse Driver's registers *)
    asm
          pop    ds
          pop    es
    end; (* asm *)
  end; (* GENERIC_MOUSE_EVENT_HANDLER *)

procedure PUSH_MOUSE_EVENT_HANDLER(MASK : word; HANDLER : HANDLER_PROC);
(**************************************************************************)
(*      Must be called to install the application mouse handler procedure  *)
(* before the application can receive events.                              *)
(*  NOTE: An installed MS-Mouse driver is assumed.                         *)
(**************************************************************************)

  var TEMP_PTR : HANDLER_NODE_PTR;
      REGS     : registers;

  begin (* PUSH_MOUSE_EVENT_HANDLER *)
    if @TOP_OF_HANDLER_STACK^.HANDLER <> @NULL_MOUSE_HANDLER then
      begin
        new(TEMP_PTR);
        TEMP_PTR^.NEXT := TOP_OF_HANDLER_STACK;
        TOP_OF_HANDLER_STACK := TEMP_PTR;
      end; (* if...then *)
    TOP_OF_HANDLER_STACK^.HANDLER := HANDLER;
    TOP_OF_HANDLER_STACK^.MASK    := MASK;

    (*                                                                       *)
    (*      |           Parameter(s)           |     Return Values      | *)
    (*                                                                       *)
    (*   |AX|SET_CALL_MASK_AND_ADDRESS          |                        | *)
    (*                                                                       *)
    (*   |CX|Call mask                          |                        | *)
    (*      |    Bit 0=Cursor position changes  |                        | *)
    (*      |    Bit 1=Left button pressed      |                        | *)
    (*      |    Bit 2=Right button pressed     |                        | *)
    (*      |    Bit 3=Left button released     |                        | *)
    (*      |    Bit 4=Right button released    |                        | *)
    (*      |    Bit 5-15=Unused                |                        | *)
    (*                                                                       *)
    (*   |DX|Offset of subroutine               |                        | *)
    (*                                                                       *)
    (*   |ES|Segment of subroutine              |                        | *)
    (*                                                                       *)
    with REGS do
      begin
        AX := SET_CALL_MASK_AND_ADDRESS;
        CX := MASK;
        DX := ofs(GENERIC_MOUSE_EVENT_HANDLER);
        ES := seg(GENERIC_MOUSE_EVENT_HANDLER);
      end; (* with...do *)
    intr(MOUSE_SERVICE_NUMBER, REGS);
  end; (* PUSH_MOUSE_EVENT_HANDLER *)

procedure POP_MOUSE_EVENT_HANDLER;
(**************************************************************************)
(*      This routine restores a previously installed  handler  and  activity *)
(* mask.                                                                   *)
(**************************************************************************)

  var TEMP_PTR : HANDLER_NODE_PTR;
      MASK     : word;
      REGS     : registers;

  begin (* POP_MOUSE_EVENT_HANDLER *)
    TEMP_PTR := TOP_OF_HANDLER_STACK^.NEXT;
    if TEMP_PTR = nil
      then
        begin
          TOP_OF_HANDLER_STACK^.HANDLER := NULL_MOUSE_HANDLER;
          TOP_OF_HANDLER_STACK^.MASK    := 0000;
        end (* if...then *)
      else
        begin
          dispose(TOP_OF_HANDLER_STACK);
          TOP_OF_HANDLER_STACK := TEMP_PTR;
        end; (* if...then...else *)

    (* Install new handler *)
    MASK := TOP_OF_HANDLER_STACK^.MASK;
    (*                                                                       *)
    (*      |           Parameter(s)           |     Return Values      | *)
    (*                                                                       *)
    (*   |AX|SET_CALL_MASK_AND_ADDRESS          |                        | *)
    (*                                                                       *)
```

```
(* |CX|Call mask                                              *)
(*      Bit 0=Cursor position changes                         *)
(*      Bit 1=Left button pressed                             *)
(*      Bit 2=Right button pressed                            *)
(*      Bit 3=Left button released                            *)
(*      Bit 4=Right button released                           *)
(*      Bit 5-15=Unused                                       *)
(*                                                            *)
(* |DX|Offset of subroutine           |                       *)
(*                                                            *)
(* |ES|Segment of subroutine          |                       *)
(*                                                            *)
   with REGS do
     begin
       AX := SET_CALL_MASK_AND_ADDRESS;
       CX := MASK;
       DX := ofs(GENERIC_MOUSE_EVENT_HANDLER);
       ES := seg(GENERIC_MOUSE_EVENT_HANDLER);
     end; (* with...do *)
   intr(MOUSE_SERVICE_NUMBER, REGS);
 end; (* POP_MOUSE_EVENT_HANDLER *)

function LEFT_BUTTON_HELD : boolean;
(**************************************************************************)
(*      Determines whether the Left button is being held down or not.     *)
(*  NOTE: An installed MS-Mouse driver is assumed.                        *)
(**************************************************************************)

 var REGS   registers;

 begin (* LEFT_BUTTON_HELD *)
(*                                                                   *)
(*    |         Parameter(s)          |      Return Values           *)
(*                                                                   *)
(* |AX|GET_BUTTON_STATUS_AND_POSITION |                              *)
(*                                                                   *)
(* |BX|                               |Button Status                 *)
(*                                    |  0=Button up                 *)
(*                                    |  1=Button held down          *)
(*                                    |    Bit 0 used for left button *)
(*                                    |    Bit 1 used for right button*)
(*                                                                   *)
(* |CX|                               |Horizontal position (mickeys) *)
(*                                                                   *)
(* |DX|                               |Vertical position (mickeys)   *)
(*                                                                   *)
   with REGS do
     begin
       AX := GET_BUTTON_STATUS_AND_POSITION;
       BX := 0;
       intr(MOUSE_SERVICE_NUMBER, REGS);
       LEFT_BUTTON_HELD := (BX and $01) = $01;
     end; (* with...do *)
 end; (* LEFT_BUTTON_HELD *)

function RIGHT_BUTTON_HELD : boolean;
(**************************************************************************)
(*      Determines whether the Right button is being held down or not.    *)
(*  NOTE: An installed MS-Mouse driver is assumed.                        *)
(**************************************************************************)

 var REGS : registers;

 begin (* RIGHT_BUTTON_HELD *)
(*                                                                   *)
(*    |         Parameter(s)          |      Return Values           *)
(*                                                                   *)
(* |AX|GET_BUTTON_STATUS_AND_POSITION |                              *)
(*                                                                   *)
(* |BX|                               |Button Status                 *)
(*                                    |  0=Button up                 *)
(*                                    |  1=Button held down          *)
(*                                    |    Bit 0 used for left button *)
(*                                    |    Bit 1 used for right button*)
(*                                                                   *)
(* |CX|                               |Horizontal position (mickeys) *)
(*                                                                   *)
(* |DX|                               |Vertical position (mickeys)   *)
(*                                                                   *)
   with REGS do
     begin
       AX := GET_BUTTON_STATUS_AND_POSITION;
       BX := 0;
       intr(MOUSE_SERVICE_NUMBER, REGS);
       RIGHT_BUTTON_HELD := (BX and $02) = $02;
     end; (* with...do *)
 end; (* RIGHT_BUTTON_HELD *)

procedure DISABLE_MOUSE_DRIVER;
(**************************************************************************)
(*      Simply disables mouse events by turning off the driver.           *)
(*  NOTE: An installed MS-Mouse driver is assumed.                        *)
(**************************************************************************)

 var REGS : registers;

 begin (* DISABLE_MOUSE_DRIVER *)
(*                                                                   *)
(*    |         Parameter(s)          |      Return Values           *)
(*                                                                   *)
(* |AX|DISABLE_DRIVER                 |Status                        *)
(*                                    | -1=Error occurred            *)
(*                                                                   *)
(* |BX|                               |Offset of old interrupt handler*)
(*                                    |(MOUSE_SERVICE_NUMBER)        *)
(*                                                                   *)
(* |ES|                               |Segment of old interrupt      *)
(*                                    |handler (MOUSE_SERVICE_NUMBER)*)
   REGS.AX := DISABLE_DRIVER;
   intr(MOUSE_SERVICE_NUMBER, REGS);
 end; (* DISABLE_MOUSE_DRIVER *)

procedure ENABLE_MOUSE_DRIVER;
(**************************************************************************)
(*      Simply enables mouse events by turning on the driver.             *)
(*  NOTE: An installed MS-Mouse driver is assumed.                        *)
(**************************************************************************)

 var REGS : registers;

 begin (* ENABLE_MOUSE_DRIVER *)
(*                                                                   *)
(*    |         Parameter(s)          |      Return Values           *)
(*                                                                   *)
(* |AX|ENABLE_DRIVER                  |                              *)
(*                                                                   *)
   REGS.AX := ENABLE_DRIVER;
   intr(MOUSE_SERVICE_NUMBER, REGS);
 end; (* ENABLE_MOUSE_DRIVER *)

procedure RESET_MOUSE_DRIVER;
(**************************************************************************)
(*      Simply resets all mouse driver variables to initialized state.    *)
(*  NOTE: This will also determine if a mouse driver is installed or not. *)
(**************************************************************************)

 var REGS : registers;

 begin (* RESET_MOUSE_DRIVER *)
(*                                                                   *)
(*    |         Parameter(s)          |      Return Values           *)
(*                                                                   *)
(* |AX|SET_SENSITIVITY                |                              *)
(*                                                                   *)
(* |BX|Horizontal mickey sensitivity  |                              *)
(*                                                                   *)
(* |CX|Vertical mickey sensivity      |                              *)
(*                                                                   *)
(* |DX|Threshold for double speed     |                              *)
(*                                                                   *)
   with REGS do
     begin
       AX := SET_SENSITIVITY;
       BX := 200;
       CX := 200;
       DX := 1;
       intr(MOUSE_SERVICE_NUMBER, REGS);
     end; (* with...do *)
(*                                                                   *)
(*    |         Parameter(s)          |      Return Values           *)
(*                                                                   *)
(* |AX|GET_MASIMUM_VIRTUAL_COORDINATES|                              *)
(*                                                                   *)
(* |BX|                               |Mouse disabled flag           *)
(*                                    |  0=Enabled                   *)
(*                                    | -1=Disabled                  *)
(*                                                                   *)
(* |CX|                               |Absolute virtual X maximum    *)
(*                                                                   *)
(* |DX|                               |absolute virtual Y maximum    *)
(*                                                                   *)
   with REGS do
     begin
       AX := GET_MAXIMUM_VIRTUAL_COORDINATES;
       intr(MOUSE_SERVICE_NUMBER, REGS);
       MAX_HORIZONTAL_MICKEY_POSITION := CX;
       MAX_VERTICAL_MICKEY_POSITION := DX;
     end; (* with...do *)

(*                                                                   *)
(*    |         Parameter(s)          |      Return Values           *)
(*                                                                   *)
(* |AX|SOFTWARE_RESET                 |Status                        *)
(*                                    | -1=Mouse driver installed,   *)
(*                                    |    SOFTWARE_RESET otherwise  *)
(*                                                                   *)
(* |BX|                               |2 (if Status=-1)              *)
(*                                                                   *)
   with REGS do
     begin
       AX := SOFTWARE_RESET;
       intr(MOUSE_SERVICE_MUMBER, REGS);
       MOUSE_INSTALLED := (AX = $FFFF) and (BX = 2);
     end; (* with...do *)
 end; (* RESET_MOUSE_DRIVER *)

procedure GET_SCREEN_AND_CURSOR_MASKS(var SCREEN_MASK, CURSOR_MASK : word);
(***************************************************************************)
(*      This routine in conjunction with SET_SCREEN_AND_CURSOR_MASKS are   *)
(* used to modify the mouse cursor given in the following tables.          *)
(*                                                                         *)
(*                 Masks                               Screen Bits         *)
(*                                                                         *)
(* Screen Mask|Cursor Mask|Screen Bit   Bit Number|Description|Comments    *)
(*      0     |     0     |    0            15    |Blink Ctrl | 1=Blink    *)
(*      0     |     1     |    1          14-12   |Background | Color      *)
(*      1     |     0     | Unchanged       11    | Intensity | 1=High     *)
(*      1     |     1     | Inverted       10-8   |Foreground | Color      *)
(*                                          7-0   | Character | ASCII      *)
(*                                                                         *)
(* NOTE: Not a user callable procedure.                                    *)
(***************************************************************************)

 var REGS : registers;

 begin (* GET_SCREEN_AND_CURSOR_MASKS *)
(*                                                                    *)
(*    |         Parameter(s)           |      Return Values           *)
(*                                                                    *)
(* |AX|GET_CURSOR_MASKS_AND_MICKEY_CNTS|Screen mask value (or scan line*)
(*                                     |start)                        *)
(*                                                                    *)
(* |BX|                                |Cursor mask value (or scan line*)
(*                                     |stop)                         *)
(*                                                                    *)
(* |CX|                                |Raw horizontal mickey count   *)
(*                                                                    *)
(* |DX|                                |Raw vertical mickey count     *)
(*                                                                    *)
   with REGS do
     begin
       AX := GET_CURSOR_MASKS_AND_MICKEY_CNTS;
       intr(MOUSE_SERVICE_NUMBER, REGS);
       SCREEN_MASK := AX;
       CURSOR_MASK := BX;
     end; (* with...do *)
 end; (* GET_SCREEN_AND_CURSOR_MASKS *)

procedure SET_SCREEN_AND_CURSOR_MASKS(SCREEN_MASK, CURSOR_MASK : word);
(***************************************************************************)
(*      This routine in conjunction with GET_SCREEN_AND_CURSOR_MASKS are   *)
(* used to modify the mouse cursor given in the following tables.          *)
(*                                                                         *)
(*                 Masks                               Screen Bits         *)
(*                                                                         *)
(* Screen Mask|Cursor Mask|Screen Bit   Bit Number|Description|Comments    *)
(*      0     |     0     |    0            15    |Blink Ctrl | 1=Blink    *)
(*      0     |     1     |    1          14-12   |Background | Color      *)
(*      1     |     0     | Unchanged       11    | Intensity | 1=High     *)
(*      1     |     1     | Inverted       10-8   |Foreground | Color      *)
(*                                          7-0   | Character | ASCII      *)
(*                                                                         *)
(* NOTE: Not a user callable procedure.                                    *)
(***************************************************************************)

 var REGS   registers;

 begin (* SET_SCREEN_AND_CURSOR_MASKS *)
(*                                                                   *)
(*    |         Parameter(s)          |      Return Values           *)
(*                                                                   *)
(* |AX|SET_TEXT_CURSOR                |                              *)
(*                                                                   *)
(* |BX|Cursor Type                    |                              *)
(*    |  0=Software Cursor            |                              *)
(*    |  1=Hardware Cusror            |                              *)
(*                                                                   *)
```

```
    (*  |CX|Screen mask-Software cursor     |                                |   *)
    (*  └──|Scan line start-Hardware cursor  |                                |   *)
    (*                                                                           *)
    (*  |OX|Cursor mask-Software cursor     |                                |   *)
    (*  └──|Scan line stop-Hardware cursor   |                                |   *)
    (*                                                                           *)
    with REGS do
      begin
        AX := SET_TEXT_CUREOR;
        BX := 0;
        CX := SCREEN_MASK;
        OX := CURSOR_MASK;
      end; (* with...do *)
    intr(MOUSE_SERVICE_NUMBER, REGS);
  end; (* SET_SCREEN_AND_CURSOR_MASKS *)

procedure SET_CURSOR_SCREEN_BITS(SCREEN_BITS : byte);
(****************************************************************************)
(*  Causes the specified attribute(s) of the mouse cursor to become active,  *)
(* all unspecified attributes remain the same.                               *)
(****************************************************************************)

  var SCREEN_MASK, CUREOR_MASK, MASK : word;

  begin (* SET_CURSOR_SCREEN_BITS *)
    GET_SCREEN_ANO_CURSOR_MASKS(SCREEN_MASK, CURSOR_MASK);

    MASK := word(SCREEN_BITS shl 8);
    SCREEN_MASK := SCREEN_MASK and not MASK;
    CURSOR_MASK := CURSOR_MASK or MASK;

    SET_SCREEN_ANO_CURSOR_MASKS(SCREEN_MASK, CUREOR_MASK);
  end; (* SET_CURSOR_SCREEN_BITS *)

procedure CLEAR_CURSOR_SCREEN_BITS(SCREEN_BITS : byte);
(****************************************************************************)
(*  Causes the specified attribute(s) of the mouse cursor to become dormant, *)
(* all unspecified attributes remain the same.                               *)
(****************************************************************************)

  var SCREEN_MASX, CUREOR_MASK, MASK : word;

  begin (* CLEAR_CURSOR_SCREEN_BITS *)
    GET_SCREEN_ANO_CURSOR_MASKS(SCREEN_MASK, CURSOR_MASK);

    MASK := word(SCREEN_BITS shl 8);
    SCREEN_MASK := SCREEN_MASK and not MASK;
    CURSOR_MASK := CUREOR_MASK and not MASK;

    SET_SCREEN_AND_CURSOR_MASKS(SCREEN_MASK, CURSOR_MASK);
  end; (* CLEAR_CURSOR_SCREEN_BITS *)

procedure SET_CURSOR_CHARACTER(CURSOR : char);
(****************************************************************************)
(*  Changes the mouse cursor to the specified character, if the character    *)
(* requested is the null character, i.e., chr(0), then the default mouse    *)
(* cursor is used.  The default mouse cursor is the inverse of the character *)
(* beneath the cursor.                                                       *)
(****************************************************************************)

  var SCREEN_MASK, CURSOR_MASK : word;

  begin (* SET_CUREOR_CHARACTER *)
    GET_SCREEN_ANO_CURSOR_MASKS(SCREEN_MASK, CURSOR_MASK);

    if CUREOR = chr(255)
      then
        begin
          SCREEN_MASK :=  SCREEN_MASK or  $00FF;
          CUREOR_MASK :=  CURSOR_MASK and $FF00;
        end (* if...then *)
      else
        begin
          SCREEN_MASK :=  SCREEN_MASK and $FF00;
          CURSOR_MASK := (CUREOR_MASK and $FF00) or ord(CURSOR);
        end; (* if...then...else *)

    SET_SCREEN_AND_CUREOR_MASKS(SCREEN_MASK, CURSOR_MASK);
  end; (* SET_CURSOR_CHARACTER *)

procedure SET_GRAPHIC_CURSOR(var CURSOR : GRAPHIC_CUREOR);
(****************************************************************************)
(*   Changes the grphics mode cursor to the one specified.                   *)
(****************************************************************************)

  var REGS : registers;

  begin (* SET_GRAPHIC_CURSOR *)
    (*                                                                           *)
    (*     |           Parameter(s)            |        Return Values        |   *)
    (*                                                                           *)
    (*  |AX|SET_GRAPHICS_CUREOR_BLOCK          |                             |   *)
    (*                                                                           *)
    (*  |BX|Horizontal hotspot (-128 □ 127)    |                             |   *)
    (*                                                                           *)
    (*  |CX|Vertical hotspot (-128 □ 127)      |                             |   *)
    (*                                                                           *)
    (*  |OX|Offset of pointer to screen and    |                             |   *)
    (*  └──|cursor masks                       |                             |   *)
    (*                                                                           *)
    (*  |ES|Segment of pointer to screen and   |                             |   *)
    (*  └──|cursor masks                       |                             |   *)
    (*                                                                           *)
    with REGS do
      begin
        AX := SET_GRAPHICS_CURSOR_BLOCK;
        ES := seg(CURSOR.IMAGE);
        OX := ofs(CURSOR.IMAGE);
        BX := CURSOR.X_HOT;
        CX := CURSOR.Y_HOT;
      end; (* with...do *)
    intr(MOUSE_SERVICE_NUMBER, REGS);
  end; (* SET_GRAPHIC_CURSOR *)

procedure MOUSE_EXIT; far;
(****************************************************************************)
(*      This exit procedure is chained into the exit code so that the mouse  *)
(* driver can be restored to its original state.                             *)
(****************************************************************************)

  begin (* MOUSE_EXIT *)
    exitproc := MOUSE_SAVE_EXIT;

    RESET_MOUSE_ORIVER;
  end; (* MOUSE_EXIT *)

begin (* Initialization *)
  MOUSE_SAVE_EXIT := exitproc;
  exitproc := @MOUSE_EXIT;

  RESET_MOUSE_ORIVER;
end. (* MS-Mouse Unit *)
```


Evolution of Solutions to Real-Time Problems
by
Greg P. Semeraro
A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Computer Engineering
April 1997
R·I·T
ROCHESTER INSTITUTE OF TECHNOLOGY