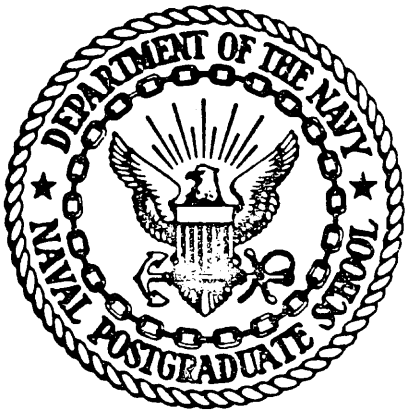
ALL APPLICATION DIGITAL COMPUTER OPERATING SYSTEM DESIGN AND SPECIFICATION

Bernard Janov

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

ALL APPLICATION DIGITAL COMPUTER
OPERATING SYSTEM
DESIGN AND SPECIFICATION

by

Bernard Janov

and

Vernon Michael Johns

June 1974

Thesis Advisor: G. L. Barksdale

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) All Application Digital Computer Operating System Design And Specification		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; June 1974 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Bernard Janov and Vernon Michael Johns		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1974 13. NUMBER OF PAGES 215
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Operating Systems
Structured Programming
Decision Hiding
Hierarchical Structures
All Applications Digital Computer (AADC)
Modularity
PL/I Programmed Operating System

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper discusses the design and specification of a general purpose operating system for the proposed All Application Digital Computer. The objective is to develop a system which is relatively hardware independent, adaptable, comprehensive, maintainable and functional in a multiprocessing, multiprogramming environment. The operating system model is defined by using the techniques of Structured Programming, Decision Hiding, and Multi-Level Hierarchical Ordering. An initial set of functional requirements and system

20. (cont'd)

constraints are postulated from which additional functions are defined and assigned to modules for further specification. The modules are grouped into two classes; system processes which provide services to applications programs such as input/output operations, and primitives which allow for the dynamic creation and control of processes as well as the interprocess communications. Finally, formal parameter specifications are developed which identify the module interfaces, functions, and proposed implementation.

All Application Digital Computer Operating System Design And Specification

by

Bernard Janov
Lieutenant, United States Navy
B.S., Villanova University, 1968

and

Vernon Michael Johns
Lieutenant, United States Navy
B.S., University of Colorado, 1967

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1974

ABSTRACT

This paper discusses the design and specification of a general purpose operating system for the proposed All Application Digital Computer. The objective is to develop a system which is relatively hardware independent, adaptable, comprehensive, maintainable and functional in a multiprocessing, multiprogramming environment. The operating system model is defined by using the techniques of Structured Programming, Decision Hiding, and Multi-Level Hierarchical Ordering. An initial set of functional requirements and system constraints are postulated from which additional functions are defined and assigned to modules for further specification. The modules are grouped into two classes; system processes which provide services to applications programs such as input/output operations, and primitives which allow for the dynamic creation and control of processes as well as the interprocess communications. Finally, formal parameter specifications are developed which identify the module interfaces, functions, and proposed implementation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

I. INTRODUCTION

In the 1960's the Navy saw a proliferation of various types of computers and found itself faced with enormous and expensive computer procurement and support problems. Thus, in 1968 the Naval Air Systems Command undertook the design and development of a modular digital computer system (Advanced Avionic Digital Computer) for future naval air computing requirements. The impetus for the project was cost reduction through the application of standardization and modularity. By using standardized modular hardware and software components, the proposed AADC system could be configured as a simple minicomputer, a complex multiprocessor system, or anything in between. In 1972, the Department of the Navy recognized the potential of this computer and expanded its role from the Advanced Avionic Digital Computer to the All Application Digital Computer (AADC). The AADC system, which is still in the advanced development stage, is intended to satisfy the entire spectrum of Naval Airborne and general purpose computing requirements for the 1978-1990 time period [Refs. 1 and 2].

A. MODULAR OPERATING SYSTEMS

The advent of complex, multipurpose computer systems necessitated the development of an operating system which guided a computer in the performance of its tasks and assisted the applications programs with certain supporting functions. Shaw [Ref. 3] defined an operating system as "an organized collection of (systems) programs that acts as an interface between machine hardware and users, providing users with a set of facilities to simplify the design, coding, debugging, and maintenance of programs; and, at the same time, controlling the allocation of resources to assure efficient operation." The complexity of these operating

systems varied according to the size and purpose of the computer installation; i.e., from single CPU monoprogrammed to multiprocessor, multiprogrammed systems. Most of the existing third generation computer operating systems were designed as highly interdependent, complex systems. Once these systems had been implemented, any changes, improvements, or corrections were difficult and costly. Benson [Ref. 4] described some of the consequences of these complex systems as:

> Instead of the programming task becoming easier with more sophisticated machines, it has become increasingly more difficult. Large and intricate computer programs are being constructed and as a direct consequence the task of demonstrating that the programs are correct, that is they are producing in all cases the correct output for the input data, is nearly impossible.
>
> Another consequence of increased program complexity is the increased cost of design and implementation. Although hardware costs are generally decreasing, the cost of software is increasing dramatically...
>
> In an attempt to solve the above two problems the technique of "structured programming" has been proposed.

The primary advantages of the Modular Design Method ("Structured Programming") have been increased system reliability, reduced complexity, and ease of modification. Consequently, the software development and support costs have been reduced.

B. RESEARCH GOALS

The goals of this research were to develop a modularized, general purpose operating system for the AADC, and to specify the modules and module interfaces in a functional notation. D. L. Parnas [Ref. 5] described the criteria to be used for module specification as:

> 1. The specification must provide to the intended user all the information that he will need to use the program correctly, and nothing more.
>
> 2. The specification must provide to the implementer all the information about the intended use that he needs to complete the program, and no additional information; in particular, no information about the structure of the calling program should be conveyed.

3. The specification must be sufficiently formal that it can conceivably be machine tested for consistency, completeness (in the sense of defining the outcome of all possible uses) and other desirable properties of a specification.

4. The specification should discuss the program in the terms normally used by user and implementer alike rather than some other area of discourse.

These goals were achieved by employing the modular design concept in a "top down" approach to design and implementation. To attain these objectives, the research was organized as follows:

1. Define the design philosophy and heirarchy,

2. Define preliminary specifications for each module,

3. Design the modules,

4. Implement the modules,

5. Test the model,

6. Define formal specifications for each module.

II. AADC OPERATING SYSTEM DESIGN METHODOLOGY

By applying the concept of modularity and the "decision hiding" criteria, the operating system has been defined as a set of hierarchical modules which have facilitated the "top down" design and implementation. As presented in the following sections of this chapter, these techniques have provided the means of expressing a large system as a logical ordering of less complex components.

A. MODULAR DESIGN CONCEPT

The modular design concept has been described by Gouthier and Pont [Ref. 6] as:

> A well defined segmentation of the project effort ensures system modularity. Each task forms a separate, distinct program module. At implementation time each module and its inputs and outputs are well defined; there is no confusion in the intended interface with other system modules. At checkout time the integrity of the module is tested independently; there are few scheduling problems in synchronizing the completion of several tasks before checkout can begin. Finally, the system is maintained in modular fashion; system errors and dificiencies can be traced to specific system modules, thus limiting the scope of detailed error searching.

In recent years, there have been a number of proponents (Dijkstra, Parnas, Brinch Hansen, to name a few) for a modular design approach for developing large programs. The basic justification for this approach has been to allow the design of a system to proceed in a hierarchical way. By treating the system as a set of basic components, specifying the requirements/functions of each and then treating each component as a system (module), the original system has been divided into a set of independent modules. Consequently, the hierarchical organization in the operating system model has simplified module specification, implementation and testing.

Many operating systems in past years have been designed with inadequate methods of program construction. Although specific requirements have been given, the design has proceeded with the impetus on efficient resource utilization at the expense of long term systems reliability and maintainability. This conventional approach has restricted the versatility of the computer system and thereby limited the applications programmer's ability to design more advanced programs requiring extended computer capabilities. Additionally, any attempt to expand or modify the existing operating system to provide more services had been difficult because of the rigid set of specifications in its basic design [Refs. 7 and 8]. D. L. Parnas [Ref. 9] has proposed a solution to the problem as:

> The basic justification for the design methodology presented an old precept from engineering design: a problem must be defined before it is solved. The result was a methodology which laid great stress on specifying the behavior of a system or a component in a system before producing the design...
>
> The premise that we should proceed by specifying the behavior of a system before designing its components implies that we can no longer look at an operating system as an item to be placed on a previously designed piece of hardware. The actual design should begin with a specification of the overall behavior of the hardware-software combination. It continues by dividing the system into components and they, in turn, are designed with little or no attention to the question of what will be hardware and what will be software until very late in the design.

The principle goals of the Modular Design Concept as applied to the development of an AADC operating system have been to define the overall system requirements and to satisfy these requirements by designing and implementing independent modules with well-defined interfaces.

B. THE DECISION HIDING CRITERIA

Unlike the conventional modular design which had defined operating system modules according to function, the decision hiding technique has been used to modularize a large system in a different way. Decision hiding has been proposed as a

means of decomposing a large system into modules based on isolating the decisions made at each stage of the design. Initially, a set of basic decisions has been specified to define the operating system (i.e., multiprocessor, monoprogrammed, shared resources, etc.). Once decision hiding has been applied to obtain a module for a specific decision, the technique may be re-applied to define decisions for submodules.

Modularization by this technique has provided "a mechanism for improving the flexibility and comprehensibility of a system while allowing the shortening of its development time" [Ref. 10]. For example, in the design of a large system program by the conventional method, the system's data structures; i.e., control blocks, queues, etc., have been accessed directly by several modules. Since the data structures have served as interface variables between certain modules, a change in the format has necessitated a corresponding change in all the modules using these structures. The decision hiding criteria added the requirement that each data structure be defined separately and managed by a specific module. Consequently, modification of a data structure format has required a change to only one module, the data structure management module, since the interfaces have been defined explicitly.

C. THE HIERARCHICAL STRUCTURE

The modular design concept using decision hiding has resulted in the decomposition of the operating system into a set of distinct modules. This technique has enhanced system reliability and comprehensibility, increased flexibility and made testing easier. Further improvements in these areas have been achieved by ordering the modules into a hierarchical structure in which the specific relations between the modules have been explicitly defined.

Since one module frequently has provided services to several modules, the system hierarchy has been established by identifying the use or dependency relationships. For example, those modules which identify the basic functions of the operating system have been assigned to the top level - level N. Each subsequent level (N-1,N-2,...,1) has been formed as a collection of those modules that provide the services required in the preceeding level.

Two benefits have been gained from the establishment of a hierarchy. Firstly, the upper levels have been simplified because they use the lower levels as primitive operations. Secondly, the lower levels may be used as a subsystem because they do not require the services of the modules in the higher levels. Therefore, the technique of hierarchical structuring combined with the previously described design techniques have established a means of developing a well-defined operating system.

D. MODULE INTERFACE DECISIONS

The three design methodologies define the technique used in designing a modular operating system. As the functions and requirements were identified and assigned to separate modules, the specification of the module interfaces with the rest of the system was necessary to permit independent module development. These interface decisions define the conventions for passing information between modules; and, in some instances, they specify the data format and type. Once the module interfaces were defined, system design proceeded with strict compliance to these constraints. In this manner, system comprehensibility was enhanced and, although flexibility appears to be limited, the algorithm developed for each module may be readily changed provided the interface requirements are not violated. Since the operating system will be composed of functionally related

modules, the module interface decisions specifically identify the logical flow of information between modules and the set of constraints to be observed in developing them.

III. PRELIMINARY DESIGN SPECIFICATIONS

The essential requirement in preliminary design specification has been defining the purpose for which the AADC operating system was being designed and then to determine the set of basic functions which satisfy this purpose. Since the AADC operating system was to be designed as a multipurpose, modular system which served as an interface between user programs and hardware, the specifications have been determined by considering the requirements of an operating system in a multiprogramming, multiprocessing environment. The decision to design the operating system in this environment was predicated on the fact that monoprogrammed and single processor, multiprogrammed systems could be obtained as logical subsets of the design by restricting the number of user jobs in the system and number of proeessors, respectively.

The first task in developing the preliminary design specifications was to identify the services to be provided to the user and the essential managerial functions to be performed by the operating system given the constraints of multiprogramming and multiprocessing. Firstly, the operating system had to perform monitoring operations to supervise the activity in the system and to detect and rectify, where possible, software or hardware errors. Secondly, a means for managing resources, such as processors, files, input/output devices, space, etc., had to be provided for more efficient resource utilization. Thirdly, a technique for passing information between separate programs (system or user jobs) had to be designed; and finally, the operating system had to provide a simple method for controlling input and output operations, a means by which a user may enter his program into the system and obtain the requested output.

In addition to these functions and services, several requirements have been specified in the AADC Develcpment Project which have a direct effect on the design and implementation of the operating system. In particular, the operating system will be executable on one processor at any point in time; i.e., a dedicated system processor. Secondly, the user has been given the ability to cause the creation of separate processes[1] that may execute independently from the parent process. Another requirement has been the implementaticn of paging techniques; however, the page fault recognition and page replacement algorithms have been scheduled for implementation in hardware [Refs. 2 and 11].

Finally, a standardized, simple communications technique has been specified to facilitate interprocess communication and process to hardware communication. In current operating systems, the communications facilities have varied according to the type of communication; i.e., user processes pass information to system processes via a program interrupt facility, system processes communicate with other system processes via shared tables and lock/unlock mechanisms, etc. Wecker [Ref. 12] has described the problems resulting from these implementations as:

> By building systems with all these varied communication facilities, we tend to overcomplicate and overburden the operating system and the user programs. This nonuniformity of communication techniques within a system leads to problems of synchronization and scheduling and to systems where the overall design and structure become obscure and maintenance becomes difficult...
>
> It would, therefore, be very desirable to build an operating system where data exchange techniques are simplified and which will execute efficiently on our spectrum of multiple processor hardware configurations. These goals can be achieved by having all processes in the system communicate via explicit data exchanges.

---

[1] A process is a task or algorithm which competes for resources and can be characterized by its state and environment.

The communications technique utilized in this operating system has been designed to pass information through the use of message semaphores and buffers [Refs. 3, 8, 12, and 13]. The interprocess communication requirement and the functional requirements identified in the preceeding paragraphs are summarized in Table I.

TABLE I BASIC FUNCTIONAL REQUIREMENTS FOR THE AADC OPERATING SYSTEM

FUNCTION	DEFINITION
Monitoring	Supervise the activity in the system; Detect and rectify hardware and software errors.
Resource Allocation	Supervise the allocation of resources to competing system and user processes.
Input/Output Control	Method for users to enter programs into the system and obtain output.
Communication Requirements	Technique for passing information from process to process, process to hardware, and in operator-system communications.
Multiprogramming and Multiprocessing	Scheduling technique for optimizing system utilization while providing processor execution time equitably.
Dedicated Processor	Operating system executable on only one processor at any point in time.
User Created Processes	Method by which a user process can create independent/dependent user processes.
Paging	(To be implemented in hardware)

Having identified the basic functional requirements of the operating system, the preliminary design specifications were determined in the manner described in the preceeding

chapter. To facilitate design and hierarchical ordering, the modules were divided into two distinct classes (processes and primitives). Process modules were designed to perform specific functions while competing for system resources; whereas the primitive modules were designed to perform the common services required by user and system processes. In particular, primitives were used to provide the mechanism for resource allocation and process communication, and to protect critical sections.

The operating system processes have been assigned to level 2, whereas the primitives which provide services to the processes were assigned to level 1. Within each of these levels, the modules have been ordered hierarchically depending on the services they provided to other modules on the same level. For example, a system monitor module required the services of an input/output control module to receive instructions from or pass information to the computer system operator. In this case, the monitor would be assigned to level 2.2 and the I/O controller to level 2.1. Hence, any module on level 2 may utilize the services provided by modules on level 1; however within the levels, services may be obtained from modules on an adjacent, lower level. The remaining sections of this chapter are concerned with the preliminary design specifications of the operating system modules. The Fundamental Algorithm Technique used by Knuth [Ref. 14] was adopted to define the program logic of the processes and primitives described in the following sections.

A. OPERATING SYSTEM PROCESSES - LEVEL 2

The first stage in the preliminary design specifications for the operating system processes was to determine which of the basic functions could best be performed by a process and then to determine which processes provided services to other

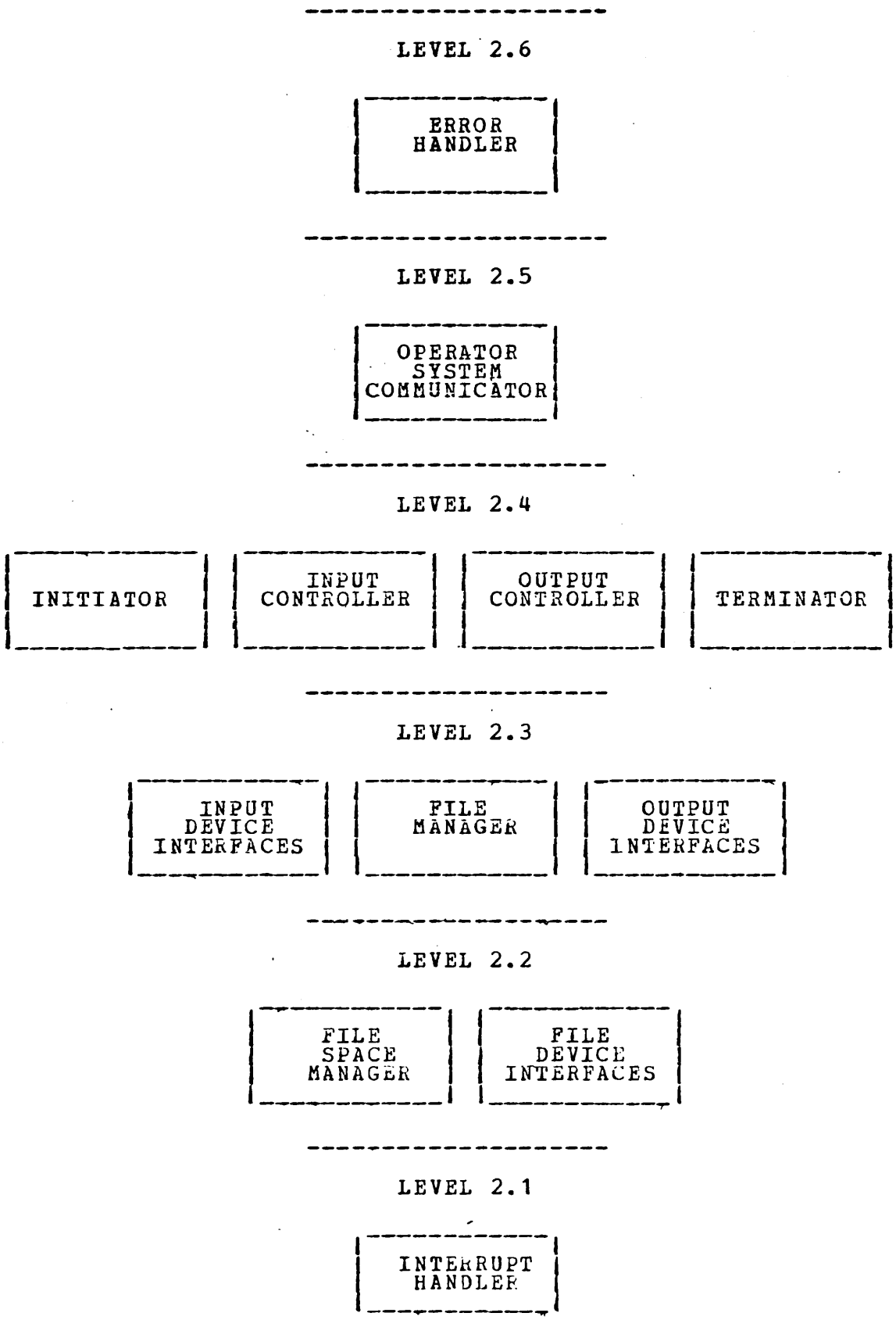


FIGURE 1. THE HIERARCHY OF OPERATING SYSTEM PROCESSES

processes. In particular, the monitoring function and the input/output control functions were selected to be performed by processes. In designing the modules to satisfy these requirements and to provide the capability for multiprogramming-multiprocessing, nine (9) processes were specified. The need for additional processes to serve as interfaces between selected hardware devices (i.e., line printers, card readers, consoles, disk drives, etc.) and system processes was also considered. Figure 1 is the hierarchical ordering of the operating system processes.

1. Error Handler - Level 2.6

In any multi-purpose operating system, there must be an internal method for the handling of user, systems and hardware generated error conditions. Corrective action may be accomplished by canceling the ill-behaved user process, terminating and then replacing a system process, or notifying the system operator of the problem and waiting for instructions. In any case, the action to be performed was determined by the designers of the computer system (including both hardware and software) with particular emphasis on the precise hardware configuration.

Since the physical machine was unspecified, a module to perform these functions was designed to simulate the monitoring function; however the error correction routine has been left undefined. Once the specific actions are identified, the appropriate code may be inserted (provided the interface requirements are not violated). The following algorithm describes the basic logic of the Error Handler:

1.1 [Wait for message] Request(error message).

1.2 [Interpret message] Decode the error message and determine the action to be taken.

1.3 [Recoverable] If correctable Then take predetermined action Otherwise Go to step 1.5.

1.4 [Inform operator] Release (operator message) and Go to step 1.1. (Notify the operator of the error and action taken.)

1.5 [Nonrecoverable] Release (operator message). (Notify the operator of the error and request instructions.)

1.6 [Wait for answer] Request(answer).

1.7 [Answer received] Perform the action specified and Go to step 1.1

ALGORITHM 1 ERROR HANDLER

The "wait for an error condition", "wait for answer", and "notify the operator" were performed in this implementation through the use of the message passing primitives (Request and Release) to be defined later. Since information was being exchanged with the operator, the Error Handler needed the use of an I/O device(s). To hide the actual manner in which this was performed and to simplify the I/O controllers, this service was assigned to an independent module -- the Operator System Communicator.

2. Operator System Communicator - Level 2.5

In addition to providing services to the Error Handler, the Operator System Communicator was designed to assist the computer operator in controlling the system. The operator needed the ability to 'start up' a specific job; to add, delete or modify the system configuration (start up a new disk drive, delete a card reader, or modify a disk pack); or to obtain information concerning the status of queues, availibility of core, etc. The Operator System Communicator was implemented to provide a centralized control point and a uniform way of obtaining system information or giving orders to the system. As in the case of the Error Handler, the full design of this module depended upon knowledge of the possible configuration and the repertoire of "operator-system" instructions. All messages are passed via the Request and Release primitives.

2.1 [Wait for message] Request(operator I/O).

2.2 [Interpret message] Determine who sent the message and the action to be taken.

2.3 [From operator] Perform action specified, and Go to step 2.5.

2.4 [From system] Perform the action specified. (i.e., pass the message to the operator, etc.).

2.5 [Answer requested] If (answer requested) Then Release(answer) . Go to step 2.1.

ALGORITHM 2 OPERATOR SYSTEM COMMUNICATOR

3. Input/Output Controller - Level 2.4

Since there is generally a great disparity between the rate of information transfer among input devices and main storage access time, the concept of spooling has been adopted to assist the system in satisfying the multiprogramming functional requirement. In particular, spooling operations utilize the services of file management and employ the concept of buffering (a buffer is a storage area used to give a better match between processor speeds and I/O device speeds) in accomplishing its function. Watson [Ref. 15] describes the I/O problem as follows:

> The I/O system must cope with a wide variety of devices and therefore is quite complicated at detailed levels of design. This complication arises because of the large number of special situations which can arise in handling communication with each type of device. To create a design which is as conceptually simple as possible, the designer should probably isolate as many device-dependent characteristics as possible in separate routines (often called device drivers) and then interface these routines with more general routines which are device-independent.
>
> One can recognize four major functions in handling I/O devices:
>
> 1 Buffering of information transmitted between I/O devices, auxiliary storage and memory
>
> 2 Proper handling of interrupts or other device to processor signals and their interface to the rest of the system
>
> 3 Reserving and allocating I/O resources

4 Protection of the resources dedicated to one user or process from interference by another.

The Input Controller was designed to "spool" user jobs into a file on some auxiliary device and to enter the job control language (JCL) and various administrative data into a job queue. The Initiator can then retrieve this data and create the user process. To avoid the possibility of "deadlocks" for file space (i.e., the Input Controller gets blocked indefinitely for space to spool the incoming data), the responsibility for requesting file space was assigned to the input device interface processes. Hence, the Input Controller, which is device independent, receives JCL information, file information, etc. from the various interface processes, stores this information in the job queue and sends a message to the Initiator.

3.1 [Wait for a message] Request (output).

3.2 [Interpret message] Determine the action required.

3.3 [JCL] If (JCL or file information) Then store the data and Go to step 3.1.

3.4 [Input for Operator System Communicator] If (Operator input) Then Release(Operator I/O) and Go to step 3.1.

3.5 [EOF] If [End Of File (EOF)] Then enter data in the job queue, Release(New Job) (to the Initiator), and Go to step 3.1.

3.6 [From Operator System Communicator] If (Configuration Modification) Then take appropriate action and Go to step 3.1.

ALGORITHM 3 INPUT CONTROLLER

In item 3.6 of the above algorithm, the message received by the Input Controller was from the Operator System Communicator. The decision was made to have the Input Controller create the interface processes for the input

devices since it had to maintain separate files tc store incoming data from each device. Thus, if the operator entered the command to start up ancther card reader, the Communicator would instruct the Input Ccntroller to do it, wait for an answer that it had been done, and then inform the operator.

The Output Controller was designed to provide a uniform, generalized technique for handling the various requested output operations. In keeping the design simple and device independent, the Controller was implemented to utilize the services of the device interface processes and the File Manager. In particular, the Output Contrcller, upon receiving a request to output a message, assigned a device and then passed the information to the device interface process. In the case of printing files, the data was passed from the File Manager via the Controller tc the preassigned (maybe user or system specified) interface process. In the event a device was not available, the Controller queued the output request.

4.1 [Wait for message] Request(output message).

4.2 [Interpret message] Determine the action to be taken.

4.3 [Operator System Communicator message] Take specified action and Go to step 4.1. (i.e., Release message to the operator, add a new device, etc.).

4.4 [File Manager message] Release(full buffer) and Go to step 4.1. (pass the information to be output to the appropriate device interface process).

4.5 [Terminator message] If (Device available) Then get data frcm print queue, take action to output the job, and Go to step 4.1; Otherwise queue the print job and Go to step 4.1.

ALGORITHM 4 OUTPUT CONTROLLER

4. Initiator - Level 2.4

One of the principle functions of the operating system was to prepare a user program for execution. From the JCL and file information stored in the job queue, the system determined and then allocated the initial resources required by the job. In so doing, the user job was transformed into a process which was assigned an internal identification, given resources or access to them, and entered on the "ready active" queue (a list of processes waiting for a processor). Furthermore, a system table [Process Control Block (PCB)] containing management information about the job was established for use by the operating system as the process proceeded through execution. The Initiator was designed and implemented to prepare the user program for execution.

```
5.1  [Get internal name] Request (Iname)
     (Wait for space for a PCB if
     necessary).

5.2  [Get a job from the job queue]
     Request(Newjob).

5.3  [Interpret JCL] Determine which
     resources are required by the
     process and verify that this process
     may have access.

5.4  [Obtain required resources]
     For (each required resource) Do
     Request(resource).

5.5  [Create Process Control Block]
     Call Create PCB.

5.6  [Activate the process] Call Activate
     (new process).

5.7  [Free job queue space] Release(Job
     queue).

5.8  [Finished] Go to step 5.1.
```

ALGORITHM 5 INITIATOR

5. Terminator - Level 2.4

When a process completes execution (either normally or abnormally), the various resources created by the process or assigned to it were destroyed or deallocated,

respectively. Additionally, any output requirements were satisfied, including operator notification if necessary. Since in the design of the operating system, both system and user processes were given the ability to create subordinate processes, the termination of the parent process necessitated the termination of its progeny (dependent processes). The Terminator was designed to perform these functions when notified by the system or user to terminate a process.

6.1 [Wait for termination message] Request(termination message).

6.2 [Verify message] If (Invalid) Then Release(Error message) and Go to step 6.1.

6.3 [Compile list of processes] Insert process to be terminated and all dependent processes on the list and i <- n (number of items in the list).

6.4 [Select process for termination] Term_Process <- list(i).

6.5 [Print output file] Release(Output).

6.6 [Free all resources] Deallocate resources.

6.7 [Free Internal Name] Release(job name).

6.8 [Finished] i <- i-1; If i = 0 Then Go to step 6.1 Otherwise Go to step 6.4.

ALGORITHM 6 TERMINATOR

6. File Management - Levels 2.3 and 2.2

The design and subsequent implementation of file management techniques simplified the development of a multiprogramming operating system. Serving as an interface between processes and the auxiliary storage devices (disks, drums, tape drives), the File Manager was designed to perform such functions as controlling access to files, creating and destroying files, opening and closing files, and providing backup and restoration services, if possible. Additionally, characteristics of the file, i.e., record

size, data storage methods, etc., and of the device were hidden from the Manager to eliminate file type and device dependencies.

In multiprogramming systems, simultaneous requests from separate processes for access to a shared file required the File Manager to determine if multiple access can be permitted. Since the requests may be for read access, write access, or both, the File Manager was implemented to satisfy multiple read access requests but restricted write access to only one process at a time. As in the case of the Input and Output controllers, the File Manager was assigned the additional function of creating, destroying and supervising the device interface processes which perform the reading and writing of data.

7.1 [Wait for a message] Request (File Operation).

7.2 [Interpret message] Determine action to be taken.

7.3 [Configuration modification] If (configuration modification) Then perform required action and Go to step 7.1. (From Operator System Communicator - add, delete or modify the status of a storage device).

7.4 [New file added] If (new file) Then update master list of known files and Go to step 7.1. (From the File Space Manager - a new file has been added).

7.5 [Directory information] If (directory information) Then update directory and release available space to the File Space Manager as necessary and Go to step 7.1. (From an Interface Process when the storage device is added to the system).

7.6 [Operation on a file] If (file operation) Then [If (access allowed) Then perform operation Otherwise Release (Error message)] Go to step 7.1. (A process has requested a file read, write, open, close, etc. The access check can include deadlock checks as necessary).

ALGORITHM 7 FILE MANAGER

When a process requests access to a file, the File Manager tested for potential deadlock situations (a deadlock occurs when process 1 had been given write access to file A, process 2 had been given write access to file B, and then processes 1 and 2 request access to files B and A, respectively, and consequently, become blocked indefinitely). Many solutions have been presented to solve the deadlock problem, some of which include: requesting and then assigning all resources at the time of process creation, the concept of "sacrificing", and more complex schemes [Refs. 16 and 17]. The concept of "sacrificing" was implemented in this design. In particular, once a process requested a write access which was not immediately serviceable or had write access and then requested read access which was not serviceable, all file resources owned by this process became preemptable. In this scheme, the user must be careful to request file access at points where preemption does not prohibit recovery.

In creating a file, available storage space was found, tagged as being non-available and reserved for access by the process requesting a new file. To prevent the File Manager from being blocked indefinitely while attempting to service a request for space allocation, the responsibility for this function has been assigned to a separate process, File Space Manager. Thus, processes desiring a new file made their request to this process rather than the File Manager.

The File Space Manager was designed to manage the space available in auxiliary storage and to create new files upon request by a process. The allocation-deallocation and accounting functions were implemented as a coordinated operation with the File Manager. In particular, the File Manager was assigned the responsibilty to determine the change in available space as files were destroyed or devices

were added, removed or modified (change in disk pack, tape, etc.) and then, to inform the Space Manager of the change and its effect on available space.

8.1 [Wait for a message] Request (space operation).

8.2 [Interpret message] Determine action to be taken.

8.3 [System configuration change] If (Device deleted) Then Go to step 8.1 Otherwise If (Outstanding space requests) Then Go to step 8.7.

8.4 [Space freed] If (space freed and outstanding space requests) Then Go to step 8.7; Otherwise Go to step 8.1 (from File Manager - a file has been destroyed).

8.5 [Permanent file] If (permanent file) Then [If (space available) Then update available space and Go to step 8.7 Otherwise Release (Error message) and Go to step 8.1.

8.6 [Temporary file] If (temporary file Then [While (space not available) Request (space from File manager)]; Update available temporary space.

8.7 [Create a file] Get an RCB; update space available; update file directory for the selected storage device; and Release (answer) to the requesting process; Go to step 8.1.

ALGORITHM 8 FILE SPACE MANAGER

Upon fielding a request from a process to create a temporary file which required more space then what was available, the Space Manager was blocked until a change in the system occurred or a previously created file was destroyed. When space was not available for a permanent file, an error condition was set and the Space Manager proceeded to service other requests.

7. Interrupt Handler - Level 2.1

The interrupt concept, a second and third generation advance, was devised to facilitate multiprogramming operations. Sayers [Ref. 18] defined an interrupt as "a

break in the normal flow of program execution,.. usually caused by a hardware-generated signal, such as an I/O event, a program error, a machine error, or an operator-initiated signal." The interrupt was used primarily to compensate for the speed differential between I/O operations and central processing by permitting the processor to continue execution of a process while servicing the I/O requests of other processes. For example, a process was blocked upon requesting I/O, the I/O operation was started and the processor was allocated to another process. Once the I/O operation was completed, the system was notified via an interrupt signal. At this point, the process which was blocked on this I/O operation was permitted to preempt the current process and resume execution, or was placed in a "ready-active" queue to wait for a processor.

The Interrupt Handler, as implemented in this operating system, was designed to service various interrupts by determining which interrupt was set and performing any predefined action. The Interrupt Handler, therefore, served as an interface between the hardware devices and the processes using these devices. When an interrupt occured, the Handler was invoked directly (by hardware) to service the interrupt. In particular, the state of the current process was saved, the Handler released a message tc the process waiting for the interrupt and then, control returned to the interrupted process.

9.1 [Wait for interrupt] Wait to be invoked by hardware.

9.2 [Save state of current process]
Save_process <- Current_process;
Call Save_state (Current_process).

9.3 [Reset Current_process]
Current_Process <- Interrupt_Hndlr.

9.4 [Determine action] Identify interrupt and required action. (i.e., which process is concerned with this interrupt and the proper message semaphore).

9.5 [Send message] Release (message to process).

9.6 [Restore the preempted process] Call Restore_state (Save_process) and Go to step 9.1.

ALGORITHM 9 INTERRUPT HANDLER

B. OPERATING SYSTEM PRIMITIVES - LEVEL 1

In a multiprogramming, multiprocessing environment, mechanisms must be provided for the synchronization of processes during information transfer; for allocation and deallocation of resources; for process creation, control and termination; and for protection of critical sections of code, of data structures and of resources. The primitives were designed to perform these functions.

In particular, the technique for interprocess communication was defined through the use of the Request, Release and Allocator primitives; processor allocation was achieved by the Scheduler primitive. The primitives Savestate, Restorestate, Interrupt Enabler and Interrupt Disabler assist in process control and protection. Finally, the various system data structures (i.e., Resource Control Block - RCB, Process Control Block - PCB, etc.) were "hidden" from the rest of the system by providing the primitives RCB Handler and PCB Structures.

A hierarchical organization of the primitives was defined in that various primitives require the services of other ones. For example, all primitives invoke the Interrupt Disabler at the start of execution and the Interrupt Enabler upon completing execution. This was necessary to preclude preemption while executing critical sections of code. Figure 2 represents the primitive hierarchy.

page 35 was a duplicate of 36

at any point in time. Request and Release messages that cannot be immediately matched are queued on the message semaphore and ordered by the priority of the invoking process.

TABLE II. MESSAGE BUFFER

FIELD	PURPOSE
RELEASOR	Internal name of the process releasing a message.
ANSWER-REQUIRED	Indicates if an answer to the release is required.
MESSAGE-SEMAPHORE	Internal identification of the message class identifier to be used in the answer.
BUFFER LOCATION	Identifies the input/output buffer.
DATA FIELDS	Used to pass action identifiers (interface variables); device, process, resource external names; and administrative data.

Request and Release may be considered as a software interrupt scheme utilized by processes. A process invoking Request "enables the interrupt" associated with the specified message semaphore and a process invoking Release "sets the interrupt". After a Request has been matched with a Release, the "message semaphore interrupt" is disabled until Request is invoked again with that semaphore.

The ability to Release/Request a message to/from a specific process was incorporated to provide a usable system with a reasonable number of message semaphores. For example, to perform a file operation using the services of the File Manager, process A only needs the message semaphore ($FILEOP) and the message format utilized by the File Manager; no other information is needed. Process A releases a $FILEOP message (restricted to the File Manager for requests) and specifies the operation desired. If an answer is required for synchronization, the message semaphore process A will use in the Request is also specified. When a

Request from the File Manager is matched with the Release from process A, the message data and A's internal name are provided to the File Manager. Upon completion of the desired operation, a message will be Released to process A specifically, on the desired semaphore. In this manner, several processes may be blocked with a Request on a single message semaphore, such as $WAIT; and another process may Release messages to a specific process, not necessarily the process at the head of the queue.

```
10.1  [Disable Interrupts]
      Call Interrupt_Disabler.

10.2  [Verify semaphore]
      Call RCB_Data ("validate").
      If (Invalid) Then set error code and
      Go to step 10.6.

10.3  [Verify authorization to access]
      Call PCB_Data("access");
      If (Unauthorized) Then Set error
      code and Go to step 10.6.

10.4  [Determine necessary action]
      Call Allocator("data",Match).

10.5  [If a match is found then activate
      the process]  If (Match) Then
      Call PCB_Data("get priority", PRI);
      Call PCB_Data("put status",READYA);
      Call PCB_Data("get type - System or
      User",Type_Proc);
      Call RCB_Put_Queue(PIName,
      Ready_A_Que, Type_Process,
      Priority);
      Call Scheduler.

10.6  [Enable Interrupts]
      Call Interrupt_Enabler.
```

AIGORITHM 10 RELEASE

```
11.1  [Disable Interrupts]
      Call Interrupt_Disabler.

11.2  [Verify semaphore]
      Call RCB_Data ("validate"); If
      (Invalid) Then set error code and Go
      to step 11.6.

11.3  [Verify authorization to access]
      Call PCB_Data("access");
      If (Unauthorized) Then set error
      code and Go to step 11.6.

11.4  [Determine necessary action]
      Call Allocator("data",Match).
```

```
11.5  [Test for a match] If (Match) Then
      Go to step 11.7.

11.6  [Block the process]
      Call PCB_Data("status = BLOCKED");
      Call Savestate("invoking process");
      Call Scheduler.

11.7  [Enable Interrupts]
      Call Interrupt_Enabler.
```

ALGORITHM 11 REQUEST

The Allocator was designed to perform the functions common to Request and Release: matching messages or queueing messages if no match is found. The entry point RCB_Match to RCB Structures was implemented to simplify the Allocator and hide the data structure.

```
12.1  [Check for a match ]
      Call RCB_Match("data").

12.2  [Match and a Request] If (Match &
      Request) Then Do Transfer message to
      Requestor's buffer; and Return.

12.3  [Match and a Release] If (Match &
      Release) Then Do; Transfer message
      to the Blocked Process' Buffer;
      Enter Releasor's internal name in
      the buffer; and Return.

12.4  [No_Match and a Release] If
      (No_Match & Release) then Allocate a
      temporary message buffer; and Enter
      the message data into that buffer.

12.5  [No match-queue message by Priority]
      Call PCB_Data("get priority");
      Call RCB_PutQue("message, priority,
      and semaphore"); Return.
```

ALGORITHM 12 ALLOCATOR

2. Primitive - Scheduler - Level 1.3

The processor which is freed when a process is blocked must be reassigned to another process to efficiently use the processor(s) in a multiprogramming environment. Many scheduling algorithms have been proposed; such factors as the priority and the past I/O behavior of processes can be used to determine which process should be scheduled, or

if a currently running process should be preempted. Other techniques include first-in-first-out (FIFO) scheduling and round-robin scheduling, in which each process is allocated a time-slice on a recurring basis. The scheduling algcrithm is usually tailored to achieve the goal of the system, such as maximizing throughput in a batch system, or immediately starting critical processes in a command/control system. Since the Scheduler is run on the designated operating system processor, that processor must be started last to preclude preempting the Scheduler.

```
13.1  [Initialize variables]
      Processor <- User_Processor(1);
      Que <- User; i <- 2;
      Cycle_finished <- False.

13.2  [Check processor] If (Processcr is
      not allocated) Then Go to step 13.6.

13.3  [Check the queue] Call RCB_Find("get
      priority of the top element in Que);
      If (Que is empty) Then Go to step
      13.7.

13.4  [Check for preemption] If
      (preemption is desirable) Then Call
      Savestate(Current_Process,Processor)
      Otherwise Go to step 13.7.

13.5  [Change status and queue current
      process] Call PCB_Data("put status
      of current process to REDYA"); Call
      RCB_Put_Que("insert current process
      on Que".

13.6  [Start-up highest priority process]
      Call RCB_Get_que("process at the
      head of Que");
      Call PCB_Data("put status of New
      Process to RUNNING");
      Call Restorestate (New Process,
      Processor).

13.7  [Check for a new cycle]
      If (Cycle_Finished) Then Go to step
      13.1.

13.8  [Select next processor]
      If (more user processors) Then Do;
      Processor <- User_processor(i);
      i <- i + 1; End; Else Do;
      Processor <- System_Processor;
      Que <- OS_Que;
      Cycle_Finished <- True; End.

13.9  [Loop] Go to step 13.2.
```

ALGORITHM 13 SCHEDULER

3. Device Directory - Level 1.3

The Device Directory primitive was designed to interface with the Operator System Communicator, Input Controller, Output Controller and File Manager processes to provide information concerning system configuration, I/O devices and device interface processes. During system initialization and, when required, for system reconfiguration, this primitive would be invoked tc cbtain the data used in creating Resource Control Blocks for the devices and Process Control Blocks for the interface processes.

The Directory was designed to store two classes of data; system configuration (menu) and resource data. System configuration data defines several possible configurations which may be selected at system "start_up". For example, Menu A may identify Cardreader1, Lineprinter3, Console2, etc. and Menu B identifies Cardreader2, Lineprinter3, Console1, etc. The resource data contained in the Directory may consist of the interface process's external name, code location, priority, etc. and the device's access qualifier (shared or private), blocksize (buffer length), interrupt identifier, and so forth. This data is used by the Operator System Communicator to identify the devices to be activated and, hence, the message to be sent to the appropriate data transfer controller (Input Controller, Output Controller or File Manager). The various data transfer controllers use this information to create the interface process and device RCB. The Device Directory primitive provides a centralized and standard method for accessing the required information to select, activate, and modify the system configuration; consequently, a modification to the current functions of and services provided by the operating system.

```
14.1 [Disable the interrupts]
     Call Interrupt_Disabler.

14.2 [Identify information type] If
     Menu_Data Then Look_Up("menu");
     Otherwise Look_Up("device").

14.3 [Entry found] If Entry_Found Then
     transfer data Otherwise set error
     code.

14.4 [Enable the interrupts]
     Call Interrupt_Enabler.
```

ALGORITHM 14 DEVICE DIRECTORY

4. Data Structures - Level 1.2

The data structures primitives were designed primarily to eliminate the data dependencies and alleviate the critical section problem that has plagued many second and third generation computer operating systems. For example, concurrent access to a common data element was prevented by designing these modules as non-interruptable primitives and defining interface variables through which the processes and other primitives access or store data in the structures.

Every process and resource is represented in the system by its state, identification and accounting/administrative information. A corresponding data structure was defined to contain this descriptor data for use in basic process and resource operations (i.e., scheduling, message handling, etc.). Tables III and IV represent a flexible and general purpose model structure for process and resource descriptors, respectively. For example, a field may be added or deleted provided the interface variable associated with the field is also appropriately modified. If a data type were changed, code to convert the data to the type expected by other modules need only be added to the data structure module. The Resource Control Block was designed to be functional for the types of rescurces identified

TABLE III. PROCESS CONTROL BLOCK DATA STRUCTURE

FIELD	PURPOSE
EXTERNAL NAME	User/operator specified process name.
PARENT	Internal name of process which created this process or which is the parent of a sibling which created this process.
CHILD	Internal name of a dependent process created by this process or another child of this process.
LEFT SIBLING	Link to related independent processes. Value is a process internal name.
RIGHT SIBLING	Link to related independent processes. Value is a process internal name.
PAGE TABLE POINTER	Address of the process's Page Table.
FILE-SHARE-WRITE COUNTER	Counter of the number of outstanding file open requests for write access to a non-owned, shared file.
PROCESS TYPE	Identifies system versus user process.
RESOURCE VECTOR	Identifies resources accessible or acquired.
FILE ACCESS STATUS	Identifies the process as a reader or writer, or if it is in a sacrificed condition.
MESSAGE BUFFER	Pointer to the process's message buffer.
STATUS	The status of the process in the system; i.e., Blocked, Running, Ready, etc.
PRIORITY	Identifies relative importance of the process; used in queueing and scheduling operations.
QUANTUM	Specified time interval in which a processor is assigned to this process for execution.
CYCLE TIME	Reschedule time period for a recurrent process.
PROCESSOR	Internal name of the processor allocated to the process for execution.
STATE VECTOR	Save area for the initial or last state of execution; i.e., instruction counter and processor register values.

TABLE IV. RESOURCE CONTROL BLOCK DATA STRUCTURE

FIELD	EXAMPLE RESOURCE TYPES AND FIELD USE			
	SEMAPHORE	DEVICE	STORAGE	FILES
ASSIGNED	T. or F.	T. or F.	T. or F.	T. or F.
CREATOR	system proc. iid	system proc. iid	system proc. iid	system proc. iid
OWNER	creator	interface proc. iid	File Manager	proc. iid
EXTERNAL NAME	---	---	---	--
DEVICE STATUS	N/A	active or hold	active or hold	active or hold
DEVICE IID	N/A	N/A	RCB Nr.	RCB Nr.
SHARED-PRIVATE	S. or P.	S. or p.	S. or P.	S. or p.
COUNTER-SIZE	counter	number	space available	file length
STORAGE DEVICE NAME	N/A	external name	N/A	external name
OPEN FILE	N/A	N/A	N/A	Read/Write
FILE TYPE	N/A	N/A	Temporary Permanent	Temporary Permanent both
LEFT QUEUE POINTER	Request messages	process accessing	not used	process accessing
RIGHT QUEUE POINTER	Release messages	process waiting for access	file creation requests	file open requests
	SAMPLE LEFT/RIGHT QUEUE			
FROM	message addressor	proc. iid	proc. iid	proc. iid
TO	message addressee	message semaphore	message semaphore	message semaphore
PRIORITY	proc. pri.	proc. pri.	proc. pri.	proc. pri.
FILE DATA	N/A	N/A	file iid	Read/Write
MESSAGE POINTER	---	N/A	N/A	N/A
UP-LINK	---	---	---	---
DOWN-LINK	---	---	---	---

during the design of the operating system; i.e., semaphores (message class identifiers), devices (readers, writers, consoles, etc.), secondary storage devices (tapes, drums, disk packs, etc.), and files.

```
15.1     [Disable interrupts]
         Call Interrupt_Disabler.

15.2     [Create a PCB]
         If (action = Create_PCB) Then
         Begin.

15.2.1   [Check for space and authorization]
         If((space available) & (Process has
         authorization)) Then assign a PCB
         Number and enter data passed;
         Else set Error code.

15.2.2   [Finished] Go to step 15.8; End.

15.3     [Find PCB Number corresponding to an
         External Name]
         If (action = Find_PIName) Then Do;
         Search active PCBs for specified
         external name; If (found) Then
         PIName_Parameter <- PCB_Number;
         Else PIName_Parameter <- 0; Go to
         step 15.8; End.

15.4     [Check validity of PCB Number]
         If (PCB Number is invalid) Then Do;
         Set Error code; go to step 15.8;
         End.

15.5     [Free a PCB]
         If (action = Release_PCB) Then
         Begin.

15.5.1   [Check authorization]
         If (Process has authorization) Then
         set specified PCB status to
         inactive; Else set Error code.

15,5.2   [Finished] Go to step 15.8; End.

15.6.    [Put data into a PCB]
         If (action = Put_Data) Then Begin.

15.6.1   [Check authorization]
         If (Process has authorization) Then
         store the data; Else set Error code.

15.6.2   [Finished] Go to step 15.8; End.

15.7     [Get data from a PCB]
         If (action = Get_Data) Then
         Data_Parameter <= Desired PCB Field.

15.8     [Enable interrupts]
         Call Interrupt_Enabler.
```

ALGORITHM 15 PCB STRUCTURES

```
16.1      [Disable interrupts]
          Call Interrupt_Disabler.

16.2      [Create an RCB]
          If (action = Create_RCB) Then Begin.

16.2.1    [Check if space is available]
          If (space for specified RCB type is
          available) Then
          assign RCB and enter data;
          Else set Error code.

16.2.2    [Finished] Go to step 16.14.

16.3      [Find an RCB Number corresponding to
          an External Name]
          If (action = Find_RIName) Then
          Begin.

16.3.1    [Search active RCBs]
          Search active RCBs for the External
          Name specified.

16.3.2    [Return value]
          If (External Name is found) Then
          RIName_Parameter <- RIName;
          Else RIName_Parameter <- 0.

16.3.3    [Finished] Go to step 16.14; End.

16.4      [Check validity of RCB Number]
          If (Invalid RCB_Number) Then Do;
          set Error code;
          go to step 16.14; End.

16.5      [Free an RCB]
          If (action = Release_RCB) Then set
          RCB_Number specified to unassigned.

16.6      [Enter data in an RCB]
          If (action = Put_Data) Then
          enter specified data element.

16.7      [Retrieve data from an RCB]
          If (action = Get_Data) Then
          Data_Parameter <= desired RCB field.

16.8      [Find a Process in a queue]
          If (action = Find_Proc) Then Begin.

16.8.1    [Search the specified Queue for the
          specified PIName]
          Search Queue for PIName;
          If (found) Then return the data and
          queue position;
          Else return "Process not found".

16.8.2    [Finished] Go to step 16.14.
```

```
16.9      [Find the Process at the specified
          position in a Queue]
          If (action = Find_At_Position) Then
          Begin.

16.9.1    [Find the Queue element at the
          specified position]
          Find the specified Queue element;
          If ( specified Queue element is
          active) Then return data in the
          Queue element; Else return "Queue
          element not active".

16.9.2    [Finished] Go to step 16.14; End.

16.10     [Enter an element in a Queue]
          If (action = Put_Que) Then enter the
          data in the specified Queue

16.11     [Remove an element from a queue]
          If (action = Get_Que) Then Begin.

16.11.1   [Find the specified queue element]
          Find Queue element;
          If (Queue element is found) Then
          Remove the specified Queue element;
          Else return "element not found".

16.11.2   [Finished] Go to step 16.14; End.

16.12     [Transfer a queue element from the
          specified queue to the opposite
          queue] If (action = Transfer_Que)
          Then Begin.

16.12.1   [Find the specified queue element]
          Find Queue element;
          If (Queue element is found) Then Do;
          Remove Queue element; Insert Queue
          element on the opposite queue; End;
          Else return "element not found".

16.12.2   [Finished] Go to step 16.14; End.

16.13     [Compare specified data with queue
          element data in a specified queue]
          If (action = Match) Then Begin.

16.13.1   [Check each active element in the
          queue for compatible data ]
          Match_Found <- False;
          For i = (first queue element) Until
          (Last queue element) Do;
          compare data;
          If (compatible) Then Do;
          Match_Found <- True; exit For loop;
          End; End;
          If (Match_Found) Then Remove queue
          element and return data;
          Else return "match not found".

16.13.2   [Finished] End.

16.14     [Enable interrupts]
          Call Interrupt_Enabler.
```

ALGORITHM 16 RCB HANDLER

5. Primitive-Hardware Interface - Level 1.1

A processor has a normal complement of registers used to fetch, select and execute instructions, and a mechanism for interrupt recognition. To provide to the operating system the facilities for altering process execution (i.e., preempt and schedule/reschedule processes) and to protect vital operations on commonly accessed data structures, a set of primitives have been designed to enable and disable interrupts and to save, set and reset the processor registers. The primitives, Interrupt_Enabler, Interrupt_Disabler, Savestate and Restorestate, defined below, perform these functions when invoked by processes and other primitives.

```
17.1  [Specific or all interrupts]
      If (Interrupt# = 0) Then Enable all
      the interrupts in the current
      Savevector; Otherwise Enable
      Interrupt#.
```

ALGORITHM 17 INTERRUPT ENABLER

```
18.1  [Specific or all interrupts]
      If (Interrupt# = 0) Then Disable all
      interrupts and save status in
      Savevector; Otherwise Disable
      Interrupt#.
```

ALGORITHM 18 INTERRUPT DISABLER

```
19.1  [Disable all interrupts]
      Call Interrupt_Disabler.

19.2  [Get copy of processor registers]
      Temp_regs = CPUREGS(Processor#).

19.3  [Save copy in PCB]
      Call PCB_Data("temp_regs",process).

19.4  [Enable the interrupts]
      Call Interrupt_Enabler.
```

ALGORITHM 19 SAVESTATE

```
20.1  [Disable all interrupts]
      Call Interrupt_Disabler.

20.2  [Get copy of saved Statevector]
      Call PCB_Data("temp_Regs",Process).

20.3  [Set processor registers]
      CPUREGS(Processor#) = Temp_Regs.

20.4  [Enable the interrupts]
      Call Interrupt_Enabler.
```

ALGORITHM 20 RESTORESTATE

IV. DESIGN VALIDATION

This chapter is concerned with implementation of the modules specified in Chapter III, testing the design correctness and demonstrating the feasibility of the proposed operating system model. Time constraints precluded the development of the entire system, so the decision was made to verify selected functions (i.e., interprocess communication and synchronization using semaphores and messages). For example, the Error Handler was implemented to receive error messages and to generate messages for the system operator, but error correction routines were left undefined. Similarly, the Operator System Communicator module was provided a minimum set of control message codes to identify selected actions (add or delete a device, etc.) and to handle message passing. Certain primitives, including those which perform the primary functions of dynamic creation and control of processes and resources as well as interprocess communication, were implemented in detail. The criterion that modules were to be independent allowed the implementation to proceed along a path through the processes to the primitives with incomplete or partially developed modules at several nodes. Model testing was performed using three additional modules: a hardware simulator, an initialization procedure and a preprocessor macro. The Formal Module Specifications with accompanying PL/I implementations, and the test modules are described in appendices C and D, respectively.

A. THE HARDWARE SIMULATOR

The Hardware Driver module was designed to perform hardware functions such as recognition of interrupts and execution of code at an address specified by the instruction counter. Additionally, code was included to create the

desired system state so that system behavior could be validated.

One of the processor "registers" was used as an instruction counter to indicate the current status of program execution and another "register" was used to identify the process which was scheduled to execute. An integer mapping was used to identify the processes (i.e., the Error Handler was identified as process one (1), the Operator System Communicator as process two (2), etc.). The process identification and instruction counter were separated to avoid encoding and decoding information in the Hardware Driver, and to allow the model to be expandable to simulate paging.

B. THE INITIALIZATION MODULE

A special module was implemented for initializing interface variables and for creating the environment necessary to start up the system. PCB's were created for the operating system processes as were the semaphore RCB's they required for interprocess communication. The initialization procedure in the PCB Structures module was used to create the PCB for the Error Handler; the PCB's for the remaining system processes were created dynamically in the Initialization routine with the Error Handler identified as the running process. Message and I/O buffers were allocated and assigned to each of the system processes, and the internal identification of the resources required by each process were entered in their respective access vectors.

The initialization process could have been implemented using PL/I initial attributes (initializing variables at compile time) within the various modules. However, in consonance with the Decision Hiding Criteria, the initial-

ization of the interface variables was implemented in a separate module. Additionally, the initial configuration was more easily modified during the design and testing phase.

C. THE PREPROCESSOR MACRO

A preprocessor macro (SIMULTR) was used to adapt the PL/I modules for testing. Simulation interrupt points (SIM_INTERRUPT_PTs) were placed in the modules where a process might be blocked (i.e., after invoking the primitive Request). The macro was implemented to include the code necessary to generate a label at the selected interrupt points, set the simulation instruction counter to the appropriate value and exit the module. The additional statements SIM_START and SIM_END were necessary to generate the code to initialize the simulation and generate declarations.

D. TESTS OF SELECTED OPERATING SYSTEM FUNCTIONS

Selected system processes were implemented to verify interprocess communication techniques. Since the test program required to realistically demonstrate this function would encompass the operations performed by Request, Release, Allocator, PCB Structures and RCB Handler, these primitives were implemented in detail to provide a basis for testing the model. Subsequently, test programs were designed and implemented to assure the correctness of the data storage and retrieval operations, queue manipulation techniques, and message handling procedures.

In particular, the Initialization Module was invoked to initialize the interface variables, establish the PCB's for the selected processes and to create RCB's for the message class identifiers (semaphores). Once the initial environment was established, the ability to send messages to a specific process was tested by multiple invocations of

Release with a specific semaphore, a varying addressee, and a unique message. Request was then invoked with the same semaphore and both specific and general addressees to verify the logic of Release, Allocator and RCB Structures. The roles of Release and Request were then reversed. The primitives were tested in a similar manner using messages with general addressees.

Having successfully tested these primitives, the Scheduler primitive and the Hardware Driver were implemented to schedule and simulate running the selected system processes. In particular, a test program was included in the Hardware Driver to simulate the system operator entering commands to add an input device, an output device, and a disk unit. These messages were released to the Input Controller to be passed to the Operator System Communicator. The system processes, upon being selected for execution and then invoked by the Hardware Driver, initiated a Request for a message. Since there was a matching Release on the queue for the Input Controller, it was received, interpreted and passed to the Operator System Communicator which in turn received and interpreted the message. The simulation continued until the initial three messages released by the Hardware Driver and the subsequent responses were received by the appropriate system processes.

The write statements generated by the Preprocessor Macro were used to trace the flow of messages from the Input Controller to the Operator System Communicator, and from that process to the Input Controller, Output Controller and File Manager. The logic used to create the interface processes was not validated; only the sequence in which the messages were requested, released and received was verified. Message flow to the Error Handler was also demonstrated during the preliminary system tests; however, comprehensive error generation and validation was not accomplished.

V. SUMMARY AND RECOMMENDATIONS

The emergence of sophisticated, versatile computer systems has resulted in the need for operating systems which are efficient, adaptable, maintainable and general purpose. Adaptability permits expansion or modification within reasonable cost and time constraints. Maintainability makes system error detection and correction more feasible. Generality must exist if the operating system is to be useful to the many classes of potential users.

A. SUMMARY

As presented in Chapter II, the first step in the design of an operating system was the identification and specification of the overall system behavior in a hardware-software environment. Before specifying and implementing the system modules, a set of primary functions were postulated and the design was then carried cut by employing the techniques of Structured Programming, Decision Hiding and Hierarchical Ordering. At each level of development, new functions were identified and the decision was made to either implement the functions within the scope of the current module being designed or to implement it in a separate module. The principle consideration for specifying new modules was the need for having one module perform common services for the other modules, thus reducing the system complexity.

The modules were grouped into two classes (processes and primitives) and ordered hierarchically. As the design proceeded, hardware dependencies were identified and program stubs were inserted to identify the areas of incomplete specification. However, the functions to be performed at these stubs were defined so that the appropriate code may be

written and inserted as the target machine and system configuration are specified. In this manner, it was possible to accomplish the goal of designing a well defined general purpose operating system.

Though the goal was far reaching, it did not seem practical to approach the design of an operating system by considering each segment of the system independently (for example, designing interprocess communication without considering data structures and data access methods). The design proceeded in a "top-down" fashion, specifying modules to perform the various functions required by the previously defined modules. As a result of having a set of machine independent module definitions, it was then possible to implement and test the postulated function of each module separately at first, and then collectively. Finally, formal specifications identifying the parameters, data structures, module interfaces, and primary functions were developed for each module.

Many of the features of the design are of particular interest. The interprocess communication modules define the techniques which aid in resource acquisition and process control (interrupt) and which provide the ability for synchronization of asynchronous operations. The data structures modules are the focal point for accessing, transferring and storing of data while allowing for the design and implementation of other modules which are independent of the data structures. This latter case defines a mechanism for generating adaptive changes to an existing data structure without necessitating major changes to other modules providing the change is within the interface constraints.

B. RECOMMENDATIONS

As stated previously, the primary goal of this project was the design of a general purpose, modular operating system which, at least conceptually, was hardware independent. Since module functions were postulated within a set of generalized constraints, extentions of this work may proceed in several directions. Firstly, each module specification should be more closely investigated to determine if a better technique for performing the function may replace the proposed one.

Secondly, investigation into the hardware versus software dilemma should continue with the emphasis on identifying operating system functions which, if performed by hardware, would enhance system performance without degradation of system flexibility, adaptability, and generality. In particular, the primitives Request and Release as designed in this system are two functions which could conceivably be implemented in hardware. Additionally, disabling and enabling interrupts, saving and restoring process states should be performed by simple machine instructions.

Thirdly, a target machine should be specified for which a simulation model may be developed to test the correctness of the design decisions in a real time and multiprocessing, multiprogramming environment. The stubs should be replaced with workable test programs to support the simulation.

The programming language PL/I was used for module implementation and testing rather than a conventional use of an assembler language. The use of PL/I enhanced the comprehensability of the program logic and provided several mechanisms which aided in module implementation and testing; i.e., based and pointer variables, on conditions, generic

entry pcint specifications, and preprocessor functions. These high level language constructs and the addition of a construct for supporting queueing operations are recommended for inclusion in CS-4 (see Appendix B). Additionally, a compiler for the language should be designed which employs the latest techniques for code optimization since ccmpile time is, in general, less critical than execution time.

APPENDIX A: GLOSSARY

Access Vector: PCB data element that identifies the resources and primitives which may be used by the process.

Active: A status value which indicates that a process is waiting for a processor.

Algorithm: A prescribed set of well-defined rules for the solution of a problem in a finite number of steps.

Assignment Vector: PCB data element which identifies the resources a process currently controls, must be a subset of the access vector.

Blocked-For-Timer (BlockedT): A status value which indicates that a process is waiting for a deadline.

Buffer: A temporary storage area used for the transmission of data.

Child: PCB data element that identifies the first dependent process in the list of related dependent processes.

Cycle Time: PCB data element that identifies the reschedule time interval for a recurrent process.

Deadline: The latest time at which the execution of a process may begin.

Deadlock: The condition that results from the allocation of resources among certain processes in such a way that it is impossible to grant additional requests to these processes.

File: A collection of related data items treated as a unit.

Hardware: Physical equipment, e.g., mechanical, electrical, or electronic devices.

Interface: The linkage and conventions established for communication between two independent elements, usually between a process and another process, computer operator, I/O device, etc.

Interrupt: A break in the normal flow of execution usually caused by a hardware-generated signal.

JCL: Job Control Language - used to provide job specifications.

Job Queue: Contains the JCL and administrative data of a process prior to its creation.

Left Sibling: PCB data element that links the process to and identifies a related independent process having the same parent.

LSI: Large scale integration.

Module: One buliding block or logical unit which is used in the construction of a system.

Multiprocessing: The use of two or more processors to logically or functionally divide processes and to simultaneously execute various processes or segments of processes asynchronously.

Multiprogramming: The use of the computing system to perform interleaved execution of two or more different processes which simultaneously contend for system resources.

Operating System: A set of programs and routines which guide a processor in the performance of its tasks and assist the programs and programmers with supporting functions.

Page: A process section of convienient size for transmission between secondary storage and main storage.

Page Table: PCB data element that contains the location of the process' pages in secondary storage.

Parent: PCB data element that identifies the process on which this process is immediately dependent.

PCB: Process Control Block, a collection of control data concerning a process.

Primative: An algorithm that is invoked by a process and is executed as a part of the invoking process.

Preemption: The seizing of resources previously allocated to processes.

Print Queue: Contains the administrative data identifying processes' output files.

Priority: PCB data element that identifies the order of precedence for competing processes.

Process: An algorithm which requires resources and can be characterized by its state and environment.

Processor State Vector: PCB data element that specifies the necessary information to start (restart) the process; i.e., processor's registers contents.

Quantum: PCB data element that identifies a limited or algorithmically specified time interval during which a processor is assigned to a process in a multiprogramming environment for sharing the processor among competing processes.

Queue: An ordered or unordered list of processes waiting for some resource or service.

RCB: Resource Control Block, a collection of data concerning a resource.

Resource: Any facility of the computer system or operating system required by a process.

Right Sibling: PCB data element that links the process to and identifies a related independent process having the same parent.

Running: A status value which indicates that a process is in execution.

Semaphore: Identifies a resource or class of resources.

Software: Programs or routines to be executed on computer hardware.

Spooling: A technique for interleaving I/O operations and process execution.

Status: PCB data element that identifies the current state of the process in the system.

Stub: Denotes a logical break in a program at which point a subprogram or macro call may be inserted when the functional subspecification is implemented.

Suspend: A status value which indicates that a process is not contending for resources and cannot be scheduled for execution until a system or process imposed condition has been satisfied.

APPENDIX B: MODELLING LANGUAGE

The work done by D. L. Parnas in the development of SODAS [Ref. 9] and the high level language operating systems, such as the Burroughs MCP, MULTICS and Project SUE, have provided the motivation for modelling and implementing the AADC operating system in a high level language. Furthermore, the AADC program specifications have included the development of a new high level language, tentatively called CS-4, and the design and implementation of an operating system for AADC in the proposed language [Refs. 11 and 19]. Basically, CS-4 has developed as an extension to the Navy Tactical Compiler Monitor System (CMS-2) with the impetus on utilizing the proposed AADC hardware features and facilitating the programming of efficient executive and applications programs [Ref. 20].

A comparison of selected features of CS-4 to those defined in the languages available at the Naval Postgraduate School has been the basis for the selection of a high level language for modelling. The results of the comparison, presented in Table V, have led to the selection of PL/I in that it is more representative of CS-4 than the other languages. Additionally, PL/I has provided a number of I/O control, storage allocation and system control features that facilitated testing and simulating the model. Those features of CS-4 considered germaine to the modelling effort are defined below:

1) Compound Data Structures: A hierarchical set of variables that refer to an aggregate of data items that may or may not have different attributes (data types),

2) Logical, Boolean, Arithmetic and Conditional Operators: A set of symbols each specifying an operation to be performed; the result of which depends upon the type of data and context in which it occurs, i.e.,
IF {[(A-1) > B] OR (C=3)} THEN ...
the minus sign, -, is arithmetic, the 'OR' is boolean

and the greater than, >, and equal, =, signs are conditional operators,

3) External Data Declaration: An explicit or contextual declaration of an identifier such that it is only known within the scope of the declaration,

4) External Procedure Declaration: An explicit or contextual declaration of a procedure such that the procedure is only known within the scope of the declaration,

5) Variable Size Array Declarations at Runtime: The dimension of an array is determined and space allocated during execution of the program,

6) Flow Control: The ability to control the execution of a specific instruction or set of instructions:

   i. Algolic Case Statment
   ii. If and compound If Statment
   iii. Do Statment
   iv. While Statment,

7) Limited Scope Variables: An identifier which is known only within the scope of its declaration,

8) Macro Definition: A compile time feature which provides the ability to specify an instruction or set of instructions which replace the macro name where it occurs in the source program,

9) Character String: A string composed of zero or more characters from the complete set of characters whose bit configuration is recognizable by the computer system in use,

TABLE V. COMPARISON OF SELECTED HIGH LEVEL LANGUAGES

CS-4 FEATURE	ALGOL	FORTRAN	XPL	PL-360	PL/I	APL
Compound Data Structures	Yes[1]	No	No	No	Yes	Yes[1]
Logical, Boolean, Conditional, and Arithmetic Operators	Yes[1]	Yes[1]	Yes[1]	Yes[1]	Yes[1]	Yes[1]
External Data Declaration	No	Yes[1]	No	Yes[1]	Yes	No
External Procedure Declaration	No	Yes[1]	No	Yes[1]	Yes	No
Variable Size Array Declaration-Runtime	No	No	No	No	Yes[1]	Yes[1]
Flow Control	Yes	Yes[1]	Yes	Yes	Yes[1]	Yes
Limited Scope Variables	Yes[1]	Yes[1]	Yes[1]	Yes[1]	Yes	Yes[1]
Macro Definition	No	No	Yes[1]	Yes[1]	Yes[1]	Yes[1]
Character String	Yes	No	Yes	Yes	Yes	Yes

1. Limited Implementation

APPENDIX C: MODULE SPECIFICATION AND IMPLEMENTATION

PREPROCESSOR FUNCTION

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

    NAME CHANGER PREPROCESSOR MACRO, USED TO SHORTEN
    IDENTIFIERS WITH THE EXTERNAL ATTRIBUTE TO SEVEN
    CHARACTERS DUE TO A PL/I RESTRICTION

    "ENAELED" BY  %INCLUDE NAMCHGR;                                  */

%DECLARE(
          CURRENT_PROCESS,
          CREATE_RCB,
          DESTROY_RCB,
          DEVICE_DIRECTORY,
          DEV_L_LIMIT,
          DEV_U_LIMIT,
          ERROR_HANDLER,
          FILE_L_LIMIT,
          FILE_MANAGER,
          FILE_SPACE_MANAGER,
          FILE_U_LIMIT,
          FIND_PINAME,
          FIND_INAME,
          INITIALIZATION,
          INITIATOR,
          INPUT_CONTROLLER,
          INTERRUPT_HANDLER,
          OPERATOR_SYSTEM_COMMUNICATOR,
          OUTPUT_CONTROLLER,
          PCBSTRINT,
          PCT_L_LIMIT,
          PCT_U_LIMIT,
          PRIMITIVE_ACTIVATE,
          PRIMITIVE_ALLOCATOR,
          PRIMITIVE_CREATE_PCB,
          PRIMITIVE_DESTROY,
          PRIMITIVE_GETQUE,
          PRIMITIVE_INTERRUPT_DISENABLER,
          PRIMITIVE_INTERRUPT_ENABLER,
          PRIMITIVE_PCBDATA,
          PRIMITIVE_PUTQUE,
          PRIMITIVE_RCBDATA,
          PRIMITIVE_RCB_HANDLER,
          PRIMITIVE_RCB_MATCH,
          PRIMITIVE_RELEASE,
          PRIMITIVE_RESTORESTATE,
          PRIMITIVE_REQUEST,
          PRIMITIVE_SAVESTATE,
          PRIMITIVE_SCHEDULER,
          PROCESSOR,
          RCB_BITDATA,
          RCB_CHARDAT,
          RCB_FIND,
          RCB_FIXEDAT,
          RCB_TRANSFERQ,
          SEM_LIMIT,
          SEM_L_LIMIT,
          SYS_PROCESSOR,
          TERMINATOR
     ) CHARACTER ;
```

```
%CREATE_RCB = 'CREAT_R';
%CURRENT_PROCESS = 'CURPROC' ;
%DESTROY_RCB = 'DSTRY_R';
%DEVICE_DIRECTORY = 'DEV_DIC';
%DEV_L_LIMIT = 'LDEVLIM';
%ERROR_HANDLER = 'ERRHDLR';
%DEV_U_LIMIT = 'UDEVLIM';
%FILE_L_LIMIT = 'LFILLIM';
%FILE_MANAGER= 'FILEMAN';
%FILE_SPACE_MANAGER = 'FSPAMN';
%FILE_U_LIMIT = 'UFILLIM';
%FIND_INAME = 'FNDINAM';
%FIND_PINAME = 'FPINAME';
%INITIALIZATION= 'INITIZE';
%INITIATOR ='INITATR' ;
%INPUT_CONTROLLER='IN_CONT';
%INTERRUPT_HANDLER = 'INTHDLR' ;
%OPERATOR_SYSTEM_COMMUNICATOR ='CP_COMM' ;
%OUTPUT_CONTROLLER = 'OUT_CON' ;
%PCBSTRINT = 'PCBINIT';
%PCT_L_LIMIT = 'LPCTLIM';
%PCT_U_LIMIT = 'UPCTLIM';
%PRIMITIVE_ACTIVATE ='ACTIVAT';
%PRIMITIVE_ALLOCATOR = 'ALLOCTR' ;
%PRIMITIVE_CREATE_PCB = 'CREAT_P';
%PRIMITIVE_DESTROY ='DESTROY';
%PRIMITIVE_GETQUE = 'GETQUER';
%PRIMITIVE_INTERRUPT_DISENABLER = 'DISNABL' ;
%PRIMITIVE_INTERRUPT_ENABLER = 'ENABLE ' ;
%PRIMITIVE_PCBDATA ='PCBSTR';
%PRIMITIVE_PUTQUE='PUTQUE';
%PRIMITIVE_RCBDATA='RCBHDLR';
%PRIMITIVE_RCB_HANDLER = 'RCBHDLR';
%PRIMITIVE_RCB_MATCH='RCBMTCH';
%PRIMITIVE_RELEASE = 'RELEASE';
%PRIMITIVE_RESTORESTATE = 'RESTATE';
%PRIMITIVE_REQUEST = 'REQUEST';
%PRIMITIVE_SAVESTATE = 'SAVSTAT';
%PRIMITIVE_SCHEDULER = 'SCHEDLR';
%PROCESSOR = 'PRCSSR';
%RCB_BITDATA = 'BITDATA';
%RCB_CHARDAT='CHARDAT';
%RCB_FIND='RCBFIND';
%RCB_FIXEDAT='FIXEDAT';
%RCB_TRANSFERQ = 'TRANSQR';
%SEM_LIMIT = 'USEMLIM';
%SEM_L_LIMIT = 'LSEMLIM';
%SYS_PROCESSOR = 'SYSPSCR' ;
%TERMINATOR = 'TERMNTR';


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

    USEFUL DECLARATIONS NOT PROVIDED BY PL/I.

    INCLUDED BY  %INCLUDE GENDEC;                                  */

DCL TRUE BIT(1) STATIC INITIAL('1'B);
DCL FALSE BIT(1) STATIC INITIAL('0'B);

%DECLARE FOREVER CHAR;
%FOREVER='WHILE (TRUE)';
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    PREPROCESSOR IMPLEMENTATION OF A CASE STATMENT.

    "ENABLED" BY  %INCLUDE CASESTM.                                   */

%DCL DO_ACTION_OF ENTRY RETURNS(CHAR);
%DCL CASE ENTRY RETURNS(CHAR);
%DCL END_OF_CASES ENTRY RETURNS(CHAR);
%DCL ENDCASE CHAR;
%DCL (@I,@I1,@NEST) FIXED;
%DCL @K FIXED;

%ENDCASE ='GO TO END_OF_CASE_STMT';
%@I1 = -1;
%@I= -1;
%@NEST = 0;


%DO_ACTION_OF: PROC RETURNS (CHAR);

   @NEST = @NEST + 1;
   RETURN('BEGIN;  GO TO');
%END;


%CASE: PROC RETURNS(CHAR);

   IF(@NEST = 1) THEN @I = @I + 1;
     ELSE @I1 = @I1 + 1;
   RETURN('CASELAB');
%END;


%END_OF_CASES: PROC RETURNS(CHAR);

   IF (@NEST = 1) THEN DO; @K = @I; @I = -1; END;
     ELSE DO; @K = @I1; @I1=-1; END;
   @NEST = @NEST -1;
   RETURN('DCL CASELAB(@K) LABEL;
     END_OF_CASE_STMT:  END');
%END;


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

    DECLARATIONS FOR IDENTIFIERS USED AS INTERFACE VARIABLES
    WITH REQUEST AND RELEASE, AND AS VALUES PASSED IN
    MESSAGES.

    INCLUDED IN PROGRAM BY %INCLUDE REQRELD;                          */

DECLARE ( /*  REQUEST - RELEASE INTERFACE  */
          ANYPROC,
          $ENDJOB,
          $ERROR,
          $FILEOP,
          $IBUFF,
          $INAME,
          $INPUT,
          $INTDEV,
          $INTRPT,
          $JCBUSP,
          $NEWJOB,
          $OBUFF,
          $OPR_IO,
          $OUTPUT,
          $PUTOUT,
          $PRINTQ,
          $RFILEW,
          $SPACE,
          $WAIT
    ) FIXED BINARY(15) EXTERNAL;
```

```
DECLARE( /*  OPERATION IDENTIFIERS FOR FILE SPACE MANAGER */
         $$ADD,      /*  ADD A DEVICE  */
         $$DELET,    /*  DELETE A DEVICE  */
         $$EXTNT,    /*  REQUEST FOR AN EXTENT  */
         $$PERMF,    /*  PERMANENT TYPE FILE  */
         $$SPACE,    /* SPACE REQUESTED  */
         $$TEMPF     /*  TEMPORARY TYPE FILE  */
       ) FIXED BINARY(15) EXTERNAL;
DECLARE( /* OPERATION IDENTIFIERS FOR FILE MANAGER    */
         $$CLOSF,    /* CLOSE  FILE */
         $$DIRAD,  /* DIRECTORY ADD */
         $$DIRDL,  /* DIRECTORY DELETE */
         $$DIRRD,  /* DIRECTORY READ */
         $$DSTYF,    /* DESTROY FILE */
         $$EOF,  /* END OF FILE */
         $$INTPC,    /* INTERFACE PROCESS IDENTIFIER */
         $$OPENF,    /* OPEN FILE */
         $$OPCOM,    /* OPERATOR SYSTEM COMMUNICATOR ID.  */
         $$READ,     /*  READ OPERATION  */
         $$SPCMN,    /* SPACE MANAGER IDENTIFIER */
         $$WRITE     /*  WRITE OPERATION  */
       ) FIXED BINARY(15) EXTERNAL;

DECIARE( /* OPERATION IDENTIFIERS FOR OPERATOR SYS COMMUN */
         $$DCNE,
         $$FAIL,
         $$OPRTR,
         $$PASS   /* PASS TO OPERATOR */
       ) FIXED BINARY(15) EXTERNAL;

DECLARE( /* OPERATION IDENTIFIERS FOR OUTPUT CONTROLLER */
         $$FILOP,
         $$JCL,
         $$STOP,
         $$TERM
        ) FIXED BINARY(15) EXTERNAL;

DCL MESSAGE POINTER STATIC ;
DCL 1 MESSAGE_BUFFER BASED(MESSAGE),
    2 RELEASCR FIXED BINARY(15),
    2 ANSWER_REQUEST BIT(1),
    2 MSG_SEMAPHORE  FIXED BINARY(15),
    2 BUFFER_LOCATION POINTER,
    2 DATA_FIELD,
      3 CHAR_FIELD1 CHAR(8),
      3 CHAR_FIELD2 CHAR(8),
      3 CHAR_FIELD3 CHAR(8),
      3 CHAR_FIELD4 CHAR(8),
      3 FIELD1 FIXED BINARY(15),
      3 FIELD2 FIXED BINARY(15),
      3 FIELD3 FIXED BINARY(15),
      3 FIELD4 FIXED BINARY(15),
      3 FIELD5 FIXED BINARY(15),
      3 FIELD6 FIXED BINARY(15);

DECIARE 1 OUTPUT_BUFFER BASED (OBUFFPTR),
        2 OUT_BUFF  CHAR(132);
DECIARE 1 INPUT_BUFFER  BASED (IBUFFPTR),
        2 IN_BUFF CHAR(80);
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

    DECLARATIONS FOR IDENTIFIERS USED AS INTERFACE VARIABLES
    WITH PCB STRUCTURES.

    INCLUDED BY  %INCLUED PCBDCL;                                    */

DECLARE PRIMITIVE_PCBDATA GENERIC

        (DATCHAR ENTRY /* PUT OR GET CHARACTER TYPE DATA */
                      (FIXED BINARY,/* PINAME = PCB NUMBER*/
                       BIT(1),/* @PUT OR @GET */
                       FIXED BINARY,/* FIELD IDENTIFIER */
                       CHAR(8),/* DATA FIELD */
                       FIXED BINARY /* ERROR PARAMETER */),

         DATFIXB ENTRY /* PUT OR GET FIXED BINARY DATA */
                      (FIXED BINARY,/* PINAME = PCB NUMBER*/
                       BIT(1), /* @PUT OR @GET */
                       FIXED BINARY,/* FIELD IDENTIFIER */
                       FIXED BINARY, /* DATA FIELD */
                       FIXED BINARY  /* ERROR PRAMETER */),

         DATPTR ENTRY /* PUT OR GET POINTER DATA */
                      (FIXED BINARY,/* PINAME = PCB NUMBER*/
                       BIT(1), /* @PUT OR @GET */
                       FIXED BINARY,/* FIELD IDENTIFIER */
                       POINTER, /* DATA FIELD */
                       FIXED BINARY  /* ERROR PRAMETER */),

         DATBIT2 ENTRY /* BIT(2) ARRAY DATA */
                      (FIXED BINARY,/* PINAME = PCB NUMBER*/
                       BIT(1), /* @PUT OR @GET */
                       FIXED BINARY,/* FIELD IDENTIFIER */
                       FIXED BINARY,/* DESIRED ARRAY LOWER
                                       LIMIT                         */
                       FIXED BINARY,/* DESIRED ARRAY UPPER
                                       LIMIT                         */
                       (*)BIT(2),/*ARRAY DATA FIELD  */
                       FIXED BINARY  /* ERROR PRAMETER */),

         DATARYS ENTRY /* FIXED BINARY(15) ARRAY DATA */
                      (FIXED BINARY,/* PINAME = PCB NUMBER*/
                       BIT(1), /* @PUT OR @GET */
                       FIXED BINARY,/* FIELD IDENTIFIER */
                       FIXED BINARY,/* DESIRED ARRAY LOWER
                                       LIMIT                         */
                       FIXED BINARY,/* DESIRED ARRAY UPPER
                                       LIMIT                         */
                       (*)FIXED BINARY,/* ARRAY DATA FIELD*/
                       FIXED BINARY  /* ERROR PRAMETER */),

         DATARYL ENTRY /* FIXED BINARY(31) ARRAY DATA */
                      (FIXED BINARY,/* PINAME = PCB NUMBER*/
                       BIT(1), /* @PUT OR @GET */
                       FIXED BINARY,/* FIELD IDENTIFIER */
                       (*)FIXED BINARY(31),/* DATA FIELD  */
                       FIXED BINARY  /* ERROR PRAMETER */),

         DATBIT1 ENTRY /* BIT(1) DATA */
                      (FIXED BINARY,/* PINAME = PCB NUMBER*/
                       BIT(1), /* @PUT OR @GET */
                       FIXED BINARY,/* FIELD IDENTIFIER */
                       BIT(1), /* DATA FIELD  */
                       FIXED BINARY  /* ERROR PRAMETER */));
```

```
DECLARE PCB_IIM FIXED BINARY(15) EXTERNAL;

DECLARE(
          @BORMNR,
          @BRMVEC,
          @CHILD,
          @CYCLE,
          @FSTAT,
          @FWCNTR,
          @LFISIB,
          @MSGPTR,
          @NR_PGS,
          @PARENT,
          @PE_NR,
          @PRIRTY,
          @QUANTM,
          @RESVEC,
          @RGISIB,
          @STATE,
          @STATUS,
          @SYSPRO,
          @XNAME
       ) FIXED BINARY(15) EXTERNAL;

DECLARE(
          @GET,
          @PUT
       ) BIT(1) EXTERNAL;

DECLARE( /* PROCESS STATUS IDENTIFIERS  */
          @@BLKDR,   /*  WAITING FOR A RESOURCE  */
          @@BLKDT,   /*  BLOCKED AWAITING TIMER  */
          @@REDYA,   /* WAITING FOR A PROCESSOR  */
          @@RUN,     /*  RUNNING  */
          @@SUSPD    /*  SUSPENDED  */
       ) CHARACTER(8) EXTERNAL;


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

    DECLARATIONS FOR IDENTIFIERS USED AS INTERFACE VARIABLES
    WITH RCB HANDLER.

    INCLUDED BY  %INCLUDE RCBDCL;                              */

DECLARE PRIMITIVE_RCBDATA GENERIC

         (RCB_CHARDAT ENTRY /* PUT OR GET CHARACTER DATA  */
             (FIXED BINARY, /* RCB NUMBER = RESOURCE INTERNAL
                               NAME                          */
              BIT(1), /* #PUT OR #GET  */
              FIXED BINARY, /*  FIELD IDENTIFIER */
              CHAR(8), /* DATA ARGUMENT */
              FIXED BINARY /* ERROR ARGUMENT */ ),

          RCB_FIXBDAT ENTRY /*PUT OR GET FIXED BINARY DATA */
             (FIXED BINARY, /* RCB NUMBER = RESOURCE INTERNAL
                               NAME                          */
              BIT(1), /* #PUT OR #GET  */
              FIXED BINARY, /*  FIELD IDENTIFIER */
              FIXED BINARY, /* DATA ARGUMENT */
              FIXED BINARY /* ERROR ARGUMENT */ ),

          RCB_BITDATA ENTRY /* PUT OR GET BIT(1) DATA  */
             (FIXED BINARY, /* RCB NUMBER = RESOURCE INTERNAL
                               NAME                          */
              BIT(1), /* #PUT OR #GET  */
              FIXED BINARY, /*  FIELD IDENTIFIER */
              BIT(1), /* DATA ARGUMENT */
              FIXED BINARY /* ERROR ARGUMENT */ ));
```

```
DECLARE( /* FIELD IDENTIFIERS ASSOCIATED WITH
            PRIMITIVE_RCBDATA  */

     /***FIELD       | ASSOCIATED VALUES PASSED OR RETURNED
         IDENTIFIER  | IN THE DATA ARGUMENT                    */

         #ASSGND,   /* VALID RCB NUMBER CHECK */
         #CNT_SZ,   /* COUNTER OR SIZE - FIXED BINARY          */
         #CRATR,    /* PINAME OF CREATOR - FIXED BINARY        */
         #DINAME,   /* DEVICE INAME (FILE LOCATION) - F.B.     */
         #DSTAT,    /* DEVICE STATUS ##GO OR ##HOLD -BIT(1)    */
         #L_QUE,    /* RETURNS EMPTY OR NOT EMPTY - BIT(1)     */
         #OFILE,    /* OPEN FILE STATUS - ##NOAVL,##AVAIL,
                       ##READ, OR ##WRITE; DEVICE AVAILABLE
                       FOR ##TEMPF, ##PERMF, OR #TORPF FILES
                       FIXED BINARY                            */
         #OWNER,    /* PINAME OF OWNER  -  FIXED BINARY        */
         #PCTNAM,   /* EXTERNAL NAME OR PACK, CELL OR TAPE
                          - FILE LOCATION - CHAR(8)            */
         #R_QUE,    /* RETURNS EMPTY OR NOT EMPTY - BIT(1)     */
         #S_OR_P,   /* SHARED OR PRIVATE ##SHRD OR ##PRIV
                                          - BIT(1)             */
         #TFILE,    /* FILE TYPE - #TEMPF OR ##PERMF   F.B.    */
         #XNAME     /* RES. EXTERNAL NAME - CHAR(8)            */
      )  FIXED BINARY EXTERNAL;

DECLARE( /* RCB TYPES  */
         #DEVICE,
         #FILE,
         #PCT,
         #SEMFOR
      ) FIXED BINARY EXTERNAL;

DECLARE( /* OPERATIONS FOR RCB_FIND  */
         #FNDOP1,
         #FNDOP2,
         #FNDOP3,
         #FNDOP4,
         #FNDOP5
      ) FIXED BINARY EXTERNAL;

DECLARE( /* STATUS IDENTIFICATION FOR QUEUED FILE OPERATIONS
                AND FILE USE STATUS IDENTIFIERS              */
         ##AVAIL,   /*  FILE IS NOT CURRENTLY OPEN          */
         ##NCAVL,   /*  FILE IS NOT CURRENTLY ACTIVE        */
         ##PERMF,   /*  PERMANENT FILE  */
         ##PRIV,    /*  PRIVATE FILE  */
         ##READ,    /* READ OPERATION */
         ##READA,   /* READ - ANSWER REQUIRED - STATUS =>
                       PROCESS' FILES HAVE BEEN SACRIFICED   */
         ##READS,   /* READ SACRIFICE - NO ANSWER REQ'D.     */
         ##SHRD,    /*  SHARED FILE  */
         ##TEMPF,   /* TEMPORARY FILE */
         ##TORPF,   /* TEMPORARY OR PERMANENT FILE  */
         ##WRITE,   /* WRITE OPERATION */
         ##WRITA,   /* WRITE  ANSWER REQUIRED - SACRIFICED */
         ##WRITH,   /* WRITE  HOLD - AWAITING PERMISSION   */
         ##WRITS    /* WRITE SACRIFICE - NO ANSWER REQ'D.  */
       ) FIXED BINARY EXTERNAL;

DECLARE( /* STATUS OF PROCESS W.R.T. FILE OPERATIONS       */
         ##READR,   /* ONLY READING FROM SHARED FILES      */
         ##SACR,    /* SACRIFICED                          */
         ##WRITR    /* WRITING INTO SHARED FILES - NOT
                       SACRIFICED                          */
       ) FIXED BINARY EXTERNAL;
```

```
DECLARE( /*  READY ACTIVE QUEUE IDENTIFIERS  */
         #REDYA
      )FIXED BINARY(15) EXTERNAL,
        (#OS,
         #USER
       ) BIT(1) EXTERNAL;

DECLARE( /* RESOURCE VECTOR STATUS VARIABLES     */
         ##ACCES,  /*  ACCESS ALLOWED    */
         ##ACQRD,  /*  PROCESS HAS ACQUIRED THIS DEVICE */
         ##NOACC,  /*  ACCESS NOT ALLOWED  */
         ##SACRF  /* PROCESS HAS BEEN SACRIFICED  W.R.T.
                     THIS DEVICE - MUST REASSIGN       */
       ) BIT(2) EXTERNAL;

DECLARE( /* QUEUE STATUS, QUEUE IDENTIFIER AND DEVICE
                  STATUS VARIABLES                     */
         ##CLOSE,
         ##GO,
         ##HOLD,
         ##OPEN,

         #GET,
         #PUT,

         #LEFT,
         #RIGHT
        ) BIT(1) EXTERNAL;

DECLARE (
         DEV_L_LIMIT,
         DEV_U_LIMIT,
         FILE_L_LIMIT,
         FILE_U_LIMIT,
         PCT_L_LIMIT,
         PCT_U_LIMIT,
         SEM_LIMIT,
         SEM_L_LIMIT
       ) FIXED BINARY EXTERNAL;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

    DECLARATION AND PROCEDURE CALL NECESSARY TO OBTAIN A
    MESSAGE BUFFER  AND KNOWLEDGE OF A PROCESS' OWN
    INTERNAL NAME.

    INCLUDED BY  %INCLUDE RRMSG;                          */

DECLARE MYNAME FIXED BINARY(15) STATIC INIT(0);
CALL PRIMITIVE_PCBDATA(MYNAME,@GET,@MSGPTR,MESSAGE,ERRCR);

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

    DECLARATIONS FOR IDENTIFIERS USED AS INTERFACE VARIABLES
    BY OPERATING SYSTEM PRIMITIVES AND PROCESSES.

    INCLUDED BY  %INCLUDE OSDCL;                          */

DCL CURRENT_PROCESS(4) FIXED BINARY EXTERNAL;

DECLARE(
         ALL_INT,
         NUMBINT,
         NUMBCPU,
         PROCESSOR,
         SYS_PROCESSOR
       ) FIXED BINARY EXTERNAL;

DECLARE SAVEINTS (16) BIT(1) STATIC;
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    DECLARATIONS FOR PROCESSOR REGISTERS, FOR SIMULATICN.
    INCLUDED BY  %INCLUDE REGSTRS;                                 */
DECLARE CPUREGS(4,10) FIXED BINARY(31) EXTERNAL;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    DECLARATIONS FOR AN ARRAY TO SIMULATE ENABLING AN
    INTERRUPT.
    INCLUDED BY  %INCLUDE INTACTV;                                 */
DECLARE INTRUPT(16) BIT(1) EXTERNAL;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   DECLARATION, FOR SIMULATION PURPOSES, OF THE INTERRUPT
    LOCATIONS.
    INCLUDED BY  %INCLUDE INTSET;                                  */
DECLARE INTRSET(16) BIT(1) EXTERNAL;
```

MODULE SPECIFICATION

NAME: ERROR HANDLER TYPE: PROCESS

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
Not Applicable: All communications handled via messages.			

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
Request	Prccess I-Name, Semaphore, Message Pointer, Error Parameter	a) $Error - Semaphore used to identify messages for this process. b) $Obuff - Semaphore used to obtain an output buffer.
Release	Process I-Name, Semaphore, Message Pointer, Error Parameter	a) $Opr_IO - Semaphore used to pass a message via the Cp-erator System Communicator to the Computer Operator.
PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to Process Control Block module used to enter or obtain data.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE
Not applicable for processes.	

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Message Buffer	---	Based	Dynamically allocated, pcinter qualified structure used for passing messages.
	Releasor	Integer	Internal name of process releasing a message.
	Answer-Request	Bit(1)	Boolean indicating if answer required.
	Message-Semaphore	Integer	Semaphore to be used in the answer.
	Buffer-Location	Pointer	Qualified I/O buffer containing message to/from operator.
	Field1	Integer	Internal name of process which caused the error.
	Field2	Integer	The error condition code.
	Field3-6	Integer	Not used.
	Char-Field1-4	Char(8)	Not used.

MODULE DESCRIPTION

Nct inplemented - Dummy module used to accept error messages and to pass a decoded message to the computer operator. Actual inplementation is dependent on the hardware and the detectien or correction techniques employed.

MODULE IMPLEMENTATION

```
%INCLUDE NAMCHGR;                    /*****   ERROR HANDLER   *****/
(CHECK (ERROR)):

ERROR_HANDLER: PROC OPTIONS(MAIN);

%INCLUDE SIMULTR;
  SIM_START(PROCESS_ERROR_HANDLER)

%INCLUDE GENDEC;
%INCLUDE PCBDCL;
%INCLUDE CASESTM;
%INCLUDE REQRELD;

  ON CHECK (ERROR)
     BEGIN;
       IF (ERROR ¬= 0) THEN DO;
            /* NON RECOVERABLE CONDITION: ERROR DETECTED
               IN ERROR HANDLER                          */
         STOP;
                              END;
     END;

%INCLUDE RRMSG;

     DCL ERROR FIXED BINARY INIT(0);
     DCL HARDWARE_ERROR BIT(1) ;
    DCL (FINAME,ERRCOD) FIXED BINARY STATIC;
    DCL (FNAME,RNAME) CHAR(8) STATIC;
    DCL INDX FIXED BINARY (15,0);
    DCL CBPTR POINTER STATIC;

    DCL RCB_ERR_CODES (101 : 120) CHAR (40) STATIC
        INITIAL(
                 'INVALID RCB TYPE SPECIFIED IN CREATE RCB',
                 'SYSTEM CONDITION: ALL RCB SPACE IN USE',
                 'INVALID RESOURCE INTERNAL NAME SPECIFIED',
                 'ATTEMPTED ACCESS TO AN UNALLOCATED RCB',
                 'ACCESSED EMPTY QUEUE OR UNOPENED FILE',
                 'READ/WRITE OPERATION ON AN UNOPENED FILE',
                 'RESTRICTED ACCESS: WRITE INTO QUE-HEADER',
                 'UNIDENTIFIED OPERATION IN RCB FIND',
                 'UNAUTHORIZED ACCESS TO A RESOURCE',
                 'INVALID FILE OPERATION SPECIFIED',
                 'UNIDENTIFIED RESOURCE EXTERNAL NAME',
                 'UNATHORIZED WRITE ACCESS TO SHARED FILE',
                 'DEVICE IN HOLD STATUS: FILE UNACCESSIBLE',
                 'PERMANENT FILE CREATION UNAUTHORIZED',
                 'INSUFFICIENT SPACE FOR PERMANENT FILE',
                 'WRITE OPERATION TO FILE IN READ STATUS',
                 'UNAUTHORIZED RESOURCE DESTROY ATTEMPTED',
                 'RESOURCE ACCESSED HAS BEEN DESTROYED',
                 'ATTEMPTED TO MODIFY RCB ASSIGNED FIELD',
                 '
               );

    DCL PCB_ERR_CODES (201 : 206) CHAR (40) STATIC
        INITIAL(
                 'INVALID PROCESS INTERNAL NAME SPECIFIED',
                 'ATTEMPTED ACCESS TO UNALLOCATED PCB',
                 'INVALID PCB FIELD REFERENCE SPECIFIED',
                 'INVALID OPERATION ON PCB FIELD ATTEMPTED',
                 'SPACE NOT AVAILABLE FOR A NEW PCB',
                 'PROCESS NOT FOUND WITH SPECIFIED NAME'
               );
```

```
  DCL PROCESS_ERR_CODES (301 : 310) CHAR (40) STATIC
      INITIAL(
                'INVALID OP_SYS_COMM ACTION IDENTIFIER',
                '                                     ',
                'UNIDENTIFIED RESOURCE X_NAME IN MSG BUFF',
                'UNIDENTIFIED MESSAGE TO INPUT CONTROLLER',
                'INVALID ACTION IDENTIFIER TO OUTPUT CONT',
                'INVALID ACTION IDENTIFIER TO OUTPUT CONT',
                'INVALID SEMAPHORE PASSED TO REL/REQ PRIM',
                'UNAUTHORIZED USE OF MESSAGE SEMAPHORE',
                '                                     ',
                '                                     '
              );

   DO FOREVER;
  CALL PRIMITIVE_REQUEST(ANYPROC,$OBUFF,MESSAGE,ERROR);
  OBPTR = BUFFER_LOCATION;

SIM_INTERRUPT_PT

        CALL PRIMITIVE_REQUEST(ANYPROC,$ERROR,MESSAGE,
                               ERROR);

SIM_INTERRUPT_PT

        CALL ERROR_INTERPRETER;
        IF HARDWARE_ERROR THEN CALL HDWR_ERR_HNDLR;
           ELSE CALL SOFTWARE_ERR_HNDLR;
   END;

SIM_END

   ERROR_INTERPRETER: PROC;

        /* IDENTIFICATION OF THE ERROR TYPE AND CAUSE
           IS DETERMINED IN THIS SUBROUTINE; I.E., TABLE
           LOOKUP.                                        */
        HARDWARE_ERROR = FALSE;
        RETURN;
   END ERROR_INTERPRETER;

   HDWR_ERR_HNDLR: PROC;
        /* DETERMINATION OF RECOVERABLE/NON-RECOVERABLE
           AND THE APPROPRIATE ACTION INCLUDING A MESSAGE
           TO THE OPERATOR, IF NECESSARY, IS ACCOMPLISHED
                                                         */
        RETURN;
   END HDWR_ERR_HNDLR;
```

```
SOFTWARE_ERR_HNDLR: PROC;

    /* APPROPRIATE ACTION TO CORRECT THE ERROR OR
    PREVENT FURTHER ERROR IS TAKEN. FOR EXAMPLE, A
    USER PROCESS ATTEMPTING TO WRITE INTO A RESTRICTED
    FILE MIGHT BE TERMINATED.          ***/

       /*** IDENTIFY PROCESS AND ERROR CODE ***/
    IF (FIELD1 = 0) THEN PINAME = RELEASOR;
    ELSE PINAME =FIELD1;
    ERRCOD = FIELD2;

       /*** SET UP MESSAGE FOR THE OPERATOR ***/
    CALL PRIMITIVE_PCBDATA(PINAME,@GET,@XNAME,PNAME,
                           ERROR);
    OBUFPTR, BUFFER_LOCATION = OBPTR;
    INDX = ERRCOD /^100;
    DO ACTION_OF  CASE(INDX);
     CASE(1):      /*  RESOURCE ERROR CODES  */
       OBUFPTR -> OUT_BUFF = PNAME || ' ' ||
                             RCB_ERR_CODES(ERRCOD);
     ENDCASE;
     CASE(2):      /*  PROCESS ERROR CODES  */
       OBUFPTR -> OUT_BUFF = PNAME || ' ' ||
                                   PCB_ERR_CODES(ERRCOD);
     ENDCASE;
     CASE(3):      /*  PROCESS ERROR CODES  */
       OBUFPTR -> OUT_BUFF = PNAME || ' ' ||
                             PROCESS_ERR_CODES(ERRCOD);
     ENDCASE;
    END_OF_CASES;
    MSG_SEMAPHORE = $OBUFF;
    ANSWER_REQUEST = TRUE;
    FIELD1 = $$PASS;

       /*** RELEASE MESSAGE TO THE OPERATOR ***/
    CALL PRIMITIVE_RELEASE(ANYPROC,$OPR_IO,MESSAGE,
                           ERROR);
    /* ADD CODE TO TERMINATE PROCESS CAUSING THE
       ERROR, IF NECESSARY.                          */
    RETURN;
 END SOFTWARE_ERR_HNDLR;

END ERROR_HANDLER;
```

MODULE SPECIFICATION

NAME: OPERATOR SYSTEM COMMUNICATOR TYPE: PROCESS

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS

Not Applicable: All communications handled via messages.

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
Request	Process I-Name, Semaphore, Message Pointer, Error parameter	a) $Opr_IO - Semaphore used to identify messages for this process. b) $Obuff - Semaphore used to obtain an output buffer.
Release	Process I-Name, Semaphore, Message Pointer, Error Parameter	a) $Output - Semaphore used to send messages to the Output Controller. b) $Input - Semaphore used to send messages to the Input Controller. c) $Fileop - Semaphore used to send messages to the File Manager. d) $Error - Semaphore used to send messages to the Error Handler. e) $Wait - General message class identifier. f) $Ibuff - Semaphore used to free input buffers.
Find-Piname	Process X-Name, Process I-Name, Error Parameter	Entry point to PCB Handler used to obtain the internal name of a process identified by external name.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE

Not applicable for processes.

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Message Buffer	---	Based	Dynamically allocated, pointer qualified structure used for passing messages.
	Releasor	Integer	Internal name of process releasing a message.
	Answer-Request	Bit(1)	Boolean indicating if answer required.
	Message-Semaphore	Integer	Semaphore to be used in the answer.
	Buffer-Location	Pointer	Qualified I/O buffer containing message to/from operator.
	Field1	Integer	Action identifiers: $$PASS for messages to other processes or the operator; $$OPCOM for messages to this process.
	Field2	Integer	Value equals $$OPRTR then message from operator; $$DONE or $$FAIL indicates task completion code for messages sent to other processes.
	Field3-6	Integer	Not used.
	Char-Field1	Char(8)	Resource external name in answers to task messages.
	Char-Field2-4	Char(8)	Not used.

MODULE DESCRIPTION

A simplistic implementation has been completed to reify the concept of using a focal point for system-operator communications. The exact hardware configuration; a repertoire of instructions; and a complete specification of functions to be performed are essential for a complete implementation. At present, the process passes messages and initiates action messages upon receipt of instructions from the operator.

MODULE IMPLEMENTATION

```
%INCLUDE NAMCHGR ;      /***** OPERATOR SYSTEM COMMUNICATOR*/
(CHECK (ERROR)):

OPERATOR_SYSTEM_COMMUNICATOR: PROC OPTIONS(MAIN);

%INCLUDE SIMULTR;
  SIM_START(PROCESS_OPERATOR_SYSTEM_COMMUNICATOR)

    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     THIS MODULE SERVES AS THE FOCAL POINT FOR SYSTEM TO
     OPERATOR AND OPERATOR TO SYSTEM MESSAGES.  IT INTER-
     PRETS MESSAGES AND DETERMINES THE ACTION TO BE TAKEN.
     IN PARTICULAR, IT ISSUES ACTION MESSAGES TO THE INPUT
     CONTROLLER, OUTPUT CONTROLLER, AND FILE MANAGER TO ADD
     OR DELETE A DEVICE AND ITS ASSOCIATED INTERFACE PROCESS
     AS WELL AS PASSING MESSAGES FROM THE SYSTEM TO THE
     OPERATOR.                                              ***/

%INCLUDE GENDEC;
%INCLUDE CASESTM;
%INCLUDE PCBDCL;
%INCLUDE REQRELD;

   ON CHECK (ERROR)
      BEGIN;
        IF (ERROR ¬= 0) THEN DO;
           (NOCHECK (ERROR)):
             BEGIN;
               ANSWER_REQUEST = FALSE;
               MSG_SEMAPHORE = 0;
               FIELD1 = RELEASOR;
               FIELD2 = ERROR;
               MSG_SEMAPHORE = 0;
               CALL PRIMITIVE_RELEASE(ANYPROC,$ERROR,
                                      MESSAGE,ERROR);
               ERROR = 0;
             END;
             GO TO START;
                           END;
      END;

%INCLUDE RRMSG;

    DCL ERROR FIXED BINARY STATIC INIT(0);
    DCL BUFFER_USED BIT(1) STATIC INIT('1'B);
    DCL BUFFER_POINTER POINTER STATIC;

  CALL COMMUNICATIONS_INITIALIZER;
   START:
  DO FOREVER;
    IF BUFFER_USED THEN DO;
      CALL PRIMITIVE_REQUEST(ANYPROC,$OBUFF,MESSAGE,ERROR);

  SIM_INTERRUPT_PT

      BUFFER_USED = FALSE;
      BUFFER_POINTER = BUFFER_LOCATION;
    END;

    CALL PRIMITIVE_REQUEST(ANYPROC,$OPR_IO,MESSAGE,ERROR);

  SIM_INTERRUPT_PT
```

```
CALL MESSAGE_INTERPRETER;
DO_ACTION_OF_CASE(I);

CASE(1):
        /* PASS MESSAGE TO OPERATOR */
     FIELD4 = RELEASOR;
     CALL PRIMITIVE_RELEASE(ANYPROC,$OUTPUT,MESSAGE,
                            ERROR);
  ENDCASE;

CASE(2): ;
        /* MESSAGE FROM OPERATOR OR SYSTEM  */
     DCL (START_PT,MSG_LEN,PINAME) FIXED BINARY (15,0);
     DCL MSG_CODE CHAR(4) STATIC;
     DCL (MSG_ITEM1,MSG_ITEM2) CHAR(8) STATIC;
     DCL PCODE CHAR(1) STATIC;
     DCL MSG_SEM FIXED BINARY STATIC INITIAL (0);
     DCL SYS_MSG CHAR(40);
     DCL ACTION FIXED BINARY STATIC INITIAL (0);
  IF (FIELD2 = $$OPRTR) THEN DO; /* MSG FROM OPERATOR */
       /*** INTERPRET MESSAGE IN INPUT BUFFER  ***/
     START_PT = 3;
     MSG_LEN  = 4;
     IBUFPTR = BUFFER_LOCATION;
     MSG_CODE = SUBSTR(IBUFPTR -> IN_BUFF,
                       START_PT,MSG_LEN);
     START_PT = 7;
     MSG_LEN = 8;
     MSG_ITEM1 = SUBSTR(IBUFPTR -> IN_BUFF,
                        START_PT,MSG_LEN);
     IF (MSG_CODE = 'ADD ') THEN ACTION = $$ADD;
        ELSE IF (MSG_CODE = 'DEL ') THEN ACTION=$$DELET;
             ELSE DO;  /* MESSAGE TO SYSTEM PROCESS */;
                  CALL FIND_PINAME(MSG_ITEM1,PINAME,
                                   ERROR);
                  FIELD1 = $$OPCOM;
                  CALL PRIMITIVE_RELEASE(PINAME,$WAIT,
                                         MESSAGE,ERROR);
                  END;
     IF (ACTION ¬= 0) THEN DO;
        /*** IDENTIFY RESOURCE TYPE ***/
        START_PT = 1;
        MSG_LEN = 1;
        PCODE = SUBSTR(MSG_ITEM1,START_PT,MSG_LEN);
        IF (PCODE = 'I') THEN MSG_SEM = $INPUT;
           ELSE IF (PCODE = 'O') THEN MSG_SEM = $OUTPUT;
                ELSE IF (PCODE='F') THEN DO;
                     START_PT = 15;     MSG_LEN = 8;
                     MSG_ITEM2=SUBSTR(IBUFPTR ->
                               IN_BUFF,START_PT,MSG_LEN);
                     MSG_SEM = $FILEOP;
                               END;
                     ELSE DO; /* UNIDENTIFIED RESOURCE
                                 EXTERNAL NAME.       */
                  CALL PRIMITIVE_RELEASE(RELEASOR,$IBUFF,
                                         MESSAGE,ERROR);
                  BUFFER_LOCATION,OBUFPTR = BUFFER_POINTER;
                  OBUFPTR -> OUT_BUFF = MSG_CODE ||
                    MSG_ITEM1 || 'INVALID RESOURCE NAME';
                  FIELD1 = $$OPCOM;
                  FIELD2 = $$PASS;
                  ANSWER_REQUEST = FALSE;
                  MSG_SEMAPHORE = 0;
                  FIELD3,FIELD4,FIELD5,FIELD6 = 0;
                  CHAR_FIELD1,CHAR_FIELD2,CHAR_FIELD3,
                       CHAR_FIELD4 = '';
                  CALL PRIMITIVE_RELEASE(ANYPROC,$OUTPUT,
                                         MESSAGE,ERROR);
                  BUFFER_USED = TRUE;
                              END;
```

```
        IF ((ACTION ¬= 0) & (MSG_SEM ¬= 0)) THEN DO;
           /*** SEND TASK MESSAGE TO PROCESS CONCERNED ***/
           ANSWER_REQUEST = TRUE;
           MSG_SEMAPHORE = $OPR_IO;
           FIELD1 = $$OPCOM;
           FIELD2 = ACTION;
           CHAR_FIELD1 = MSG_ITEM1;
           IF (PCODE = 'F') THEN CHAR_FIELD2 = MSG_ITEM2;
           CALL PRIMITIVE_RELEASE(ANYPROC,MSG_SEM,MESSAGE,
                                  ERROR);

                                                  END;
                                              END;
                                 END;
     ELSE DO;      /* MESSAGE ANSWER FROM SYSTEM PROCESS */
           /*** RELEASE MESSAGE TO THE OPERATOR ***/
           ACTION = FIELD2;
           IF (ACTION = $$DONE) THEN
                    SYS_MSG = 'ACTION COMPLETED FOR';
              ELSE SYS_MSG = 'UNABLE TO PERFORM ACTION ON';
           BUFFER_LOCATION,OBUFPTR = BUFFER_POINTER;
           OBUFPTR -> OUT_BUFF = SYS_MSG ||
                    CHAR_FIELD1 || CHAR_FIELD2;
           FIELD1 = $$OPCOM;
           FIELD2 = $$PASS;
           ANSWER_REQUEST = FALSE;
           MSG_SEMAPHORE = 0;
           FIELD3,FIELD4,FIELD5,FIELD6 = 0;
           CHAR_FIELD1,CHAR_FIELD2,CHAR_FIELD3,
               CHAR_FIELD4 = '';
           CALL PRIMITIVE_RELEASE(ANYPROC,$OUTPUT,MESSAGE,
                                  ERROR);
           BUFFER_USED = TRUE;
           END;
     ENDCASE;

   END_OF_CASES;

 END;

 SIM_END


 MESSAGE_INTERPRETER: PROC;

    /* INTERPRET MESSAGE TO DETERMINE REQUIRED ACTION AND
       SET THE CASE STATEMENT INDEX                  ***/

   IF (FIELD1 = $$PASS) THEN I = 1;
      ELSE IF (FIELD1 = $$OPCOM) THEN I = 2;
           ELSE ERROR = 301;
 END MESSAGE_INTERPRETER;

 COMMUNICATIONS_INITIALIZER: PROC;

    /* ISSUE REQUESTS AND RELEASES TO INITIALIZE THE
       SYSTEM INPUT / OUTPUT CONFIGURATION              */

 END COMMUNICATIONS_INITIALIZER;

END OPERATOR_SYSTEM_COMMUNICATOR;
```

MODULE SPECIFICATION

NAME: INPUT CONTROLLER TYPE: PROCESS

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
Not Applicable: All communications handled via messages.			

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
Request	Process I-Name, Semaphore, Message Pointer, Error Parameter	a) $Input - Semaphore used to identify messages for this process.
Release	Process I-Name, Semaphore, Message Pointer, Error Parameter	a) $Error - Semaphore used to send a message to Error Handler. b) $Intdev - Semaphore used to send a message to a newly created interface process to identify a device internal name. c) $Ibuff - Semaphore used to release input buffers to an interface process. d) $Opr_IO - Semaphore used to send a message to Operator System Communicator when passing messages from the operator or replying to a message from Op-Sys-Comm.
Device Directory	Resource X-Name, Access Identifier, Process X-Name, Page Table Length, Page Table Vector, Priority, Interrupt Number, Located-Boolean	This module is invoked to get data which identifies a device interface process and which is required to create a PCB and an RCB for the interface process and device, respectively.
PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to PCB Structures module to enter or obtain data.
PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Vector Lower Limit, Vector Upper Limit, Vector Parameter, Error Parameter	Generic entry point to PCB Structures module to enter or obtain data or portions of the data stored as a vector (i.e., Resource Access Vector, Page Table Vector, etc.).
RCBData	Resource I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to RCB Handler module to enter or get data concerning a resource.

RCBPUTQ	Resource I-Name, Left/Right Queue, Process I-Name, Data Parameter, Priority, Message Pointer, Data Parameter, Error Parameter	Entry point to RCB Handler used to insert a process or a message on a specified resource queue by priority. Queues used by this process are the message semaphore and the Ready Active queues.
Find I-Name	Resource Type, Resourse X-Name, Resource I-Name, Error Parameter	Entry point to RCB Handler used to get the internal name for the resource specified by type (file, device, etc.) and external name.
Interrupt Enabler	Interrupt Number, Interrupt Save-Vector	This module is invoked tc enable a specific interrupt or enable all interrupts disabled by this process.
Interrupt Disabler	Interrupt Number, Interrupt Save-Vector	This module is invoked tc disable a specific or all interrupts and saving the status of the interrupts in a save-vector for enabling.
Create RCB	Resource Type, Resource X-Name, Resource Owner, Sz-Cntr Parameter, Access Identifier, PCI Name, Dev/Int Identifier, File Descriptor, File Descriptor, Resource I-Name, Error Parameter	Entry point to RCB Handler used to create an RCB of the type specified; enter descriptor data in the appropriate RCB fields; and return the resource internal name. Not all fields are used by each type of resource.
GETPCB	Parent I-Name, Rgt-Sib I-Name, Process X-Name, Priority, System Process ID, Init State Vector, Process I-Name, Cyclic Process Id, Error Parameter	Entry point to PCB Structures used to create a PCB for a process; enter data in the PCB fields; and return the process internal name.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE

Not applicable for processes.

DAIA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Message Buffer	---	Based	Dynamically allocated, pcinter qualified structure used for passing messages.
	Releasor	Integer	Internal name of process releasing a message.
	Answer-Request	Bit(1)	Boolean indicating if answer required.
	Message-Semaphore	Integer	Semaphore to be used in the answer.
	Buffer-Location	Pointer	Qualified I/O buffer containing message to/from operator.
	Field1	Integer	a) $$Opcom - Messae from Operator System Communicator. b) $$JCL - Message from interface process.
	Field2	Integer	a) $$ADD - Task identifier to create an interface process and device RCB.
	Field3-6	Integer	Not used.
	Char-Field1	Char(8)	Identifies external name of the device to create/destroy.
	Char-Field2-4	Char(8)	Not used.

MODULE DESCRIPTION

Partially implemented process performing such functions as creating a device and its associated interface process; passing messages from the operator to the Operator System Communicator, etc. Implementation limited until the hardware environment, JCL code, etc. are defined.

MODULE IMPLEMENTATION

```
%INCLUDE NAMCHGR;                    /*****  INPUT CONTROLLER *****/
(CHECK (ERROR)):

INPUT_CONTROLLER: PROC OPTIONS(MAIN);

%INCLUDE SIMULTR;
  SIM_START(PROCESS_INPUT_CONTROLLER)

    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     THIS MODULE HAS BEEN IMPLEMENTED TO CENTRALIZE AND
     FACILITATE THE "INPUT" OPERATIONS FROM VARIOUS DEVICES.
     IT PERFORMS SUCH FUNCTIONS AS CREATING AND DESTROYING
     INTERFACE PROCESSES AND DEVICE RCB'S; ENTERING A JOB'S
     JCL IN THE JOB QUEUE; RELEASING MESSAGES TO THE INI-
     TIATOR WHEN A JOB IS READY FOR CREATION; AND PASSING
     MESSAGES TO THE OPERATOR SYSTEM COMMUNICATOR.      ***/

%INCLUDE GENDEC;
%INCLUDE REQFLD;
%INCLUDE PCBDCL;
%INCLUDE RCBDCL;
%INCLUDE CASESTM;

  ON CHECK (ERROR)
     BEGIN;
       IF (ERROR ¬= 0) THEN DO;
          (NOCHECK (ERROR)):
            BEGIN;
              FIELD1 = RELEASOR;
              FIELD2 = ERROR;
              MSG_SEMAPHORE = 0;
              ANSWER_REQUEST = FALSE;
              CALL PRIMITIVE_RELEASE(ANYPROC,$ERROR,MESSAGE,
                                     ERROR);
              ERROR = 0;
            END;
            GO TO START;
                              END;
     END;

%INCLUDE RRMSG;

    DCL ERROR FIXED BINARY STATIC INIT(0);
     DCL (S_OR_P,NRPGS,PRI,PINAME,RINAME,CHILD,INTRPTNR)
          FIXED BINARY;
     DCL FOUND BIT(1);
     DCL (PXNAME,RXNAME) CHAR(8) STATIC;
     DCL PGTABVEC(1) FIXED BINARY;
     DCL STATE_VEC(10) FIXED BINARY (31,0);
     DCL RES_VEC(SEM_L_LIMIT : PCT_U_LIMIT) BIT(2);
     DCL NRDEVICES FIXED BINARY STATIC INIT(0);
     DCL TEMP_PTR POINTER;
     DCL DUMMY(16) BIT(1);
     DCL I1 FIXED BINARY(15) STATIC INIT(1),
         I0 FIXED BINARY(15) STATIC INIT(0),
         NULP POINTER STATIC ;
     NULP = NULL;
```

```
START: DO FOREVER;
   CALL PRIMITIVE_REQUEST(ANYPROC,$INPUT,MESSAGE,ERROR);

SIM_INTERRUPT_PT

   CALL MESSAGE_INTERPRETER;
   DO ACTION OF  CASE(I);
   CASE(1): /* MESSAGE FROM OPERATOR-SYSTEM COMMUNICATOR
               TO ADD AN INPUT DEVICE                    */
            /*** GET DATA FROM THE DIRECTORY REQUIRED TO
                 CREATE AN INTERFACE PROCESS AND DEVICE
                 RCB.                                    ***/
              CALL DEVICE_DIRECTORY(CHAR_FIELD1,S_OR_P,
                PXNAME,NRPGS,PGTABVEC,PRI,INTRPTNR,FOUND);
              IF (FOUND = TRUE) THEN DO;
                 RXNAME = CHAR_FIELD1;
                 CALL PRIMITIVE_PCBDATA(MYNAME,@GET,@CHILD,
                                        CHILD,ERROR);
                 STATE_VEC(1) = PGTABVEC(1);
                 STATE_VEC(2) = 1;      /* IC REGISTER */
            /*** CREATE PCB FOR INTERFACE PROCESS    ***/
                 CALL GETPCB(MYNAME,CHILD,PXNAME,PRI,TRUE,
                             STATE_VEC,PINAME,IO,ERROR);
            /*** RESET FAMILY LINKAGE.   ***/
                 IF (CHILD ¬= 0) THEN
                    CALL PRIMITIVE_PCBDATA(CHILD,@PUT,
                                           @LFTSIB,PINAME,ERROR);
                 CALL PRIMITIVE_PCBDATA(MYNAME,@PUT,@CHILD,
                                        PINAME,ERROR);
            /*** CREATE AN RCB FOR THE DEVICE.   ***/
                 CALL CREATE_RCB(#DEVICE,RXNAME,PINAME,IO,
                      S_OR_P,(' '),INTRPTNR,IO,IO,RINAME,
                      ERROR);
                   /* SET UP RESOURCE ACCESS VECTOR */
                 RES_VEC = '00'B;
                 RES_VEC(RINAME)  = ##ACCES;
                 RES_VEC($IBUFF)  = ##ACCES;
                 RES_VEC($INPUT)  = ##ACCES;
                 RES_VEC($SPACE)  = ##ACCES;
                 RES_VEC($ERROR)  = ##ACCES;
                 RES_VEC($FILEOP) = ##ACCES;
                 RES_VEC($INTRPT) = ##ACCES;
                 RES_VEC($INTDEV) = ##ACCES;
                 RES_VEC($WAIT)   = ##ACCES;
                 CALL PRIMITIVE_PCBDATA(PINAME,@PUT,@RESVEC,
                    SEM_L_LIMIT,PCT_U_LIMIT,RES_VEC,ERROR);
            /*** INSERT PAGE TABLE VECTOR AND MESSAGE BUFFER
                 POINTER IN THE PCB.                     ***/
                 CALL PRIMITIVE_PCBDATA(PINAME,@PUT,@BFRVEC,
                       I1,NRPGS,PGTABVEC,ERROR);
                 ALLOCATE MESSAGE_BUFFER SET(TEMP_PTR);
                 CALL PRIMITIVE_PCBDATA(PINAME,@PUT,@MSGPTR,
                                        TEMP_PTR,ERROR);
            /*** RELEASE MESSAGE TO THE NEW PROCESS IDENTI-
                 FYING THE DEVICE INTERNAL NAME.         ***/
                 FIELD1 = RINAME;
                 CALL PRIMITIVE_RELEASE(PINAME,$INTDEV,
                                        MESSAGE,ERROR);
            /*** ALLOCATE AND RELEASE INPUT BUFFERS TO BE
                 USED BY THE NEW PROCESS.                ***/
                 FIELD1,FIELD2,FIELD3,FIELD4 = 0;
                 MSG_SEMAPHORE = 0;
                 ANSWER_REQUEST = FALSE;
                 CHAR_FIELD1,CHAR_FIELD2 = ' ';
                 ALLOCATE INPUT_BUFFER  SET  (IBUFPTR);
                 BUFFER_LOCATION = IBUFPTR;
                 CALL PRIMITIVE_RELEASE(PINAME,$IBUFF,
                                        MESSAGE,ERROR);
                 ALLOCATE INPUT_BUFFER  SET  (IBUFPTR);
                 BUFFER_LOCATION = IBUFPTR;
                 CALL PRIMITIVE_RELEASE(PINAME,$IBUFF,
                                        MESSAGE,ERROR);
```

```
            /*** INSERT THE PROCESS ON THE READY ACTIVE
                 QUEUE AND ENABLE THE DEVICE INTERRUPT. ***/
                 NRDEVICES = NRDEVICES + 1;
                 CALL PRIMITIVE_INTERRUPT_ENABLER(INTRETNR,
                                                  DUMMY);
                 CALL RCBPUTQ(#REDYA,#OS,PINAME,IO,PRI,NULP,
                              IO,ERROR);
                 FIELD1 = $$OPCOM;
                 FIELD2 = $$DONE;
                                      END;
               ELSE DO; FIELD2 = $$FAIL;
                        CHAR_FIELD2 = 'BAD NAME';
                    END;
            /*** RELEASE ACTION TAKEN MESSAGE TO OPERATOR
                 SYSTEM COMMUNICATOR.                    ***/
             CALL PRIMITIVE_RELEASE(ANYPRCC,$OPR_IO,
                                    MESSAGE,ERROR);
    ENDCASE;
    CASE(2):      /* MESSAGE FROM OPERATOR SYSTEM
                     COMMUNICATOR; DELETE A DEVICE */
             CALL FIND_INAME(#DEVICE,CHAR_FIELD1,RINAME,
                             ERROR);
             CALL PRIMITIVE_RCBDATA(RINAME,#GET,#OWNER,
                                    PINAME,ERROR);
             /* DISENABLE THE INTERRUPT, LOCATE AND FREE
                BUFFERS, DESTROY OUTSTANDING FILES,
                OUTPUT MESSAGE TO OPERATOR CONCERNING
                USER PROCESS EFFECTED, DESTROY DEVICE
                RCB AND PROCESS PCB, RELEASE ACTION
                DONE MESSAGE TO OPERATOR-SYSTEM
                COMMUNICATOR.  ALL TO BE IMPLEMETED      */
    ENDCASE;
    CASE(3):;/* SAVE JCL AND FILE INFORMATION UNTIL EOF.
                IF EOF, ENTER NEW JOB IN JOB QUEUE       */
             DCL JCLCODE CHAR(2);
             DCL MSG_PT FIXED BINARY INIT(1);
             DCL MSG_LEN FIXED BINARY INIT(2);
             IBUFPTR = BUFFER_LOCATION;
             JCLCODE = SUBSTR(IBUFPTR -> IN_BUFF,
                              MSG_PT,MSG_LEN);
             IF (JCLCODE = '@@') THEN DO;  /* OPCOM MSG */
                 FIELD2 = $$OPRTR;
                 FIELD1 = $$OPCOM;
                 ANSWER_REQUEST = FALSE;
                 CALL PRIMITIVE_RELEASE(ANYPRCC,$OPR_IC,
                                        MESSAGE,ERROR);
                                     END;
        /* ELSE DO; SAVE JCL FOR THE PROCESS, FILE INFO,
                    OR ENTER IN JOB QUEUE IF JOB EOF.    */
        ENDCASE;
    END_OF_CASES;
  END;

  SIM_END

  MESSAGE_INTERPRETER: PROC;
     /* THIS SUBROUTINE DETERMINES WHO DID THE RELEASE ON
        $INPUT AND WHAT ACTION IS REQUIRED.              */
    IF ((FIELD1 = $$OPCOM) & (FIELD2 = $$ADD)) THEN I = 1;
       ELSE IF ((FIELD1=$$OPCOM)&(FIELD2=$$DELET)) THEN I=2;
            ELSE IF (FIELD1 = $$JCL) THEN I = 3;
                 ELSE ERROR = 304;
  END MESSAGE_INTERPRETER;
END INPUT_CONTROLLER;
```

MODULE SPECIFICATION

NAME: OUTPUT CONTROLLER TYPE: PROCESS

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS

Not Applicable: All communications handled via messages.

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
Request	Process I-Name, Semaphore, Message Pointer, Error Parameter	a) $Output - Semaphore used to identify messages for this process. b) $Obuff - Semaphore used to obtain an output buffer. c) $Printq - Semaphore used to get next print job.
Release	Process I-Name, Semaphore, Message Pointer, Error Parameter	a) $Error - Semaphore used to send messages to the Error Handler. b) $Intdev - Semaphore used to send message to an interface process to identify a device internal name. c) $Obuff - Semaphore used to release output buffers. d) $Opr_IO - Semaphore used to send messages to Operator System Communicator in reply to $$OPCOM messages. e) $Putout - Semaphore used to send output commands to interface processes. f) $Fileop - Semaphore used to send messages to the File Manager to open, read or destroy a file.
Find Piname	Process X-Name, Process I-Name, Error Parameter	Entry point to PCB Handler used to obtain the internal name of a process identified by external name.
Device Directory	Resource X-Name, Access Identifier, Process X-Name, Page Table Length, Page Table Vector, Priority, Interrupt Number, Located-Boolean	This module is invoked to get data which identifies a device interface process and which is required to create a PCB and an RCB for the interface process and device, respectively.
PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to PCB Structures module to enter or obtain data.
PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Vector Lower Limit, Vector Upper Limit, Vector Parameter, Error Parameter	Generic entry point to PCB Structures module to enter or obtain data or portions of the data stored as a vector (i.e., Resource Access Vector, Page Table Vector, etc.).

RCBData	Resource I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to RCB Handler module to enter or get data concerning a resource.
RCBPUTQ	Resource I-Name, Left/Right Queue, Process I-Name, Data Parameter, Priority, Message Pointer, Data Parameter, Error Parameter	Entry point to RCB Handler used to insert a process or a message on a specified resource queue by priority. Queues used by this process are the message semaphore and the Ready Active queues.
Find I-Name	Resource Type, Resourse X-Name, Resource I-Name, Error Parameter	Entry point to RCB Handler used to get the internal name for the resource specified by type (file, device, etc.) and external name.
Interrupt Enabler	Interrupt Number, Interrupt Save-Vector	This module is invoked tc enable a specific interrupt or enable all interrupts disabled by this process.
Interrupt Disabler	Interrupt Number, Interrupt Save-Vector	This module is invoked to disable a specific or all interrupts and saving the status of the interrupts in a save-vector for enabling.
Create RCB	Resource Type, Resource X-Name, Resource Owner, Sz-Cntr Parameter, Access Identifier, PCT Name, Dev/Int Identifier, File Descriptor, File Descriptor, Resource I-Name, Error Parameter	Entry point to RCB Handler used to create an RCB of the type specified; enter descriptor data in the appropriate RCB fields; and return the resource internal name. Not all fields are used by each type of resource.
GETPCB	Parent I-Name, Rgt-Sib I-Name, Process X-Name, Priority, System Process ID, Init State Vector, Process I-Name, Cyclic Process Id, Error Parameter	Entry point to PCB Structures used to create a PCB for a process; enter data in the PCB fields; and return the process internal name.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE

Not applicable for processes.

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Message Buffer	---	Based	Dynamically allocated, pointer qualified structure used for passing messages.

	Releasor	Integer	Internal name of process releasing a message.
	Answer-Reguest	Bit(1)	Boolean indicating if answer required.
	Message-Semaphore	Integer	Semaphore to be used in the answer.
	Buffer-Location	Pointer	Qualifies an output buffer used to store data read from files and to pass data for output to an interface process.
	Field1	Integer	a) $$OPCOM - identifies a message from the Operator System Communicator. b) $$TERM - identifies a message from the Terminator to print a job. c) $$READ, $$EOF or $$OPENF - identifies message-type answer from File Manager.
	Field2	Integer	a) $$PASS - pass message to the system operator. b) $$ADD, $$DELET or $$STOP - identifies task specified by the Operator System Communicator. c) File internal name to be printed.
	Field3	Integer	File record to be printed.
	Field4	Integer	Not used.
	Char-Field1	Char(8)	File external name.
	Char-Field2-4	Char(8)	Not used.
Output Admin Table	---	Array	This structure is used to determine availability of devices, the interface process internal name, and the file being printed on that device.
	Interface Process	Integer	Process internal name.
	Assigned	Bit(1)	Identifies available devices.
	File Name	Integer	File internal name.

MODULE DESCRIPTION

The Output Controller performs such functions as creating and destroying interface processes and device RCB's; assigning output files to the device; opening and destroying output files; passing output buffers with file data or system messages to the appropriate device; etc.

MODULE IMPLEMENTATION

```
%INCLUDE NAMCHGR;       /*****  OUTPUT CONTRCLLER *****/
(CHECK (ERROR)):

OUTPUT_CCNTRCLLER: PROC OPTIONS(MAIN);

    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     THIS PROCESS HAS BEEN IMPLEMENTED TO CENTRALIZE AND
     FACILITATE THE "OUTPUT" REQUIREMENTS TO VARIOUS DEVICES
     THAT MAY, AND GENERALLY DO, EXIST IN AN ALL APPLICATION
     COMPUTER SYSTEM.  THE PROCESS INTERACTS THROUGH THE USE
     OF MESSAGES TO AND FROM OTHER PROCESSES SUCH AS THE
     TERMINATOR, OPERATOR SYSTEM COMMUNICATOR, FILE MANAGER,
     ERROR HANDLER, AND VARIOUS INTERFACE PROCESSES.  IN
     ADDITION, IT INVOKES SEVERAL OF THE PRIMITIVES TO
     ACCOMPLISH ITS FUNCTIONS.                    * * * * */

%INCLUDE SIMULTR;
  SIM_START(PROCESS_OUTPUT_CONTROLLER)

%INCLUDE GENLEC;
%INCLUDE REQRELD;
%INCLUDE CASESTM;
%INCLUDE PCBDCL;
%INCLUDE RCBDCL;
  ON CHECK (ERROR)
     BEGIN;
       IF (ERROR ¬= 0) THEN DO;
          (NOCHECK (ERROR)):
             BEGIN;
               FIELD1 = RELEASOR;
               MSG_SEMAPHORE = $OUTPUT;
               CALL PRIMITIVE_RELEASE(ANYPROC,$ERROR,
                                      MESSAGE,ERROR);
               ERROR = 0;
             END;
          GO TO START;
       END;
     END;
%INCLUDE RRMSG;

    DCL 1 OUTPUT_ADMIN_TABLE (10) STATIC,
       2 INTF_PROCS FIXED BINARY INIT((10) 0),
       2 ASSIGNED BIT(1) INIT((10)(1) '0'B),
       2 FINAME FIXED BINARY INIT((10) 0);
    DCL CPR_CUT_DEV FIXED BINARY STATIC;
    DCL CUT_JOBS FIXED BINARY STATIC;
    DCL NRDEVICES FIXED BINARY STATIC INIT(0);
    DCL I1 FIXED BINARY INIT(1);
    DCL NULP POINTER;
    DCL NEWDEVICE FIXED BINARY;
    DCL I FIXED BINARY STATIC;
    DCL RECORD FIXED BINARY STATIC;
    DCL ERROR FIXED BINARY STATIC INIT(0);
    DCL INAME FIXED BINARY;
    DCL BUFFER_USED BIT(1) STATIC INIT('1'B);
    DCL BUFFER_POINTER POINTER STATIC;
```

```
START: DO FOREVER;
   IF BUFFER_USED THEN DO;
      /*  GET AN OUTPUT BUFFER  */
     CALL PRIMITIVE_REQUEST(ANYPROC,$OBUFF,MESSAGE,ERROR);

SIM_INTERRUPT_PT

      BUFFER_POINTER = BUFFER_LOCATION;
      BUFFER_USED = FALSE;
   END;
   CALL PRIMITIVE_REQUEST(ANYPROC,$OUTPUT,MESSAGE,ERROR);

SIM_INTERRUPT_PT

   CALL MESSAGE_INTERPRETER;

   DO ACTION_OF CASE(I) ;
     CASE(1):
       /* OPERATOR SYSTEM COMMUNICATOR MESSAGE      */
       IF (FIELD2 = $$PASS) THEN
          /*** OUTPUT THE MESSAGE  ***/
          CALL PRIMITIVE_RELEASE(OPR_OUT_DEV,$PUTOUT,
                                 MESSAGE,ERROR);
       ELSE DO;
          /*** PERFORM TASK SPECIFIED  ***/
            CALL OP_COMM_MSG_HANDLER;
            IF ((NEWDEVICE ¬= 0) & (OUT_JOBS ¬= 0)) THEN
              CALL START_NEWJOB(NRDEVICES);
            END;
       ENDCASE;
     CASE(2):
       /* TERMINATOR MESSAGE                          */
          /*** INCREMENT PRINT JOB COUNTER AND DETERMINE
               IF A DEVICE IS AVAILABLE.            ***/
       OUT_JOBS = OUT_JOBS + 1;
       DO I = 1 TO NRDEVICES WHILE (ASSIGNED(I)); END;
       IF (I <= NRDEVICES) THEN CALL START_NEWJOB(I);
       ENDCASE;
     CASE(3):
       /*  FILE MANAGER MESSAGE                       */
       CALL FILE_MSG_HANDLER;
       ENDCASE;
     END_OF_CASES;
END;

MESSAGE_INTERPRETER: PROC;
   /*  DETERMINE WHICH PROCESS DID A RELEASE ON THE
       $OUTPUT SEMAPHORE AND SET THE CASE STATEMENT
       INDEX                    ***/
   IF (FIELD1 = $$OPCOM) THEN I = 1;
      ELSE IF (FIELD1 = $$TERM) THEN I = 2;
           ELSE I = 3;
END MESSAGE_INTERPRETER;
```

```
CP_COMM_MSG_HANDLER: PROC;
   /* TAKE THE APPROPRIATE ACTION; I.E., ADD A DEVICE AND
      AN INTERFACE PROCESS, DELETE A DEVICE AND INTERFACE
      PROCESS, OR STOP PRINTING A JOB.                    */
   DCL (S_OR_P,NRPGS,PRI,PINAME,RINAME,CHILD,INTRNR,INDX)
        FIXED BINARY;
   DCL FOUND BIT(1);
   DCL (PXNAME,RXNAME) CHAR(8) STATIC;
   DCL PGTABVEC(1) FIXED BINARY;
   DCL STATE_VEC(10) FIXED BINARY (31,0);
   DCL RES_VEC(SEM_L_LIMIT : PCT_U_LIMIT) BIT(2);
   DCL DUM(16) BIT(1);
   DCL TEMP_PTR POINTER;
          /* DETERMINE ACTION REQUIRED AND SET INDEX  */
  IF (FIELD2 = $$ADD) THEN INDX = 1;
     ELSE IF (FIELD2 = $$DELET) THEN INDX = 2;
          ELSE IF (FIELD3 = $$STOP) THEN INDX = 3;
               ELSE ERROR = 306;
  DO_ACTION OF CASE(INDX);
     CASE(1):     /* ADD A DEVICE AND INTERFACE PROCESS. */
        RXNAME = CHAR_FIELD1;
        CALL DEVICE_DIRECTORY(RXNAME,S_OR_P,PXNAME,NRPGS,
                              PGTABVEC,PRI,INTRNR,FOUND);
        IF (FOUND = FALSE) THEN DO;
           /*** INVALID EXTERNAL NAME  ***/
            FIELD2 = $$FAIL;  CHAR_FIELD2 = 'BAD NAME';
                              END;
        ELSE DO;
           /*** CREATE PCB FOR INTERFACE PROCESS, SETUP
                FAMILY LINKAGE, AND CREATE DEVICE RCB. ***/
             CALL PRIMITIVE_PCBDATA(MYNAME,@GET,@CHILD,
                                    CHILD,ERROR);
             STATE_VEC(1) = PGTABVEC(1);
             STATE_VEC(2) = 1;     /*  IC REGISTER  */
             CALL GETPCB(MYNAME,CHILD,PXNAME,PRI,TRUE,
                         STATE_VEC,PINAME,(0),ERROR);
             IF (CHILD ¬= 0) THEN
                CALL PRIMITIVE_PCBDATA(CHILD,@PUT,@LFISIB,
                                       PINAME,ERROR);
             CALL PRIMITIVE_PCBDATA(MYNAME,@PUT,@CHILD,
                                    PINAME,ERROR);
             CALL CREATE_RCB(#DEVICE,RXNAME,PINAME,(0),
                 S_OR_P,(' '),INTRNR,(0),(0),RINAME,ERROR);
             /* SET UP RESOURCE VECTOR FOR PINAME */
             RES_VEC = '00'B;
             RES_VEC(RINAME) = ##ACCES;
             RES_VEC($PUTOUT) = ##ACCES;
             RES_VEC($ERROR) = ##ACCES;
             RES_VEC($OBUFF) = ##ACCES;
             RES_VEC($INTRPT) = ##ACCES;
             RES_VEC($INTDEV) = ##ACCES;
             RES_VEC($WAIT) = ##ACCES;
             CALL PRIMITIVE_PCBDATA(PINAME,@PUT,@RESVEC,
                     SEM_L_LIMIT,PCT_U_LIMIT,RES_VEC,ERROR);
             CALL PRIMITIVE_PCBDATA(PINAME,@PUT,@ERMVEC,
                                    11,NRPGS,PGTABVEC,ERROR);
           /*** GET MESSAGE BUFFER FOR INTERFACE PROCESS */
             ALLOCATE MESSAGE_BUFFER SET(TEMP_PTR);
             CALL PRIMITIVE_PCBDATA(PINAME,@PUT,@MSGPTR,
                                    TEMP_PTR,ERROR);
           /*** RELEASE MESSAGE TO THE PROCESS IDENTIFYING
                THE DEVICE INTERNAL NAME.                ***/
             FIELD1 = RINAME;
             CALL PRIMITIVE_RELEASE(PINAME,$INTDEV,
                                    MESSAGE,ERROR);
```

```
            /*** GET OUTPUT BUFFERS AND INSERT ON THE
                 AVAILABILITY QUEUE (RELEASE MESSAGES). ***/
              FIELD1,FIELD2,FIELD3,FIELD4 = 0;
              MSG_SEMAPHORE = 0;
              ANSWER_REQUEST = FALSE;
              CHAR_FIELD1,CHAR_FIELD2 = ' ';
              ALLOCATE OUTPUT_BUFFER SET (OBUFPTR);
              BUFFER_LOCATION = OBUFPTR;
              CALL PRIMITIVE_RELEASE(MYNAME,$OBUFF,
                                     MESSAGE,ERROR);
              ALLOCATE OUTPUT_BUFFER SET (OBUFPTR);
              BUFFER_LOCATION = OBUFPTR;
              CALL PRIMITIVE_RELEASE(MYNAME,$OBUFF,
                                     MESSAGE,ERROR);
            /*** INITIALIZE LOCAL VARIABLES AND INSERT
                 THE INTERFACE PROCESS ON THE READY 'A'
                 QUEUE.  ENABLE THE DEVICE INTERRUPT. ***/
              NRDEVICES = NRDEVICES + 1;
              INTF_PROCS(NRDEVICES) = PINAME;
              IF (NRDEVICES = 1) THEN OPR_OUT_DEV = PINAME;
              CALL PRIMITIVE_INTERRUPT_ENABLER(INTRNR,DUM);
              NULP = NULL;
              CALL RCBPUTQ(#REDYA,#OS,PINAME,(0),PRI,NULP,
                           (0),ERROR);
              NEWDEVICE = PINAME;
              FIELD1 = $$OPCOM;
              FIELD2 = $$DONE;
              END;
            /*** RELEASE ACTION TAKEN MESSAGE TO OPERATOR
                 SYSTEM COMMUNICATOR                    ***/
          CALL PRIMITIVE_RELEASE(ANYPROC,$OPR_IO,MESSAGE,
                                 ERROR);
        ENDCASE;
        CASE(2):     /*  DELETE A DEVICE AND DESTROY THE
                         INTERFACE PROCESS                */
          CALL FIND_INAME(#DEVICE,CHAR_FIELD1,RINAME,ERROR);
          CALL PRIMITIVE_RCBDATA(RINAME,#GET,#DINAME,INTRNR,
                                 ERROR);
          CALL PRIMITIVE_INTERRUPT_DISENABLER(INTRNR,DUM);
                    /* LOCATE BUFFERS AND DESTROY THEM,
                       DESTROY RCB, PCB, MESSAGE CONTAINER
                       AND ALL OUTSTANDING RELEASES/REQUESTS
                       CONCERNING THE INTERFACE PROCESS    */
        ENDCASE;
        CASE(3):     /* STOP PRINTING CURRENT JOB  */
            /*** GET INTERNAL NAME OF THE DEVICE AND
                 INTERFACE PROCESS.                     ***/
          RXNAME = CHAR_FIELD1;
          CALL FIND_INAME(#DEVICE,RXNAME,INAME,ERROR);
          CALL PRIMITIVE_RCBDATA(INAME,#GET,#OWNER,PINAME,
                                 ERROR);
          DO I = 1 TO NRDEVICES
                    WHILE (INTF_PROCS(I) ¬= PINAME); END;
            /*** RESET LOCAL VARIABLES, RELEASE MESSAGE TO
                 OPERATOR SYSTEM COMMUNICATOR, AND
                 CHECK FOR NEW JOB TO PRINT.            ***/
          FINAME(I) = 0;
          ASSIGNED(I) = '0'B;
          FIELD1 = $$OPCOM;
          FIELD2 = $$DONE;
          CALL PRIMITIVE_RELEASE(ANYPROC,$OPR_IO,MESSAGE,
                                 ERROR);
          IF (OUT_JOBS ¬= 0) THEN CALL START_NEWJOB(I);
        ENDCASE;
    END OF CASES;
END OP_COMM_MSG_HANDLER;
```

```
FILE_MSG_HANDLER: PROC;
   /* THIS SUBROUTINE  PASSES OUTPUT BUFFERS TO THE
      APPROPRIATE OUTPUT DEVICE; CHECKS FOR OUTSTANDING
      JOBS TO PRINT; IF ANY, GETS THE FILE NAME FROM
      PRINT QUEUE AND ASSIGNS THE FILE TO THE DEVICE
      FOR PRINTING.                              ***/
  DCL FXNAME CHAR(8) STATIC;
      DO I = 1 TO NRDEVICES WHILE (FINAME(I) ¬= FIELD2);
      END;
         /*** PASS OUTPUT BUFFER TO APPROPRIATE
              INTERFACE PROCESS.                        ***/
      IF (FIELD1 = $$READ) THEN DO;
         RECORD = FIELD3 + 1;
         MSG_SEMAPHORE = $OUTPUT;
         CALL PRIMITIVE_RELEASE(INTF_PROCS(I),$PUTOUT,
                                MESSAGE,ERROR);
         /*** GET ANOTHER OUTPUT BUFFER, INITIALIZE
              MESSAGE BUFFER, AND RELEASE MESSAGE TO
              FILE MANAGER TO READ THE NEXT RECORD.  ***/
         BUFFER_LOCATION = BUFFER_POINTER;
         FIELD1 = $$READ;
         FIELD2 = FINAME(I);
         FIELD3 = RECORD;
         FIELD4,FIELD5,FIELD6 = 0;
         CHAR_FIELD1,CHAR_FIELD2,CHAR_FIELD3,
            CHAR_FIELD4 = '';
         MSG_SEMAPHORE = $OUTPUT;
         ANSWER_REQUEST = TRUE;
         CALL PRIMITIVE_RELEASE(ANYPROC,$FILEOP,MESSAGE,
                                ERROR);
         BUFFER_USED = TRUE;
                                END;
      ELSE IF (FIELD1 = $$EOF) THEN DO;
         /*** PRINT TASK FINISHED: REINITIALIZE LOCAL
              VARIABLES, TEST FOR NEW PRINT JOB, RELEASE
              THE OUTPUT BUFFER, AND RELEASE MESSAGE TO
              FILE MANAGER TO DESTROY THE FILE    ***/
           ASSIGNED(I) = FALSE;
           FINAME(I) = 0;
           IF (OUT_JOBS ¬= 0) THEN CALL START_NEWJOB(I);
           CALL PRIMITIVE_RELEASE(MYNAME,$OBUFF,MESSAGE,
                                  ERROR);
           FIELD1 = $$DSTYF;
           CALL PRIMITIVE_RELEASE(ANYPROC,$FILEOP,
                                  MESSAGE,ERROR);
                                    END;
      ELSE DO;    /*  FILE OPENED */
         /*** GET OUTPUT BUFFER, INITIALIZE MESSAGE
              BUFFER, AND RELEASE MESSAGE TO THE FILE
              MANAGER TO READ THE FIRST RECORD.       ***/
           BUFFER_LOCATION = BUFFER_POINTER;
           FIELD1 = $$READ;
           FIELD2 = FINAME(I);
          FIELD3 = 1;
           FIELD4,FIELD5,FIELD6 = 0;
           CHAR_FIELD1,CHAR_FIELD2,CHAR_FIELD3,
              CHAR_FIELD4 = '';
          MSG_SEMAPHORE = $OUTPUT;
          ANSWER_REQUEST = TRUE;
          CALL PRIMITIVE_RELEASE(ANYPROC,$FILEOP,
                                 MESSAGE,ERROR);
           BUFFER_USED = TRUE;
          END;
END FILE_MSG_HANDLER;
```

```
    /*** THIS SUBROUTINE OBTAINS THE EXTERNAL NAME OF THE
         FILE TO BE PRINTED FROM THE PRINT QUEUE (CHAR-
         FIELD1), RELEASES A MESSAGE TO THE FILE MANAGER
         TO OPEN THE FILE FOR A READ OPERATION, UPDATES
         LOCAL VARIABLES TO REFLECT THE ASSIGNMENT, AND
         CHANGES OWNERSHIP OF THE FILE TO THE OUTPUT
         CONTROLLER.                                    ***/
  START_NEWJOB: PROC(INDEX);
    DCL INDEX FIXED BINARY;
    CALL PRIMITIVE_REQUEST(ANYPROC,$PRINTQ,MESSAGE,ERROR);
    FIELD1 = $$OPENF;
    FIELD2 = ##READ;
    MSG_SEMAPHORE = $OUTPUT;
    ANSWER_REQUEST = TRUE;
    CALL PRIMITIVE_RELEASE(ANYPROC,$FILEOP,MESSAGE,ERROR);
    ASSIGNED(INDEX) = TRUE;
    FINAME(INDEX), INAME = FIELD2;
    OUT_JOBS = OUT_JOBS - 1;
    CALL PRIMITIVE_RCEDATA(INAME,#PUT,#OWNER,MYNAME,ERROR);
  END START_NEWJOB;

  SIM_END

END  OUTPUT_CONTROLLER;
```

MODULE SPECIFICATION

NAME: INITIATOR TYPE: PROCESS

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS

Not Applicable: All communications handled via messages.

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
Request	Process I-Name, Semaphore, Message Pointer, Error Parameter	a) $Iname - Semaphore used to limit the number of processes in the system. Requests decrement counter while releases by Terminator increment the counter. b) $Newjob - Semaphore used to identify messages to this process.

**********Module Implementation Incomplete**********

**********Anticipated External Calls**********

PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to PCB Structures module to enter or obtain data.
PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Vector Lower Limit, Vector Upper Limit, Vector Parameter, Error Parameter	Generic entry point to PCB Structures module to enter or obtain data or portions of the data stored as a vector (i.e., Resource Access Vector, Page Table Vector, etc.).
RCBData	Resource I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to RCB Handler module to enter or get data concerning a resource.
RCBPUTQ	Resource I-Name, Left/Right Queue, Process I-Name, Data Parameter, Priority, Message Pointer, Data Parameter, Error Parameter	Entry point to RCB Handler used to insert a process or a message on a specified resource queue by priority. Queues used by this process are the message semaphore and the Ready Active queues.
GETPCB	Parent I-Name, Rgt-Sib I-Name, Process X-Name, Priority, System Process ID, Init State Vector, Process I-Name, Cyclic Process Id, Error Parameter	Entry point to PCB Structures used to create a PCB for a process; enter data in the PCB fields; and return the process internal name.

Create RCB	Resource Type, Resource X-Name, Resource Owner, Sz-Cntr Parameter, Access Identifier, PCT Name, Dev/Int Identifier, File Descriptor, File Descriptor, Resource I-Name, Error Parameter	Entry point to RCB Handler used to create an RCB of the type specified; enter descriptor data in the appropriate RCB fields; and return the resource internal name. Not all fields are used by each type of resource.
Find I-Name	Resource Type, Resourse X-Name, Resource I-Name, Error Parameter	Entry point to RCB Handler used to get the internal name for the resource specified by type (file, device, etc.) and external name.
RCBGETQ	Resource I-Name, Left/Right Queue, Process I-Name, Q_Data Parameter, Message Pointer, Q_Data Parameter, Found Boolean, Q_Status Boolean, Error Parameter	This entry point to RCB Handler is used to remove a process from the specified queue for the indicated resource. The data stored in the queue are returned if the process is found and the queue status is also provided.
RCB-Find	Resource I-Name, Left/Right Queue, Find Operation, Queue Position, Process I-Name, Data Parameter, Data Parameter, Error Parameter	This entry point to RCB Handler is used to determine queue position of a process and put or get a copy of data. The process, position, or both may be specified to select a specific process, any process at the specified position or a specific process at a specified position.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE

Not applicable for processes.

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Message Buffer	---	Based	Dynamically allocated, pointer qualified structure used for passing messages.
	Releasor	Integer	Internal name of process releasing a message.
	Answer-Request	Bit(1)	Boolean indicating if answer required.
	Message-Semaphore	Integer	Semaphore to be used in the answer.
	Buffer-Location	Pointer	Not used.
	Field1-6	Integer	Use undefined.
	Char-Field1-4	Char(8)	Use undefined.

**********Additional Structures Undefined**********

MODULE DESCRIPTION

Initiator design and implementation is incomplete: job types, JCI used, etc. must be known before implementation.

MODULE IMPLEMENTATION

```
%INCLUDE NAMCHGR;                    /*****  INITIATOR *****/
(CHECK (ERROR)):
INITIATOR: PROC OPTIONS(MAIN);
%INCLUDE GENDEC;
%INCLUDE PCBDCL;
%INCLUDE REQRELD;
%INCLUDE RRMSG;
    ON CHECK(ERROR)
        BEGIN;
          IF (ERROR ¬= 0) THEN DO;
              (NOCHECK(ERROR)): BEGIN;
                FIELD1 = RELEASOR;
              FIELD2 = ERROR;
              CALL PRIMITIVE_RELEASE(ANYPROC,$ERROR,MESSAGE,
                                     ERROR);
              GO TO START;
              END;
          END;
        END;
    DCL (NEW_P_I_NAME,PARENT,PRIORITY) FIXED BINARY;
    DCL  EXTERNAL_NAME CHAR(8);
    DCL ERROR FIXED BINARY STATIC INIT(0);

  START:
    DO FOREVER ;
          /*** CHECK FOR AVAILABLE PCB SPACE ***/
        CALL PRIMITIVE_REQUEST(ANYPROC,$INAME,MESSAGE,ERROR);

          /*** GET NEXT JOB TO BE CREATED ***/
        CALL PRIMITIVE_REQUEST(ANYPROC,$NEWJOB,MESSAGE,
                               ERROR);

          /*** INTERPRET DATA FROM JOB QUEUE ***/
        CALL JCL_INTERPRETER;

          /*** GET THE REQUIRED RESOURCES ***/
        CALL GET_REQUIRED_RESOURCES;

          /*** CREATE A PCB FOR THE PROCESS ***/
          /***  CALL CREATE_PCB ***/

          /*** FREE JOB QUEUE SPACE ***/
        CALL PRIMITIVE_RELEASE(ANYPROC,$JOBQSP,MESSAGE,
                               ERROR);

          /*** INSERT THE PROCESS ON THE READY A QUEUE **/
        CALL RCBPUTQ(#READYA,#USER,P_I_NAME,(0),PRIORITY,
                     MSGPTR,(0),ERROR);

    END ;  /* MAIN PROGRAM SCOPE */

   JCL_INTERPRETER: PROC ;

        /*  INTERPRET  JCL TO IDENTIFY REQUIRED RESOURCES
            AND CAPABILITIES                                  */

   RETURN;
   END JCL_INTERPRETER;

   GET_REQUIRED_RESOURCES: PROC ;

        /* VERIFY THAT THE PROCESS BEING CREATED CAN ACCESS
            THE RESOURCES HE REQUESTED ; CREATE RCB' AS
            NECESSARY AND SET UP ACCESS VECTOR                */

   RETURN;
   END GET_REQUIRED_RESOURCES;
END INITIATOR;
```

MODULE SPECIFICATION

NAME: TERMINATOR TYPE: PROCESS

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS

Not Applicable: All communications handled via messages.

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
Request	Process I-Name, Semaphore, Message Pointer, Error Parameter	a) $Endjob - Semaphore used to identify messages to this process.

**********Module Implementation Incomplete**********

**********Anticipated External Calls**********

Find Piname	Process X-Name, Process I-Name, Error Parameter	Entry point to PCB Handler used to obtain the internal name of a process identified by external name.
PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to PCB Structures module to enter or obtain data.
PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Vector Lower Limit, Vector Upper Limit, Vector Parameter, Error Parameter	Generic entry point to PCB Structures module to enter or obtain data or portions of the data stored as a vector (i.e., Resource Access Vector, Page Table Vector, etc.).
RCBData	Resource I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to RCB Handler module to enter or get data concerning a resource.
RCBPUTQ	Resource I-Name, Left/Right Queue, Process I-Name, Data Parameter, Priority, Message Pointer, Data Parameter, Error Parameter	Entry point to RCB Handler used to insert a process or a message on a specified resource queue by priority. Queues used by this process are the message semaphore and the print queues.
Destroy-RCB	Resource I-Name, Process I-Name, Process Vector, Error Parameter	This entry point to RCB Handler is used to destroy an RCB. All processes on the RCB queue are identified and returned to the calling process so that an error message may be sent to the Error Handler for each process in the list.

RCBGETQ	Resource I-Name, Left/Right Queue, Process I-Name, Q_Data Parameter, Message Pointer, Q_Data Parameter, Found Boolean, Q_Status Boolean, Error Parameter	This entry point to RCB Handler is used to remove a process from the specified queue for the indicated resource. The data stored in the queue are returned if the process is found and the queue status is also provided.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE

Not applicable for processes.

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Message Buffer	---	Based	Dynamically allocated, pointer qualified structure used for passing messages.
	Releasor	Integer	Internal name of process releasing a message.
	Answer-Request	Bit(1)	Boolean indicating if answer required.
	Message-Semaphore	Integer	Semaphore to be used in the answer.
	Buffer-Location	Pointer	Not used.
	Field1-6	Integer	Use undefined.
	Char-Field1-4	Char(8)	Use undefined.

**********Additional Structures Undefined**********

MODULE DESCRIPTION

Terminator design and implementation is incomplete. Note: termination of a process results in the termination of all dependent processes; deallocating resources; and printing output files.

MODULE IMPLEMENTATION

```
%INCLUDE NAMCHGR;                    /***** TERMINATOR ******/
(CHECK (ERROR)):
TERMINATOR: PROC OPTIONS(MAIN);

%INCLUDE GENDEC;
%INCLUDE PCBDCL;
%INCLUDE REQRELD;
%INCLUDE RRMSG;
    ON CHECK (ERROR)
       BEGIN;
       IF (ERROR ¬= 0) THEN DO;
          (NOCHECK(ERROR)): BEGIN;
            FIELD1 = RELEASOR;
            FIELD2 = ERROR;
            CALL PRIMITIVE_RELEASE(ANYPROC,$ERROR,MESSAGE,
                                   ERROR);
            ERROR = 0;
            GO TO START;
            END;
          END;
       END;
 DCL P_I_NAME FIXED BINARY;
    DCL ERROR FIXED BINARY STATIC INIT(0);
 DCL (INVALID,TERM_FLAG) BIT(1);
  START:
  DO FOREVER;
            /*** GET TERMINATION MESSAGE ***/
     CALL PRIMITIVE_REQUEST(ANYPROC,$ENDJOB,MESSAGE,ERROR);

            /*** VERIFY THE MESSAGE ***/
     CALL TERMINATION_VALIDATOR;

            /*** VALID: THEN START TERMINATING ***/
     TERM_FLAG = TRUE;
       DO WHILE (TERM_FLAG) ;
            /*** SELECT THE PROCESS TO BE TERMINATED ***/
       CALL TERM_SELECTOR;

            /*** INSERT CODE TO DETERMINE OUTPUT FILES ***/
            /*** RELEASE MESSAGE TO OUTPUT CONTROLLER ***/
       CALL PRIMITIVE_RELEASE(ANYPROC,$OUTPUT,MESSAGE,
                              ERROR);

            /*** DESTROY THE PROCESS ***/
       /*  CALL PRIMITIVE_DESTROY(PINAME,PROCVEC,ERRCR);*/

            /*** RELEASE ERROR MESSAGES FOR ANY PROCESS
                 QUEUED ON A RESOURCE JUST DESTROYED ***/
            /******* INSERT CODE ******/
            /*** FREE THE JOB QUEUE SPACE ***/
       CALL PRIMITIVE_RELEASE(ANYPROC,$INAME,MESSAGE,
                              ERROR);
       END;
  END;
  TERMINATION_VALIDATOR: PROC;
     /*  THIS SUBROUTINE VERIFIES THE JOB TERMINATION
         REQUEST; I.E., THE VALIDITY OF PROCESS 'A' TO
         DESTROY PROCESS 'B'.                          */
     INVALID=FALSE;
  END TERMINATION_VALIDATOR;
  TERM_SELECTOR: PROC;
     /* THIS SUBROUTINE DETERMINES WHICH PROCESS OR
        MEMBER OF A PROCESS'S FAMILY IS TO BE DESTROYED
        FIRST.                                         */
     TERM_FLAG=FALSE;
  END TERM_SELECTOR;
END TERMINATOR;
```

MODULE SPECIFICATION

NAME: FILE_MANAGER TYPE: PROCESS

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS

Not Applicable: All communications handled via messages.

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
Request	Process I-Name, Semaphore, Message Pointer, Error Parameter	a) $Fileop - Semaphore used to identify messages for this process.
Release	Process I-Name, Semaphore, Message Pointer, Error Parameter	a) $Error - Semaphore used to send messages to the Error Handler. b) $Intdev - Semaphore used to send message to an interface process to identify a device internal name. c) $Rfilew - Semaphore used to release messages to an interface process to perform a read or write operation. d) $Opr_IO - Semaphore used to send a message to Operator System Communicator when replying to a message from Op-Sys-Comm. e) $Space - Semaphore used to send messages to the File Space Manager when a file is destroyed or when a device is added or deleted. f) $Wait - Semaphore used to send a message to a process which sent an open-file message. g) $XXXXX - Semaphore used when specified in answer to a message received.
Find Piname	Process X-Name, Process I-Name, Error Parameter	Entry point to PCB Handler used to obtain the internal name of a process identified by external name.
Device Directory	Resource X-Name, Access Identifier, Process X-Name, Page Table Length, Page Table Vector, Priority, Interrupt Number, Located-Boolean	This module is invoked to get data which identifies a device interface process and which is required to create a PCB and an RCB for the interface process and device, respectively.
PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to PCB Structures module to enter or obtain data.

PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Vector Lower Limit, Vector Upper Limit, Vector Parameter, Error Parameter	Generic entry point to PCB Structures module to enter or obtain data or portions of the data stored as a vector (i.e., Resource Access Vector, Page Table Vector, etc.).
RCBData	Resource I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to RCB Handler module to enter or get data concerning a resource.
RCBPUTQ	Resource I-Name, Left/Right Queue, Process I-Name, Data Parameter, Priority, Message Pointer, Data Parameter, Error Parameter	Entry point to RCB Handler used to insert a process or a message on a specified resource queue by priority. Queues used by this process are the message semaphore and the Ready Active queues.
Find I-Name	Resource Type, Resourse X-Name, Resource I-Name, Error Parameter	Entry point to RCB Handler used to get the internal name for the resource specified by type (file, device, etc.) and external name.
Interrupt Enabler	Interrupt Number, Interrupt Save-Vector	This module is invoked to enable a specific interrupt or enable all interrupts disabled by this process.
Interrupt Disabler	Interrupt Number, Interrupt Save-Vector	This module is invoked to disable a specific or all interrupts and saving the status of the interrupts in a save-vector for enabling.
Create RCB	Resource Type, Resource X-Name, Resource Owner, Sz-Cntr Parameter, Access Identifier, PCI Name, Dev/Int Identifier, File Descriptor, File Descriptor, Resource I-Name, Error Parameter	Entry point to RCB Handler used to create an RCB of the type specified; enter descriptor data in the appropriate RCB fields; and return the resource internal name. Not all fields are used by each type of resource.
GETPCB	Parent I-Name, Agt-Sib I-Name, Process X-Name, Priority, System Process ID, Init State Vector, Process I-Name, Cyclic Process Id, Error Parameter	Entry point to PCB Structures used to create a PCB for a process; enter data in the PCB fields; and return the process internal name.
Destroy-RCB	Resource I-Name, Process I-Name, Process Vector, Error Parameter	This entry point to RCB Handler is used to destroy an RCB. All processes on the RCB queue are identified and returned to the calling process so that an error message may be sent to the Error Handler for each process in the list.

RCBGETQ	Resource I-Name, Left/Right Queue, Process I-Name, Q_Data Parameter, Message Pointer, Q_Data Parameter, Found Boolean, Q_Status Boolean, Error Parameter	This entry point to RCB Handler is used to remove a process from the specified queue for the indicated resource. The data stored in the queue are returned if the process is found and the queue status is also provided.
RCB-Transfer-Queue	Resource I-Name, Left/Right Queue, Process I-Name, From Q_Status, To Q_Status, Xfered Boolean, Error Parameter	This entry point to RCB Handler is used to transfer a process from the queue specified to the opposite queue. The status of the To-Queue before transfer and From-Queue after transfer are returned.
RCB-Find	Resource I-Name, Left/Right Queue, Find Operation, Queue Position, Process I-Name, Data Parameter, Data Parameter, Error Parameter	This entry point to RCB Handler is used to determine queue position of a process and put or get a copy of data. The process, position, or both may be specified to select a specific process, any process at the specified position or a specific process at a specified position.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE
Not applicable for processes.	

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Message Buffer	---	Based	Dynamically allocated, pointer qualified structure used for passing messages.
	Releasor	Integer	Internal name of process releasing a message.
	Answer-Request	Bit(1)	Boolean indicating if answer required.
	Message-Semaphore	Integer	Semaphore to be used in the answer.
	Buffer-Location	Pointer	Qualified I/O buffer containing data read from a file or to be written into a file.
	Field1	Integer	a) $$OPCOM - Message from Operator System Communicator. b) $$INTPC - Message from interface process. c) $$OPENF - Message to open a file. d) $$READ or $$WRITE - Message to do a file read or write operation. e) $$CLOSF - Message to close a file. f) $$DSTYF - Message to destroy a file.

Field2	Integer	a) $$OPRTR - Operator message reply to a request for access to a restricted file. b) $$ADD or $$DELET - $$CPCOM message to add or delete a device and interface process. c) File internal name identifying a file to be closed, accessed on a read or write operation, or destroyed. d) ##READ or ##WRITE - Identifies type access requested in a file open message. e) $$EOF - End of file read. f) Directory entry number.
Field3	Integer	a) Record number to be read from or written into. b) File length for files in the directory for the PCT.
Field4	Integer	Device internal name identified in $$INTPC message.
Field5	Integer	a) $$TEMPF or $$PERMF - Identified in $$INTPC messages.
Field6	Integer	a) ##SHRD or ##PRIV - Identified in $$INTPC messages.
Char-Field1	Char(8)	File or device external name.
Char-Field2	Char(8)	Pack, tape, cell, etc. (PCT) external name.
Char-Field3-4	Char(8)	Not used.

MODULE DESCRIPTION

File Manager has been implemented to perform the standard file operations of opening, closing, destroying, reading from and writing into files. Creating files is performed by an independent module to prevent file creation when space is not available from interfering with the other file operations. In addition, this process controls the allocation of files and prevents deadlocks using preemptive techniques; i.e., once a process has been assigned access to a file if subsequent open requests cannot be satisfied immediately and a potential deadlock exists, the process's files are sacrificed (can be allocated to other processes) until such time as all files required can be allocated concurrently. Interface processes and device RCB's are also created and destroyed by this process.

MODULE IMPLEMENTATION

```
%INCLUDE NAMCHGR;                    /***** FILE MANAGER *****/
(CHECK (ERROR)):

FILE_MANAGER: PROC OPTIONS(MAIN);

%INCLUDE SIMULTR;
  SIM_START(PROCESS_FILE_MANAGER)

%INCLUDE GENDEC;
%INCLUDE REQRELD;
%INCLUDE CASESTM;
%INCLUDE RCBDCL;
%INCLUDE PCBDCL;
  DCL ERROR FIXED BINARY INITIAL(0);
  DCL I FIXED BINARY;
        DCL  RCBNR FIXED BINARY;
        DCL  PCTNAME CHAR(8);
        DCL  NULP POINTER;

  ON CHECK (ERROR)
    BEGIN;
      IF (ERROR ¬= 0) THEN DO;
        FIELD1 = RELEASOR;
        FIELD2 = ERROR;
        IF (ERROR = 112) THEN DO;
           ANSWER_REQUEST = TRUE;
           MSG_SEMAPHORE = $FILEOP;
                              END;
        ELSE DO;
           ANSWER_REQUEST = FALSE;
           MSG_SEMAPHORE = 0;
             END;
        (NOCHECK (ERROR)):
           BEGIN;
             ERROR = 0;
             CALL PRIMITIVE_RELEASE(ANYPROC,$ERROR,MESSAGE,
                                    ERROR);
           END;
        IF (ERROR ¬= 118) THEN GO TO START;
                          END;
    END;

%INCLUDE RRMSG;

  START:
  DO FOREVER;
     CALL PRIMITIVE_REQUEST(ANYPROC,$FILEOP,MESSAGE,ERROR);

  SIM_INTERRUPT_PT

     CALL MESSAGE_INTERPRETER(I);
```

```
   DO_ACTION_OF  CASE(I);

     CASE(1):
       /* MESSAGE FROM OPERATOR SYSTEM COMMUNICATOR     */
       CALL  OP_COMM_MSG_HANDLER;
     ENDCASE;

     CASE(2):
       /* MESSAGE FROM DEVICE INTERFACE PROCESS
          PERFORM SUCH ACTIONS AS UP DATING THE
          MASTER FILE LIST, FILE DIRECTORY ON THE
          DRUM, PACK, TAPE, ETC., RELEASING MESSAGE
          TO STORAGE MEMORY MANAGER.                    */
       IF (FIELD2 ¬= $$EOF) THEN DO;

         /*** IF EXISTING FILE IS TEMPORAY THEN DELETE IT
              IT FROM THE DIRECTORY ELSE CREATE AN RCB   */
          IF (FIELD5 ¬= $$TEMPF) THEN DO;
             CALL PRIMITIVE_RCBDATA(FIELD4,#GET,#PCTNAM,
                                    PCTNAME,ERROR);
             CALL CREATE_RCB(#FILE,CHAR_FIELD1,MYNAME,
                  FIELD3,FIELD6,PCTNAME,FIELD4,##AVAIL,
                  ##PERMF,RCBNR,ERROR);
                                    END;
          ELSE DO;
             ANSWER_REQUEST = FALSE;
             FIELD1 = $$DIRDL;
             CALL PRIMITIVE_RELEASE(RELEASOR,$RFILEW,
                                    MESSAGE,ERROR);
               END;

          /*** GET THE NEXT DIRECTORY ENTRY ***/
          FIELD1 = $$DIRRD;  FIELD2 = FIELD2 + 1;
          ANSWER_REQUEST = TRUE;
          CALL PRIMITIVE_RELEASE(RELEASOR,$RFILEW,
                                 MESSAGE,ERROR);
                                 END;

          /*** EOF: CREATE AN RCB FOR THE PCT AND RELEASE
               A MESSAGE TO THE FILE SPACE MANAGER     ***/
       ELSE DO;
          FIELD1 = $$OPCOM;  FIELD2 = $$DONE;
          CALL PRIMITIVE_RCBDATA(FIELD4,#GET,#XNAME,
                                 CHAR_FIELD1,ERROR);
          CALL PRIMITIVE_RCBDATA(FIELD4,#GET,#PCTNAM,
                                 CHAR_FIELD2,ERROR);
          CALL CREATE_RCB(#PCT,CHAR_FIELD2,MYNAME,FIELD3,
                          FIELD6,(' '),FIELD4,(0),FIELD5,
                          RCBNR,ERROR);
          CALL PRIMITIVE_RELEASE(ANYPROC,$OPR_IO,MESSAGE,
                                 ERROR);
          FIELD1 = $$ADD;   FIELD2 = RCBNR;
          CALL PRIMITIVE_RELEASE(ANYPROC,$SPACE,MESSAGE,
                                 ERROR);
            END;
       FIELD1,FIELD2,FIELD3,FIELD4,FIELD5,FIELD6 = 0;
       CHAR_FIELD1,CHAR_FIELD2,CHAR_FIELD3,CHAR_FIELD4
         = '                ';
     ENDCASE;

     CASE(3):
       /*  MESSAGE FROM A PROCESS TO DO A FILE OPEN,
           CLOSE, DESTROY, READ, WRITE, ETC.           */
       CALL  FILE_OPERATION_CONTROLLER;
     ENDCASE;

   END_OF_CASES;
END;

SIM_END
```

```
MESSAGE_INTERPRETER: PROC(I);
   /*  DETERMINE WHICH PROCESS DID A RELEASE ON $FILEOP
       AND SET THE CASE STATEMENT INDEX                    */

   DCL I FIXED BINARY;
   IF (MESSAGE -> FIELD1 = $$OPCOM) THEN I = 1;
      ELSE IF (MESSAGE -> FIELD1 = $$INTPC) THEN I = 2;
           ELSE I = 3;
   RETURN;
END MESSAGE_INTERPRETER;

OP_COMM_MSG_HANDLER: PROC;
   /* THIS SUBROUTINE PERFORMS SUCH FUNCTIONS AS CREATING
      OR DESTROYING A DEVICE RCB AND AN INTERFACE PROCESS,
      UP_DATING THE PACK,TAPE, ETC MOUNTED ON THE DEVICE,
      AND DETERMINING ACTION TO BE TAKEN ON A REPLY TO A
      USER'S REQUEST FOR FILE OWNERSHIP ON A SHARED FILE*/
   DCL (RXNAME,PXNAME) CHAR(8);
   DCL (S_OR_P,NRPGS,PRI,PINAME,RINAME,CHILD,INTRNR,INDX)
        FIXED BINARY;
   DCL VALX BIT(1);
   DCL PGTABVEC(1) FIXED BINARY;
   DCL STATE_VEC(10) FIXED BINARY(31,0);
   DCL RESVEC(SEM_L_LIMIT : PCT_U_LIMIT) BIT(2);
   DCL DUM(16) BIT(1);
   DCL TEMP_PTR POINTER;
   DCL I1 FIXED BINARY INIT(1);

     /* DETERMINE ACTION TO BE PERFORMED AND SET INDEX */
   IF (FIELD2 = $$OPRTR) THEN INDX = 1;
      ELSE IF (FIELD2 = $$ADD) THEN INDX = 2;
           ELSE IF (FIELD2 = $$DELET) THEN INDX = 3;
                ELSE ERROR = 309;

   DO_ACTION_OF  CASE(INDX);
      CASE(1):   /* MESSAGE FROM OPERATOR */
                       /* ADD CODE */
      ENDCASE;

      CASE(2):   /* ADD A DEVICE, INTERFACE PROCESS AND
                    PCT IF APPLICABLE                     */
            RXNAME = CHAR_FIELD1;

         /*** GET DATA REQUIRED TO CREATE THE INTERFACE
              PROCESS AND DEVICE RCB FROM THE DEVICE
              DIRECTORY.                                  ***/
            CALL DEVICE_DIRECTORY(RXNAME,S_OR_P,PXNAME,
                 NRPGS,PGTABVEC,PRI,INTRNR,VALX);
            IF (VALX = FALSE) THEN DO;
               FIELD2 = $$FAIL;
               CHAR_FIELD2 = 'BAD NAME';
               CALL PRIMITIVE_RELEASE(ANYPROC,$OPR_IC,
                                      MESSAGE,ERROR);
                                  END;
            ELSE DO;

         /*** CREATE INTERFACE PROCESS AND SET FAMILY
              LINKAGE.                                    ***/
               CALL PRIMITIVE_PCBDATA(MYNAME,@GET,@CHILD
                                     ,CHILD,ERROR);
               STATE_VEC(1) = PGTABVEC(1);
               STATE_VEC(2) = 1;
               CALL GETPCB(MYNAME,CHILD,PXNAME,PRI,TRUE,
                           STATE_VEC,PINAME,(0),ERROR);
               IF (CHILD ¬= 0) THEN
                  CALL PRIMITIVE_PCBDATA(CHILD,@PUT,
                                         @LFTSIB,PINAME,ERROR);
               CALL PRIMITIVE_PCBDATA(MYNAME,@PUT,
                                      @CHILD,PINAME,ERROR);
               CALL CREATE_RCB(#DEVICE,RXNAME,PINAME,(0)
                    ,S_OR_P,CHAR_FIELD2,INTRNR,(0),(0),
                    RINAME,ERROR);
```

```
                    /* SETUP RESOURCE VECTOR FOR PINAME */
                    RESVEC = '00'B;
                    RESVEC(RINAME) = ##ACCES;
                    RESVEC($FILEOP) = ##ACCES;
                    RESVEC($ERROR) = ##ACCES;
                    RESVEC($RFILEW) = ##ACCES;
                    RESVEC($WAIT) = ##ACCES;
                    RESVEC($INTDEV) = ##ACCES;
                    RESVEC($INTRPT) = ##ACCES;
                    RESVEC($OBUFF) = ##ACCES;
                    RESVEC($IBUFF) = ##ACCES;
                    CALL PRIMITIVE_PCBDATA(PINAME,@PUT,
                         @RESVEC,SEM_L_LIMIT,PCT_U_LIMIT,
                         RESVEC,ERROR);
                    CALL PRIMITIVE_PCBDATA(PINAME,@PUT,
                         @BRMVEC,IT,NRPGS,PGTABVEC,ERROR);

               /*** GET MESSAGE BUFFER FOR THE PROCESS     ****/
                    ALLOCATE MESSAGE_BUFFER SET(TEMP_PTR);
                    CALL PRIMITIVE_PCBDATA(PINAME,@PUT,
                                 @MSGPTR,TEMP_PTR,ERROR);

               /*** RELEASE MESSAGE TO THE PROCESS IDENTIFYING
                    THE DEVICE INTERNAL NAME.                ***/
                    FIELD1 = RINAME;
                    CALL PRIMITIVE_RELEASE(PINAME,$INTDEV,
                                           MESSAGE,ERROR);
               /*** ENABLE THE DEVICE INTERRUPT  ***/
                    CALL PRIMITIVE_INTERRUPT_ENABLER(INTRNM,
                                                     DUM);
                    NULP = NULL;
               /*** ENTER THE PROCESS ON THE READY 'A' QUEUE */
                    CALL RCBPUTQ(#REDYA,#OS,PINAME,(0),PRI,
                                 NULP,(0),ERROR);
                    IF (CHAR_FIELD2 = 'NO MOUNT') THEN DO;
               /* RELEASE MESSAGE TO OPERATOR SYSTEM COMM. */
                         FIELD1 = $$OPCOM;
                         FIELD2 = $$DONE;
                         CALL PRIMITIVE_RELEASE(ANYPROC,$CPR_IO
                                                ,MESSAGE,ERROR);
                    END;
                    ELSE DO;
               /*** SEND MESSAGE TO INTERFACE PROCESS TO READ
                    THE FIRST DIRECTORY ENTRY.               ***/
                         FIELD1 = $$DIRRD;
                         FIELD2 = 1;
                         ANSWER_REQUEST = TRUE;
                         CALL PRIMITIVE_RELEASE(PINAME,$RFILEW,
                                                MESSAGE,ERROR);
                    END;
               END;
          ENDCASE;
          CASE(3):    /* DELETE A DEVICE AND DESTROY THE
                         INTERFACE PROCESS                   */
               CALL FIND_INAME(#DEVICE,CHAR_FIELD1,RINAME,
                               ERROR);
               CALL PRIMITIVE_RCEDATA(RINAME,#GET,#DINAME,
                                      INTRNM,ERROR);
               CALL PRIMITIVE_INTERRUPT_DISENABLER(INTRNM,
                                                   DUM);
                    /* DETERMINE FILES ASSOCIATED WITH THE
                       DEVICE AND THE PCT; DESTROY RCE'S */
          ENDCASE;
     END_OF_CASES;
END OF_COMM_MSG_HANDLER;
```

```
FILE_OPERATION_CONTROLLER: PROC;
   DCL (DATA,DUMMY,POSIT,FSIZE,INAME) FIXED BINARY;
   DCL (FINAME,PINAME,READ_WRITE,J) FIXED BINARY;
   DCL NO_OP FIXED BINARY INITIAL(0);
   DCL FXNAME CHAR(8);

         /*** DETERMINE ACTION AND SET INDEX  ***/
   IF (MESSAGE -> FIELD1 = $$OPENF) THEN J = 1;
      ELSE IF ((MESSAGE -> FIELD1 = $$READ) |
               (MESSAGE -> FIELD1 = $$WRITE)) THEN J = 2;
           ELSE IF (MESSAGE -> FIELD1 = $$CLOSF) THEN J=3;
                ELSE IF (MESSAGE->FIELD1=$$DSTYF)THEN J=4;
                     ELSE ERROR = 110;


   DO_ACTION_OF  CASE(J);

      CASE(1):  /* OPEN FILE: SHOULD INCLUDE SUCH TESTS
                   AS; LEGAL FILE, ACCESS, DEVICE STATUS,
                   READ OR WRITE OPERATION, IF WRITE AND
                   FILE OPEN CAN'T BE SATISFIED AT THIS
                   TIME THEN CHECK FOR POTENTIAL DEADLOCK*/

         /*** GET FILE INTERNAL NAME  ***/
                FXNAME = MESSAGE -> CHAR_FIELD1;
                CALL FIND_INAME(#FILE,FXNAME,FINAME,ERROR);
                CALL FILE_ALLOCATOR(FINAME,##OPEN,RELEASOR,
                                    FIELD2);
      ENDCASE;

      CASE(2):  /*  READ/WRITE OPERATION: CHECK FILE
                   EXISTANCE, ACCESS, AND RELEASE MESSAGE
                   TO THE APPROPRIATE DEVICE INTERFACE
                   PROCESS                                 */
                FINAME = FIELD2  ;
                  /* CHECK IF MESSAGE RELEASOR DID AN
                     OPEN ON THIS FILE.                    */
                CALL RCB_FIND(FINAME,#LEFT,#FNDOP1,POSIT,
                              RELEASOR,DATA,DUMMY,ERRCR);
                IF (POSIT = 0) THEN ERROR = 106;
                  /* VERIFY IF OPERATION (READ/WRITE)
                     REQUESTED IS LEGAL.                   */
                IF ((FIELD1 = ##WRITE) & (DATA = ##READ))
                     THEN ERROR = 116;
                  /* GET DEVICE INTERNAL NAME AND THEN
                     THE INTERFACE PROCESS'S INTERNAL
                     NAME.                                 */
                CALL PRIMITIVE_RCBDATA(FINAME,#GET,#DINAME,
                                       DATA,ERROR);
                CALL PRIMITIVE_RCBDATA(DATA,#GET,#OWNER,
                                       DATA,ERROR);
                  /* RELEASE A MESSAGE TO THE INTERFACE
                     PROCESS TO PERFORM THE OPERATION. */
                FIELD4 = RELEASOR;
                CALL PRIMITIVE_RELEASE(DATA,$RFILEW,
                                       MESSAGE,ERROR);
      ENDCASE;

      CASE(3):  /* CLOSE FILE:  VERIFY FILE EXISTANCE,
                   REMOVE PROCESS INAME FROM FILE RCB,
                   CHECK FOR OUTSTANDING OPENS, IF ANY
                   SELECT ONE OR MORE AND RELEASE A MESSAGE
                   TO THOSE PROCESSES, IF NONE CLOSE THE
                   FILE.                                   */

                FINAME = MESSAGE -> FIELD2;
                PINAME = MESSAGE -> RELEASOR;
                CALL FILE_ALLOCATOR(FINAME,##CLOSE,PINAME,
                                    NO_OP);
      ENDCASE;
```

```
      CASE(4):; /* DESTROY FILE:  PERFORM TESTS ON LEGAL
                   FILE, ACCESS, AUTHORIZED TO DESTROY,
                   IF ANY PROCESSES ARE USING OR WAITING TO
                   USE THE FILE, RELEASE A MESSAGE TO THE
                   SUPERVISOR FOR EACH PROCESS, DESTROY
                   FILE RCB, UP DATE MASTER FILE LIST, UP-
                   DATE FILE DIRECTORY, AND SEND MESSAGE TO
                   STORAGE MEMORY MANAGER.                 */
                DCL PRO_VEC(0 : PCB_LIM) BIT(1)
                      INITIAL ((PCB_LIM +1) (1) '0'B);
                FINAME = FIELD2;
           /*** GET SIZE OF THE FILE AND PCT NAME  ***/
                CALL PRIMITIVE_RCBDATA(FINAME,#GET,#CNT_SZ,
                                       FSIZE,ERROR);
                CALL PRIMITIVE_RCBDATA(FINAME,#GET,#PCTNAM,
                                       FXNAME,ERROR);
                CALL FIND_INAME(#PCT,FXNAME,INAME,ERROR);
                CALL DESTROY_RCB(FINAME,RELEASOR,PRO_VEC,
                                 ERROR);
                  /* RELEASE MESSAGE TO SUPERVISOR FOR
                     EACH PROCESS THAT WAS QUEUED UP ON
                     THIS FILE.                          */
                IF (PRO_VEC(0)) THEN
                   DO I = 1 TO PCB_LIM;
                      IF (PRO_VEC(I)) THEN DO;
                         FIELD3 = I;
                         ERROR = 118;
                                          END;
                   END;
           /* GET SPACE AVAILABLE IN PCT AND UPDATE IT */
                CALL PRIMITIVE_RCBDATA(INAME,#GET,#CNT_SZ,
                                       DATA,ERROR);
                DATA = DATA + FSIZE;
                CALL PRIMITIVE_RCBDATA(INAME,#PUT,#CNT_SZ,
                                       DATA,ERROR);

           /* RELEASE MESSAGE TO UPDATE DIRECTORY */
                FIELD1 = $$DIRDL;
                FIELD2 = FINAME;
                FIELD3 = FSIZE;
                CALL PRIMITIVE_RCBDATA(INAME,#GET,#DINAME,
                                       FINAME,ERROR);
                CALL PRIMITIVE_RCBDATA(FINAME,#GET,#OWNER,
                                       FINAME,ERROR);
                CALL PRIMITIVE_RELEASE(FINAME,$RFILEW,
                                       MESSAGE,ERROR);

           /* INFORM SPACE MANAGER OF THE CHANGE IN
              SPACE AVAILABLE FOR FILES.       */
                FIELD1 = $$ADD;
                FIELD2 = INAME;
                FIELD3 = FSIZE;
                CALL PRIMITIVE_RELEASE(ANYPROC,$SPACE,
                                       MESSAGE,ERROR);
      ENDCASE;

   END_OF_CASES;  RETURN;
END FILE_OPERATION_CONTROLLER;
```

```
FILE_ALLOCATOR: PROC(FINAME,OPERATION,PINAME,TYPE);
  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   THIS SUBROUTINE DETERMINES IF A PROCESS CAN HAVE ACCESS
   TO A FILE, IF ACCESS IS DELAYED, DOES A DEADLOCK EXIST
   OR CAN IT OCCUR; IF A FILE IS CLOSED, ARE THERE OUT-
   STANDING REQUESTS FOR THE FILE AND IF SO SATISFIES
   THEM; ETC.                                          * * * * * * */
  DCL (OPERATION,FOUND,Q_EMPTY,BDATA) BIT(1);
  DCL  DUMMY_PARM  FIXED BINARY INITIAL(0);
  DCL  DUMMY_PTR  POINTER ;
  DCL (FINAME,PINAME,TYPE,FSTAT,PRI,TPRI,TSTAT,
       PRIVATE,DATA) FIXED BINARY;
  DCL  TOPELEM FIXED BINARY INITIAL(1);
  DCL  FILE(1) BIT(2);
  DCL TBIT2 BIT(2);
  DCL CDATA CHAR(8);
  DUMMY_PTR = NULL;
  FOUND = FALSE;

  CALL PRIMITIVE_PCBDATA(PINAME,@GET,@RESVEC,FINAME,
                         FINAME,FILE,ERROR);
  CALL PRIMITIVE_RCBDATA(FINAME,#GET,#S_OR_P,PRIVATE,
                         ERROR);
  IF (OPERATION = ##OPEN) THEN DO;     /* OPEN FILE */
       /*** CHECK ACCESS AUTHORIZATION  ***/
     IF (FILE(1) ¬= ##ACCES) THEN DO; ERROR = 109; RETURN;
                             END;
       /*** CHECK IF THE FILE CAN BE OPENED  ***/
     CALL PRIMITIVE_RCBDATA(FINAME,#GET,#DSTAT,BDATA,
                            ERROR);
     IF (BDATA = ##HOLD) THEN DO; ERROR=113; RETURN; END;
     CALL PRIMITIVE_PCBDATA(PINAME,@GET,@PRIRTY,PRI,
                            ERROR);
       /*** ASSIGN THE FILE TO THE PROCESS  ***/
     FILE(1) = ##ACQRD;
     CALL PRIMITIVE_PCBDATA(PINAME,@PUT,@RESVEC,FINAME,
                            FINAME,FILE,ERROR);
     IF ((TYPE = ##WRITE) & (PRIVATE = ##SHRD)) THEN DO;
       /*** WRITE ACCESS INTO SHARED FILE: CHECK
            OWNERSHIP                                  ***/
        CALL PRIMITIVE_RCBDATA(FINAME,#GET,#OWNER,DATA,
                               ERROR);
        IF (DATA ¬= PINAME) THEN DO;
          /*** NOT OWNER: PUT ON QUEUE AND RELEASE ERROR
           MESSAGE TO THE OPERATOR                     ***/
           CALL RCBPUTQ(FINAME,#RIGHT,PINAME,DUMMY_PARM,
                        PRI,DUMMY_PTR,##WRITH,ERROR);
           ERROR = 112; RETURN;  END;
          /*** AUTHORIZED ACCESS: UPDATE WRITE COUNTER **/
        CALL PRIMITIVE_PCBDATA(PINAME,@PUT,@FWCNTR,DATA,
                               ERROR);
                                          END;
     CALL PRIMITIVE_PCBDATA(PINAME,@GET,@FSTAT,TSTAT,
                            ERROR);
     IF (TSTAT = ##SACR) THEN DO;
       /*** PROCESS IN SACRIFICE STATE ***/
        CALL DEADLOCK_SETTER(FINAME,PINAME,PRI,TYPE,TSTAT,
                             PRIVATE);
        RETURN;              END;
     IF ((TSTAT = ##READR) & (TYPE = ##WRITE) &
         (PRIVATE = ##SHRD)) THEN
       /*** UPDATE PROCESS'S FILE ACCESS STATUS  ***/
        CALL PRIMITIVE_PCBDATA(PINAME,@PUT,@FSTAT,##WRITR,
                               ERROR);
       /*** DETERMINE IF THE FILE IS AVAILABLE   ***/
     CALL PRIMITIVE_RCBDATA(FINAME,#GET,#OFILE,FSTAT,
                            ERROR);
     IF (FSTAT = ##AVAIL) THEN DO;
       /*** FILE AVAILABLE: ASSIGN THE PROCESS, ENTER
            PROCESS IN THE QUEUE AND RELEASE A MESSAGE
                                                    ***/
```

```
        CALL PRIMITIVE_RCBDATA(FINAME,#PUT,#OFILE,TYPE,
                               ERROR);
        CALL RCBPUTQ(FINAME,#LEFT,PINAME,DUMMY_PARM,PRI,
                     DUMMY_PTR,TYPE,ERROR);
        CALL PRIMITIVE_RCBDATA(PINAME,#GET,#XNAME,CDATA,
                               ERROR);
        MESSAGE -> CHAR_FIELD1 = CDATA;
        MESSAGE -> FIELD1 = FINAME;
        CALL PRIMITIVE_RELEASE(PINAME,MSG_SEMAPHORE,
                               MESSAGE,ERROR);
                             END;
     ELSE DO;
        /*** FILE ALL READY OPENED  ***/
        IF ((TYPE = ##WRITE) | (TYPE ¬= FSTAT)) THEN DO;
          /*** ACCESS NOT AVAILABLE AT THIS TIME  ***/
          CALL DEADLOCK_SETTER(FINAME,PINAME,PRI,TYPE,
                               TSTAT,PRIVATE);
          RETURN;                                  END;
          /*** ACCESS MAY BE AVAILABLE; CHECK OUTSTANDING
               OPEN QUEUE AND PERMIT THIS OPEN IF QUEUE
               IS EMPTY OR THE PRIORITY OF THIS PROCESS
               IS GREATER THAN THE PRIORITY OF THE PROCESS
               ON TOP OF THE QUEUE                  ***/
        CALL RCB_FIND(FINAME,#RIGHT,#ENDOP5,TOPELEM,
                      DUMMY_PARM,DUMMY_PARM,TPRI,ERRCR);
        IF (PRI < TPRI) THEN
           CALL RCBPUTQ(FINAME,#RIGHT,PINAME,DUMMY_PARM,
                        PRI,DUMMY_PTR,TYPE,ERROR);
        ELSE DO;
           CALL RCBPUTQ(FINAME,#LEFT,PINAME,DUMMY_PARM,
                        PRI,DUMMY_PTR,TYPE,ERROR);
           CALL PRIMITIVE_RCBDATA(PINAME,#GET,#XNAME,
                                  CDATA,ERROR);
           MESSAGE -> CHAR_FIELD1 = CDATA;
           MESSAGE -> FIELD1 = PINAME;
           CALL PRIMITIVE_RELEASE(PINAME,MSG_SEMAPHORE,
                                  MESSAGE,ERROR);
             END;
       END;
     RETURN;
   END;

  ELSE DO;                  /*  OPERATION IS CLOSE FILE  */
        /*** REMOVE THE PROCESS FROM THE QUEUE AND
             DETERMINE IF ANY OUTSTANDING REQUESTS FOR
             OPEN CAN BE SATISFIED                  ***/
    IF ((FILE(1)=##ACCES) | (FILE(1)=##NOACC))THEN RETURN;
    IF (FILE(1) = ##ACQRD) THEN
       CALL RCBGETQ(FINAME,#LEFT,PINAME,DUMMY_PARM,
                    DUMMY_PTR,DATA,FOUND,Q_EMPTY,ERRCR);
    IF ((¬FOUND) | (FILE(1) = ##SACRF)) THEN
       CALL RCBGETQ(FINAME,#RIGHT,PINAME,DUMMY_PARM,
                    DUMMY_PTR,DATA,FOUND,Q_EMPTY,ERRCR);
    IF (((DATA = ##WRITE) | (DATA = ##WRITH) |
         (DATA = ##WRITA) | (DATA = ##WRITS)) &
         (PRIVATE = ##SHRD)) THEN DO;
       CALL PRIMITIVE_PCBDATA(PINAME,@GET,@FWCNTR,DATA,
                              ERROR);
       IF (DATA = 0) THEN CALL PRIMITIVE_PCBDATA(PINAME,
                          @PUT,@FSTAT,##READ,ERROR);
    TBIT2 = FILE(1);  FILE(1) = ##ACCES;
    CALL PRIMITIVE_PCBDATA(PINAME,@PUT,@RESVEC,FINAME,
                           FINAME,FILE,ERROR);
    IF ((Q_EMPTY = FALSE) | (¬FOUND) | (TBIT2 = ##SACRF))
       THEN RETURN;
    CALL PRIMITIVE_RCBDATA(FINAME,#PUT,#OFILE,##AVAIL,
                           ERROR);
    DUMMY_PARM = 0;
    CALL RESOLVE_DEADLOCK(FINAME,DUMMY_PARM);
       END;
  RETURN;
END FILE_ALLOCATOR;
```

```
RESOLVE_DEADLOCK: PROC(RCBNR,EXEMPT_PRO);
 /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  THIS SUBROUTINE TESTS IF ANY OUTSTANDING OPEN REQUESTS
  CAN BE SATISFIED, IF A SACRIFICE HAS OCCURRED AND THE
  FILE IS AVAILABLE, CAN THE PROCESS'S SACRIFICED FILES
  BE RE-ASSIGNED, ETC.                    * * * * * * * * * * * * */
 DCL (STARTING,RQ_STAT,LQ_STAT,XFERED) BIT(1);
 DCL (ELEM,RCBNR,PINAM,DATA,PRI,FSTAT,DUMMY_PARM,
      POSIT,EXEMPT_PRO)  FIXED BINARY;
 DCL  CDATA CHAR(8);
 DCL  FILES(FILE_L_LIMIT : FILE_U_LIMIT) BIT(2);
 STARTING = TRUE;
 ELEM = 1;
 DO FOREVER;
  TOP:
       /*** GET THE TOP ELEMENT ON THE OUTSTANDING
            QUEUE                                              ***/
   CALL RCB_FIND(RCBNR,#RIGHT,#FNDOP5,ELEM,PINAM,DATA,
                 PRI,ERROR);
   ELEM = ELEM + 1;
   IF (DATA = 0) THEN RETURN;  /* R_QUE IS EMPTY */

   IF ((DATA = ##READ) | ((DATA = ##WRITE) & STARTING))
      THEN DO;          /* OPEN REQUEST NOT SACRIFICED
                           TRANSFER TO OPEN QUEUE AND
                           RELEASE MESSAGE TO PROCESS */
      STARTING = FALSE;
      CALL RCB_TRANSFERQ(RCBNR,#RIGHT,PINAM,RQ_STAT,
                         LQ_STAT,XFERED,ERROR);
      CALL PRIMITIVE_RCBDATA(RCBNR,#PUT,#OFILE,DATA,
                             ERROR);
      CALL PRIMITIVE_RCBDATA(RCBNR,#GET,#XNAME,CDATA,
                             ERROR);
      MESSAGE -> CHAR_FIELD1 = CDATA;
      MESSAGE -> FIELD1 = RCBNR;
      CALL PRIMITIVE_RELEASE(PINAM,$WAIT,MESSAGE,ERROR);
      IF ((DATA=##WRITE)|(RQ_STAT=##CLOSE)) THEN RETURN;
        END;

   ELSE DO;                 /* OPEN REQUEST SACRIFICED */
     IF (¬STARTING & (¬((DATA=##READA)|(DATA=##READS))))
       THEN RETURN;

            /* TEST SACRIFICED OPEN REQUESTS FOR
               READ OR WRITE IF NO TRANSFER WAS
               MADE ABOVE; ELSE ONLY TEST READS.  */

     IF ((DATA ¬= ##WRITH) | (PINAM ¬= EXEMPT_PRO)) THEN
        DO;
        CALL PRIMITIVE_PCBDATA(PINAM,@GET,@RESVEC,
             FILE_L_LIMIT,FILE_U_LIMIT,FILES,ERRCR);
        DO I = FILE_L_LIMIT TO FILE_U_LIMIT;
          IF ((FILES(I)=##SACRF) & (I¬=RCBNR)) THEN DO;
             /* I IS AN RCB INTERNAL NAME FOR WHICH
                PROCESS PINAM DID AN OPEN REQUEST  */
           CALL PRIMITIVE_RCBDATA(I,#GET,#OFILE,FSTAT,
                                  ERROR);
           CALL RCB_FIND(I,#RIGHT,#FNDOP1,DUMMY_PARM,
                         PINAM,DATA,DUMMY_PARM,ERRCR);
           IF ((FSTAT = ##WRITE) | (DATA = ##WRITS) |
               (DATA = ##WRITA) | (DATA = ##WRITH))
               THEN IF (STARTING) THEN GO TO TOP;
                    ELSE RETURN;
                                                    END;
        END;
```

```
                /* GOT HERE --- TRANSFER THIS PROCESS'S
                   OUTSTANDING OPEN REQUESTS AND RELEASE
                   MESSAGES TO THE PROCESS WHEN REQUIRED
                                                        */
            DCL IDATA FIXED BINARY;
            FSTAT = ##READR;
            DO I = FILE_L_LIMIT TO FILE_U_LIMIT;
              IF (FILES(I) = ##SACRF) THEN DO;
               CALL RCB_TRANSFERQ(I,#RIGHT,PINAM,RQ_STAT,
                                  LQ_STAT,XFERED,ERROR);
               CALL RCB_FIND(I,#LEFT,#FNDOP1,POSIT,PINAM,
                             DATA,PRI,ERROR);
               IF ((DATA = ##READA) | (DATA = ##READS)) THEN
                  IDATA = ##READ; ELSE LO; IDATA = ##WRITE;
                                           FSTAT = ##WRITR; END;
               CALL PRIMITIVE_RCBDATA(I,#PUT,#OFILE,IDATA,
                                      ERROR);
               CALL RCB_FIND(I,#LEFT,#FNDOP4,POSIT,PINAM,
                             IDATA, PRI,ERROR);
               IF ((DATA = ##READA) | (DATA = ##WRITA)) THEN
                  DO;
                     CALL PRIMITIVE_RCBDATA(I,#GET,#XNAME,
                                            CDATA,ERROR);
                     MESSAGE -> CHAR_FIELD1 = CDATA;
                     MESSAGE -> FIELD1 = I;
                     CALL PRIMITIVE_RELEASE(PINAM,$WAIT,
                                            MESSAGE,ERROR);
                  END;
               FILES(I) = ##ACQRD;
                                                END;
            END;
            CALL PRIMITIVE_PCBDATA(PINAM,@PUT,@RESVEC,
                 FILE_L_LIMIT,FILE_U_LIMIT,FILES,ERRCR);
            CALL PRIMITIVE_PCBDATA(PINAM,@PUT,@FSTAT,FSTAT,
                                   ERROR);
            END;
         END;
      END;
   END RESOLVE_DEADLOCK;
```

```
DEADLOCK_SETTER: PROC(FINAME,PINAME,PRI,TYPE,STAT,
                      SHAR_PRIV);
  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   THIS SUBROUTINE TRANSFERS ALL OF THE FILE OPEN REQUESTS
   FOR THE PROCESS (PINAME) TO THE OUTSTANDING QUEUE AND
   SETS THEIR STATUS TO ##READH OR ##WRITH ACCORDINGLY.
   IF A FILE BECOMES AVAILABLE , DEADLOCK RESOLVER IS
   INVOKED                                                  ***/
  DCL SHAR_PRIV FIXED BINARY;
  DCL (FINAME,PINAME,PRI,TYPE,STAT) FIXED BINARY,
       CAS_INDX FIXED BINARY(3),
      (LQSTAT,RQSTAT,TRANS) BIT(1),
      (OP,POS,FSTAT,CPRI,DATA) FIXED BINARY,
       CHEKK (FILE_L_LIMIT:FILE_U_LIMIT) BIT(1),
       DUMMY_PARM FIXED BINARY INIT (0),
       DUMMY_PTR  POINTER ;
  DUMMY_PTR = NULL;

    /* DETERMINE NECESSARY ACTION                          */
  IF(SHAR_PRIV = ##PRIV) THEN OP=1; ELSE OP = STAT;
  IF ((SHAR_PRIV = ##SHRD) & ( TYPE = ##WRITE)) THEN DO;
      CALL PRIMITIVE_PCBDATA(PINAME,@GET,@FWCNTR,DATA,
                             ERROR);
      DATA = DATA + 1;
      CALL PRIMITIVE_PCBDATA(PINAME,@PUT,@FWCNTR,DATA,
                             ERROR);
  END;
  DO_ACTION_OF CASE(OP);

    CASE(1): /* PROCESS HAS NOT OPENED A SHARED FILE ON
                A WRITE                                  */
             IF (TYPE =##READ) THEN CAS_INDX = 1;
              ELSE CAS_INDX = 2;
    ENDCASE;

    CASE(2): /* PROCESS HAS OPENED A SHARED FILE ON A
                WRITE AND IS NOT CURRENTLY SACRIFICED    */
             CAS_INDX = 2;
    ENDCASE;

    CASE(3): /* PROCESS IS CURRENTLY SACRIFICED          */
             CAS_INDX = 3;
    ENDCASE;

  END_OF_CASES;

  DO_ACTION_OF CASE(CAS_INDX);

    CASE(1): /* READ ACCESS REQUESTED - NO SACRIFICE
                CONDITION                                */
             CALL RCBPUTQ(FINAME,#RIGHT,PINAME,DUMMY_PARM,
                          PRI,DUMMY_PTR,TYPE,ERROR);
    ENDCASE;

    CASE(2): /* SACRIFICE ALL OF THIS PROCESS' FILES     */
            BEGIN;
             DCL FILES (FILE_L_LIMIT:FILE_U_LIMIT) BIT(2);
             CALL PRIMITIVE_PCBDATA(PINAME,@GET,@RESVEC,
                   FILE_L_LIMIT,FILE_U_LIMIT,FILES,ERROR);
             DO I = FILE_L_LIMIT TO FILE_U_LIMIT;
               CHEKK(I) = FALSE;
               IF ((FILES(I) = ##ACQRD) & (I ¬= PINAME))
                THEN DO;
                 FILES(I)= ##SACRF;
                  CALL RCB_TRANSFERQ(I,#LEFT,PINAME,LQSTAT,
                             RQSTAT,TRANS,ERROR);
```

```
                    /* RESET DATA IF THE OPEN QUE IS EMPTY
                       AND SET CHECK TO INDICATE THE FILE MAY
                       BE REASSIGNED                          */
                    IF (LQSTAT =##CLOSE) THEN DO;
                      CALL PRIMITIVE_RCBDATA(FINAME,#PUT,
                           #OFILE,##AVAIL,ERROR);
                      DATA = 0;
                      CALL PRIMITIVE_RCBDATA(FINAME,#PUT,
                           #OWNER,DATA,ERROR);
                      CHEKK(I) = TRUE;
                    END;
                    CALL RCB_FIND(I,#RIGHT,#FNDOP1,POS,PINAME,
                         FSTAT,CPRI,ERROR);
                    /* CHANGE THE PROCESS' OPERATION STATUS
                       ON THE QUEUE, IF NECESSARY             */

                    IF ((FSTAT > 0) & (FSTAT <=2)) THEN DO;
                      DO_ACTION_OF CASE(FSTAT);

                        CASE(1): /* CURRENTLY READ           */
                          IF (TRANS) THEN FSTAT =##READS;
                            ELSE FSTAT =##READA;
                          ENDCASE;

                        CASE(2): /* CURRENTLY WRITE          */
                          IF (TRANS) THEN FSTAT = ##WRITS;
                            ELSE FSTAT = ##WRITA;
                          ENDCASE;
                      END_OF_CASES;
                      CALL RCB_FIND(I,#RIGHT,#FNDOP2,POS,
                                    PINAME,FSTAT,CPRI,ERROR);
                    END;
                  END; /* END IF ACQUIRED */
                END; /* END DO LOOP */

                /* SET THE PROCESS' STATUS TO SACRIFICED   */
                CALL PRIMITIVE_PCBDATA(PINAME,@PUT,@FSTAT,
                     ##SACR,ERROR);
                 CALL PRIMITIVE_PCBDATA(PINAME,@PUT,@RESVEC,
                     FILE_L_LIMIT,FILE_U_LIMIT,FILES,ERROR);
                /* NOW ENTER THE FILE TO BE OPENED IN
                   SACRIFICE STATUS - ANSWER REQUIRED      */
                GO TO CASE(3);
              END;
      ENDCASE;

      CASE(3): /* ENTER THE FILE TO BE OPENED IN SACRIFICE
                  STATUS, ANSWER REQUIRED  */
               IF (TYPE =##READ) THEN FSTAT =##READA;
                ELSE  FSTAT = ##WRITA;
               CALL RCBPUTQ(FINAME,#RIGHT,PINAME,DUMMY_PARM,
                            PRI,DUMMY_PTR,FSTAT,ERROR);
               DCL FILES(1) BIT(2);
               FILES(1) = ##SACRF;
               CALL PRIMITIVE_PCBDATA(PINAME,@PUT,@RESVEC,
                    FINAME,FINAME,FILES,ERROR);
      ENDCASE;
    END_OF_CASES;

    /* ASSIGN THE FILES THAT HAVE BECOME AVAILABLE */
    DO I = FILE_L_LIMIT TO FILE_U_LIMIT;
      IF CHEKK(I) THEN CALL RESOLVE_DEADLOCK(I,PINAME);
    END;
  END DEADLOCK_SETTER;

END  FILE_MANAGER;
```

MODULE SPECIFICATION

NAME: FILE_SPACE_MANAGER TYPE: PROCESS

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS

Not Applicable: All communications handled via messages.

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
Request	Process I-Name, Semaphore, Message Pointer, Error Parameter	a) $Space - Semaphore used to identify messages for this process.
Release	Process I-Name, Semaphore, Message Pointer, Error Parameter	a) $Rfilew - Semaphore used to send messages to an interface process to update the directory. b) $XXXXXX - Semaphore used to send a message to the process requesting a file. c) $Error - Semaphore used to send messages to the Error Handler.
PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to PCB Structures module to enter or obtain data.
PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Vector Lower Limit, Vector Upper Limit, Vector Parameter, Error Parameter	Generic entry point to PCB Structures module to enter or obtain data or portions of the data stored as a vector (i.e., Resource Access Vector, Page Table Vector, etc.).
RCBData	Resource I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to RCB Handler module to enter or get data concerning a resource.
RCBPUTQ	Resource I-Name, Left/Right Queue, Process I-Name, Data Parameter, Priority, Message Pointer, Data Parameter, Error Parameter	Entry point to RCB Handler used to insert a process or a message on a specified resource queue by priority. Queues used by this process are the message semaphore and the PCT queues.
Find I-Name	Resource Type, Resource X-Name, Resource I-Name, Error Parameter	Entry point to RCB Handler used to get the internal name for the resource specified by type (file, device, etc.) and external name.

Create RCB	Resource Type, Resource X-Name, Resource Owner, Sz-Cntr Parameter, Access Identifier, PCT Name, Dev/Int Identifier, File Descriptor, File Descriptor, Resource I-Name, Error Parameter	Entry point to RCB Handler used to create an RCB of the type specified; enter descriptor data in the appropriate RCB fields; and return the resource internal name. Not all fields are used by each type of resource.
RCBGETQ	Resource I-Name, Left/Right Queue, Process I-Name, Q_Data Parameter, Message Pointer, Q_Data Parameter, Found Boolean, Q_Status Boolean, Error Parameter	This entry point to RCB Handler is used to remove a process from the specified queue for the indicated resource. The data stored in the queue are returned if the process is found and the queue status is also provided.
RCB-Find	Resource I-Name, Left/Right Queue, Find Operation, Queue Position, Process I-Name, Data Parameter, Data Parameter, Error Parameter	This entry point to RCB Handler is used to determine queue position of a process and put or get a copy of data. The process, position, or both may be specified to select a specific process, any process at the specified position or a specific process at a specified position.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE
Not applicable for processes.	

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Message Buffer	---	Based	Dynamically allocated, pointer qualified structure used for passing messages.
	Releasor	Integer	Internal name of process releasing a message.
	Answer-Request	Bit(1)	Boolean indicating if answer required.
	Message-Semaphore	Integer	Semaphore to be used in the answer.
	Buffer-Location	Pointer	Not used by this process.

Field1	Integer	a) $$ADD or $$DELET - Message from File Manager indicating that a device has been added or deleted. b) $$SPACE - Message from File Manager indicating that a file has been destroyed. c) $$PERMF or $$TEMPF - Message to create a permanent or temporary file. d) $$EXTNT - Message for a file extention.
Field2	Integer	a) PCT internal name - Identifies the PCT which was added or deleted, or on which a file was destroyed. b) $$SHRD or $$PRIV - Identifies the file access-type.
Field3	Integer	File space requested.
Field4-6	Integer	Not used.
Char-Field1	Char(8)	Identifies file external name.
Char-Field2	Char(8)	Identifies PCT external name.
Char-Field3-4	Char(8)	Not used.

MODULE DESCRIPTION

File Space Manager has been implemented to perform file creation as an independent system function. It fields requests for file creation, determines type, access, location, if specified, and availability of space. If a permanent file is requested on a specified PCT and space is not available an error condition is raised which terminates the process. For temporary files, a dummy file is created until space becomes available and then a message is released to the process requesting the file.

MODULE IMPLEMENTATION

```
%INCLUDE NAMCHGR;  /***** FILE SPACE MANAGER  ****/
(CHECK (ERROR)):

FILE_SPACE_MANAGER: PROC OPTIONS(MAIN);


%INCLUDE SIMULTR;
  SIM_START(PROCESS_FILE_SPACE_MANAGER)

%INCLUDE GENDEC;
%INCLUDE REQRELD;
%INCLUDE RCBDCL;
%INCLUDE PCBDCL;
%INCLUDE CASESTM;

    DCL (INAME,FDATA,I)  FIXED BINARY;
    DCL SPACE_VEC(PCT_L_LIMIT : PCT_U_LIMIT) BIT(1);
    DCL  OUT_SPACE_REQ FIXED BINARY STATIC INITIAL(0);
    DCL  ESTAT BIT(1);
    DCL  ERROR FIXED BINARY  INITIAL(0);
    ON CHECK (ERROR)
       BEGIN;
         IF (ERROR ¬= 0) THEN DO;
            FIELD1 = RELEASOR;  MSG_SEMAPHORE = $SPACE;
            FIELD2 = ERROR;
            (NOCHECK (ERROR)) : BEGIN;  ERROR = 0;
                     CALL PRIMITIVE_RELEASE(ANYPROC,$ERROR,
                        MESSAGE,ERROR);                END;
            GO TO START;
                              END;
       END;

%INCLUDE RRMSG;

  START:
  DO FOREVER;
     CALL PRIMITIVE_REQUEST(ANYPROC,$SPACE,MESSAGE,ERROR);

  SIM_INTERRUPT_PT

     CALL MESSAGE_INTERPRETER;

     DO_ACTION_OF  CASE(I);
        CASE(1):   /* SPACE MODIFICATION FROM FILE MANAGER
                      A DEVICE HAS BEEN ADDED OR DELETED  */
                IF (FIELD1 = $$DELET) THEN
                   SPACE_VEC(FIELD2) = FALSE;
                ELSE DO;  SPACE_VEC(FIELD2) = TRUE;
                          IF (OUT_SPACE_REQ ¬= 0) THEN
                             CALL SPACE_RESOLVER(FIELD2);
                     END;
          ENDCASE;
        CASE(2):   /* SPACE FREED BY FILE MANAGER; I.E.,
                      A FILE WAS DESTROYED.                */
              IF (OUT_SPACE_RLQ ¬= 0) THEN
                          CALL SPACE_RESOLVER(FIELD2);
          ENDCASE;
```

```
      CASE(3):    /* SPACE REQUESTED FOR PERMANENT FILE,
                     DEVICE SPECIFIED BY EXTERNAL NAME. */
                  /* GET INTERNAL NAME, IF NOT FOUND ERROR
                     CONDITION IS SET IN PRIMITIVE.      */
                  CALL FIND_INAME(#PCT,CHAR_FIELD2,INAME,
                                  ERROR);
                  /* CHECK DEVICE STATUS  IF IN HOLD SET
                     ERROR; I.E., NO FILE CAN BE CREATED */
                  CALL PRIMITIVE_RCBDATA(INAME,#GET,#DSTAT,
                                         BSTAT,ERROR);
                  IF (BSTAT = ##HOLD) THEN ERROR = 113;
                     /* STAT IS GO; CHECK IF PERMANENT FILES
                        CAN BE CREATED ON THIS UNIT.       */
                  CALL PRIMITIVE_RCBDATA(INAME,#GET,#OFILE,
                                         FDATA,ERROR);
                  IF (FDATA = ##TEMPF) THEN ERROR = 114;
                     /* NOW CHECK SPACE AVAILABLE; IF SPACE
                        NOT AVAILABLE SET ERROR CONDITION  */
                  CALL PRIMITIVE_RCBDATA(INAME,#GET,#CNT_SZ,
                                         FDATA,ERROR);
                  IF (FIELD3 > FDATA) THEN ERROR = 115;
                     /* ALL TESTS COMPLETED SUCCESSFULLY;
                        CREATE THE FILE, ENTER IN DIRECTORY,
                        UPDATE SPACE, ETC.                 */
                  CALL FILE_CREATOR(INAME);
                  FDATA = FDATA - FIELD3;
                  CALL PRIMITIVE_RCBDATA(INAME,#PUT,#CNT_SZ,
                                         FDATA,ERROR);
        ENDCASE;
      CASE(4):    /* CREATE A TEMPORARY FILE WHERE SPACE
                     IS AVAILABLE; ELSE CREATE A DUMMY
                     FILE UNTIL SPACE BECOMES AVAILABLE. */
                  DO J = PCT_L_LIMIT TO PCT_U_LIMIT - 1;
                     IF (SPACE_VEC(J)) THEN DO;
                        CALL PRIMITIVE_RCBDATA(J,#GET,#CFILE,
                                               FDATA,ERROR);
                        IF (FDATA ¬= ##PERMF) THEN DO;
                           CALL PRIMITIVE_RCBDATA(J,#GET,
                                #CNT_SZ,FDATA,ERROR);
                           IF (FIELD3 <= FDATA) THEN
                                                GO TO CREATEF;
                                                END;
                                          END;
                  END;
                  IF (J > PCT_U_LIMIT) THEN J = 0;
                     /*  CREATE A DUMMY FILE  */
            CREATEF: CALL FILE_CREATOR(J);
                  FDATA = FDATA - FIELD3;
                  IF (J ¬=0) THEN CALL PRIMITIVE_RCBDATA(J,
                     #PUT,#CNT_SZ,FDATA,ERROR);
        ENDCASE;
      CASE(5):    /* A FILE EXTENTION REQUESTED; CHECK
                     FOR SPACE AVAILABLE ON SAME UNIT;
                     IF NOT AVAILABLE SET ERROR CONDITION
                     ELSE ALLOCATE SPACE, UPDATE DIRECTORY,
                     UPDATE FILE RCB, ETC.              */
                  /* TO BE IMPLEMENTED LATER  */
        ENDCASE;
    END_OF_CASES;
END;  /* END DO FOREVER  */

SIM_END
```

```
MESSAGE_INTERPRETER: PROC;
   IF ((FIELD1 = $$ADD) | (FIELD1 = $$DELET)) THEN I = 1;
      ELSE IF (FIELD1 = $$SPACE) THEN I = 2;
           ELSE IF (FIELD1 = $$PERMF) THEN I = 3;
                ELSE IF (FIELD1 = $$TEMPF) THEN I = 4;
                     ELSE IF (FIELD1 = $$EXTNT) THEN I=5;
                          ELSE ERROR = 110;
   RETURN;
END MESSAGE_INTERPRETER;

FILE_CREATOR: PROC(PCTINAM);

     /* THIS SUBROUTINE CREATES A FILE (REAL OR DUMMY) AND
        RELEASES MESSAGES TO THE PROCESS REQUESTING A FILE
        AND TO UPDATE THE DIRECTORY, OR QUEUES REQUESTS
        UNTIL SPACE BECOMES AVAILABLE                        */

   DCL (PCTINAM,RCBNR,FDATA,AVAIL,PINAME) FIXED BINARY;
   DCL CDATA CHAR(8);
   DCL DUMMY_PTR POINTER;
   IF (PCTINAM ¬= 0) THEN DO;
        /*  GET DATA FOR CREATING A FILE  */
     CALL PRIMITIVE_RCBDATA(PCTINAM,#GET,#XNAME,CDATA,
                            ERROR);
     CALL PRIMITIVE_RCBDATA(PCTINAM,#GET,#DINAME,FDATA,
                            ERROR);
     CALL PRIMITIVE_RCBDATA(FDATA,#GET,#CWNER,PINAME,
                            ERROR);
     AVAIL = ##AVAIL;
                          END;
              /*  SET VARIABLES FOR CREATING A DUMMY FILE*/
   ELSE DO; CDATA = ' '; FDATA = 0; AVAIL = ##NOAVL; END;
   CALL CREATE_RCB(#FILE,CHAR_FIELD1,RELEASOR,FIELD3,
                   FIELD2,CDATA,FDATA,AVAIL,FIELD1,RCBNR,
                   ERROR);
   IF (PCTINAM ¬= 0) THEN DO;
      /*  FILE CREATED: RELEASE A MESSAGE TO THE INTERFACE
          PROCESS TO UPDATE THE DIRECTORY                    */
               ANSWER_REQUEST = FALSE;
               FIELD5 = FIELD1;
               FIELD6 = FIELD2;
               FIELD1 = $$DIRAD;
               FIELD2 = 0;
               FIELD4 = FDATA;
               CALL PRIMITIVE_RELEASE(PINAME,$RFILEW,
                                      MESSAGE,ERROR);
                 /*  RELEASE A MESSAGE TO THE PROCESS
                     REQUESTING A FILE.                      */
               FIELD1 = RCBNR;
               CHAR_FIELD2 = CDATA;
               CALL PRIMITIVE_RELEASE(RELEASOR,
                    MSG_SEMAPHORE,MESSAGE,ERROR);
                    END;
   ELSE DO;  /* DUMMY FILE CREATED; QUEUE UP INFO UNTIL
                SPACE AVAILABLE AND MESSAGE CAN BE
                RELEASED                                    */
        DUMMY_PTR = NULL;
        CALL PRIMITIVE_PCBDATA(RELEASOR,@GET,@PRIRTY,
                               FDATA,ERROR);
        CALL RCBPUTQ(PCT_U_LIMIT,$RIGHT,RELEASOR,
             MSG_SEMAPHORE,FDATA,DUMMY_PTR,RCBNR,ERROR);
        OUT_SPACE_REQ = OUT_SPACE_REQ + 1;
        END;
END FILE_CREATOR;


SPACE_RESCIVER: PROC(PCTINAM);

      /*  THIS SUBROUTINE ATTEMPTS TO RESOLVE OUTSTANDING
          FILE CREATION REQUESTS                            */
```

```
      DCL (PCTINAM,DAT,POSIT,PINAM,FINAM,PRI,SEMA,DAT2,DATA)
          FIXED BINARY;
      DCL (FOUND,Q_EMPTY) BIT(1), DUMMY_PTR POINTER;
      DCL CDATA CHAR(8);
      POSIT = 0;

      DO WHILE (OUT_SPACE_REQ ¬= 0);  POSIT = POSIT + 1;
         /* GET THE SPACE AVAILABLE; AN OUTSTANDING REQUEST
            FOR A FILE FROM THE QUEUE; AND THE SPACE REQUIRED
            FOR THIS FILE REQUEST.                          */
      CALL PRIMITIVE_RCBDATA(PCTINAM,#GET,#CNT_SZ,DAT,ERROR);
      CALL RCB_FIND(PCT_U_LIMIT,#RIGHT,#FNDOP5,POSIT,PINAM,
                    FINAM,PRI,ERROR);
      CALL PRIMITIVE_RCBDATA(FINAM,#GET,#CNT_SZ,DAT2,ERROR);
      IF (DAT >= DAT2) THEN DO;
            /*  SUFFICIENT SPACE IS AVAILABLE: REMOVE THE
                FILE REQUEST FROM THE QUEUE.                */
         CALL RCBGETQ(PCT_U_LIMIT,#RIGHT,PINAM,SEMA,DUMMY_PTR
                      ,FINAM,FOUND,Q_EMPTY,ERROR);
            /* UPDATE THE SPACE AVAILABLE                   */
         DATA = DAT - DAT2;
         CALL PRIMITIVE_RCBDATA(PCTINAM,#PUT,#CNT_SZ,
                                DATA,ERROR);
         OUT_SPACE_REQ = OUT_SPACE_REQ - 1;
            /*  INITIALIZE FILE RCB ELEMENTS.               */
         CALL PRIMITIVE_RCBDATA(FINAM,#PUT,#OFILE,##AVAIL,
                                ERROR);
         CALL PRIMITIVE_RCBDATA(PCTINAM,#GET,#DINAME,DATA,
                                ERROR);
         CALL PRIMITIVE_RCBDATA(FINAM,#PUT,#DINAME,DATA,
                                ERROR);

            /* RELEASE A MESSAGE TO THE INTERFACE PROCESS
               TO UPDATE THE PCT DIRECTORY.                 */
         FIELD4 = DATA;
         CALL PRIMITIVE_RCBDATA(DATA,#GET,#OWNER,DATA,ERROR);
         ANSWER_REQUEST = FALSE;
         FIELD1 = $$DIRAD;
         FIELD2 = FINAM;
         FIELD3 = DAT2;
         CALL PRIMITIVE_RCBDATA(FINAM,#GET,#TFILE,FIELD5,
                                ERROR);
         CALL PRIMITIVE_RCBDATA(FINAM,#GET,#S_OR_P,FIELD6,
                                ERROR);
         CALL PRIMITIVE_RELEASE(DATA,$RFILEW,MESSAGE,ERROR);

             /* SETUP AND RELEASE A MESSAGE TO THE PROCESS
                REQUESTING A FILE.                          */
         CALL PRIMITIVE_RCBDATA(PCTINAM,#GET,#XNAME,CDATA,
                                ERROR);
         CALL PRIMITIVE_RCBDATA(FINAM,#PUT,#XNAME,CDATA,
                                ERROR);
         FIELD1 = FINAM;
         CHAR_FIELD2 = CDATA;
         ANSWER_REQUEST = FALSE;
         MSG_SEMAPHORE = 0;
         FIELD2, FIELD3 = 0;
         CHAR_FIELD1 = ' ';
         CALL PRIMITIVE_RELEASE(PINAM,SEMA,MESSAGE,ERROR);
         POSIT = POSIT - 1;
         DAT = DAT - DAT2;
                              END;
         /*  CHECK FOR MORE OUTSTANDING REQUESTS TO BE
             SATISIFIED.                                    */
      IF (POSIT = OUT_SPACE_REQ) THEN RETURN;
      END;
      RETURN;
   END SPACE_RESOLVER;

END FILE_SPACE_MANAGER;
```

MODULE SPECIFICATION

NAME: INTERRUPT HANDLER TYPE: PROCESS

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
None			

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
Release	Process I-Name, Semaphore, Message Pointer, Error Parameter	a) $Error - Semaphore used to send messages to the Error Handler. b) $XXXXXX - Semaphore used to send messages to an interface process or process a process expecting an interrupt message.
Interrupt Disabler	Interrupt Number, Interrupt Save-Vector	This module is invoked to disable all interrupts while the handler is in execution.
Interrupt Enabler	Interrupt Number, Interrupt Save-Vector	This module is invoked to enable a specific interrupt or enable all interrupts disabled by this process.
Savestate	CPIname, Processor, Error Parameter	This module is invoked to save the current state of execution of a process being preempted.
Restore-State	PIName, Processor, Error Parameter	This module is invoked to restore the state of the preempted process.
RCBData	Resource I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to RCB Handler module to enter or get data concerning a resource.
Find I-Name	Resource Type, Resourse X-Name, Resource I-Name, Error Parameter	Entry point to RCB Handler used to get the internal name for the resource specified by type (file, device, etc.) and external name.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE
Invcked directly by hardware when an interrupt occurs.	

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Message Buffer	---	Based	Dynamically allocated, pcinter qualified structure used for sending information concerning the interrupt via a Release.
Saveint		Bit(1) Array	Array used to save the status of the interrupts; arguement in call to the Disabler and Enabler.

MODULE DESCRIPTION

This process is invoke-scheduled by hardware when ever an interrupt occurs. It determines which interrupt was set, sets up a message to the appropriate interface process and resets the interrupt. The current process is temporarily preempted until the interrupt is handled. This module has only been partially implemented.

```
%INCLUDE NAMCHGR;              /***** INTERRUPT HANDLER *****/

(CHECK (ERROR)):
INTERRUPT_HANDLER: PROC OPTIONS(MAIN);

%INCLUDE GENDEC;
%INCLUDE CSDCL;
%INCLUDE PCBDCL;
%INCLUDE REQRELD;
%INCLUDE RRMSG;

  ON CHECK (ERROR)
     BEGIN;
       IF (ERROR ¬= 0) THEN DO;
          (NOCHECK(ERROR)): BEGIN;
            FIELD1 = MYNAME;
            FIELD2 = ERROR;
            CALL PRIMITIVE_RELEASE(ANYPROC,$ERROR,MESSAGE,
                                   ERROR);
            ERROR = 0;
            GO TO RETURN_PT;
          END;
       END;
     END;

DCL  (SAV_PROC,INAME,INTF_PROC,SEMAPHORE) FIXED BINARY;
     DCL ERROR FIXED BINARY;

             /*** DISENABLE ALL INTERRUPTS ***/
     CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

             /*** SAVE THE STATE OF CURRENT PROCESS ***/
  SAV_PROC=CURRENT_PROCESS(SYS_PROCESSOR);
     CALL PRIMITIVE_SAVESTATE(SAV_PROC,SYS_PROCESSOR,ERROR);
             /*** SET CURRENT PROCESS TO INTERRUPT HANDLER */
     CURRENT_PROCESS(SYS_PROCESSOR) = MYNAME;

             /*** IDENTIFY THE INTERRUPT ***/
     CALL INTERRUPT_IDENTIFIER;

             /*** RELEASE THE MESSAGE TO PROCESS CONCERNED */
  CALL PRIMITIVE_RELEASE(INTF_PROC,SEMAPHORE,MESSAGE,ERROR);

             /*** RESTORE THE STATE OF INTERRUPTED PROCESS */
  CALL PRIMITIVE_RESTORESTATE(SAV_PROC,SYS_PROCESSOR,ERROR);

             /*** REENABLE THE INTERRUPTS ***/
     CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);

  INTERRUPT_IDENTIFIER: PROC;
     /* IDENTIFY WHICH INTERRUPT IS SET, SET MESSAGE BUFFER,
        RESET THE INTERRUPT, SET LOCAL VARIABLES ,INTF PROC
        AND SEMAPHORE,APPROPRIATELY AND TAKE CARE OF ANY
        SPECIAL ACTION REQUIRED                              */
  END INTERRUPT_IDENTIFIER;

  RETURN_PT:
END  INTERRUPT_HANDLER;
```

MODULE SPECIFICATION

NAME: RELEASE TYPE: PRIMITIVE

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
Addressee		Integer	Process, specified or unspecified, expected to do a matching Request.
Semaphore		Integer	Message/Resource class identifier.
Message		Pointer	Qualifies message buffer which contains the information to be entered in the message buffer request.
	Error	Integer	Error condition code.

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
Interrupt Disabler	Interrupt Number, Save Vector	This module is invoked to disable all interrupts while Release is in execution. The status of the interrupts is saved in the Save Vector.
Interrupt enabler	Interrupt Number, Save Vector	This module is invoked to reenable all interrupts which were disabled by the current module in execution.
RCBData	Resource I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to RCB Handler module used by this module to verify the semaphore passed as an arguement.
RCBPUTQ	Resource I-Name, Left/Right Queue, Process I-Name, Data Parameter, Priority, Message Pointer, Data Parameter, Error Parameter	Entry point to RCB Handler used to insert a process or a message on a specified resource queue by priority. Queues used by this module are the message semaphore and the Ready Active queues.
PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Vector Lower Limit, Vector Upper Limit, Vector Parameter, Error Parameter	Generic entry point to PCB Structures module used by this module to verify access authorization for the process using the semaphore.
PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to PCB Structures module to obtain the unblocked process's type and priority and change status to ready active.

Scheduler	(NONE)	This primitive is invoked to schedule a process for execution if the current process doing the release matched an outstanding request from a blocked process.
Allocator	Caller, Addressee, Semaphore, Message, Req_Rel Boolean, Match Boolean, Error Parameter	This primitive is invoked to determine if a matching request has been made for this message or the resource being released for reallocation to a process. If Match is true the process is unblocked.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE
Invoked by all system and user processes.	

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Saveint		Bit(1) Array	Array used to save the status of the interrupts; arguement in call to Disabler and Enabler.

MODULE DESCRIPTION

This primitive has been designed to provide a uniform method for interprocess communication and resource allocation. The message/resource class identifier and process access authorization are verified. The Allocator is then invoked to determine if the release can be matched. If it is, the process is unblocked and the Scheduler is invoked.

MODULE IMPLEMENTATION

```
%INCLUDE NAMCHGR;                    /*****  RELEASE  *****/
    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     THIS PRIMITIVE DETERMINES IF THE CURRENT RELEASE
     MATCHES AN OUTSTANDING REQUEST. IF IT DOES, THE BLOCKED
     PROCESS IS ACTIVATED AND THE SCHEDULER IS INVOKED.  **/

(CHECK (ERROR)):
PRIMITIVE_RELEASE: PROC(TO,SEMAPHORE,MSG_PTR,ERPRM)
                        OPTIONS(MAIN);

%INCLUDE GENDEC;
%INCLUDE OSDCL;
%INCLUDE RCBDCL;
%INCLUDE PCBDCL;

  ON CHECK (ERROR)
     BEGIN;
       IF (ERROR ¬= 0) THEN DO;
          ERPRM = ERROR;
          GO TO RETURN_POINT;
       END;
     END;

  DCL (CALLER,ADDRESSEE,SEMAPHORE,ERPRM,PRI,TO)
       FIXED BINARY;
  DCL ERROR FIXED BINARY INIT (0);
  DCL MSG_PTR POINTER;
  DCL (VALID,MATCH) BIT(1);
  DCL (SYSPRO,QUE) BIT (1);
  DCL REL BIT(1) STATIC INITIAL('1'B);
  DCL RES_VEC (SEM_L_LIMIT : SEM_LIMIT) BIT(2);
  DCL IO FIXED BINARY STATIC INIT(0);
  DCL NULP POINTER STATIC;

            /*** DISENABLE ALL INTERRUPTS***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

     /*  VERIFY SEMAPHORE AND ACCESS    */
    CALL PRIMITIVE_RCBDATA(SEMAPHORE,#GET,#ASSGND,VALID,
                           ERROR);
    IF (¬VALID) THEN ERROR = 308;
     CALLER = CURRENT_PROCESS(PROCESSOR);
    CALL PRIMITIVE_PCBDATA(CALLER,@GET,@RESVEC,SEM_L_LIMIT,
                           SEM_LIMIT,RES_VEC,ERROR);
    IF (RES_VEC(SEMAPHORE)=##NOACC) THEN ERROR = 307;

            /*** INVOKE THE ALLOCATOR ***/
     ADDRESSEE = TO;
     CALL PRIMITIVE_ALLOCATOR(CALLER,ADDRESSEE,SEMAPHORE,
                              MSG_PTR,REL,MATCH,ERROR);

     IF (MATCH) THEN DO; /*** ACTIVATE THE PROCESS ***/
        CALL PRIMITIVE_PCBDATA(ADDRESSEE,@PUT,@STATUS,
                               @@READYA,ERROR);
        CALL PRIMITIVE_PCBDATA(ADDRESSEE,@GET,@PRIRTY,PRI,
                               ERROR);
        CALL PRIMITIVE_PCBDATA(ADDRESSEE,@GET,@SYSPRO,
                               SYSPRO,ERROR);
        IF (SYSPRO = TRUE) THEN QUE = #OS; ELSE QUE = #USER;
        NULP = NULL;
        CALL RCBPUTQ(#READYA,QUE,ADDRESSEE,IO,PRI,NULP,IO,
                     ERROR);
        CALL PRIMITIVE_SCHEDULER;
                     END;
  RETURN_POINT:  /*** REENABLE THE INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);
END PRIMITIVE_RELEASE;
```

MODULE SPECIFICATION

NAME: REQUEST TYPE: PRIMITIVE

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
Addressee		Integer	Process, specified or unspecified, expected to do a matching Release.
Semaphore		Integer	Message/Resource class identifier.
Message		Pointer	Qualifies message buffer in which the matching release message information is tc be of a process doing a matching entered.
	Error	Integer	Error condition code.

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
Interrupt Disabler	Interrupt Number, Save Vector	This module is invoked tc disable all interrupts while Request is in execution. The status of the interrupts is saved in the Save Vector.
Interrupt enabler	Interrupt Number, Save Vector	This module is invoked tc reenable all interrupts which were disabled by the current module in execution.
RCBData	Resource I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to RCB Handler module used by this module to verify the semaphore passed as an arguement.
PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Vector Lower Limit, Vector Upper Limit, Vector Parameter, Error Parameter	Generic entry point to PCB Structures module used by this module to verify access authorization for the process using the semaphore.
PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to PCB Structures module to change the requesting process's status to blocked if no match .

Scheduler	(NONE)	This primitive is invoked to schedule a process for execution if the current process doing the request gets blocked for an answer or a resource.
Savestate	Requestor, Processor, Error Parameter	This primitive is invcked to save the current state of execution of the process which did an unmatched request.
Allccatcr	Caller, Addressee, Semaphore, Message, Req_Rel Boolean, Match Boolean, Error Parameter	This primitive is invoked to determine if a matching release has been made or if the requested resource is available for allocation to this process. If Match is false the process is blocked.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE
Invoked by all system and user processes.	

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Saveint		Bit(1) Array	Array used to save the status of the interrupts; arguement in call to Disabler and Enabler.

MODULE DESCRIPTION

This primitive has been designed to provide a uniform method for interprocess communication and resource allocation. The message/resource class identifier and process access authorization are verified. The Allocator is then invoked tc determine if the request can be satisfied. If not, the process is blocked and the Scheduler is invoked.

MODULE IMPLEMENTATION

```
%INCLUDE NAMCHGR;                    /***** REQUEST  *****/
    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      THIS PRIMITIVE DETERMINES IF THERE IS A MATCHING
      RELEASE FOR THE CURRENT REQUEST. IF NOT, THE INVOKING
      PROCESS IS BLOCKED, THE REQUEST QUEUED, AND SCHEDULER
      INVCKED.                                                  **/

(CHECK (ERROR)):
PRIMITIVE_REQUEST: PROC(TO,SEMAPHORE,MSG_PTR,ERPRM)
                        OPTIONS(MAIN);

%INCLUDE PCBDCL;
  %INCLUDE OSDCL;
%INCLUDE RCBDCL;

  DCL (SEMAPHORE,ADDRESSEE,CALLER,ERPRM,TO) FIXED BINARY;
  DCL ERROR FIXED BINARY INIT (0);
  DCL MSG_PTR POINTER;
  DCL REQ BIT(1) STATIC INITIAL('0'B);
  DCL VALID BIT(1) INITIAL('1'B);
  DCL RES_VEC (SEM_L_LIMIT : SEM_LIMIT) BIT(2);
  DCL MATCH BIT(1);

  ON CHECK (ERROR)
     BEGIN;
       IF (ERROR ¬= 0) THEN DO;
          ERPRM = ERROR;
          GO TO RETURN_POINT;
       END;
     END;

            /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

        /*  VERIFY SEMAPHORE AND ACCESS  */
    CALL PRIMITIVE_RCBDATA(SEMAPHORE,#GET,#ASSGND,VALID,
                           ERROR);
    IF (¬VALID) THEN ERROR = 308;
    CALLER = CURRENT_PROCESS(PROCESSOR);
    CALL PRIMITIVE_PCBDATA(CALLER,@GET,@RESVEC,SEM_L_LIMIT,
                           SEM_LIMIT,RES_VEC,ERROR);
    IF (RES_VEC(SEMAPHORE)=##NOACC) THEN ERROR = 307;

            /*** INVOKE THE ALLOCATOR ***/
    ADDRESSEE = TO;
    CALL PRIMITIVE_ALLOCATOR(CALLER,ADDRESSEE,SEMAPHORE,
                             MSG_PTR,REQ,MATCH,ERROR);

    IF (¬MATCH) THEN DO; /*** BLOCK THE INVOKING PROCESS **/
       CALL PRIMITIVE_PCBDATA(CALLER,@PUT,@STATUS,@@BLKDR,
                              ERROR);
       CALL PRIMITIVE_SAVESTATE(CALLER,PROCESSOR,ERROR);
       CURRENT_PROCESS(PROCESSOR) = 0;
       CALL PRIMITIVE_SCHEDULER;
                    END;

  RETURN_POINT:
            /*** REENABLE THE INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);
END  PRIMITIVE_REQUEST;
```

MODULE SPECIFICATION

NAME: ALLOCATOR TYPE: PRIMITIVE

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
Addressor		Integer	Process internal name which initiated the call to Request or Release.
Addressee		Integer	Process internal name to which the message is addressed or from which the message is expected; may be unspecified.
Semaphore		Integer	Message/Resource class identifier.
Message		Pointer	Qualifies message container.
Req_Rel		Bit(1)	Boolean identifying message type; request or release.
	Match	Bit(1)	Boolean specifying if the current message matched a queued message.
	Error	Integer	Error condition code.

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to PCB Structures module to obtain the priority of the process which initiated the message when no match was found.
RCBPUTQ	Resource I-Name, Left/Right Queue, Process I-Name, Data Parameter, Priority, Message Pointer, Data Parameter, Error Parameter	Entry point to RCB Handler used to insert a process or a message on a specified resource queue by priority when no match was found. The queue is specified by the semaphore and Req_Rel .
RCB-Match	Addressor, Addressee, Semaphore, Message, Req_Rel, Match, Error Parameter	This primitive is invoked to compare outstanding messages on the specified semaphore queue (if any) and returns the message buffer pointer if a match is found.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE
Request & Release	Resource allocation and message matching or queueing unmatched messages.

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Message Buffer		Based	Structure used to store unmatched message releases until matching request is received.

MODULE DESCRIPTION

This primitive has been designed to manage the allocation of resources, handle interprocess communication, and assist the primitives Request and Release. Unmatched message releases are queued on the specified semaphore queue after transfering the message data to a temporary container whereas unmatched message requests result in the queueing of the message and the process doing the request. Matched releases unblock processes which had outstanding requests satisfied and in either case, matched requests or releases, the data in the queued release or current release is transferred into the requestor's message container.

MODULE IMPLEMENTATION

```
%INCLUDE NAMCHGR;                    /*****  ALLOCATOR *****/

(CHECK (ERROR)):

PRIMITIVE_ALLOCATOR: PROC(ADDRESSOR,ADDRESSEE,SEMAPHORE,
                          MSG_PTR,REQ_REL,MATCH,ERRORPARM)
                     OPTIONS(MAIN);

%INCLUDE PCBDCL;
%INCLUDE GENDEC;
%INCLUDE RCBDCL;
%INCLUDE REQRELD;

  ON CHECK (ERROR)
     BEGIN;
       IF (ERROR ¬= 0) THEN DO;
          ERRORPARM = ERROR;
          GO TO RETURN_POINT;
       END;
     END;

    DCL (ADDRESSOR,ADDRESSEE,SEMAPHORE,PRI,DATA,ERRORPARM)
         FIXED BINARY;
    DCL ERROR FIXED BINARY INIT(0);
    DCL (MSG_PTR,MSGPTR) POINTER;
    DCL (MATCH, REQ_REL) BIT(1);

    /*  DETERMINE IF THE CURRENT MESSAGE MATCHES A QUEUED
        MESSAGE.                                          */
    CALL PRIMITIVE_RCB_MATCH(ADDRESSOR,ADDRESSEE,SEMAPHORE,
                             MSGPTR,REQ_REL,MATCH,ERROR);

    IF ((MATCH) & (¬REQ_REL)) THEN DO; /* MATCH & REQUEST */
     IF (MSG_PTR ¬= NULL) THEN DO;
       MSG_PTR -> MESSAGE_BUFFER = MSGPTR -> MESSAGE_BUFFER;
       FREE MSGPTR -> MESSAGE_BUFFER;
     END;
       RETURN;                          END;

    IF ((MATCH) & (REQ_REL)) THEN DO; /* MATCH & RELEASE */
     IF (MSG_PTR ¬= NULL) THEN
       MSGPTR -> MESSAGE_BUFFER = MSG_PTR -> MESSAGE_BUFFER;
       MSGPTR -> RELEASOR = ADDRESSOR;
       RETURN;                         END;

    IF ((¬MATCH) & (REQ_REL) & (MSG_PTR ¬= NULL)) THEN DO;
                                       /* NO MATCH & RELEASE */
       ALLOCATE MESSAGE_BUFFER SET (MSGPTR);
       MSGPTR -> MESSAGE_BUFFER = MSG_PTR -> MESSAGE_BUFFER;
       MSGPTR -> RELEASOR = ADDRESSOR;
                                       END;

    ELSE MSGPTR = MSG_PTR;             /* NO MATCH & REQUEST */

    /*  NO MATCH: QUEUE THE MESSAGE BY PRIORITY           */
    CALL PRIMITIVE_PCBDATA(ADDRESSOR,@GET,@PRIRTY,PRI,
                           ERROR);
    CALL RCBPUTQ(SEMAPHORE,REQ_REL,ADDRESSOR,ADDRESSEE,PRI,
                 MSGPTR,DATA,ERROR);

RETURN_POINT:
END PRIMITIVE_ALLOCATOR;
```

MODULE SPECIFICATION

NAME: SCHEDULER TYPE: PRIMITIVE

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
None			

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
Interrupt Disabler	Interrupt Number, Save Vector	This module is invoked to disable all interrupts while Scheduler is in execution. The status of the interrupts is saved in the Save Vector.
Interrupt enabler	Interrupt Number, Save Vector	This module is invoked tc reenable all interrupts which were disabled by the current module in execution.
Savestate	CPIname, Processor, Error Parameter	This module is invoked tc save the current state of execution of a process being preempted.
Restore-State	PIName, Processor, Error Parameter	This module is invoked tc enter the newly scheduled process's state vector into tne the allocated processor's registers.
PCBData	Process I-Name, Put/Get Identifier, Field Identifier, Data Parameter, Error Parameter	Generic entry point to PCB Structures module to enter or obtain data.
RCBGETQ	Resource I-Name, Left/Right Queue, Process I-Name, Q_Data Parameter, Message Pointer, Q_Data Parameter, Found Boolean, Q_Status Boolean, Error Parameter	This entry point to RCB Handler is used to remove a process from the specified queue for the indicated resource. The data stored in the queue are returned if the process is found and the queue status is also provided.
RCBPUTQ	Resource I-Name, Left/Right Queue, Process I-Name, Data Parameter, Priority, Message Pointer, Data Parameter, Error Parameter	This entry point to RCB Handler is used to enter a preempted process on the ready active queue after doing a savestate operation.
RCB-Find	Resource I-Name, Left/Right Queue, Find Operation, Queue Position, Process I-Name, Data Parameter, Data Parameter, Error Parameter	This entry point to RCB Handler is used to determine queue position of a process and put or get a copy of data. The process, position, or both may be specified to select a specific process, any process at the specified position or a specific process at a specified position.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE
Request	Current process blocked on unsatisfied request; hense processor available.
Release	Current process did a matching release which unblocked a process; hense, preemption possible.

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Saveint		Bit(1) Array	Array used to save the status of the interrupts; arguement in call to Disabler and Enabler.
Current_ Process		Integer Vector	This static array identifies processes assigned a processor which is identified by the array index.
CPU_CPROC_ Pri		Integer Vector	Array of priorities for the current processes.

MODULE DESCRIPTION

This primitive has been partially implemented to perform scheduling of processes with preemption only among operating system processes. System processes execute only on the system processor while user processes execute on the other processors.

MODULE IMPLEMENTATION

```
%INCLUDE NAMCHGR;       /***** SCHEDULER  *****/
   /* THIS PRIMITIVE SCHEDULES PROCESSES TO BE RUN ON
      PROCESSORS.                                        */

(CHECK (ERROR)):
PRIMITIVE_SCHEDULER: PROC OPTIONS(MAIN);

%INCLUDE OSDCL;
%INCLUDE PCBDCL;
%INCLUDE RCBDCL;
%INCLUDE GENDEC;

  ON CHECK (ERROR)
     BEGIN;
       IF (ERROR ¬= 0) THEN GO TO RETURN_POINT; END;

    DCL (PRI,POSIT,PINAME,DUMMYFB) FIXED BINARY(15),
        ERROR FIXED BINARY(15) INIT(0),
         CPU_CPROC_PRI(4) FIXED BINARY(15) STATIC INIT((4)0)
        ,(FOUND,DUMMYB1,RE_SCHED,PREEMPT) BIT(1),
         DUMMYPTR  POINTER;

            /*** DISENABLE ALL INTERRUPTS ***/
    CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINIS);

            /*** SCHEDULE USER PROCESS IF PROCESSOR IS
                 AVAILABLE                              ***/
    DO I = 1 TO NUMBCPU;
      IF ((I ¬=SYS_PROCESSOR) & (CURRENT_PROCESS(I)   = 0))
        THEN DO;
          POSIT = 1;
          CALL RCB_FIND(#REDYA,#USER,#FNDOP5,POSIT,PINAME,
                        DUMMYFB,PRI,ERROR);
          IF(PINAME ¬= 0) THEN DO;
            CALL RCBGETQ(#REDYA,#USER,PINAME,DUMMYFB,
                         DUMMYPTR,DUMMYFB,FOUND,DUMMYB1,ERROR);
            CALL PRIMITIVE_PCBDATA(PINAME,@PUT,@STATUS,
                                   @@RUN,ERROR);
            CPU_CPROC_PRI(I) = PRI;
            CALL PRIMITIVE_RESTORESTATE(PINAME,I,ERROR);
          END;
        END;
    END;
```

```
          /*** SCHEDULE SYSTEM PROCESSES:  PREEMPT IF
               NECESSARY                                    ***/
  POSIT = 1;
  CALL RCB_FIND(#REDYA,#OS,#FNDOP5,POSIT,PINAME,DUMMYFB,
               PRI,ERROR);
  IF (PINAME ¬= 0) THEN DO;
    IF (CURRENT_PROCESS(SYS_PROCESSOR) = 0) THEN
      RE_SCHED = TRUE;
    ELSE DO;
      RE_SCHED = FALSE;
      IF (PRI > CPU_CPROC_PRI(SYS_PROCESSOR)) THEN DO;
        PREEMPT = TRUE;
        CALL RCBPUTQ(#REDYA,#OS, CURRENT_PROCESS(
             SYS_PROCESSOR),DUMMYFB,CPU_CPROC_PRI(
             SYS_PROCESSOR),DUMMYPTR,DUMMYFB,ERROR);
        CALL PRIMITIVE_SAVESTATE(CURRENT_PROCESS(
             SYS_PROCESSOR),SYS_PROCESSOR,ERROR);
        CALL PRIMITIVE_PCBDATA(CURRENT_PROCESS(
             SYS_PROCESSOR),@PUT,@STATUS,@@REDYA,ERROR);
      END;
      ELSE PREEMPT = FALSE;
    END;
    IF (RE_SCHED | PREEMPT) THEN DO;
      CPU_CPROC_PRI(SYS_PROCESSOR) = PRI;
      CALL RCBGETQ(#REDYA,#OS,PINAME,DUMMYFB,DUMMYPTR,
           DUMMYFB,FOUND,DUMMYB1,ERROR);
      CALL PRIMITIVE_PCBDATA(PINAME,@PUT,@STATUS,@@RUN,
           ERROR);
      CALL PRIMITIVE_RESTORESTATE(PINAME,SYS_PROCESSOR,
           ERROR);
    END;
  END;

          /*** REENABLE THE INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);
RETURN_POINT:

END PRIMITIVE_SCHEDULER;
```

MODULE SPECIFICATION

NAME: DEVICE DIRECTORY TYPE: PRIMITIVE

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
X-Name		Integer	Identifies device external name.
	Sha_Pri	Integer	Device access type: shared or private.
	Xnam_Pro	Char(8)	External name of the interface process.
	NR_PGS	Integer	Number of pages in page table.
	PgIocVec	Integer Array	Vector of the interface process's page addresses.
	Priority	Integer	Priority of interface process.
	Interrupt	Integer	Device interrupt identifier.
	Found	Bit(1)	Boolean indicating that the device specified by external name was/was not in the Device Directory.

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
Interrupt Disabler	Interrupt Number, Save Vector	This module is invoked to disable all interrupts while Device Directory is executing. The status of the interrupts is saved in the Save Vector.
Interrupt enabler	Interrupt Number, Save Vector	This module is invoked to reenable all interrupts which were disabled by the current module in execution.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE
File Manager, Input Controller, & Output Controller	This primitive is invoked to obtain the required data to create a device RCB and a device interface process.

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Saveint		Bit(1) Array	Array used to save the status of the interrupts; arguement in call to Disabler and Enabler.
Directory		Static Array	Dictionary, indexed by external name, of the computer system I/O devices.
	Device Name	Char(8)	Device external name.
	Access Type	Integer	Device access specification: shared or private.
	Process Name	Char(8)	Interface process external name.
	Nr. Pgs	Integer	Number of pages in interface processes page table.
	Page Vector	Integer Array	List of page addresses for the Interface process.
	Priority	Integer	Priority of interface process.
	Interrupt Number	Integer	Identifies device interrupt number.

MODULE DESCRIPTION

This primitive has been designed to provide the system with the necessary information to create an RCB for an I/O device and an associated interface process. The device external name need only be specified to obtain this information.

MODULE IMPLEMENTATION

```
%INCLUDE NAMCHGR;                    /* DEVICE DIRECTORY */

        /* * * * * * * * * * * * * * * * * * * * * * * * * * * *
         THIS PRIMITIVE IS USED BY SYSTEM PROCESSES TO GET
         THE REQUIRED INFORMATION TO CREATE AN RCB FOR A
         DEVICE AND A PCB FOR THE INTERFACE PROCESS.          */

DEVICE_DIRECTORY:PROC(XNAME,SHA_PRI,XNAM_PRO,NRPGS,PGLCCVEC,
                      PRI,INTRPTNR,FOUND) OPTIONS(MAIN);

%INCLUDE GENDEC;
%INCLUDE CSDCL;

     DCL (XNAME,XNAM_PRO) CHAR(8);
     DCL (SHA_PRI,NRPGS,PRI,INTRPTNR) FIXED BINARY;
     DCL PGLCCVEC(*) FIXED BINARY;
     DCL FOUND BIT(1);
     DCL NR_ENTRIES FIXED BINARY STATIC INIT(3);

     DCL 1 DIRECTORY (3) STATIC,
         2 DEVNAME CHAR(8) INIT('IPDEV1','FPDEV1','OPDEV1'),
         2 SHARPRI FIXED BINARY INIT(1,1,1),
         2 PROCXNAM CHAR(8) INIT('INPRO1','FPRO1','OUTPRO1'),
         2 NUM_PGS FIXED BINARY INIT(1,1,1),
         2 PG_LOC_VEC (1) FIXED BINARY INIT(9,10,11),
         2 PRIORITY FIXED BINARY INIT(45,47,45),
         2 INTERRUPT_LOC FIXED BINARY INIT(1,2,3);

           /*** DISENABLE ALL INTERRUPTS ***/
     CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

     /*  SEARCH THE DIRECTORY FOR THE EXTERNAL NAME          */
     DO I = 1 TO NR_ENTRIES WHILE (DEVNAME(I) ¬=XNAME); END;
     IF (I > NR_ENTRIES) THEN DO; FOUND=FALSE; RETURN; END;
     /*  IF FOUND TRANSFER THE DATA                          */
     SHA_PRI = SHARPRI(I);
     XNAM_PRO = PROCXNAM(I);
     NRPGS = NUM_PGS(I);
     PGLCCVEC = PG_LOC_VEC(I,*);
     PRI = PRIORITY(I);
     INTRPTNR = INTERRUPT_LOC(I);
     FOUND = TRUE;

           /*** ENABLE THE INTERRUPTS ***/
     CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);

     RETURN;
END DEVICE_DIRECTORY;
```

MODULE SPECIFICATION

NAME: PCB_STRUCTURES TYPE: PRIMITIVE

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
Parameters specified with the appropriate entry point.			

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
Interrupt Disabler	Interrupt Number, Save Vector	This module is invoked to disable all interrupts while PCB Handler is in execution. The status of the interrupts is saved in the Save Vector.
Interrupt enabler	Interrupt Number, Save Vector	This module is invoked to reenable all interrupts which were disabled by the current module in execution.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE
Invcked by processes and other primitives.	

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Saveint		Bit(1) Array	Array used to save the status of the interrupts; arguement in call to Disenabler and Enabler.
PCB_REF		Pointer Array	This vector contains the PCB reference pointers for created processes. The index number is the process internal name. The dimension is bounded by the external variable PCB_LIMIT, initialized at IPL, which specifies the maximum number of processes that can exist at any time in the system.
PCB		Based	Process Control Block.
	External Name	Char(8)	Process external name.
	Parent	Integer	Process creator.
	Child	Integer	Internal name of dependent, related process.
	Left-Sibling	Integer	Links independent, related processes; value is a process internal name or zero.

Right-Sibling	Integer	Links independent, related processes; value is a process internal name or zero.
PCE_Page Table_Vec	Pointer	Qualifies the Page Table structure containing the page addresses.
File_ Write_Cntr	Integer	Value is the number of current file open requests for write access into a shared file.
Sys_ Process	Bit(1)	Boolean used to qualify a process as a system/user process.
Resource Vector	Bit(2) Array	Array used for identifying resource acquisition and access authorization. Values are; a) ##NOACC - access unauthorized b) ##ACCES - access authorized c) ##ACQRD - resource acquired d) ##SACRF - allocated resources have been sacrificed. The array index is a resource internal name.
File Status	Integer	Shared file usage identifier. Values are: a) ##READR - read only, b) ##WRITR - write or read and write access, c) ##SACR - sacrificed writer.
Message_ Pointer	Pointer	Qualifies the message buffer allocated to the process.
Status	Char(8)	Process system status. Values: a) @@BLKDR - blocked for a resource b) @@BLKDT - blocked for time c) @@REDYA - blocked for a processor d) @@SUSPD - process suspended e) @@RUN - process scheduled.
Priority	Integer	Process priority.
Quantum	Integer	Maximum execution time alloted for each allocation of the CPU to the process.
Cycle_ Time	Integer	Reschedule time period for a recurrent process.
Processor	Integer	Processor allocated to the process.
Registers	Integer Vector	State vector: status of CPU registers saved when execution interrupted ot the initial state of execution.

Page_Table		Based	Contains the number of pages and the page addresses.
	Number_Pages	Integer	
	Table	Integer Array	Contains the location of each page of a process's code.

MODULE_DESCRIPTION

This module has been designed to be independent and to provide sufficient flexibility for modification. A PCB structure has been defined above which is allocated dynamically. Data is entered or retrieved by invoking the module at the data-type-specified entry point. The entry point specifications have been done separately.

MODULE IMPLEMENTATION

```
%INCLUDE NAMCHGR;       /*****  PCB STRUCTURES *****/

PCBSTR: PROC OPTIONS(MAIN);

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

     OPERATING SYSTEM SUBMODULE 1 OF DATA STRUCTURES.

  THIS MODULE IS DESIGNED TO BE INDEPENDENT OF THE OTHER
  O.S. MODULES AND TO PROVIDE SUFFICIENT FLEXIBILITY FOR
  EASE OF MODIFICATION. A PRIMITIVE PCB STRUCTURE IS
  DEFINED FOR WHICH SPACE IS ALLOCATED DYNAMICALLY.
  INFORMATION IS ENTERED INTO AND RETRIEVED FROM ELEMENTS
  OF THE STRUCTURE THROUGH A CALL TO THIS MODULE AT A
  SPECIFIED ENTRY POINT DEPENDING ON THE TYPE OF DATA
  (IE. CHARACTER,BIT STRING, INTEGER, INTEGER ARRAY).
  A SIMPLIFIED PAGE TABLE IS SIMILARLY DEFINED AND
  ALLOCATED, AND IS LINKED TO THE APPROPRIATE PCB.
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */


%INCLUDE GENDEC;
%INCLUDE CSDCL;
%INCLUDE PCBDCL;
%INCLUDE CASESTM;

     DCL NUME_PAGES FIXED BINARY;

     DCL PCB_REF_ARRAY(50)  POINTER STATIC;


     DCL 1 PCB BASED (PCB_PTR) ALIGNED,

         2 EXTERNAL_NAME CHAR(8),

         2 IMMEDIATE_RELATIVES,
           3 ( PARENT,
               CHILD,
               LEFT_SIBLING,
               RIGHT_SIBLING  )FIXED BINARY,

         2 PCB_PAGE_TABLE_PTR  POINTER,

         2 RESOURCES,
           3  FILE_SHARE_WRITE_CTR FIXED BINARY,
           3  VALID_SYS_PROCESS_BIT(1),
           3 RESOURCE_VEC(120) BIT(2),
           3  FILE_STATUS FIXED BINARY,
           3  MESSAGE_POINTER POINTER,

          2 PROCESS_EXECUTION_DATA,
              3 STATUS  CHAR(8),
              3 ( PRIORITY,
                  QUANTUM,
                  CYCLE_TIME,
                  PROCESSR  ) FIXED BINARY,
              3 REGISTERS(10)  FIXED BINARY(31);

     DCL 1 PAGE_TABLE BASED (PAGE_TABLE_PTR),
         2 NUMBER_PAGES FIXED BINARY,
         2 TABLE(NUME_PAGES REFER (NUMBER_PAGES))
             FIXED BINARY(15);
```

```
/*      INITIALIZE PCB REFERENCE VECTOR TO NULL AND LABEL
        INDICIES. THIS INITIALIZATION IS INTENDED TO BE
        DONE AT IPL TIME BY THE SYSTEM SUPERVISOR.          */

PCBSTRINT:  ENTRY ;

    DCL FIRST BIT(1) STATIC INIT('1'B);

            /*** DISENABLE ALL INTERRUPTS ***/
    CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

    IF FIRST THEN DO;
       FIRST = FALSE;
      PCB_REF_ARRAY = NULL;
      ALLOCATE PCB SET(PCB_PTR);
      PCB_REF_ARRAY(1) = PCB_PTR;
      PCB_PAGE_TABLE_PTR = NULL;
      QUANTUM,
      CHILD,
      LEFT_SIBLING,
      RIGHT_SIBLING,
      PARENT,
      FILE_STATUS,
      FILE_SHARE_WRITE_CTR,
      CYCLE_TIME  =  0;
      PRIORITY = 50;
      VALID_SYS_PROCESS = TRUE;
      STATUS = @@READY;
      EXTERNAL_NAME = 'ERRHNDLR';
      RESOURCE_VEC = '00'B;
      REGISTERS = 0;
    END;
            /*** ENABLE THE INTERRUPTS ***/
    CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);
     RETURN;
```

ENTRY POINT SPECIFICATION

MODULE NAME: PCB_STRUCTURES ENTRY NAME: GETPCB

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
Parent		Integer	Process creator.
Right_ Sibling		Integer	Child of the parent---may be zero.
External_ Name		Char(8)	External name of the process.
Priority		Integer	Priority of the process.
Sys_Proc		Bit(1)	System/User process boolean. Values: #OS or #USER.
State		Integer Array	Initial state vector used to set the CPU registers upon allocation of a processor for execution.
	Internal_ Name	Integer	Process internal name.
Cycle_ Time		Integer	Reschedule time period for a recurrent process.
	Error	Integer	Error condition code. Values: a) 205 - PCB space unavailable

ENTRY POINT DESCRIPTION

This entry point is invoked when a new process is being created. A PCB is allocated and initialized provided space is available and the number of allowable processes (PCB_LIMIT) is not exceeded. A process internal name is returned to the invoking module and PCB reference pointer is saved in the PCB Reference vector until the process is destroyed. The entry point is restricted from direct access by a user process.

ENTRY POINT IMPLEMENTATION

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      THIS ENTRY POINT IS INVOKED WHEN A NEW PCB IS TO BE
      ALLOCATED AND INITIALIZED.  AN INTERNAL NAME IS
      ASSIGNED AND A PCB ALLOCATED IF SPACE IS AVAILABLE
      (IF THE NUMBER OF ALLOWABLE PROGRAMS IN THE SYSTEM
      IS NOT EXCEEDED....PRESENTLY 50).                          */

GETPCB: ENTRY (FATHER,BROTHER,EXT_NAME,PRI,SYS_PROC,STATE,
               INTERNAL_NAME,CYCLE,ERROR) ;

    DCL (FATHER,BROTHER,PRI,CYCLE,INTERNAL_NAME)
       FIXED BINARY;
    DCL EXT_NAME CHAR(8);
    DCL SYS_PROC BIT(1);
    DCL STATE(*) FIXED BINARY(31);

            /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

            /*  GET INTERNAL NAME.  */
    DO INTERNAL_NAME =1 TO PCB_LIM
       WHILE (PCB_REF_ARRAY(INTERNAL_NAME)¬= NULL);
     END;
    IF (INTERNAL_NAME > PCB_LIM) THEN DO;
       INTERNAL_NAME = 0;  ERROR = 205;  GO TO R1;
       END;

            /*   ALLOCATE AND INITIALIZE THE PCB.  */
    ALLOCATE PCB SET( PCB_PTR);
    PCB_REF_ARRAY(INTERNAL_NAME)=PCB_PTR;
    PCB_PAGE_TABLE_PTR=NULL;
    QUANTUM,CHILD,LEFT_SIBLING,PROCESSR = 0;
    RESOURCE_VEC = '0'B;
    STATUS= @@SUSPD;
    PARENT=FATHER;
    RIGHT_SIBLING=BROTHER;
    PRIORITY=PRI;
    EXTERNAL_NAME=EXT_NAME;
    CYCLE_TIME=CYCLE;
    RESOURCE_VEC = '00'B;
    VALID_SYS_PROCESS = SYS_PROC;
    DO I=1 TO 10;
       REGISTERS(I)=STATE(I);
    END;

R1:
            /*** ENABLE THE INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);
    RETURN;
```

ENTRY POINT SPECIFICATION

MODULE NAME: PCB_STRUCTURES ENTRY NAME: RELPCB

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
Internal_ Name		Integer	Internal name of the process being destroyed.
	Error	Integer	Error condition code. Values: a) 201 - Invalid process name. b) 202 - Inactive PCB number.

ENTRY_POINT_DESCRIPTION

Deallocation of a PCB and associated Page Table, and freeing of the internal name for future use is accomplished when this entry point is invoked. Entry is restricted from direct access by user processes.

ENTRY_POINT_IMPLEMENTATION

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      DEALLOCATION OF A PCB AND ASSOCIATED PAGE TABLE, AND
      FREEING OF THE INTERNAL NAME FOR FUTURE USE IS
      ACCOMPLISHED WHEN THIS ENTRY POINT IS INVOKED.     */

RELPCB: ENTRY(INTERNAL_NAME,ERROR) ;

          /*** DISENABLE ALL INTERRUPTS ***/
   CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

          /*  VERIFY PCB NUMBER AND ACCESS.    */
    IF INVALID(INTERNAL_NAME) THEN GO TO R2;

          /*  DEALLOCATE THE PCB AND PAGE TAELE.   */
       IF(PCB_PAGE_TABLE_PTR¬=NULL) THEN DO;
          PAGE_TABLE_PTR=PCB_PAGE_TABLE_PTR;
          FREE PAGE_TABLE;
       END;
       PCB_REF_ARRAY(INTERNAL_NAME)=NULL;
       FREE PCB;

  R2:
    CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);
          /*** ENABLE THE INTERRUPTS ***/
     RETURN;
```

ENTRY POINT SPECIFICATION

MODULE NAME: PCB_STRUCTURES ENTRY NAME: FIND_PINAME

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
PXname		Char(8)	Process external name.
	PIname	Integer	Process internal name.
	Error	Integer	Error condition code. Values: a) 206 - Process not found.

ENTRY POINT DESCRIPTION

This entry pcint searches the list of active PCB's to find the process specified by external name and return the internal name if found.

ENTRY POINT IMPLEMENTATION

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   THIS ENTRY POINT SEARCHES THE LIST OF ACTIVE PCB'S
   TO LOCATE THE PROCESS WITH EXTERNAL NAME ('PXNAME')
   AND RETURNS ITS INTERNAL NAME ('PINAME'). IF NOT
   FOUND AN ERROR CONDITION IS RAISED.                    */

FIND_PINAME: ENTRY (PXNAME,PINAME,ERROR);

     DCL PXNAME CHAR (8);
     DCL PINAME FIXED BINARY;

          /*** DISENABLE ALL INTERRUPTS ***/
   CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

     DO I = 1 TO PCB_LIM;
        PCB_PTR = PCB_REF_ARRAY(I);
        IF ((PCB_PTR ¬= NULL) & (EXTERNAL_NAME = PXNAME))
           THEN DO;  PINAME = I;  GO TO R3;  END;
     END;
     ERROR = 206;
     PINAME = 0;

  R3:
          /*** ENABLE THE INTERRUPTS ***/
   CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);

     RETURN;
```

ENTRY POINT SPECIFICATION

MODULE NAME: PCB_STRUCTURES ENTRY NAME: PCBDATA-GENERIC

PARAMETERS INPUT	OUTPUT	TYPE	CONTENTS
**********Common Parameters To Each Entry Point**********			
PCB_Numb		Integer	Process internal name.
Put/Get		Bit(1)	Operation identifier. Values: a) @PUT - enter data in specified field of the PCB. b) @GET - return copy of data stored in specified PCB field.
Field#		Integer	PCB field identifier. Values: a) @BORMNR - Page Table reference used to obtain address of a specified page. Put operation invalid. b) @BRMVEC - Page table reference used to get a copy of the page table. c) @CHILD - PCB Child field. d) @CYCLE - PCB Cycle_Time. e) @FSTAT - File_Status. f) @FWCNTR - File_Write_Cntr. g) @LFTSIB - Left_Sibling. h) MSGPTR - Message_Pointer. i) @NR_PGS - Number_Pages. j) @PARENT - Parent. k) @PE_NR - Processor. l) @PRIRTY - Priority. m) @QUANTM - Quantum. n) @RESVEC - Resource_Vector. o) @RGTSIB - Right_sibling. p) @STATE - Registers. q) @STATUS - Status. r) @SYSPRO - Sys_Process. s) @XNAME - External_Name.
	Error	Integer	Error condition codes. Values: a) 201 - Invalid process name. b) 202 - Inactive PCB number. c) 203 - Invalid field number. d) 204 - Invalid operation.

ENTRY POINT DESCRIPTION

These entry points are invoked to enter or get a copy of data from a PCB. They all have at least four parameters in common; the first three which identify the PCB number, put or get operation, and the PCB field, respectively. The Error parameter is the last parameter in each entry point.

**********Generic Entry Points**********

DATCHAR Entry Point Specification.

PARAMETERS INPUT	OUTPUT	TYPE	CONTENTS
Charparm	Charparm	Char(8)	Transfer of character data.

DATFIXB Entry Point Specification.

PARAMETERS INPUT	OUTPUT	TYPE	CONTENTS
Fixbparm	Fixbparm	Integer	Transfer of integer data. @PUT unauthorized in fields: @BRMVEC and @NR_PGS. @GET not authorized in field @ERMVEC if the Page Table is unallocated.

DATBIT2 Entry Point Specification.

PARAMETERS INPUT	OUTPUT	TYPE	CONTENTS
L_Lim		Integer	Array lower limit.
U_Lim		Integer	Array upper limit.
B2ary	B2ary	Bit(2) Array	Transfer of bits data into or from an element of, portion of or the entire resource vector.

DATAPYS Entry Point Specification.

PARAMETERS INPUT	OUTPUT	TYPE	CONTENTS
L_Lim		Integer	Array lower limit.
U_Lim		Integer	Array upper limit.
SFBary	SFBary	Integer Array	Transfer short integer array data into or from an element of, portion of, or the entire Page Table Vector. @GET is not authorized if the table is not allocated.

DATAPYL Entry Point Specification.

PARAMETERS INPUT	OUTPUT	TYPE	CONTENTS
LFBary	LFBary	Integer Array	Transfer long integer array data into or from the PCI field - Registers.

DATBIT1 Entry Point Specification.

PARAMETERS INPUT	OUTPUT	TYPE	CONTENTS
Bit1parm		Bit(1)	Transfer a copy of data from PCB field Sys_Process only. Field initialized when process created.

DATPTR Entry Point Specification.

PARAMETERS INPUT	OUTPUT	TYPE	CONTENTS
Ptrparm	Ptrparm	Pointer	Transfer pointer data identifying the process's message buffer.

ENTRY POINT DESCRIPTION

The entry point is selected by a preprocessor generic procedure which determines the appropriate entry point by the number of arguements in the call and their data type.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      THE FOLLOWING ENTRY POINTS ARE INVOKED TO ENTER IN
      OR GET A COPY OF DATA FROM A PCB. THEY ALL HAVE AT
      LEAST FIVE PARAMETERS.  THE FIRST IS THE PCB NUMBER
      THE SECOND SPECIFIES IF THE OPERATION IS A PUT OR A
      GET, THE THIRD IDENTIFIES THE FIELD,  THE NEXT
      PARAMETERS ARE DATA PARAMETERS, AND THE LAST
      PARAMETER IS AN ERROR PARAMETER.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  DCL (PCB_NUMB,FIELD#,ERROR) FIXED BINARY,
      PUTGET BIT(1);
  DCL (L_LIM,U_LIM) FIXED BINARY;
  DCL  INDX  FIXED BINARY;
  DCL   CHAR_L_LIM FIXED BINARY STATIC INIT(1),
        CHAR_U_LIM FIXED BINARY STATIC INIT(2),
        FIXE_L_LIM FIXED BINARY STATIC INIT(1),
        FIXE_U_LIM FIXED BINARY STATIC INIT(12),
        BIT2_L_LIM FIXED BINARY STATIC INIT(1),
        BIT2_U_LIM FIXED BINARY STATIC INIT(1),
        RES_VEC_U_LIM FIXED BINARY STATIC INIT(120),
        ARYS_L_LIM FIXED BINARY STATIC INIT(1),
        ARYS_U_LIM FIXED BINARY STATIC INIT(1),
        ARYL_L_LIM FIXED BINARY STATIC INIT(1),
        ARYL_U_LIM FIXED BINARY STATIC INIT(1),
        BIT1_L_LIM FIXED BINARY STATIC INIT(1),
        BIT1_U_LIM FIXED BINARY  STATIC INIT(1),
        PTR_L_LIM FIXED BINARY  STATIC INIT(1),
      PTR_U_LIM FIXED BINARY STATIC INIT(1);


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      THIS ENTRY POINT IS INVOKED TO ENTER OR GET A COPY
      OF CHARACTER TYPE DATA.                                    */

DATCHAR: ENTRY(PCB_NUMB,PUTGET,FIELD#,CHARPARM,ERROR);

  DCL CHARPARM CHAR(8);

          /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

          /* VERIFY PCB NUMBER, ACCESS AND OPERATION. */
  IF (INVALID(PCB_NUMB)) THEN GO TO R4;
  IF ((FIELD# > CHAR_U_LIM) | (FIELD# < CHAR_L_LIM)) THEN
    DO;
      ERROR = 203;
      GO TO R4;
    END;

  /* THE INDEX INTO THE CASE STATEMENT IS COMPUTED TO
  REFLECT THE FIELD AND OPERATION DESIRED                     */
  INDX = 2* FIELD# ;
  IF (PUTGET = @GET) THEN INDX = INDX-1;

  DO ACTION OF CASE(INDX);
    CASE(1): CHARPARM = STATUS; ENDCASE;
    CASE(2): STATUS = CHARPARM; ENDCASE;
    CASE(3): CHARPARM = EXTERNAL_NAME; ENDCASE;
    CASE(4): EXTERNAL_NAME = CHARPARM; ENDCASE;
  END_OF_CASES;

R4:
          /*** ENABLE THE INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);
  RETURN;
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      THIS ENTRY POINT IS USED TO ACCESS INTEGER
        (FIXED BINARY) TYPE DATA.                                    */

DATFIXB: ENTRY(PCB_NUMB,PUTGET,FIELD#,FIXBPARM,ERROR);

  DCL FIXBPARM FIXED BINARY;

          /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

          /* VERIFY PCB NUMBER, ACCESS AND OPERATION. */
  IF (INVALID(PCB_NUMB))THEN GO TO R5;
  IF ((FIELD# < FIXB_L_LIM) | (FIELD# > FIXB_U_LIM)) THEN
    DO; ERROR = 203; GO TO R5; END;

  /* THE INDEX INTO THE CASE STATEMENT IS COMPUTED TO
  REFLECT THE FIELD AND OPERATION DESIRED                            */
  INDX = 2 * FIELD#;
  IF (PUTGET = @GET) THEN INDX = INDX - 1;

  DO ACTION_OF CASE(INDX);
    CASE(1):  FIXBPARM = PRIORITY; ENDCASE;
    CASE(2): PRIORITY = FIXBPARM; ENDCASE;
    CASE(3): FIXBPARM = QUANTUM;  ENDCASE;
    CASE(4): QUANTUM = FIXBPARM;  ENDCASE;
    CASE(5): FIXBPARM = PARENT;   ENDCASE;
    CASE(6): PARENT = FIXBPARM;   ENDCASE;
    CASE(7): FIXBPARM = CHILD;    ENDCASE;
    CASE(8): CHILD = FIXBPARM;    ENDCASE;
    CASE(9): FIXBPARM = LEFT_SIBLING;  ENDCASE;
    CASE(10): LEFT_SIBLING = FIXBPARM; ENDCASE;
    CASE(11): FIXBPARM = RIGHT_SIBLING; ENDCASE;
    CASE(12): RIGHT_SIBLING = FIXBPARM; ENDCASE;
    CASE(13): FIXBPARM = PROCESSR;      ENDCASE;
    CASE(14): PROCESSR = FIXBPARM;      ENDCASE;
    CASE(15): IF (PCB_PAGE_TABLE_PTR = NULL) THEN
                FIXBPARM = 0;
              ELSE DO;
                PAGE_TABLE_PTR = PCB_PAGE_TABLE_PTR;
                FIXBPARM = NUMBER_PAGES;
              END;
     ENDCASE;
    CASE(16): ERROR = 204;  ENDCASE;
    CASE(17): IF (PCB_PAGE_TABLE_PTR = NULL) THEN DO;
                FIXBPARM = 0;  ERROR = 204;
              END;
              ELSE DO;
                PAGE_TABLE_PTR = PCB_PAGE_TABLE_PTR;
                IF ((FIXBPARM < 1) |
                    (FIXBPARM > NUMBER_PAGES)) THEN
                  ERROR = 204;
                ELSE FIXBPARM = TABLE(FIXBPARM);
              END;
    ENDCASE;
    CASE(18): ERROR = 204;  ENDCASE;
    CASE(19): FIXBPARM = FILE_STATUS;  ENDCASE;
    CASE(20): FILE_STATUS = FIXBPARM;  ENDCASE;
    CASE(21): FIXBPARM = FILE_SHARE_WRITE_CTR;  ENDCASE;
    CASE(22): FILE_SHARE_WRITE_CTR = FIXBPARM;  ENDCASE;
    CASE(23): FIXBPARM = CYCLE_TIME; ENDCASE;
    CASE(24): CYCLE_TIME = FIXBPARM; ENDCASE;
  END_OF_CASES;

R5:
          /*** ENABLE THE INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);
  RETURN;
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      THIS ENTRY POINT IS USED TO ACCESS BIT(2) ARRAY DATA
                                                                    */

DATBIT2: ENTRY(PCB_NUMB,PUTGET,FIELD#,L_LIM,U_LIM,B2ARY,
               ERROR);

  DCL  B2ARY(*) BIT(2);

          /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

          /* VERIFY PCB NUMBER, ACCESS AND OPERATION. */
  IF (INVALID(PCB_NUMB)) THEN GO TO R6;
  IF ((FIELD# < BIT2_L_LIM) | (FIELD# > BIT2_U_LIM)) THEN
    DO;
      ERROR = 203;
      GO TO R6;
    END;
  IF ((L_LIM < 1) | (U_LIM > RES_VEC_U_LIM)) THEN
    DO;
      ERROR = 204;
      GO TO R6;
    END;

  /* THE INDEX INTO THE CASE STATEMENT IS COMPUTED TO
  REFLECT THE FIELD AND OPERATION DESIRED                    */
  INDX = 2 * FIELD#;
  IF (PUTGET = @GET) THEN INDX = INDX - 1;

  DO_ACTION_OF CASE(INDX);
    CASE(1):  DO I = L_LIM TO U_LIM;
                B2ARY(I) = RESOURCE_VEC(I);
              END;
    ENDCASE;
    CASE(2):  DO I = L_LIM TO U_LIM;
                RESOURCE_VEC(I) = B2ARY(I);
              END;
    ENDCASE;
  END_OF_CASES;

R6:
          /*** ENABLE THE INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);
  RETURN;
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
     THIS ENTRY POINT IS USED TO ACCESS FIXED BINARY(15)
      ARRAY DATA                                                    */

DATARYS: ENTRY(PCB_NUMB,PUTGET,FIELD#,L_LIM,U_LIM,SFBARY,
               ERROR);

  DCL SFBARY(*) FIXED BINARY;

          /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

          /* VERIFY PCB NUMBER, ACCESS AND OPERATION. */
  IF (INVALID(PCB_NUMB)) THEN GO TO R7;
  IF ((FIELD# < ARYS_L_LIM) | (FIELD# > ARYS_U_LIM)) THEN
    DO;
      ERROR = 203;
      GO TO R7;
    END;

  /* THE INDEX INTO THE CASE STATEMENT IS COMPUTED TO
  REFLECT THE FIELD AND OPERATION DESIRED                           */
    INDX = 2 * FIELD# ;
    IF (PUTGET = @GET) THEN INDX = INDX - 1;

    DO_ACTION_OF_CASE(INDX);
      CASE(1): IF (PCB_PAGE_TABLE_PTR = NULL) THEN
                   ERROR = 204;
                 ELSE DO;
                   PAGE_TABLE_PTR = PCB_PAGE_TABLE_PTR;
                   IF ((L_LIM < 1) | (U_LIM > NUMBER_PAGES))
                    THEN ERROR = 204;
                    ELSE
                      DO I = L_LIM TO U_LIM;
                        SFBARY(I) = TABLE(I);
                      END;
                 END;
    ENDCASE;
    CASE(2): IF((L_LIM ¬= 1) | (U_LIM < 1)) THEN DO;
               ERROR = 204;
               RETURN;
             END;
             IF (PCB_PAGE_TABLE_PTR ¬= NULL) THEN DO;
                PAGE_TABLE_PTR = PCB_PAGE_TABLE_PTR;
                FREE PAGE_TABLE;
              END;
             NUMB_PAGES = U_LIM;
             ALLOCATE PAGE_TABLE SET(PAGE_TABLE_PTR);
             PCB_PAGE_TABLE_PTR = PAGE_TABLE_PTR;
             DO I = 1 TO U_LIM;
               TABLE(I) = SFBARY(I);
             END;
    ENDCASE;
  END_OF_CASES;
R7:
          /*** ENABLE THE INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);
  RETURN;
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      THIS ENTRY POINT IS USED TO ACCESS FIXED BINARY(31)
      ARRAY DATA.                                                  */

DATARYL:  ENTRY(PCB_NUMB,PUTGET,FIELD#,LFBARY,ERROR);

  DCL LFBARY(*) FIXED BINARY(31);

          /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

          /* VERIFY PCB NUMBER, ACCESS AND OPERATION. */
  IF (INVALID(PCB_NUMB)) THEN GO TO R8;
  IF ((FIELD# < ARYL_L_LIM) | (FIELD# > ARYL_U_LIM)) THEN
    DO;
      ERROR = 204;
      GO TO R8;
    END;

  /* THE INDEX INTO THE CASE STATEMENT IS COMPUTED TO
  REFLECT THE FIELD AND OPERATION DESIRED                          */
  INDX = 2 * FIELD#;
  IF (PUTGET = @GET) THEN INDX = INDX -1;

  DO_ACTION_OF CASE(INDX);
    CASE(1): DO I = 1 TO 10;
               LFBARY(I) = REGISTERS(I);
             END;
    ENDCASE;
    CASE(2): DO I = 1 TO 10;
               REGISTERS(I) = LFBARY(I);
             END;
    ENDCASE;
  END_OF_CASES;

R8:
          /*** ENABLE THE INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);
  RETURN;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      THIS ENTRY POINT IS USED TO ACCESS BIT(1) DATA.              */

DATBIT1: ENTRY(PCB_NUMB,PUTGET,FIELD#,BIT1PARM,ERROR);

  DCL BIT1PARM BIT(1);

          /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

          /* VERIFY PCB NUMBER, ACCESS AND OPERATION. */
  IF (INVALID(PCB_NUMB)) THEN GO TO R9;
  IF ((FIELD# < BIT1_L_LIM) | (FIELD# > BIT1_U_LIM)) THEN
    DO;
      ERROR = 204;
      GO TO R9;
    END;

  /* THE INDEX INTO THE CASE STATEMENT IS COMPUTED TO
  REFLECT THE FIELD AND OPERATION DESIRED                          */
  INDX = 2 * FIELD#;
  IF (PUTGET = @GET) THEN INDX = INDX - 1;

  DO_ACTION_OF CASE(INDX);
    CASE(1): BIT1PARM = VALID_SYS_PROCESS; ENDCASE;
    CASE(2): ERROR = 204;  ENDCASE;
  END_OF_CASES;

R9:
          /*** ENABLE THE INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);
  RETURN;
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      THIS ENTRY POINT IS USED TO ACCESS POINTER DATA.  */

DATPTR: ENTRY(PCB_NUMB,PUTGET,FIELD#,PTRPARM,ERROR);

  DCL PTRPARM POINTER;

          /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

          /* VERIFY PCB NUMBER, ACCESS AND OPERATION. */
  IF (INVALID(PCB_NUMB))THEN GO TO R10;
  IF ((FIELD# < PTR_L_LIM) | (FIELD# > PTR_U_LIM)) THEN
    DO;
      ERROR = 203;
      GO TO R10;
    END;

  /* THE INDEX INTO THE CASE STATEMENT IS COMPUTED TO
  REFLECT THE FIELD AND OPERATION DESIRED                    */
  INDX = 2 * FIELD#;
  IF (PUTGET = @GET) THEN INDX = INDX - 1;

  DO_ACTION_OF_CASE(INDX);
    CASE(1):  PTRPARM = MESSAGE_POINTER;  ENDCASE;
    CASE(2):  MESSAGE_POINTER = PTRPARM;  ENDCASE;
  END_OF_CASES;

R10:
          /*** ENABLE THE INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);
  RETURN;
```

LOCAL PROCEDURE

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      INVALID CHECKS THE PCB# PASSES TO INSURE THAT THE
      PROCESS CONTROL BLOCK IS CURRENTLY ACTIVE AND ALSO
      SETS THE POINTER TO THE PROPER PCB FOR DATA ACCESS.
                                                           */
  DCL INVALID ENTRY(FIXED BINARY) RETURNS(BIT(1));

INVALID: PROC(PCB#) RETURNS(BIT(1));

  DCL PCB# FIXED BINARY;
  IF((PCB# = 0) & (PUTGET = @GET)) THEN
              PCB# = CURRENT_PROCESS(PROCESSOR);
  IF ((PCB# < 1) | (PCB# > 50)) THEN DO;
     ERROR = 201 ;
     RETURN(TRUE);
   END;
  PCB_PTR = PCB_REF_ARRAY(PCB#);
  IF (PCB_PTR = NULL) THEN DO;
     ERROR = 202;
     RETURN(TRUE);
   END;
  RETURN(FALSE);
END INVALID;

END PCBSTR;
```

MODULE SPECIFICATION

NAME: RCE_HANDLER TYPE: PRIMITIVE

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
Parameters Specified With The Entry Point Specification.			

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
Interrupt Disabler	Interrupt Number, Save Vector	This module is invoked to disable all interrupts while RCB Handler is in execution. The status of the interrupts is saved in the Save Vector.
Interrupt enabler	Interrupt Number, Save Vector	This module is invoked to reenable all interrupts which were disabled by the current module in execution.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE
Invoked by other primitives and only system processes.	

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Saveint		Bit(1) Array	Array used to save the status of the interrupts; arguement in call to Disabler and Enabler.
Resource_ Control_ Block		Structure	This structure has been designed to store administrative and operational data used in the managing of system resources; u.e., Semaphores, Files, Devices, and data storage containers (PCT's).
	Assigned	Bit(1)	Determines if the resource Control Block is assigned to a resource.
	Creator	Integer	Process that created the RCB.
	Owner	Integer	Process that has been assigned control of the resource.
	Ext_Name	Char(8)	External name of the resource.
	Device_ Status	Bit(1)	Device access state. Values: a) ##GO - the device is active b) ##HOLD - access to files and PCT for this device is not authorized. c) Not applicable for Semaphore RCB's.

Device_ Iname	Integer	Device internal name.
Shared_ Private	Integer	Resource access classification Values: a) ##SHRD or ##PRIV - shared or private resource.
Cntr_Size	Integer	a) Semaphore - Buffer, Job_q, Print_Jobs, etc. counter. b) File - file length. c) Device - not used. d) PCT - file space available.
PCT_Name	Char(8)	PCT external name.
Open_File	Integer	Applicable for files. Values: a) ##NOVAIL - file unavailable b) ##AVAIL - file available c) ##READ - file opened for reading only d) ##WRITE - file opened for writing only.
File_Type	Integer	Not applicable for semaphore or device resources. Values: a) ##TEMPF - temporary file or type file allowed on PCT b) ##PERMF - permanent file or type file allowed on PCT c) ##TORPF - both file types allowed on PCT.
Left_Que	Pointer	Qualifies queued messages. a) Semaphore - outstanding process request messages b) File - current file open requests satisfied. c) Device - current device access requests satisfied d) PCT - not used.
Right_Que	Pointer	Qualifies queued messages. a) Semaphore - outstanding process release messages b) File - outstanding file open requests c) Device - outstanding device access requests d) PCT - outstanding file creation requests.

L_R_Queue		Based	This structure is used tc save data required for the management of the resources and the matching of messages.
	From	Integer	Internal name of the process doing the Request or Release saved in the Left/Right queue for Semaphore RCB's, respectively. Internal name of the process accessing or requesting access to a file or device resource; or file creation.
	To	Integer	Internal name of the process expected to do a Release or Request, saved in a Semaphore Left/Right queue. Also used to save the Semaphore to be used in an answer to a message.
	Priority	Integer	Precedence of the process.
	File_Data	Integer	Not applicable for Semaphore or Device RCB's. Values: a) Semaphores - not applicable b) Devices - not applicable c) File Left_Que - ##REAL or ##WRITE (file opened on a read or write for a process identified in the FROM Field) d) File Right_que - ##READ or ##WRITE, ##READA or ##WRITA (outstanding read/write open request sacrificed), ##READS or ##WRITS (read/write opened request sacrificed), or ##WRITH (outstanding write-open request for a non-owned shared file e) PCT Right_Que - dummy file internal name for outstanding file creation requests.
	Msg_Ptr	Pointer	A) Semaphores - qualifies the message buffer of a process doing a Request or a temporary message buffer for outstanding Releases b) Not used in the other type RCB queues.
	Q_Top	Pointer	Backward queue link.
	Q_Bottom	Pointer	Forward queue link.

MODULE DESCRIPTION

This primitive has been implemented to provide resource management facilities to the various system processes and primitives. These services include creating and destroying Resource Control blocks; entering and getting data; and controlling access to the various type resources. The entry points utilizing the data structures defined above have been specified separately.

ENTRY_POINT_IMPLEMENTATION

```
%INCLUDE NAMCHGR;  /*****  RCB HANDLER *****/
  /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  THIS PRIMITIVE HAS BEEN IMPLEMENTED TO CENTERALIZE THE
  LOCATION OF RESOURCE DATA AND TO HIDE DATA STORAGE
  METHODS. IT PROVIDES THE INTERFACE TO PROCESSES AND
  PRIMITIVES REQUIRING SUCH SERVICES AS CREATING RESOURCE
  CONTROL BLOCKS; DESTROYING RCBS; ENTERING OR ACCESSING
  DATA; MANIPULATING QUEUE ENTRIES USED TO STORE DATA
  CONCERNING AVAILIBILITY, ACCESS, AUTHORIZATION, ALLOCATION
  AND DEALLOCATION OF THE RESOURCES; AND MATCHING OF
  INTERPROCESS COMMUNICATION MESSAGES.                        */

PRIMITIVE_RCE_HANDLER: PROC OPTIONS(MAIN);

%INCLUDE GENDEC;
%INCLUDE CSDCL;
%INCLUDE RCEDCL;
%INCLUDE CASESTM;

          /*** RESOURCE CONTROL STRUCTURE ***/
    DCL 1 RESOURCE_CONTROL_BLOCK(120) STATIC,
            2 ASSIGNED BIT(1) INIT((120) (1) '0'B),
            2 CREATOR FIXED BINARY,
            2 OWNER FIXED BINARY,
            2 LEFT_QUE POINTER,
            2 RIGHT_QUE POINTER,
            2 EXT_NAME CHAR(8),
            2 DEVICE_STATUS BIT(1),
            2 DEVICE_INAME FIXED BINARY,
            2 SHARED_PRIVATE FIXED BINARY,
            2 CNTR_SIZE FIXED BINARY,
            2 PCT_NAME CHAR(8),
            2 OPEN_FILE FIXED BINARY,
            2 FILE_TYPE FIXED BINARY;
    DCL 1 L_R_QUEUE BASED(LRQ_PTR),
          2 Q_TCP POINTER,
          2 FROM FIXED BINARY,
          2 TO FIXED BINARY,
          2 PRECEDENCE FIXED BINARY,
          2 FILE_DATA FIXED BINARY,
          2 MSG_PTR POINTER,
          2 Q_BOTTOM POINTER;

    DCL (ADDRESSOR,ADDRESSEE,DATA,RINAME) FIXED BINARY;
    DCL ERROR FIXED BINARY;
    DCL (REQ_REL,Q#) BIT(1);
    DCL (MSGPTR,START,TEMP) POINTER;

          /*** LOCAL SUBROUTINE ENTRY POINTS ***/
    DCL   VALID ENTRY(FIXED BINARY) RETURNS (BIT(1)),
      REMOVEQ ENTRY(FIXED BINARY,BIT(1),POINTER,POINTER),
      INSERT ENTRY(FIXED BINARY,BIT(1),POINTER,
                   FIXED BINARY),
      LOOKUP ENTRY(POINTER,FIXED BINARY,FIXED BINARY),
      QUEPCSIT ENTRY(FIXED BINARY,POINTER);
```

ENTRY POINT SPECIFICATION

MODULE NAME: RCB_HANDLER ENTRY NAME: CREATE_RCB

PARAMETERS INPUT	OUTPUT	TYPE	CONTENTS
Type		Integer	Specifies resource type to be created. Values are: #DEVICE, #FILE, #SEMFOR, or #PCT.
RXName		Char(8)	Resource external name.
Owner		Integer	Owner's internal name.
Size		Integer	File length or number of resource units.
S_or_P		Integer	Shared or private resource.
Mounted		Char(8)	/pCT external name - for File and Device resources only.
Dev_Name		Integer	Device internal name - for PCT and File resources only.
Open		Integer	File status - ##AVAIL or ##NOAVL.
T_File		Integer	a) Files - ##TEMPF or ##PERMF b) PCT's - ##TEMPF, ##PERMF or ##TORPF.
	RIName	Integer	Resource internal name.
	Error	Integer	Error condition code. Values: a) 101 - invalid resource type b) 102 - RCB unavailable c) 103 - invalid resource name d) 104 - unallocated resource.

ENTRY_POINT_DESCRIPTION

This entry point is used to create an RCB of the type specified and to return the internal name. Only system processes may invoke this entry point.

ENTRY POINT IMPLEMENTATION

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
THIS ENTRY POINT TO RCB HANDLER IS UTILIZED TO CREATE AN
RCB OF THE TYPE SPECIFIED; TO ENTER THE DATA IN THE
VARIOUS RCB FIELDS; TO RETURN THE RCB INTERNAL NAME,
"RINAME"; AND TO RETURN AN ERROR CODE, "ERROR".          */

CREATE_RCB: ENTRY(TYPE,RXNAME,OWNR,SIZE,S_OR_P,MOUNTED,
                  DEV_NAME,OPEN,T_FILE,RINAME,ERROR);

  DCL (TYPE,OWNR,SIZE,DEV_NAME,OPEN,CREATR,LIMIT,I)
       FIXED BINARY;
  DCL (RXNAME,MOUNTED) CHAR(8);
  DCL (S_OR_P,T_FILE) FIXED BINARY;
  DCL SYSPRO BIT(1);

          /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

  CREATR=CURRENT_PROCESS(SYS_PROCESSOR);
  IF ((TYPE < 1) | (TYPE > 4)) THEN DO;
       ERROR = 101; GO TO RETURN1; END;

  DO_ACTION_OF CASE(TYPE);

    CASE(1): I=1; LIMIT=SEM_LIMIT; ENDCASE;

    CASE(2): I=DEV_L_LIMIT; LIMIT=DEV_U_LIMIT; ENDCASE;

    CASE(3): I=FILE_L_LIMIT; LIMIT=FILE_U_LIMIT; ENDCASE;

    CASE(4): I= PCT_L_LIMIT; LIMIT= PCT_U_LIMIT; ENDCASE;
  END_OF_CASES;
    DO WHILE ((ASSIGNED(I)) & (I <= LIMIT)); I=I+1; END;
    IF (I > LIMIT) THEN DO; ERROR=102; GO TO RETURN1; END;
    ASSIGNED(I) = TRUE;
    CREATOR(I) = CREATR;
    OWNER(I) = OWNR;
    LEFT_QUE(I) , RIGHT_QUE(I) = NULL;
    EXT_NAME(I) = RXNAME;
    DEVICE_INAME(I) = DEV_NAME;
    DEVICE_STATUS(I) = ##GO;
    SHARED_PRIVATE(I) = S_OR_P;
    CNTR_SIZE(I) = SIZE;
    PCT_NAME(I) = MOUNTED;
    OPEN_FILE(I) = OPEN;
    FILE_TYPE(I) = T_FILE;
    RINAME = I;
  GO TO RETURN1;
```

ENTRY POINT SPECIFICATION

MODULE NAME: RCB_HANDLER ENTRY NAME: RCB_MATCH

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
Addressor		Integer	Internal name of the process which did a Request/Release.
Addressee	Addressee	Integer	To which process the Request or Release was addressed
Semaphore		Integer	Internal name of the message semaphore used in the Request or Release.
	M_Ptr	Pointer	Qualifies the Requestor's message buffer or the temporary buffer from an unmatched Release.
Q#		Bit(1)	Left or Right queue identifier Values: #LEFT for an incoming Request or #RIGHT for an incoming Release.
	Match	Bit(1)	Boolean which identifies to the invoker if a match has been found.
	Error	Integer	Error condition code. Values: a) 103 - invalid resource name b) 104 - unallocated resource.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE
Allocator	To determine if there is an outstanding Release or Request which matches a current Request or Release, respectively.

ENTRY POINT DESCRIPTION

This entry point has been implemented to facilitate the allocation of resources and matching of interprocess communication messages. In this implementation, the actual techniques used to perform the services indicated above and the nature of the queues are hidden from the processes; hense modification has been simplified.

ENTRY POINT IMPLEMENTATION

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
THIS ENTRY POINT TO RCB HANDLER IS UTILIZED BY THE MESSAGE
ALLOCATOR TO DETERMINE IF A REQUEST/RELEASE MESSAGE
MATCHES A QUEUED-UP RELEASE/REQUEST MESSAGE. "ADDRESSOR",
"ADDRESSEE", "SEMAPHORE", AND "Q#"      MUST BE SPECIFIED
BY THE CALLING PROCEDURE. IF A MATCH IS NOT FOUND "MATCH"
IS SET TO FALSE. IF A MATCH IS FOUND  "MATCH" IS SET TO
TRUE  AND "M_PTR" (MESSAGE POINTER) AND "ADDRESSEE" ARE
SET TO THE VALUES STORED IN THE QUEUE.                      */

PRIMITIVE_RCB_MATCH: ENTRY(ADDRESSOR,ADDRESSEE,SEMAPHORE,
                           M_PTR,Q#,MATCH,ERROR);

  DCL SEMAPHORE FIXED BINARY,
      M_PTR POINTER,
      MATCH BIT(1);

          /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

  MATCH = FALSE;
  IF (VALID(SEMAPHORE)) THEN DO;
  IF (Q# =#LEFT) THEN START = RIGHT_QUE(SEMAPHORE);
     ELSE START = LEFT_QUE(SEMAPHORE);
  IF (START = NULL) THEN GO TO RETURN2;
  TEMP = START;
  IF (ADDRESSEE = 0) THEN
    DO WHILE ( ¬((TEMP-> TO = 0)|(TEMP->TO = ADDRESSOR)));
       IF (TEMP->Q_BOTTOM = START) THEN GO TO RETURN2;
         ELSE  TEMP = TEMP->Q_BOTTOM;
    END;
   ELSE
    DO WHILE(¬(((TEMP->TO = ADDRESSOR)|(TEMP-> TO = 0))
              &  (TEMP->FROM = ADDRESSEE)));
       IF (TEMP-> Q_BOTTOM = START) THEN GO TO RETURN2;
        ELSE  TEMP = TEMP -> Q_BOTTOM;
    END;
  ADDRESSEE = TEMP->FROM;
  M_PTR = TEMP-> MSG_PTR;
  MATCH = TRUE ;
  CALL REMOVEQ(SEMAPHORE,(¬Q#),TEMP,START);
  FREE TEMP -> L_R_QUEUE;
  END;
  GO TO RETURN2;
```

ENTRY POINT SPECIFICATION

MODULE NAME: RCB_HANDLER ENTRY NAME: RCBPUTQ

PARAMETERS

INPUT / OUTPUT	TYPE	CONTENTS
RIName	Integer	Resource internal name.
Req_Rel	Bit(1)	Queue identifier. Values are: #LEFT or #RIGHT; #OS or #USER for the ready active queue: #REDYA.
Addressor	Integer	Internal name of the process being inserted in the queue.
Addressee	Integer	Internal name of the process to which the message is addressed; or the semaphore to be used in reply to a file creation task.
Pri	Integer	Priority of the process being queued.
MsgPtr	Pointer	Qualifies the process's message buffer.
Data	Integer	a) Dummy file internal name for outstanding file creation requests. b) ##READ or ##WRITE for file open requests which were allocated (file left_que) c) ##READ or ##WRITE for unallocated open requests; ##READA or ##WRITA for unallocated, sacrificed file open requests; ##READS or ##WRITS for allocated, but sacrificed, file open requests; and ##WRITH for an outstanding file open for write access to non-owned, shared file.

ENTRY POINT DESCRIPTION

This entry point is invoked to enter a process by priority into a specified queue and to save specific data.

ENTRY POINT IMPLEMENTATION

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
THIS ENTRY POINT TO RCB HANDLER IS USED TO ENTER A
PROCESS ("ADDRESSOR") BY PRIORITY ("PRI") IN A SPECIFIED
QUEUE ("REQ_REL") FOR THE RESOURCE SPECIFIED BY "RINAME".
"ADDRESSOR", "ADDRESSEE", "PRI", "MSGPTR", AND "DATA" ARE
STORED IN THE APPROPRIATE QUEUE ELEMENT FIELDS.          */

RCBPUTQ: ENTRY (RINAME, REQ_REL,ADDRESSOR,ADDRESSEE,PRI,
                MSGPTR,DATA,ERROR);

  DCL PRI FIXED BINARY;

          /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

  IF (VALID(RINAME)) THEN DO;
  ALLOCATE L_R_QUEUE SET (LRQ_PTR);
  LRQ_PTR -> FROM = ADDRESSOR;
  LRQ_PTR -> TO = ADDRESSEE;
  LRQ_PTR -> PRECEDENCE = PRI;
  LRQ_PTR -> FILE_DATA = DATA;
  LRQ_PTR -> MSG_PTR = MSGPTR;
  CALL INSERT(RINAME,REQ_REL,LRQ_PTR,PRI);
  END;
  GO TO RETURN3;
```

ENTRY POINT SPECIFICATION

MODULE NAME: RCB_HANDLER ENTRY NAME: RCBGETQ

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
RIName		Integer	Resource internal name.
Req_Rel		Bit(1)	Queue identifier. Values are: #LEFT, #RIGHT, #OS, and #USER.
Addressor		Integer	Internal name of process to be removed from the queue.
	Addressee	Integer	Data stored in the TO Field of the queue.
	MsgPtr	Pointer	Process message buffer pointer
	Data	Integer	Data stored in the File_Data Field of the queue.
	Found	Bit(1)	Indicates if the specified process was found.
	Q_Empty	Bit(1)	Queue status after removal.
	Error	Integer	Error condition code. Values: a) 103 - invalid resource name b) 104 - unallocated resource.

ENTRY_POINT_DESCRIPTION

This entry is invoked to remove a process and associated data from the specified resource queue. The data is returned to the invoker and the queue element is deallocated.

ENTRY POINT IMPLEMENTATION

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
THIS ENTRY POINT TO RCB HANDLER IS USED TO REMOVE A
PROCESS ("ADDRESSOR") FROM THE SPECIFIED QUEUE ("REQ_REL")
ASSOCIATED WITH THE RESOURCE SPECIFIED BY "RINAME". THE
DATA STORED IN THE FIELDS OF THE QUEUE ELEMENT IS RETURNED
IN "MSGPTR" AND "DATA" IF THE PROPER QUEUE ELEMENT IF
FOUND. "FOUND" IS SET TO TRUE OR FALSE INDICATING THAT
THAT THE ELEMENT HAS OR HAS NOT BEEN REMOVED. "Q_EMPTY"
INDICATES THE STATUS OF THE QUE AFTER THE ELEMENT HAS
BEEN REMOVED.                                              */

RCBGETQ: ENTRY(RINAME,REQ_REL,ADDRESSOR,ADDRESSEE,MSGPTR,
               DATA,FOUND,Q_EMPTY,ERROR);

  DCL (FOUND,Q_EMPTY) BIT(1);

        /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

  FOUND, Q_EMPTY = FALSE;
  IF (VALID(RINAME)) THEN DO;
  IF (REQ_REL =#LEFT) THEN START = LEFT_QUE(RINAME);
  ELSE START = RIGHT_QUE(RINAME);
  IF (START = NULL) THEN GO TO RETURN4;
  TEMP = START -> Q_BOTTOM;
  DO WHILE ((TEMP ¬= START) & (TEMP-> FROM ¬= ADDRESSOR));
             TEMP = TEMP -> Q_BOTTOM;
  END;
  IF (TEMP -> FROM ¬= ADDRESSOR) THEN GO TO RETURN4;
  ADDRESSEE = TEMP -> TO;
  MSGPTR = TEMP -> MSG_PTR;
  DATA = TEMP -> FILE_DATA;
  CALL REMOVEQ(RINAME,REQ_REL,TEMP,START);
  IF (START = NULL) THEN Q_EMPTY = TRUE;
  FREE TEMP -> L_R_QUEUE;
  FOUND = TRUE;
  END;
  GO TO RETURN4;
```

ENTRY POINT SPECIFICATION

MODULE NAME: RCB_HANDLER ENTRY NAME: RCB_TRANSFERQ

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
RIName		Integer	Resource internal name.
Q#		Bit(1)	Queue identifier. Values are: #LEFT or #RIGHT
PIName		Integer	Process internal name which is being transferred
	Inq_Fin_Stat	Bit(1)	Status ofQ# after transfer.
	Newq_Start_Stat	Bit(1)	Status of opposite queue before transfer.
	Xfered	Bit(1)	Status of transfer.
	Error	Integer	Error condition code. Values: a) 103 - invalid resource name b) 104 - unallocated rescurce.

ENTRY_POINT_DESCRIPTION

This entry pcint is invoked to transfer a process from one queue of an RCB to the opposite queue of that RCB and to return the status of both queues after and before transfer, respectively.

ENTRY_POINT_IMPLEMENTATION

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
THIS ENTRY POINT TO RCB HANDLER IS USED TO TRANSFER A
PROCESS ("PINAME") FROM ONE QUE ("Q#") OF AN RCB
("RINAME") TO THE OTHER QUEUE. "XFERED" INDICATES THAT
THE OPERATION HAS/HAS NOT BEEN PERFORMED. "INQ_FIN_STAT"
INDICATES THE STATUS OF "Q#" AFTER THE TRANSFER.
"NEWQ_START_STAT" INDICATES THE STATUS OF THE QUEUE
OPPOSITE "Q#" BEFORE THE TRANSFER.                              */

 RCB_TRANSFERQ:  ENTRY(RINAME,Q#,PINAME,INQ_FIN_STAT,
                 NEWQ_START_STAT,XFERED,ERROR);

  DCL PINAME FIXED BINARY;
  DCL (INQ_FIN_STAT,NEWQ_START_STAT,XFERED,NEWQ)  BIT(1);
  DCL (START1,START2) POINTER;

          /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

  XFERED = FALSE;
  IF (VALID(RINAME)) THEN DO;
  IF (Q# = #LEFT) THEN DO;
    START1,TEMP = LEFT_QUE(RINAME);
    START2 = RIGHT_QUE(RINAME);
   END;
   ELSE DO;
    START1,TEMP = RIGHT_QUE(RINAME);
    START2 = LEFT_QUE(RINAME);
  END;
  IF (START2 = NULL) THEN NEWQ_START_STAT = ##CLOSE;
   ELSE  NEWQ_START_STAT = ##OPEN;
  CALL LOOKUP(TEMP,PINAME,IPOS);
  IF (IPOS = 0) THEN GO TO RETURN5;
  CALL REMOVEQ(RINAME,Q#,TEMP,START1);
  IF (START1 = NULL) THEN INQ_FIN_STAT = ##CLOSE;
   ELSE INQ_FIN_STAT = ##OPEN;
   NEWQ = ¬ Q#;
   PRI = TEMP -> PRECEDENCE;
   CALL INSERT(RINAME,NEWQ,TEMP,PRI);
   XFERED = TRUE;
  END;
  GO TO RETURN5;
```

ENTRY POINT SPECIFICATION

MODULE NAME: RCB_HANDLER ENTRY NAME: RCB_FIND

PARAMETERS INPUT	OUTPUT	TYPE	CONTENTS
RCB		Integer	Resource internal name.
Q#		Bit(1)	Queue identifier. Values are: #LEFT or #RIGHT.
FindOP		Integer	Operation identifier. Values: #FNDOP1, #FNDOP2, #FNDOP3, #FNDOP4, and #FNDOP5.
Posit	Posit	Integer	Position in queue to be sampled or in which the process was found.
PINam	PINam	Integer	Internal name of process to be found or which was found at the specified position.
Datparm	Datparm	Integer	Data found or to be entered.
	Pri	Integer	Priority of the process in the queue.
	Error	Integer	Error condition code. Values: a) 103 - invalid resource name b) 104 - unallocated resource c) 108 - invalid operation.

ENTRY POINT DESCRIPTION

This entry point provides such services as searching a queue for a process and returning position and data, sampling a specified position in a queue, and modifying data in a queue element.

ENTRY POINT IMPLEMENTATION

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
THIS ENTRY POINT TO RCB HANDLER IS USED TO DETERMINE THE
QUEUE ("#Q") POSITION ("POSIT") OF A PROCESS ("PINAM")
AND ENTER OR GET A COPY OF DATA. THE "PINAM", "POSIT", OR
BOTH MAY BE SPECIFIED TO SELECT A SPECIFIC PROCESS, ANY
PROCESS AT POSITION ("POSIT") OR A SPECIFIC PROCESS AT
A SPECIFIED POSITION.                                          */

RCB_FIND: ENTRY(RCB,Q#,FINDOP,POSIT,PINAM,DATPARM,PRI,
                ERROR);

  DCL (RCB,FINDOP,POSIT,PINAM,DATPARM,IPOS) FIXED BINARY;

          /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINIS);

  IF (VALID(RCB)) THEN DO;
  IF ((FINDOP < 1) | (FINDOP > 5)) THEN DO;
    ERROR = 108;
    GO TO RETURN6;
   END;

  IF (Q# =#LEFT) THEN START = LEFT_QUE(RCB);
   ELSE START = RIGHT_QUE(RCB);
  IF (START = NULL ) THEN DO;
    DATPARM, POSIT = 0;
    GO TO RETURN6;
   END;

  IPOS = POSIT;
  DO_ACTION_OF CASE(FINDOP);

    CASE(1): /* FIND THE PROCESS SPECIFIED BY PINAM AND
                RETURN THE POSITION AND FILE_DATA          */
       CALL LOOKUP(START,PINAM,IPOS);
       IF (IPOS = 0)  THEN DATPARM, PRI = 0;
        ELSE DO; DATPARM = START -> FILE_DATA;
                 PRI = START -> PRECEDENCE; END;
       POSIT = IPOS;
    ENDCASE;

    CASE(2): /* FIND THE PROCESS SPECIFIED BY PINAM AND
                ENTER THE DATA IN DATPARM, RETURN THE
                POSITION IN THE QUEUE                      */
       CALL LOOKUP(START,PINAM,IPOS);
       IF (IPOS ¬= 0) THEN START -> FILE_DATA = DATPARM;
       POSIT = IPOS;
    ENDCASE;
```

```
   CASE(3): /* FIND THE SPECIFIED PROC AT THE SPECIFIED
               POSITION AND RETURN FILE DATA          */
      CALL QUEPOSIT(IPOS,START);
      IF ((IPOS = 0) | (PINAM ¬= START -> FROM))THEN DO;
         DATPARM, PRI = 0;
         ERROR = 108;
       END;
       ELSE DO; DATPARM = START -> FILE_DATA;
                PRI = START -> PRECEDENCE; END;
   ENDCASE;

   CASE(4): /* FIND THE SPECIFIED PROCESS AT THE
               SPECIFIED POSITION AND ENTER THE DATA
               IN DATPARM                             */
      CALL QUEPOSIT(IPOS,START);
      IF ((IPOS =0) | (PINAM ¬= START -> FROM)) THEN
         ERROR = 108;
       ELSE
         START -> FILE_DATA = DATPARM;
   ENDCASE;

   CASE(5): /* FIND AND RETURN THE PROCESS NAME AND DATA
               AT THE SPECIFIED POSITION              */
      CALL QUEPOSIT(IPOS,START);
      IF (IPOS = 0) THEN PINAM,DATPARM,PRI = 0;
       ELSE DO;
         PINAM = START -> FROM;
         DATPARM = START -> FILE_DATA;
         PRI = START -> PRECEDENCE;
       END;
   ENDCASE;
END_OF_CASES;
END;
GO TO RETURN6;
```

ENTRY POINT SPECIFICATION

MODULE NAME: RCB_HANDLER ENTRY NAME: FIND_INAME

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
Type		Integer	Resource type identifier. Values: #PCT, #DEVICE, #FILE and #SEMFOR.
RXName		Char(8)	Resource external name.
	RIName	Integer	Resource internal name.
	Error	Integer	Error condition code. Values: a) 101 - invalid resource type b) 111 - unidentified resource external name.

ENTRY_POINT_DESCRIPTION

This entry point finds and returns a resource internal name when provided a valid external name for a resource for which an RCB has been created.

ENTRY_POINT_IMPLEMENTATION

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
THIS ENTRY POINT TO RCB HANDLER RETURNS THE INTERNAL NAME
OF A RESOURCE ("RINAME") WHEN GIVEN THE EXTERNAL NAME,
("RXNAME") AND THE RESOURCE CLASS ("TYPE") PROVIDED A
RESOURCE BY THAT NAME HAD BEEN CREATED.                        */

FIND_INAME: ENTRY(TYPE,RXNAME,RINAME,ERROR);

  DCL (BASE,TOP) FIXED BINARY;

        /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

  IF (TYPE = #SEMFOR) THEN DO; BASE = SEM_I_LIMIT;
                               TOP = SEM_LIMIT;
                          END;
    ELSE IF (TYPE = #FILE) THEN DO; BASE = FILE_L_LIMIT;
                                    TOP = FILE_U_LIMIT;
                                END;
         ELSE IF (TYPE = #DEVICE) THEN DO;
                                       BASE = DEV_L_LIMIT;
                                       TOP = DEV_U_LIMIT;
                                          END;
              ELSE IF (TYPE = #PCT) THEN DO;
                                       BASE = PCT_L_LIMIT;
                                       TOP = PCT_U_LIMIT;
                                       END;
                   ELSE DO; ERROR = 101;
                            GO TO RETURN7;  END;
  DO I = BASE TO TOP WHILE (RXNAME ¬= EXT_NAME(I)); END;
  IF (I > TOP) THEN ERROR = 111;
  ELSE RINAME = I;
  GO TO RETURN7;
```

ENTRY POINT SPECIFICATION

MODULE NAME: RCB_HANDLER ENTRY NAME: DESTROY_RCB

PARAMETERS INPUT	OUTPUT	TYPE	CONTENTS
RIName		Integer	Resource internal name.
Addressor		Integer	Internal name of the process destroying the resource.
	ProVec	Integer Vector	List of processes found on the queues of the resource being destroyed.
	Error	Integer	Error condition code. Values: a) 117 - process not authorized to destroy the resource. b) 103 - invalid resource name c) 104 - unallocated resource.

ENTRY_POINT_DESCRIPTION

This entry point is invoked to destroy a specified resource provided the addressor is the owner or the creator of the resource. Any process on the resource queue is returned to the invoker for proper disposition.

ENTRY POINT IMPLEMENTATION

```
/** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  THIS ENTRY POINT TO RCB HANDLER IS USED TO DESTROY
  A SPECIFIED RCB ("RINAME"). THE PROCESS REQUESTING
  DESTRUCTION ("ADDRESSOR") MUST BE THE CREATOR OF THE
  RCE OR AN ERROR FOR THAT PROCESS IS RAISED. ALL
  PROCESS INTERNAL NAMES QUEUED ON THIS RCB ARE
  IDENTIFIED BY SETTING A BIT FIELD IN PROVEC TO TRUE.
  THIS VECTOR IS UTILIZED BY THE INVOKING PROCESS TO
  RELEASE MESSAGES TO THE SUPERVISOR CONCERNING THE
  NON-EXISTANCE OF THIS RESOURCE AND THE PROCESSES
  USING IT.                                              */

DESTROY_RCB: ENTRY(RINAME,ADDRESSOR,PROVEC,ERROR);

DCL QNR BIT(1);
  DCL PROVEC(*) BIT(1);

          /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

  IF (VALID(RINAME)) THEN DO;
  IF ((ADDRESSOR ¬= CREATOR(RINAME)) |
      (ADDRESSOR ¬= OWNER(RINAME))) THEN DO;
                              ERROR = 117;
                              GO TO RETURN8;
                                        END;
  IF ((LEFT_QUE(RINAME)¬=NULL) | (RIGHT_QUE(RINAME)¬=NULL))
       THEN DO;
                PROVEC(0) = TRUE;
                QNR = #LEFT;
                START = LEFT_QUE(RINAME);
                DO J = 1 TO 2;
                   DO WHILE (START ¬= NULL);
                      TEMP = START -> Q_TOP;
                      IF (TEMP -> FROM ¬= ADDRESSOR) THEN
                         PROVEC(TEMP -> FROM) = TRUE;
                      CALL REMOVEQ(RINAME,QNR,TEMP,START);
                      FREE TEMP -> L_R_QUEUE;
                   END;
                   START = RIGHT_QUE(RINAME);
                   QNR = #RIGHT;
                END;
             END;
  ASSIGNED(RINAME) = FALSE;
  END;
  GO TO RETURN8;
```

ENTRY POINT SPECIFICATION

MODULE NAME: RCB_HANDLER ENTRY NAME: RCB_CHARDAT

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
RIName		Integer	Resource internal name.
PutGet		Bit(1)	Operation identifier. Values: #PUT or #GET to enter or get a copy of data, respectively.
Field		Integer	RCB field identifier. Values: a) #PCTNAM - PCT external name b) #XNAME - external name.
Cdata	Cdata	Char(8)	Transfer of character data.
	Error	Integer	Error condition code. Values: a) 103 - invalid resource name b) 104 - unallocated resource.

ENTRY POINT DESCRIPTION

Invoked to access character type data. Implemented as a generic entry point to Primitive_RCBData.

ENTRY POINT IMPLEMENTATION

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
THIS ENTRY POINT TO RCB HANDLER IS USED TO ACCESS
CHARACTER TYPE DATA. "RINAME" SPECIFICES THE RCB NUMBER;
"PUTGET" SPECIFIES THE OPERATION TO BE PERFORMED (#PUT OR
 #GET); "FIELD" SPECIFIES THE RCB FIELD (DEFINED BY THE
 #VARIABLES); AND "CDATA" CONTAINS THE DATA TO BE ENTERED
OR IS SET TO THE VALUE TO BE RETURNED.                        */

  DCL  FIELD FIXED BINARY;
  DCL  PUTGET BIT(1);

RCB_CHARDAT: ENTRY(RINAME,PUTGET,FIELD,CDATA,ERROR);

  DCL CDATA CHAR(8);

        /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

  IF (VALID(RINAME)) THEN DO;
  DO_ACTION_OF  CASE(FIELD);
    CASE(1): IF (PUTGET=#PUT) THEN EXT_NAME(RINAME)=CDATA;
             ELSE CDATA = EXT_NAME(RINAME);
       ENDCASE;
    CASE(2): IF (PUTGET=#PUT) THEN PCT_NAME(RINAME)=CDATA;
             ELSE  CDATA = PCT_NAME(RINAME);
       ENDCASE;
  END_OF_CASES;
  END;
  GO TO RETURN9;
```

ENTRY POINT SPECIFICATION

MODULE NAME: RCB_HANDLER ENTRY NAME: RCB_FIXBDAT

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
RIName		Integer	Resource internal name.
PutGet		Bit(1)	Operation identifier. Values: #PUT or #GET to enter or get a copy of data, respectively.
Field		Integer	RCB field identifier. Values: a) #CRATR - resource Creator b) #OWNER - resource Owner c) #DINAME - Device internal name d) #CNT_SZ - Counter or file length e) #OFILE - Open_File data f) #S_OR_P - Shared_Private g) #TFILE - File_Type data.
Fdata	Fdata	Integer	Transfer of integer type data.
	Error	Integer	Error condition code. Values: a) 103 - invalid resource name b) 104 - unallocated resource.

ENTRY POINT DESCRIPTION

Invoked to access integer type data. Implemented as a generic entry point to Primitive_RCBData.

ENTRY POINT IMPLEMENTATION

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
THIS ENTRY POINT TO RCB HANDLER IS USED TO ACCESS FIXED
BINARY TYPE DATA. "RINAME" SPECIFICES THE RCB NUMBER;
"PUTGET" SPECIFIES THE OPERATION TO BE PERFORMED (#PUT OR
#GET); "FIELD" SPECIFIES THE RCB FIELD (DEFINED BY THE
#VARIABLES); AND "FDATA" CONTAINS THE DATA TO BE ENTERED
OR IS SET TO THE VALUE TO BE RETURNED                          */

RCB_FIXEDAT: ENTRY(RINAME,PUTGET,FIELD,FDATA,ERROR);

  DCL FDATA FIXED BINARY;

        /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

  IF (VALID(RINAME)) THEN DO;
  DO ACTION_OF  CASE(FIELD);
    CASE(1): IF (PUTGET = #PUT)THEN CREATOR(RINAME)=FDATA;
             ELSE FDATA = CREATOR(RINAME);
        ENDCASE;
    CASE(2): IF (PUTGET=#PUT) THEN OWNER(RINAME) = FDATA;
             ELSE FDATA = OWNER(RINAME);
        ENDCASE;
    CASE(3): IF (PUTGET = #PUT) THEN
                DEVICE_INAME(RINAME) = FDATA;
             ELSE FDATA = DEVICE_INAME(RINAME);
        ENDCASE;
    CASE(4): IF (PUTGET = #PUT) THEN
                CNTR_SIZE(RINAME) = FDATA;
             ELSE FDATA = CNTR_SIZE(RINAME);
        ENDCASE;
    CASE(5): IF (PUTGET = #PUT) THEN
                OPEN_FILE(RINAME) = FDATA;
             ELSE FDATA = OPEN_FILE(RINAME);
        ENDCASE;
    CASE(6): IF (PUTGET = #PUT) THEN
                SHARED_PRIVATE(RINAME) = FDATA;
             ELSE FDATA = SHARED_PRIVATE(RINAME);
        ENDCASE;
    CASE(7): IF (PUTGET = #PUT) THEN
                FILE_TYPE(RINAME) = FDATA;
             ELSE FDATA = FILE_TYPE(RINAME);
        ENDCASE;
  END_OF_CASES;
  END;
  GO TO RETURN10;
```

ENTRY POINT SPECIFICATION

MODULE NAME: RCB_HANDLER ENTRY NAME: RCB_BITDATA

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
RIName		Integer	Resource internal name.
PutGet		Bit(1)	Operation identifier. Values: #PUT or #GET to enter or get a copy of data, respectively.
Field		Integer	RCB field identifier. Values: a) #DSTAT - Device_Status data b) #L_QUE - left_Que status c) #R_QUE - Right_Que status d) #ASGND - Assigned data.
Bdata	Bdata	Bit(1)	Transfer of bits data.
	Error	Integer	Error condition code. Values: a) 103 - invalid resource name b) 104 - unallocated resource. c) 107 - invalid put operation attempted in queue header d) 119 - invalid put operation attempted in assigned field

ENTRY_POINT_DESCRIPTION

Invoked for transfer of bit type data. Implemented as a generic entry point to Primitive_RCBData.

ENTRY_POINT_IMPLEMENTATION

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
THIS ENTRY POINT TO RCB HANDLER IS USED TO ACCESS BITS
TYPE DATA. "RINAME" SPECIFIES THE RCB NUMBER; "PUTGET"
SPECIFIES THE OPERATION TO BE PERFORMED (#PUT OR #GET);
"FIELD" SPECIFIES THE RCB FIELD (DEFINED BY THE #VARI-
ABLES); AND "BDATA" CONTAINS THE DATA TO BE ENTERED OR
IS SET TO THE VALUE TO BE RETURNED                          */

RCB_BITDATA: ENTRY(RINAME,PUTGET,FIELD,BDATA,ERROR);

  DCL BDATA BIT(1);

          /*** DISENABLE ALL INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

  IF (VALID(RINAME)) THEN DO;
  DO_ACTION_OF  CASE(FIELD);
    CASE(1): IF (PUTGET = #PUT) THEN
                              DEVICE_STATUS(RINAME) = BDATA;
             ELSE BDATA = DEVICE_STATUS(RINAME);
       ENDCASE;
    CASE(2): IF (PUTGET = #PUT) THEN ERROR = 107;
             ELSE IF (LEFT_QUE(RINAME) = NULL) THEN
                                       BDATA = TRUE;
                  ELSE BDATA = FALSE;
       ENDCASE;
    CASE(3): IF (PUTGET = #PUT) THEN ERROR = 107;
             ELSE IF (RIGHT_QUE(RINAME) = NULL) THEN
                                       BDATA = TRUE;
                  ELSE BDATA = FALSE;
       ENDCASE;
    CASE(4): IF (PUTGET= #PUT) THEN ERROR = 119;
             ELSE BDATA = ASSIGNED(RINAME);
       ENDCASE;
  END_OF_CASES;
  END;
  GO TO RETURN11;
```

LOCAL PROCEDURES

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
THE FOLLOWING LOCAL PROCEDURES ARE USED TO PERFORM
FUNCTIONS WHICH ARE COMMON TO SEVERAL ENTRY POINTS                    */


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
  VALID CHECKS THAT "RCBNR" IS WITHIN RANGE AND THAT THE
RCB IS CURRENTLY ACTIVE.                                              */

VALID: PROC(RCBNR) RETURNS(BIT(1));
  DCL RCBNR FIXED BINARY;
  IF ((RCBNR < 1) |
      ((RCBNR > SEM_LIMIT) & (RCBNR < DEV_L_LIMIT)) |
      ((RCBNR > DEV_U_LIMIT) & (RCBNR < FILE_L_LIMIT)) |
      ((RCBNR > FILE_U_LIMIT) & (RCBNR < PCT_L_LIMIT)) |
      (RCBNR > PCT_U_LIMIT)) THEN DO;
     ERROR = 103; RETURN(FALSE); END;
  IF (¬ASSIGNED(RCBNR)) THEN DO;
     ERROR = 104; RETURN(FALSE); END;
  RETURN(TRUE);
END VALID;


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
REMOVEQ REMOVES THE QUEUE ELEMENT ("ELMT") FROM THE QUEUE
("QNUM") ASSOCIATED WITH THE RCB NUMBER ("RCB"). "TOPELMT"
SPECIFIES THE TOP ELEMENT IN THE QUEUE.                               */

REMOVEQ: PROC(RCB,QNUM,ELMT,TOPELMT);
  DCL QNUM BIT(1),
      (ELMT,TOPELMT,TEMP,TEMP1) POINTER,
      RCB FIXED BINARY;
  TEMP = ELMT;
  TEMP1 = TOPELMT;
  IF ((TEMP = TOPELMT) & (TEMP -> Q_TOP = TOPELMT))
     THEN DO;
     /* REMOVE THE ONLY ELEMENT IN THE QUEUE  */
     IF (QNUM =#LEFT) THEN LEFT_QUE(RCB),TOPELMT = NULL;
     ELSE RIGHT_QUE(RCB), TOPELMT = NULL;
     RETURN;
   END;
  ELSE DO;
     IF (TEMP1 = TEMP) THEN
       /* REMOVE THE TOP ELEMENT IN THE QUEUE  */
       IF (QNUM =#LEFT) THEN
                LEFT_QUE(RCB) = TEMP -> Q_BOTTOM;
          ELSE RIGHT_QUE(RCB) = TEMP -> Q_BOTTOM;
       END;
  /* RESET THE QUEUE LINKAGE TO REMOVE THE ELEMENT  */
  TEMP1 = TEMP -> Q_BOTTOM;
  TEMP1 -> Q_TOP = TEMP -> Q_TOP;
  TEMP1 = TEMP -> Q_TOP;
  TEMP1 -> Q_BOTTOM = TEMP -> Q_BOTTOM;
  RETURN;
  END REMOVEQ;
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
INSERT ENTERS THE QUEUE ELEMENT ("PTR") IN THE RCB
("RINAME") QUEUE SPECIFIED ("L_OR_R") BY PRIORITY ("PRI").
                                                                  */

INSERT: PROC(RINAME,L_OR_R,PTR,PRI);
  DCL (RINAME,PRI) FIXED BINARY;
   DCL (PTR,START,TEMP) POINTER;
  DCL L_OR_R BIT(1);
  DCL EMPTY BIT(1) INIT('0'B);
   IF (L_OR_R =#LEFT) THEN DO;
      START = LEFT_QUE(RINAME);
      IF (START = NULL) THEN EMPTY = TRUE;
      ELSE IF (START -> PRECEDENCE >= PRI) THEN GO TO FIND;
      LEFT_QUE(RINAME) = PTR;
                              END;
  ELSE  DO;
      START = RIGHT_QUE(RINAME);
      IF (START = NULL) THEN EMPTY = TRUE;
      ELSE IF (START -> PRECEDENCE >= PRI) THEN GO TO FIND;
      RIGHT_QUE(RINAME) = PTR;
        END;
  IF (EMPTY) THEN DO;
      PTR -> Q_TOP, PTR -> Q_BOTTOM = PTR;
      RETURN;
                  END;
  TEMP = START;
  GO TO ENT;
  FIND: TEMP = START -> Q_BOTTOM;
        DO WHILE ((TEMP¬=START) & (TEMP->PRECEDENCE>=PRI));
                 TEMP = TEMP -> Q_BOTTOM;
        END;
  ENT: PTR -> Q_BOTTOM = TEMP;
       START, PTR -> Q_TOP = TEMP -> Q_TOP;
       TEMP -> Q_TOP, START -> Q_BOTTOM = PTR;
       RETURN;
END INSERT;


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
LOOKUP  SEARCHES THE QUEUE BEGINNING WITH THE ELEMENT
"START" FOR PROCESS SPECIFIED BY "NAM". IF THE PROCESS
IS FOUND "START" IS SET TO THE POINTER TO THE PROCESS
AND IPOS IS SET TO THE PROCESS' POSITION IN THE QUEUE.
IF THE PROCESS IS NOT FOUND "IPOS" IS SET TO ZERO.       */

LOOKUP: PROC(START,NAM,IPOS);
  DCL (START,TEMP) POINTER,
      (NAM,IPOS,I) FIXED BINARY;
  TEMP = START;
  I=1;
  IF (TEMP -> FROM ¬= NAM) THEN DO;
    TEMP = START -> Q_BOTTOM;
    I = 2;
    DO WHILE( (TEMP ¬= START ) & (TEMP -> FROM ¬= NAM) ) ;
      I= I+1;
      TEMP = TEMP -> Q_BOTTOM;
    END;
  END;
  IF (TEMP -> FROM ¬= NAM) THEN IPOS = 0;
   ELSE  IPOS = I;
  START = TEMP;
  RETURN;
  END LOOKUP;
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
QUEPOSIT RETURNS THE POINTER TO THE PROCESS IN THE
POSITION SPECIFIED BY "IPOS" IN THE QUEUE STARTING WITH
"START". "START" IS SET TO THE RETURN POINTER IF THE
PROCESS IS FOUND ELSE "IPOS" IS SET TO ZERO.                    */

QUEPOSIT: PROC(IPOS,START);
  DCL (IPOS,I) FIXED BINARY,
      (START,TEMP) POINTER;
  TEMP = START -> Q_BOTTOM;
  I = 1;
  DO WHILE ((I < IPOS) & (TEMP ¬= START));
    TEMP = TEMP -> Q_BOTTOM;
    I = I + 1;
   END;
  IF (I = IPOS) THEN /* DONE */ START = TEMP -> Q_TCP;
   ELSE /* NOT DONE */ IPOS = 0;
  RETURN;
END QUEPOSIT;


RETURN1: RETURN2: RETURN3: RETURN4: RETURN5: RETURN6:
RETURN7: RETURN8: RETURN9: RETURN10: RETURN11:

          /*** ENABLE THE INTERRUPTS ***/
  CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);

END PRIMITIVE_RCB_HANDLER;
```

MODULE SPECIFICATION

NAME: INTERRUPT ENABLEE TYPE: PRIMITIVE

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
Interrupt		Integer	If not zero, it identifies the interrupt to be enabled.
Save_int		Integer Array	If interrupt is zero, it identifies all interrupts to be enabled.

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
None.		

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE
Invoked by all primitives and the Interrupt Handler upon completing execution.	

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Intrupt		Bit(1) Array	External structure containing the current status of the system interrupts. A '1' indicates the interrupt is enabled and a '0' indicates that it is disabled.

MODULE DESCRIPTION

This primitive was designed to enable all the interrupts set in the array save vector if the parameter interrupt equals zero otherwise only the specified interrupt is enabled.

MODULE IMPLEMENTATION

```
%INCLUDE NAMCHGR;       /***** INTERRUPT ENABLER *****/
   /* THIS PRIMITIVE WAS DESIGNED TO ENABLE ALL THE
      INTERRUPTS SET TO TRUE IN THE ARRAY SAVINT IF
      INTERRUPT# IS ZERO (0), OR TO ENABLE THE SPECIFIC
      INTERRUPT SPECIFIED BY INTERRUPT#                    */

PRIMITIVE_INTERRUPT_ENABLER: PROC(INTERRUPT#,SAVEINT);

%INCLUDE INTACTV;

    DCL INTERRUPT# FIXED BINARY(15),
        SAVEINT (16) BIT(1);    /* DIMENSIONED NUMBINT */

    IF (INTERRUPT# = 0) THEN  /* RENABLE FROM SAVEINT */
      INTRUPT = SAVEINT;
    /* INTRUPT CONTAINS THE CURRENT STATUS OF THE SYSTEM
     INTERRUPTS. A 1 INDICATES THE INTERRUPT IS ENABLED;
     A 0 INDICATES THE INTERRUPT IS DISENABLED          */

     ELSE INTRUPT(INTERRUPT#) = '1'B;
     /* ENABLE A SINGLE INTERRUPT */

    RETURN;
END PRIMITIVE_INTERRUPT_ENABLER;
```

MODULE SPECIFICATION

NAME: INTERRUPT_DISABLER TYPE: PRIMITIVE

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
Interrupt		Integer	If not zero, it identifies the interrupt to be disabled.
	Save_int	Integer Array	If interrupt is zero, it is used to save the status of all interrupts being disabled.

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE

None.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE

Invoked by all primitives and the Interrupt Handler to prevent interrupts during execution.

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Intrupt		Bit(1) Array	External structure containing the current status of the system interrupts. A '1' indicates the interrupt is enabled and a '0' indicates that it is disabled.

MODULE DESCRIPTION

This primitive was designed to disable all the interrupts currently enabled and to return the current interrupt status in the save vector if the interrupt parameter equals zero otherwise only the interrupt specified is disabled.

MODULE IMPLEMENTATION

```
%INCLUDE NAMCHGR;       /*****  INTERRUPT DISENABLER *****/
   /* THIS PRIMITIVE WAS DESIGNED TO DISENABLE ALL THE
      INTERRUPTS CURRENTLY SET AND TO SAVE THE CURRENT
      INTERRUPT STATUS IN THE ARRAY SAVEINT, IF INTERRUPT#
      EQUALS ZERO (0) OTHERWISE DISENABLE THE SPECIFIC
      INTERRUPT INDICATED BY INTERRUPT#                    */

PRIMITIVE_INTERRUPT_DISENABLER: PROC(INTERRUPT#,SAVEINT);

%INCLUDE INTACTV;

    DCL INTERRUPT# FIXED BINARY(15),
        SAVEINT (16) BIT(1);

    IF (INTERRUPT# = 0) THEN /* DISENABLE ALL INTERRUPTS
      AND SAVE THE CURRENT CONFIGURATION IN SAVEINT  */

     DO;

    /* INTRUPT CONTAINS THE CURRENT STATUS OF THE SYSTEM
     INTERRUPTS. A 1 INDICATES THE INTERRUPT IS ENABLED;
     A 0 INDICATES AN INTERRUPT IS DISENABLED          */

       SAVEINT = INTRUPT;
       INTRUPT = '0'B;
     END;

     ELSE /* DISENABLE A SINGLE INTERRUPT */
       INTRUPT(INTERRUPT#) = '0'B;

END PRIMITIVE_INTERRUPT_DISENABLER;
```

MODULE SPECIFICATION

NAME: SAVESTATE TYPE: PRIMITIVE

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
P-Iname		Integer	Process internal name.
Processor		Integer	CPU on which the process being saved was executing.
	Error	Integer	Returns error condition code.

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
Interrupt Disabler	Interrupt Number, Save Vector	This module is invoked to disable all interrupts while Savestate is in execution.
Interrupt enabler	Interrupt Number, Save Vector	This module is invoked to reenable all interrupts which were disabled by the current module in execution.
PCBDATA	Process I-Name, Put/Get Parameter, Field Identifier, Data Parameter, Error Parameter	This module is invoked to save the contents of the CPU registers on which Process I-Name was in execution.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE
Scheduler	Save the state of a process which is being preempted.
Request	Save the state of a process whose reblocked on a request for a message or resource.
Interrupt Handler	Save the state of the current process in execution on the Operating System processor until the current interrupt is handled.

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Saveint		Bit(1) Array	Array used to save the status of the interrupts; arguement in call to Disabler and Enabler.
CPUREGS		Integer Matrix	External structure representing the CPU registers for each processor.
	Item(i,j)	Integer	Represents register 'j' on processor 'i'.
Current-Process		Integer Array	External vector used to identify the process executing on processor 'i' (Array Index).

MODULE DESCRIPTION

This primitive was designed to save the current state of the processor in the save-area (PCB Field) for the specified process.

MODULE_IMPLEMENTATION

```
%INCLUDE NAMCHGR;        /*****  SAVE STATE  *****/
   /* THIS PRIMITIVE WAS DESIGNED TO SAVE THE CURRENT STATE
      OF THE PROCESSOR ("PROCESSOR#") IN THE SAVEAREA FOR
      THE PROCESS SPECIFIED BY "PINAME". UPON COMPLETION
      THE CURRENT PROCESS FOR THE SPECIFIED PROCESSOR IS
      SET TO ZERO.                                          */

PRIMITIVE_SAVESTATE: PROC(PINAME,PROCESSOR#,ERPRM);

%INCLUDE REGSTRS;
%INCLUDE PCBDCL;
%INCLUDE OSDCL;

    ON CHECK (ERROR)
       BEGIN;
         IF (ERROR ¬= 0) THEN DO;
            ERPRM = ERROR;
            GO TO RETURN_POINT;
         END;
       END;

    DCL (PINAME,PROCESSOR#,ERPRM) FIXED BINARY(15);
    DCL ERROR FIXED BINARY(15) INIT(0);
    DCL TEMPREG (10) FIXED BINARY(31);

            /*** DISENABLE ALL INTERRUPTS ***/
    CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

            /* SAVE THE PROCESSOR STATE IN PROCESS'S PCB */
    TEMPREG = CPUREGS(PROCESSOR#,*);
    CALL PRIMITIVE_PCBDATA(PINAME,@PUT,@STATE,TEMPREG,
                           ERROR);

            /*SET CURRENT PROCESS OF THE PROCESSOR TO ZERO*/
    CURRENT_PROCESS(PROCESSOR#) = 0;

            /*** REENABLE THE INTERRUPTS ***/
    CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);
 RETURN_POINT:
 END PRIMITIVE_SAVESTATE;
```

MODULE SPECIFICATION

NAME: RESTORESTATE TYPE: PRIMITIVE

PARAMETERS

INPUT	OUTPUT	TYPE	CONTENTS
P-Iname		Integer	Process internal name.
Processor		Integer	CPU allocated to the process.
	Error	Integer	Returns error condition code.

EXTERNAL CALLS MADE TO OTHER MODULES

NAME	PARAMETERS	PURPOSE
Interrupt Disabler	Interrupt Number, Save Vector	This module is invoked to disable all interrupts while Restorestate is in execution.
Interrupt enabler	Interrupt Number, Save Vector	This module is invoked to reenable all interrupts which were disabled by the current module in execution.
PCBDATA	Process I-Name, Put/Get Parameter, Field Identifier, Data Parameter, Error Parameter	This module is invoked to set the CPU registers for the specified processor to the state vector saved in the process's PCB.

EXTERNAL CALLS MADE BY OTHER MODULES

NAME	PURPOSE
Scheduler	Restore the state of a process which was blocked, preempted or just created.
Interrupt Handler	Restore the state of the process preempted by this module after the interrupt has been handled.

DATA STRUCTURES USED

NAME	FIELD	TYPE	PURPOSE/VALUES
Saveint		Bit(1) Array	Array used to save the status of the interrupts; arguement in call to Disabler and Enabler.
CPUREGS		Integer Matrix	External structure representing the CPU registers for each processor.
	Item(i,j)	Integer	Represents register 'j' cn processor 'i'.
Current-Process		Integer Array	External vector used tc identify the process executing on processor 'i' (Array Index).

MODULE DESCRIPTION

This primitive was designed to set the registers of the specified processor to the state vector saved in the process's PCB and to set current_process for the processor to the specified prccess.

MODULE IMPLEMENTATION

```
%INCLUDE NAMCHGR;       /***** RESTORE STATE  *****/
   /* THIS PRIMITIVE WAS DESIGNED TO RESTORE THE REGISTERS
      FOR THE PROCESS SPECIFIED BY PINAME FOR THE PROCESSOR
      SPECIFIED BY PROCESSOR# AND TO SET THE CURRENT PROCESS
      FOR THE PROCESSOR TO PINAME.                          */

(CHECK (ERROR)):
PRIMITIVE_RESTORESTATE: PROC(PINAME,PROCESSOR#,ERPRM);

%INCLUDE PCBDCL;
%INCLUDE CSDCL;
%INCLUDE REGSTRS;

    ON CHECK (ERROR)
       BEGIN;
         IF (ERROR ¬= 0) THEN DO;
            ERPRM = ERROR;
            GO TO RETURN_POINT;
         END;
       END;

    DCL (PINAME,PROCESSOR#,ERPRM) FIXED BINARY(15);
    DCL ERROR FIXED BINARY(15) INIT(0);
    DCL TEMPREG(10) FIXED BINARY(31);

          /*** DISENABLE ALL INTERRUPTS ***/
    CALL PRIMITIVE_INTERRUPT_DISENABLER(ALL_INT,SAVEINTS);

        /* GET THE PROCESSOR STATE SAVED IN THE PROCESS'S
                PCB                                       */
    CALL PRIMITIVE_PCBDATA(PINAME,@GET,@STATE,TEMPREG,
                           ERROR);

        /* SET THE PROCESSOR REGISTERS AND UPDATE THE
              CURRENT PROCESS.                            */
    CPUREGS(PROCESSOR#,*) = TEMPREG;
    CURRENT_PROCESS(PROCESSOR#) = PINAME;

        /*  REENABLE THE INTERRUPTS.                      */
    CALL PRIMITIVE_INTERRUPT_ENABLER(ALL_INT,SAVEINTS);

  RETURN_POINT:
 END PRIMITIVE_RESTORESTATE;
```

APPENDIX D: MODEL INITIALIZATION AND TEST PROGRAMS

```
%INCLUDE NAMCHGR;       /***** HARDWARE DRIVER *****/
(CHECK(ERROR)):

HDRIVER: PROC OPTIONS(MAIN);

   /* THE HARDWARE DRIVER IS USED TO SIMULATE THE PROCESSORS
      EXECUTING THE SCHEDULED PROCESSES, SETTING INTERRUPTS
      AND INVOKING THE INTERRUPT HANDLER AFTER AN
      APPROPRIATE TIME LAPSE AND PERFORMING OTHER FUNCTIONS
      OF THE HARDWARE.                                        */

    ON CHECK(ERROR)
      BEGIN;
        IF (ERROR ¬= 0) THEN DO;
          PUT FILE(SYSPRINT) LIST('HARDWARE DRIVER: ERROR=',
              ERROR) SKIP;
        (NOCHECK(ERROR)):
          BEGIN;  ERROR = 0;  END;
        END;
      END;  /*  ERROR ON CONDITION  */

%INCLUDE PCBDCL;
%INCLUDE RCBDCL;
%INCLUDE CSDCL;
%INCLUDE INTACTV;
%INCLUDE INTSET;
%INCLUDE GENDEC;
%INCLUDE CASESTM;
%INCLUDE REGSIMS;

    DCL ITIME (16) FIXED BINARY(31) STATIC,
        TIMER FIXED BINARY(31),
        CPROC(4)  FIXED BINARY(15)  STATIC INIT((4)0),
        CINDX(4)  FIXED BINARY(15)  STATIC INIT((4)0),
        CREG(10)  FIXED BINARY(31)  STATIC INIT((10)0),
        ERROR FIXED BINARY INIT(0),
        IC FIXED BINARY EXTERNAL;

    CURRENT_PROCESS = 0;
    CALL INITIALIZATION;

    CALL TESTER;

    DO FOREVER;

    PUT FILE(SYSPRINT) LIST('HARDWARE DRIVER - TOP') SKIP;

        /**  EXECUTE ONE "STEP" ON EACH PROCESSOR  **/
    DO I = 1 TO NUMBCPU;
      IF (CURRENT_PROCESS(I) ¬= CPROC(I)) THEN DO;
        /***  SAVE THE IDENTIFICATION OF THE PROCESS BEING
                  RUN                                  ****/
        CPROC(I)  = CURRENT_PROCESS(I);
        CINDX(I)  = CPUREGS(I,1);
       END;
         /***  SET THE INSTRUCTION COUNTER FOR SIMULATION */
      IC = CPUREGS(I,2);
      IF (CURRENT_PROCESS(I) ¬= 0) THEN DO;

        PUT FILE(SYSPRINT) LIST('INDEX=',CINDX(I));
        PUT FILE(SYSPRINT)  LIST( ' ') SKIP;
```

```
        /**  EXECUTE THE PROCESS  **/
      DO ACTION_OF CASE(CINDX(I));
        CASE(1):  CALL SUPERVISOR;   ENDCASE;
        CASE(2):
            PUT FILE(SYSPRINT) LIST('TERMINATOR - DUMMY');
          /* CALL TERMINATOR;     */
        ENDCASE;
        CASE(3):
            PUT FILE(SYSPRINT) LIST('INITIATOR - DUMMY');
          /*  CALL INITIATOR;    */
        ENDCASE;
        CASE(4):  CALL INPUT_CONTROLLER;        ENDCASE;
        CASE(5):  CALL OUTPUT_CONTROLLER;       ENDCASE;
        CASE(6):  CALL FILE_MANAGER;            ENDCASE;
        CASE(7):  CALL OPERATOR_SYSTEM_COMMUNICATOR;
         ENDCASE;
        CASE(8):  CALL FILE_SPACE_MANAGER;      ENDCASE;
      END_OF_CASES;
      /*  CHECK TO SEE IF A PROCESS HAS BEEN PREEMPTED.
          IF SO, SAVE THE CORRECT INSTRUCTION COUNTER FOR
          SIMULATION PURPOSES.                          */
      IF (CPROC(I) ¬= CURRENT_PROCESS(I)) THEN DO;
        CREG(1) = CINDX(I);
        CREG(2) = IC;
        CALL PRIMITIVE_PCBDATA(CPROC(I),@PUT,@STATE,CREG,
                               ERROR);
       END;
       ELSE
            /***  SAVE THE SIMULATION INSTRUCTION COUNTER
                                                   *****/
        CPUREGS(I,2) = IC;
    END;
    /* CHECK INTERRUPTS */
    TIMER = TIMER + 5;  /* INCREMENT THE CLOCK */
    DO J = 1 TO NUMBINT;
            /*** IF THE TIME FOR AN INTERRUPT TO OCCUR
                 HAS ELAPSED THEN SET THE INTERRUPT   **/
      IF((INTRUPT(J)) & (ITIME(J) >= TIMER)) THEN DO;
        INTRSET(I) = TRUE;
        PUT FILE(SYSPRINT) LIST('INTERRUPT',I,'SET');
    /*
        CALL INTERRUPT_HANDLER;
  */
       END;
    END;
  END;
  END;  /* OF DO FOREVER */

STARTIO: ENTRY(INTERRUPT#);
      /****  THIS PROCEDURE SIMULATES STARTING AN I/O
            DEVICE - THE BASE TIME IS SAVED TO CHECK
            ELAPSED TIME TO SET THE INTERRUPT       ****/

  DCL INTERRUPT# FIXED BINARY(15),
      DEVTIME(16) FIXED BINARY(15) STATIC INIT((16)50);
      /* DEVTIME IS THE AVERAGE LENGTH OF TIME REQUIRED
         FOR A DEVICE TO PERFORM ITS FUNCTION          */
  ITIME(INTERRUPT#) = TIMER + DEVTIME(INTERRUPT#);
  RETURN;
```

```
  TESTER: PROC;
     /****  THIS PROCEDURE IS USED TO SET THE SYSTEM STATE
             FOR TEST PURPOSES.                              ***/
          /*** TRACE IS USED TO ENABLE PRINT STATMENTS IN
               SELECTED PROCEDURES.                          *****/
     DCL TRACE BIT(1) EXTERNAL;
     TRACE = TRUE;

%INCLUDE REQRELD;
     DCL TEST_MSGS (3) CHAR(8) INIT ('IPDEV1  ',
          'OPDEV1  ','FOPDEV1 ');

          /***  RELEASE MESSAGES TO CREATE DEVICES FOR THE
                 INPUT CONTROLLER, OUTPUT CONTROLLER AND THE
                 FILE MANAGER.                             *****/
     ALLOCATE MESSAGE_BUFFER SET(MESSAGE);
     MSG_SEMAPHORE,FIELD2,FIELD3,FIELD4,FIELD5,FIELD6  = 0;
     CHAR_FIELD1,CHAR_FIELD2,CHAR_FIELD3,CHAR_FIELD4 = '';
     ANSWER_REQUEST = FALSE;
     FIELD1 = $$JCL;
     K = 4;
     DO I =1 TO 3;
       ALLOCATE INPUT_BUFFER SET(IBUFPTR);
       BUFFER_LOCATION = IBUFPTR;
       IF (I<3) THEN INPUT_BUFFER = '@@ADD '||TEST_MSGS(I);
      ELSE INPUT_BUFFER = '@@ADD '||TEST_MSGS(3)||
                          'BIG_BIRD';
       CALL PRIMITIVE_RELEASE(K,$INPUT,MESSAGE,ERROR);
     END;

  END TESTER;

  END HDRIVER;
```

```
%INCLUDE NAMCHGR;                    /***** INITIALIZATION  *****/
(CHECK(ERROR)):
INITIALIZATION: PROC OPTIONS(MAIN);
    DCL TRACE BIT(1) EXTERNAL;
    IF TRACE THEN
        PUT FILE(SYSPRINT)LIST('ENTERING INITIALIZE');
  ON CHECK(ERROR)
        BEGIN;
        IF (ERROR ¬= 0) THEN DO;
        PUT FILE(SYSPRINT) LIST('IN INITIALIZE, ERROR = ',
              ERROR) SKIP ;
        (NOCHECK(ERROR)): BEGIN; ERROR = 0; END;
        END;
        END;

    DCL ERROR FIXED BINARY INIT(0);
%INCLUDE REQRELB;
%INCLUDE RCBDCL;
%INCLUDE PCBDCL;
%INCLUDE GENDEC;
%INCLUDE OSDCL;
%INCLUDE REGSTRS;
%INCLUDE INTACTV;
%INCLUDE INTSET;
%INCLUDE CASESTM;

          /*** INITIALIZE INTERFACE VARIABLES ***/

    ANYPROC = 0;

    ##ACCES = '01'B;
    ##ACQRD = '10'B;
    #ASSGND = 4;
    ##AVAIL = 0;
    ##CLOSE = '1'B;
    #CRATR = 1;
    #CNT_SZ = 4;
    #DEVICE = 3;
    #DINAME = 3;
    #DSTAT = 2;
    #FILE = 2;
    #FNDOP1 = 1;
    #FNDOP2 = 2;
    #FNDOP3 = 3;
    #FNDOP4 = 4;
    #FNDOP5 = 5;
    #GET = '1'B;
    ##GO = '1'B;
    ##HOLD = '0'B;
    #LEFT = '0'B;
    #L_QUE = 4;
    ##NOACC = '00'B;
    ##NOAVL = 3;
    #OFILE = 5;
    ##OPEN = '0'B;
    #OS = '0'B;
    #OWNER = 2;
    #PCT = 4;
    #PCTNAM = 2;
    ##PERMF = 1;
    ##PRIV = 1;
    #PUT = '0'B;
    ##READ = 1;
    ##READA = 6;
    ##READR = 1;
    ##READS = 4;
    #RIGHT = '1'B;
    #R_QUE = 3;
    ##SACRF = '11'B;
    ##SACR = 3;
    #SEMECR = 1;
    ##SHRD = 0;
```

```
#S_OR_P = 6;
##TEMPF = 0;
#TFILE = 7;
##TORPF = 2;
#USER = '1'B;
##WRITE = 2;
##WRITA = 7;
##WRITH = 3;
##WRITR = 2;
##WRITS = 5;
#XNAME = 1;

$$ADD = 0;
$$CLOSF = 4;
$$DELET = 1;
$$DIRAD = 18;
$$DIRDL = 19;
$$DIRRD = 17;
$$DONE = 1;
$$DSTYF = 5;
$$EOF = 15;
$$EXINT = 5;
$$FAIL = 2;
$$FILOP = 12;
$$INIPC= 7;
$$JCL = 16;
$$OPENF = 3;
$$OPCOM = 6;
$$OPRTR = 20;
$$PASS = 14;
$$PERMF = 3;
$$READ = 1;
$$SPACE = 2;
$$SPCMN = 0;
$$STOP = 10;
$$TEMPF = 4;
$$TERM = 11;
$$WRITE = 2;

@BORMNR = 9;
@BRMVEC = 1;
@CHILD = 4;
@CYCLE = 12;
@FSTAT = 10;
@FWCNTR = 11;
@GET = '1'B;
@LFTSIB = 5;
@MSGPTR = 1;
@NR_PGS = 8;
@PARENT = 3;
@PE_NR = 7;
@PRIRTY = 1;
@PUT = '0'B;
@QUANTM = 2;
@RESVEC = 1;
@RGTSIB = 6;
@STATE = 1;
@STATUS = 1;
@SYSPRO = 1;
@XNAME = 2;

@@BLKDR = 'BLOCKEDR';
@@BLKDT = 'BLOCKEDT';
@@REDYA = 'READYA';
@@RUN = 'RUNNING';
@@SUSPD = 'SUSPEND';
```

```
        /*** INITIALIZE LIMIT VARIABLES ***/

PCB_LIM = 50;

ALL_INT = 0;
NUMBINT = 16;
NUMBCPU = 4;
DEV_L_LIMIT = 101;
DEV_U_LIMIT = 110;
FILE_L_LIMIT = 51;
FILE_U_LIMIT =100 ;
PCT_L_LIMIT = 111;
PCT_U_LIMIT = 120;
SEM_L_LIMIT = 1;
SEM_LIMIT= 50;

CPUREGS = 0;
INTRUPT = '0'B;
INTRSET = '0'B;

DCL STATE(10) FIXED BINARY(31);
DCL  SYS_PROC BIT(1),EXT_NAM CHAR(8);
DCL (FATHER,BROTHER,PRI,CYCLE,INAME,LASTPROC)
     FIXED BINARY;
DCL SYSPROCESS (8) FIXED BINARY;
DCL ACCESS (120) BIT(2) INIT((120)(1)'01'B);
DCL OWNER FIXED BINARY, DUMMYFB FIXED BINARY INIT(0),
     DUMMYB1 BIT(1);
DCL DUMMYCHAR CHAR(8) INIT((8)'');
DCL NULLPTR POINTER;
DCL MSG_INIT POINTER;

    /*  INITIALIZE A MESSAGE BUFFER  */
ALLOCATE MESSAGE_BUFFER SET(MESSAGE);
MSG_INIT = MESSAGE;
FIELD1,FIELD2,FIELD3,FIELD4,FIELD5,FIELD6 = 0;
CHAR_FIELD1,CHAR_FIELD2,CHAR_FIELD3,CHAR_FIELD4 = '';
RELEASOR = 0;
MSG_SEMAPHORE = 0;
ANSWER_REQUEST = FALSE;
BUFFER_LOCATION = NULL;

        /*** CREATE PCB FOR ERROR HANDLER ***/
CALL PCBINIT;

        /***SET INITIAL SYSTEM STATE ***/
SYS_PROCESSOR ,PROCESSOR = 1;
CURRENT_PROCESS(SYS_PROCESSOR),SYSPROCESS(1) = 1;

        /*** ADD CODE FOR PAGE TABLE INITIALIZATION ***/

        /*** SET STATE FOR THE ERROR HANDLER ***/
INAME = 1;
CALL PRIMITIVE_PCBDATA(INAME,@PUT,@PE_NR,SYS_PROCESSOR,
                       ERROR);
STATE = 0;
STATE(1) = 1;
STATE(2) = 1;
CALL PRIMITIVE_PCBDATA(INAME,@PUT,@STATE,STATE,ERROR);

   /** INITIALIZE MESSAGE BUFFER FOR ERROR HANDLER **/
ALLOCATE MESSAGE_BUFFER SET(MESSAGE);
MESSAGE -> MESSAGE_BUFFER = MSG_INIT -> MESSAGE_BUFFER;
CALL PRIMITIVE_PCBDATA(INAME,@PUT,@MSGPTR,MESSAGE,
                       ERROR);

   /** INITIALIZE RESOURCE VECTOR FOR ERROR HANDLER **/
CALL PRIMITIVE_PCBDATA(INAME,@PUT,@RESVEC,
                SEM_L_LIMIT,PCT_U_LIMIT,ACCESS,ERROR);
ACCESS = '00'B;
```

```
/* INITIALIZE A PCB FOR THE TERMINATOR */
SYS_PROC = TRUE;
FATHER = 1;
BROTHER = 0;
PRI = 49;
CYCLE = 0;
EXT_NAM = 'TERMINTR';
CALL GETPCB(FATHER,BROTHER,EXT_NAM,PRI,SYS_PROC,STATE,
            INAME,CYCLE,ERROR);
 STATE(2) = 2;
CALL PRIMITIVE_PCBDATA(INAME,@PUT,@STATUS,@@REDYA,
                       ERROR);
CALL PRIMITIVE_PCBDATA(INAME,@PUT,@STATE,STATE,ERROR);
CALL PRIMITIVE_PCBDATA(INAME,@PUT,@PE_NR,SYS_PROCESSOR,
                       ERROR);
ALLOCATE MESSAGE_BUFFER SET(MESSAGE);
MESSAGE -> MESSAGE_BUFFER = MSG_INIT -> MESSAGE_BUFFER;
CALL PRIMITIVE_PCBDATA(INAME,@PUT,@MSGPTR,MESSAGE,
                       ERROR);
FATHER,SYSPROCESS(2) = INAME;

        /*** INITIALIZE PCB'S FOR SYSTEM PROCESSES ***/
DO I = 1 TO 6;
  DO ACTION_OF CASE(I);
   CASE(1): /* INITIATOR  */
     EXT_NAM = 'INITATOR';
     PRI = 48;
   ENDCASE;
   CASE(2): /*  INPUT CONTROLLER */
     EXT_NAM = 'INCONTRL';
     PRI = 48;
   ENDCASE;
   CASE(3): /* OUTPUT CONTROLLER */
     EXT_NAM = 'OUTCNTRL';
     PRI = 48;
   ENDCASE;
   CASE(4): /* FILE MANAGER */
     EXT_NAM = 'FILEMNGR';
     PRI = 45;
   ENDCASE;
   CASE(5): /* OPERATOR SYSTEM COMMUNICATOR */
     EXT_NAM = 'OP_COMM';
     PRI = 49;
   ENDCASE;
   CASE(6): /* FILE SPACE MANAGER */
     EXT_NAM = 'SPACEMAN';
     PRI = 45;
     BROTHER = 0;
     FATHER = SYSPROCESS(6);
   ENDCASE;
  END OF CASES;
  CALL GETPCB(FATHER,BROTHER,EXT_NAM ,PRI,SYS_PROC,STATE
              ,INAME,CYCLE,ERROR);
  IF TRACE THEN PUT FILE(SYSPRINT) LIST(
    'CASE INDEX =',I,'INAME =',INAME) SKIP;
  BROTHER = INAME;
  CALL PRIMITIVE_PCBDATA(INAME,@PUT,@PE_NR,SYS_PROCESSOR
                         ,ERROR);
  ALLOCATE MESSAGE_BUFFER SET(MESSAGE);
  MESSAGE->MESSAGE_BUFFER = MSG_INIT->MESSAGE_BUFFER;
  CALL PRIMITIVE_PCBDATA(INAME,@PUT,@MSGPTR,MESSAGE,
                         ERROR);
  STATE(2) = I + 2;
  CALL PRIMITIVE_PCBDATA(INAME,@PUT,@STATE,STATE,ERROR);
  CALL PRIMITIVE_PCBDATA (INAME,@PUT,@STATUS,@@REDYA,
                          ERROR);
  SYSPROCESS(I+2) = INAME;
  IF ((I>1) & (I < 6)) THEN
    CALL PRIMITIVE_PCBDATA(LASTPROC,@PUT,@LFTSIB,INAME,
                           ERROR);
  LASTPROC = INAME;
END;
```

```
/* CONNECT LINEAGE OF SYSTEM PROCESSES */
FATHER = SYSPROCESS(2);
LASTPROC = SYSPROCESS(7);
CALL PRIMITIVE_PCBDATA(FATHER,@PUT,@CHILD,LASTPROC,
                       ERROR);
/* CONNECT FILE SPACE MANAGER TO FILE MANAGER */
FATHER = SYSPROCESS(6);
LASTPROC = SYSPROCESS(8);
CALL PRIMITIVE_PCBDATA(FATHER,@PUT,@CHILD,LASTPROC,
                       ERROR);

/* SET UP SEMAPHORES  */
DO I = 1 TO 19;
  DO_ACTION_OF CASE(I);
    CASE(1): EXT_NAM  = 'ENDJOB';
             OWNER = SYSPROCESS(2); /* TERMINATOR */
     ENDCASE;
    CASE(2): EXT_NAM = 'ERROR';
             OWNER = SYSPROCESS(1); /* ERROR HANDLER */
     ENDCASE;
    CASE(3): EXT_NAM = 'FILEOP';
             OWNER = SYSPROCESS(6); /* FILE MANAGER */
     ENDCASE;
    CASE(4): EXT_NAM = 'INAME';
             OWNER = SYSPROCESS(3); /* INITIATOR */
     ENDCASE;
    CASE(5): EXT_NAM = 'INPUT';
             OWNER = SYSPROCESS(4);/* INPUT CONTROLLER*/
     ENDCASE;
    CASE(6): EXT_NAM = 'JOBQSP';
             OWNER = SYSPROCESS(4);/* INPUT CONTROLLER*/
     ENDCASE;
    CASE(7): EXT_NAM = 'NEWJOB';
             OWNER = SYSPROCESS(3); /* INITIATOR */
      ENDCASE;
    CASE(8): EXT_NAM = 'OPR_IO';
             OWNER = SYSPROCESS(7);/* CP_COMM*/
     ENDCASE;
    CASE(9): EXT_NAM = 'OUTPUT';
             OWNER = SYSPROCESS(5);/*OUTPUT CONTROLLER*/
     ENDCASE;
    CASE(10): EXT_NAM = 'REDYAQUE';
              OWNER = SYSPROCESS(1);
     ENDCASE;
    CASE(11): EXT_NAM = 'IBUFFER';
              OWNER = SYSPROCESS(1);
     ENDCASE;
    CASE(12): EXT_NAM = 'INTDEV';
              OWNER = SYSPROCESS(1);
     ENDCASE;
    CASE(13): EXT_NAM = 'OBUFFER';
              OWNER = SYSPROCESS(1);
     ENDCASE;
    CASE(14): EXT_NAM = 'PUTOUT';
              OWNER = SYSPROCESS(5);
     ENDCASE;
    CASE(15): EXT_NAM = 'WAIT';
             OWNER = SYSPROCESS(1);
     ENDCASE;
    CASE(16): EXT_NAM = 'INTERRPT';
              OWNER = SYSPROCESS(1);
     ENDCASE;
    CASE(17): EXT_NAM = 'INTDEV';
              OWNER = SYSPROCESS(1);
     ENDCASE;
    CASE(18): EXT_NAM = 'SPACE';
              OWNER = SYSPROCESS(8);
     ENDCASE;
    CASE(19): EXT_NAM = 'PRINTQ';
              OWNER = SYSPROCESS(2);
     ENDCASE;
  END_OF_CASES;
```

```
      CALL CREATE_RCB(#SEMFOR,EXT_NAM,OWNER,DUMMYFB,DUMMYFB,
                      DUMMYCHAR,DUMMYFB,DUMMYFB,DUMMYFB,
                      INAME,ERROR);
      IF TRACE THEN PUT FILE(SYSPRINT) LIST(
        'CASE INDEX =',I,'INAME =',INAME) SKIP;
      ACCESS(INAME) = ##ACCES;
      DO ACTION_OF CASE(I);
        CASE(1): $ENDJOB= INAME;          ENDCASE;
        CASE(2): $ERROR = INAME; ENDCASE;
        CASE(3): $FILEOP = INAME; ENDCASE;
        CASE(4): $INAME = INAME;
                 NULLPTR = NULL;
                 DO J = 1 TO 30;  /* 30 USER JOBS MAXIMUM */
                   CALL PRIMITIVE_RELEASE(ANYPROC,$INAME,
                                NULLPTR,ERROR);
                 END;
         ENDCASE;
        CASE(5): $INPUT = INAME;  ENDCASE;
        CASE(6): $JOBQSP = INAME; ENDCASE;
        CASE(7): $NEWJOB = INAME; ENDCASE;
        CASE(8): $OPR_IO = INAME; ENDCASE;
        CASE(9): $OUTPUT = INAME; ENDCASE;
        CASE(10): #REDYA = INAME;  ENDCASE;
        CASE(11): $IBUFF= INAME;  ENDCASE;
        CASE(12): $INTDEV = INAME;  ENDCASE;
        CASE(13): $OBUFF= INAME;  ENDCASE;
        CASE(14): $PUTOUT = INAME;  ENDCASE;
        CASE(15): $WAIT = INAME;  ENDCASE;
        CASE(16): $INTRPT = INAME;  ENDCASE;
        CASE(17): $INTDEV = INAME;  ENDCASE;
        CASE(18): $SPACE = INAME; ENDCASE;
        CASE(19): $PRINTQ = INAME; ENDCASE;
      END_OF_CASES;
    END;
    MESSAGE = MSG_INIT;
    DO I = 1 TO 8;
        INAME  = SYSPROCESS(I);
        IF (I > 1) THEN
        CALL PRIMITIVE_PCBDATA(INAME,@PUT,@RESVEC,
                    SEM_L_LIMIT,PCT_U_LIMIT,ACCESS,ERROR);
        CALL PRIMITIVE_PCBDATA(I,@GET,@PRIRTY,PRI,ERROR);
        CALL RCBPUTQ(#REDYA,#OS,I,DUMMYFB,PRI,NULLPTR,
                     DUMMYFB,ERROR);
        DO J = 1 TO 2;
          ALLOCATE OUTPUT_BUFFER SET (OBUFPTR);
          BUFFER_LOCATION = OBUFPTR;
          CALL PRIMITIVE_RELEASE(INAME,$OBUFF,MESSAGE,
                                 ERROR);
          ALLOCATE INPUT_BUFFER SET(IBUFPTR);
          BUFFER_LOCATION = IBUFPTR;
          CALL PRIMITIVE_RELEASE(INAME,$IBUFF,MESSAGE,
                                 ERROR);
        END;
    END;
     FREE MESSAGE -> MESSAGE_BUFFER;
     CURRENT_PROCESS = 0;
     CALL PRIMITIVE_SCHEDULER;
END INITIALIZATION;
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    PREPROCESSOR SIMULATION MACRO, USED TO INCLUDE THE
    INDICES AND PRINT OUTPUT MESSAGES FOR SIMULATION.

    "ENABLED" BY %INCLUDE SIMULTR;                                    */

%DCL SIM_START ENTRY(CHAR) RETURNS(CHAR);
%DCL SIM_INTERRUPT_PT ENTRY RETURNS(CHAR);
%DCL @SIMPT FIXED;
%DCL SIM_END CHAR;

%@SIMPT = 1;
%SIM_END = 'DCL SMLPT(@SIMPT) LABEL; IC = 1;
            SIMEND: RETURN;';


%SIM_START: PROC(@SIMNAM) RETURNS (CHAR);

  DCL (@SIMNAM,@SIMRET) CHAR;
   @SIMRET = 'DCL IC FIXED BINARY EXTERNAL;
              PUT FILE(SYSPRINT) LIST(''ENTERING'','' '
              ||@SIMNAM ||' '','IC='',IC); GO TO SMLPT(IC);
              SMLPT(1): ;' ;
   RETURN(@SIMRET);
%END;


%SIM_INTERRUPT_PT: PROC RETURNS(CHAR);

      @SIMPT = @SIMPT +1;
      RETURN('IC = @SIMPT; GO TO SIMEND; SMLPT(@SIMPT): ;');
%END;
```

LIST OF REFERENCES

1. Syms, G. H., All Application Digital Computer: Course Notes, p. 1.1-1.6, Naval Postgraduate School, March 1973.

2. Nissen, S. M. and Wallach, S. J., "The All Applications Digital Computer," ACM-IEEE Symposium of High-Level-Language Computer Architecture, p. 43-51, November 1973.

3. Shaw, A. C., The Logical Design of Operating Systems (draft), p. 1-5, 4-25, University of Washington Press, September 1971.

4. Benson, J. P., "Structured Programming Techniques," 1973 IEEE Symposium on Computer Software Reliability, p. 143, IEEE Society, 1973.

5. Parnas, D. L., "A Technique for Software Module Specification with Examples," Communications of the ACM, v. 15, n. 5, p. 330-336, May 1972.

6. Gouthier, R. and Pont, S., Designing systems Programs, p 10.23, Prentice Hall, 1970.

7. Brinch Hansen, P., "Structured Multiprogramming," Communications of the ACM, v. 15, n. 7, p. 574-578, July 1972.

8. Brinch Hansen, P., "The Nucleus of a Multiprogramming System," Communications of the ACM, v. 13, n. 4, p. 238-241, April, 1970.

9. Parnas, D. L., "More on Simulation Languages and Design Methodology for Computer Systems," AFIPS Conference Proceedings, v. 34, p. 739-743, 1969.

10. Parnas, D. L., "On the Criteria to be Used in Decomposing Systems into Modules," Communications of the ACM, v. 15, n. 12, p. 1053-1058, December 1972.

11. NAVAIRSYSCOM, AADC Development Program Progress Report NC-11, by R. S. Entner, June 1973.

12. Wecker, S., "A Design for a Multiple Processor Operating System," IEEE COMPCON 73, p. 143-146, March 1973.

13. A/S REGNECENTRALEN, Report Nr. 55-D140, RC 4000 Software Multiprogramming System, 2nd ed., February 1971.

14. Knuth, D. E., Fundamental Algorithms: The Art of Computer Programming, p. 1-9, Addison-Wesley Publishing Company, 1969.

15. Watson, R. W., Timesharing System Design Concepts, p. 215-216 McGraw Hill, 1970.

16. Howard, J. H. Jr., "Mixed Solutions for the Deadlock Problem," Communications of the ACM, v. 16, n. 7, p. 427-430, July 1973.

17. Frasley, D. J., "A Practical Approach to Managing Resources and Avoiding Deadlocks," Communications of the ACM, v. 16, n. 5, p. 323-329, May 1973.

18. Sayers, A. P., Operating Systems Survey, p. 104, Auerbach Publishers 1972.

19. NAVAIRSYSCOM, AADC Development Progress Report NO. 7, by R. S. Entner, p. 15, 32-33, February 1971.

20. NAVAIRSYSCOM, AADC Development Progress Report NO. 9, by R. S. Entner, November 1971.

BIBLIOGRAPHY

1. Deerfield, A., Final Report for AADC RAMM-I/O Functional Block Diagram Design Study, Raytheon Company, 1973.

2. Dijkstra, E. W., "Programming Considered as a Human Activity," Proc. of the IFIPS Congress 1965, Spartan Books, p. 213-217, 1965.

3. Dijkstra, E. W., "The Structure of 'THE' Multiprogramming System," Communications of the ACM, v. 11, n. 5, p. 341-356, May 1968.

4. IBM Corporation, IBM System/360 Operating System PL/I Language Specifications, 5th ed., 1970.

5. IBM Corporation, IBM System/360 Operating System PL/I (F) Language Reference Manual, 4th ed., June 1970.

6. Miller, J. S., and others, Preliminary Report: A Description of a Programming Language for the AADC, Intermetrics Inc., April 1973.

7. NAVAIRSYSCOM, AADC Development Program Progress Report NO. 1, by R. S. Entner, 12 November 1968.

8. NAVAIRSYSCOM, AADC Development Program Progress Report NC 2, by R. S. Entner, 14 February 1969.

9. Smith, W. R., Simulation of AADC Simplex and Multiprocessor Operation, Naval Research Laboratory, February 1972.

10. Syms, G. H., Notes on Modular Operating System Design: Specification and Simulation of Basic Modules, Naval Postgraduate School, February 1974.

11. Zurcher, F. W. and Randell, B., "Iterative Multi-Level Modelling: A Methodology for Computer Systems Design," IFIPS, v. 12, p. 867-871, 1969.

INITIAL DISTRIBUTION LIST

	No. Copies
Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
Chairman, Computer Science Group, Code 72 Naval Postgraduate School Monterey, California 93940	1
Professor G. L. Barksdale, Code 72Ba Naval Postgraduate School Monterey, California 93940	1
ENS G. M. Raetz, Code 72Rr Naval Postgraduate School Monterey, California 93940	1
Dr. G. H. Syms c/o Control Data Canada, Limited 1855 Minnesota Court Mississauga, Ontario, Canada	1
Naval Electronics Laboratory Center, Code 5200 271 Catalina Boulevard San Diego, California 92152	1
LT Bernard Janov, USN 14 West Frack Street Frackville, Pennsylvania 17931	1
LT Vernon M. Johns, USN 3056 San Juan Avenue Santa Clara, California 95051	1