AD-A115 614 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
DESIGN AND DEVELOPMENT OF A MULTIPROGRAMMING OPERATING SYSTEM F--ETC(U)
DEC 81 M S ROSS
UNCLASSIFIED AFIT/GCS/EE/81D-14 NL

AD A115614

DTIC
ELECTE
JUN 1 1982
S
D

AFIT/GCS/EE/81D-14

Design and Development of a Multiprogramming Operating System for Sixteen Bit Microprocessors

THESIS

AFIT/GCS/EE/81D-14 Mitchell S. Ross
Captain USA

AFIT/GCS/EE/81D-14

Design and Development of a Multiprogramming Operating System for Sixteen Bit Microprocessors

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements of the Degree of
Master of Science

by
Mitchell S. Ross, B. S.
Captain USA
Graduate Computer Systems
December 1981

Preface

This thesis presents a design of an operating system for the Digital Engineering Laboratory. My desire is to provide a starting point for follow-on efforts and implementation. The design is based on 16-bit microprocessors and the development uses structured analysis as a software engineering technique. I have intentionly been broad in my approach so that future studies have the flexibility to adapt this effort to their educational objectives.

I would like to express my appreciation to Dr. Gary B. Lamont, who as my research advisor gave me valuable guidance. I would also like to thank my thesis readers, Captain Roie Black and Captain Richard Conn, for their advice and assistance in improving the clarity of this thesis. In addition, I am grateful to the faculty members who provide instruction at the Air Force Institute of Technology and contributed to my education and understanding of Computer Science.

Finally, I wish to thank my wife, Cheryl, for her tolerance and encouragement during my graduate studies.

Mitchell S. Ross

Contents

List of Figures

List of Tables

Abstract

A timesharing operating system for the Air Force Institute of Technology Digital Engineering Laboratory was designed and developed with emphasis on the human interface. The functional requirements were developed by a thorough literature search on the user perceptions of computer operating systems and the justification for the success of popular systems such as UNIX, TENEX, and UCSD Pascal. Structured Analysis was used to produce a structured specification for the hierarchy of the operating system. The structured specification includes an operating system shell which allows a flexible user command structure, a hierarchical file structure, device independent input/output management, a scheduler which supports swapping, a general memory management scheme, and a system nucleus consisting of process dispatching, interrupt handling and interprocess communications. Weinberg's methodology, which is based on Yourdon and Constantine's Transform Analysis and Transaction Analysis Techniques, was used to develop the software design which consists of a set of module structure charts. The module structure charts are supported by data flow diagrams and a data dictionary.

Because of the depth needed to complete such a project, this first effort is intended to provide a basis for further expansion and development. Hence, the design is a broad overall approach aimed at 16-bit microprocessors and not detailed sufficiently for full implementation.

I. Scope of Project

Introduction

The purpose of this investigation is to develop a multi-user, multiprogramming operating system for the Digital Engineering Laboratory (DEL) at the Air Force Institute of Technology's School of Engineering. This operating system can be based on the architecture of Intel Corporation's 8086 Microprocessor. Introduced in 1978, the 8086 was one of the first 16-bit high performance microprocessors.

In some respects, the 8-bit microprocessors have been strained to perform tasks easily handled by more advanced architectures. With the advent of a new class of 16-bit microprocessors, interest is increasing in applying these machines to more complex computing problems (Ref. 32: 62). Software for these 16-bit microprocessors has just recently drawn widespread interest (Ref. 49). If contemporary software is to keep pace with more sophisticated applications such as advanced graphics, access to huge information banks, and addressing large main memories, then 16-bit operation is the solution. The operating systems for these new architectures take on a new importance.

The operating system is the view of the computer system from the user's stand-point. The user "sees" and "talks" to the computer through the operating system. Its function is to transform a hardware environment, with a low level of

execution, to an abstract machine, with a high level of execution, to interface in terms understandable to the user (Ref. 5: 1046).

Historical Perspective

The history of operating system developments is difficult to present since many concepts were introduced long before they were generally accepted and implemented. The concepts of virtual storage and paging were demonstrated in the Atlas system more than a decade before they were formally documented as an integral part of IBM's main line operating system in 1972 (Ref. 44). The early computer systems such as ENIAC in 1946 had no operating system at all. Each operator programmed the computer personnally using machine code, examining individual storage locations and loading decks.

As the expense and speed of computers increased, executive programs were created to allow several users to sequence their jobs in a "batch" fashion. This prevented the computer from sitting idle while jobs were loaded manually, thus wasting costly computer time. An executive might also have included input/ouput control services (IOCS), run time limits, and system accounting.

In the mid-1960's as CPU speed increased, input/output caused a severe bottleneck in the system. Multiprogramming was developed as a technique to improve efficiency by overlapping input/output operations with CPU processing, thus keeping the processor and input/output devices well

utilized. The basic technique is to have several jobs in memory at one time. A job executes until an input/output device is required, then it is suspended and another job is executed. This technique worked well as long as a good mix of computation and input/output existed in the jobs. Soon it was realized that multiprogramming could be forced by using timed interrrupts to switch from one job to another (preemptive scheduling). Each job was given a specific amount of time to run, such as 100ms, at which time it was interrupted by a timer. The operating system would run a job until it was interrupted or input/output was required, then the next job was executed, etc. Appendix A contains a more detailed examination of multiprogramming and its efficiency.

Timesharing became common to most large operating systems and created a new world of interactive terminals in the early 1970's (Ref. 45). All input and the majority of the output goes directly to and from the user, thus eliminating most of the electro-mechanics of card and tape readers. To minimize response time (an important requirement for timesharing operating systems) these systems relied heavily on multiprogramming and preemptive scheduling techiques. Timesharing is also refered to as "interactive" or "conversational" computing.

As the number and variety of users increased, it became evident that "software packages" were required to meet users needs. These packages were not part of the operating system

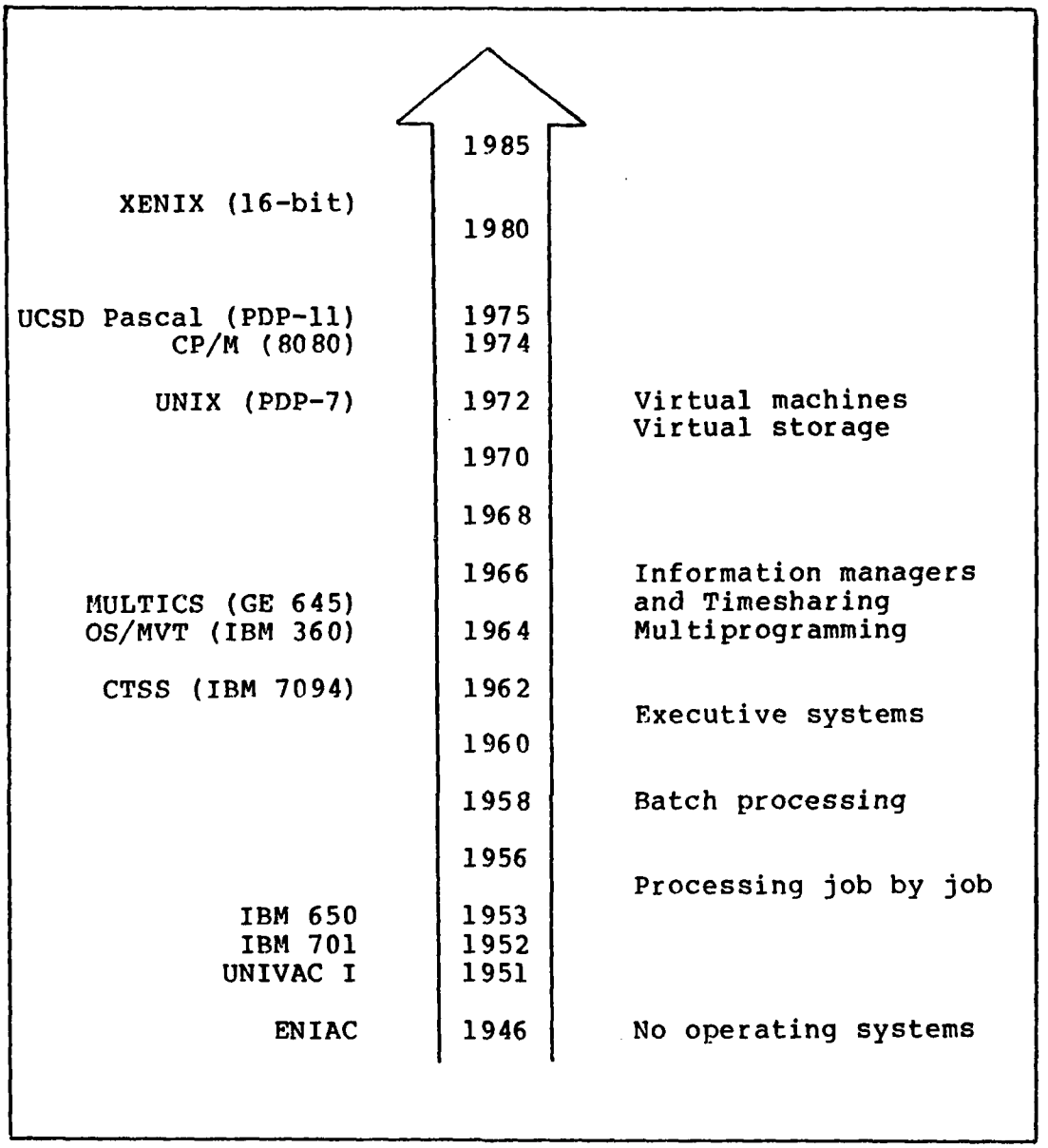


Figure 1. Evolution of Operating Systems

but were tools for the user just as the operating system was a tool to manage system resources. Examples of software packages are compilers, editors, applications programs, subroutine libraries, and utilitary routines. Also, the coming of data processing and large data banks influenced

the development of file management modules and information management facilities.

In 1972, Intel Corporation presented the 8008 microprocessor. This was the first commercially available 8-bit microprocessor and its development led to the 8080 8-bit microprosessor which became an industry standard. The 8080 microprocessor is largely responsible for the boom of computer hobbyists, economical industrial applications, and the minicomputer explosion. A microprocessor revolution took place that has yet to subside. A multitude of applications were discovered and developed due to the capabilities and availability of the microprocessor.

In 1974, Microcomputer Applications Associates (MAA) proposed a single user operating system as a companion to the 8080 microprocessor. CP/M (Control Program for Microcomputers) was a single user operating system that has been widely accepted. It is actually a very general operating system which becomes a special purpose system when it is "field-programmed" to match a particular operating environment. CP/M is now used in over 200,000 installations worldwide in over 3000 different hardware configurations (Ref. 26: 226).

Bell Laboratories, in 1971, developed UNIX, a powerful multiuser timesharing system with a vast array of software utilities which greatly increased productivity (Ref. 43). The increasing cost of software has made systems such as UNIX an attractive operating system not only as a

productivity tool, but also because of its ease of use, simplicity, and elegant design. The current thought is that UNIX or a UNIX-like operating system, such as XENIX, will become the standard system configuration in the future (Ref. 14: 252).

So the evolution of computer operating systems has come from simple, single resource managers to wide-spread standards such as CP/M and UNIX which allow the user to communicate in a more "human" way than merely toggling switches. The standards set by these and other successful operating systems provide a foundation to build on for the development of an operating system for the Intel 8086 microprocessor. Existing operating systems such as UNIX provide a basis for the design and development of this operating system. Given this historical perspective, the development of a system using the 8086 microprocessor architecture would draw on past achievements of system implementations and current design methodology.

Objectives

The objective of this investigation is to develop and design a multiuser, multiprogramming operating system for a class of 16-bit microprocessors. The development is to be based on current design methodology and modern operating system theory. The design will then be applied to the capabilities and limitations of the 8086 architecture. Consideration will be given to the number of intended users, type of peripheral devices, and efficient utilization of

computer resources in the selection of all algorithms. A foremost consideration will be the ease of use of the system and user/machine interaction. As in the development of UNIX, simplicity will be substituted for efficiency wherever possible (Ref. 54: 1932).

Approach

As with any in-depth study, a thorough literature search was conducted to gain a working knowledge of operating systems and current philosophy and methodology of their design, development, and implementation. Several successful and well known operating systems were studied to compare existing systems to the objectives of the system under design for the 8086 microprocessor.

The design of operating systems has not reached a level where there is a recognized standard approach to design. However, an operating system can be viewed as a large software system and approaches to engineering software systems are prevalent. These methods will be exercised to formulate the approach to software design. Chapter Two covers the design methodology in more detail.

A top down structured approach to the design was selected because of the amount of software involved. By using this approach, the system, as viewed by the user, will be addressed first. This will insure the user requirements are met and a primary objective of "user friendliness" will be foremost in design. Each level and modularity within levels will be developed and interfaced as well. In this

type of design, it is important to clearly define interfaces between levels to maintain modularity and proper hierarchial levels.

Since the operating system can be expected to be a large software effort, structured analysis design techniques are considered important to the success of the project. This involves such methods as graphic tools, logic modules, top down approach to design and implementation, and giving proper consideration to the user's point of view. This method produces a more efficient design and implementation (Ref. 57, 59).

Because the hardware is relatively fixed to the available 8086 microprocessor configuration in the laboratory, little emphasis was placed on the study of hardware design except as it pertained directly to the 8086 architecture. Appendix C briefly covers the hardware configuration and capabilities.

Overview of Thesis

The organization of the thesis follows the approach used in developing the operating system. Chapter One provides a brief insite to the historical importance and evolution of operating systems. Appendix A provides more information as to the importance of multiprogramming and timesharing to software productivity. Chapter Two examines the possible approaches to operating system design based on principles used in the past and more current software engineering techniques.

Chapter Three establishes functional requirements, concentrating on the man-machine interface. Appendix B provides additional material on research into the man-machine interface and how it affects the user's perception of the machine. It is basically a correlation of research on the "friendliness" of operating systems.

Chapter Four focuses on the system requirements of the operating system. Data flow diagrams are used to express the software requirements and support the narrative. Appendix D gives a brief description of the hardware system and its capabilities.

The design is developed in Chapter Five. Structure charts are constructed from requirements and data flow diagrams of Chapter Four. Throughout the design, structured analysis is used to provide a layered approach to the operating system development.

Finally, in Chaper Six, a summary of the effort is given and recommendations are made for future development.

II. Design Methodology

Introduction

Modern computing systems are a complex collection of coded routines, processors, input/output devices, and data bases capable of a large amount of interaction. A system of this complexity cannot be developed and designed without the use of an adequate design technique. Some method must be utilized to structure and decompose the system into understandable components focusing on specific interactions.

The purpose of this chapter is to study the available design methods for developing operating systems and to select and develop the techniques to be used for the operating system under development.

The Nature of Operating Systems

Many fundamental techniques of software engineering have come out of the design and development of operating systems. The reason for this is that the operating system is ususally the largest and first program developed for a given computer system. It is also the most logically complex software effort for any given system. The complexity is increased by the variety of inputs the operating system must accept and its exposure to penetration attempts. It must run continuously despite all efforts to make it fail. Unlike an applications program, an operating system cannot quit for diagnostics or abort for error conditions.

The complexity of operating systems is caused by conditions associated with the sharing of computer resources and current processes. Three of the most troublesome issues to consider during design are mutual exclusion, deadlock, and process syncronization and communication.

Since several processes are executing concurrently, two processes may attempt to use the same resource at the same time. Mutual exclusion prohibits more than one process from using the same resource. This is usually accomplished by declaring the unsharable resource a "critical section" (Ref. 62: 55).

Deadlock ocurrs when processess are waiting on each other to satisfy a condition one of the processes must resolve. This situation, called "circular waiting", (Ref. 16: 122) means the processes will wait forever. There are three design approaches identified for dealing with deadlock (Ref. 7: 46).

Syncronization and communication mechanisms between processes must be designed into the system because sharing resources requires coordination of processes in time (Ref. 31: 247). It is important to realize these same mechanisms may be used for solutions to deadlock and mutual exclusion. Several authors present different design methods for sycronization and communication of processes, most notibly Ref. 8, 16, 31, 51, 54.

Considering these conditions, the operating system requires a powerful design technique if the system is

expected to survive in this type of environment. If the operating system is slow, inefficient, or fails, then the applications packages follow suite. The operating system must be considered as the abstract machine upon which all other software functions.

The function of the operating system is to transform a hardware environment, with a very low level of execution, to an abstract machine, with a high level of execution, that will interface in terms comprehensible to the human user (Ref. 5). It must also interface with the software "users" such as compilers, data bases, and applications programs.

Design Objectives

Design objectives will vary considerably from system to system but there are general objectives that should be adhered to. One important reason for sharing the use of a computer is to make efficient use of its resources, and this should therefore be a primary objective of the operating system. However, in the process, the operating system overhead should not absorb too much of the same resources.

Another objective is reliability. Because of the number of users on a timesharing system, failure would be more disastrous than on a dedicated system. Also, correctness of the software system is crucial. The system must be predictable to the user even though the user may make unpredictable demands on the system.

Simplicity of design, implementation and use must be an issue in the design of an operating system. Without

simplicity a complete understanding of the system cannot be achieved and thus, complete control cannot be exercised over the system. To some degree these objectives are incompatible and it may be more important that they are not neglected rather than one is accomplished at the expense of the others. How well the designer is able to fulfil each objective will determine the success of the operating system design.

Design Approaches

The operating system can be viewed from three perspectives (Ref. 31: 8-20). The "process view" reduces the system to a series of user and system processes. The "resource view" consists of reducing the system to a network of processes that require resources. The "hierarchical view" consists of reducing the system to a series of nested machines where the outer most levels are dependent on the inner levels for execution.

The view used depends on which concept of the system is under study. The resource view is most useful when considering the utilization of system resources. The rate at which processes require resources and the rate required to service the process are analogous to random variables associated with a queue. The process view is most useful when considering the facilities provided to the user. Each process is determined by a procedure and a set of data with an initial state and a sequence of execution. The hierarchical view is most useful when dealing with the

functional aspects of an operating system. All three are considered at some point during the development of an operating system. The hierarchical view is the most useful to the system designer. In this manner, the designer can view the system from the user to the hardware for a top-down perspective and design approach.

Dijkstra was the first to formalize the concept of an operating system design based on a hierarchic structure (Ref. 9). His approach forms the operating system processes into a hierarchy where each level represents a successively more abstract machine then the preceding level, as shown in Figure 2. As a process moves up the hierarchy more resource management tasks are performed. Processes at one level can assume the availability of resources managed by processes at lower levels. His design is called the "THE" operating system and is the classic hierarchical operating system design.

In Dijkstra's system, level 0 is dedicated to the processor dispatching primitives and is available to higher levels. Level 1 virtualizes the memory by providing the page and segment management algorithms. Levels 0 and 1 can be referred to as a resident nucleus. Level 2 provides the virtualization of the operator console. Level 3 includes all the routines for the management of the peripheral devices. The ordinary user processes are at level 4 and the operator processes at level 5.

The abstract machine presented by the hierarchical

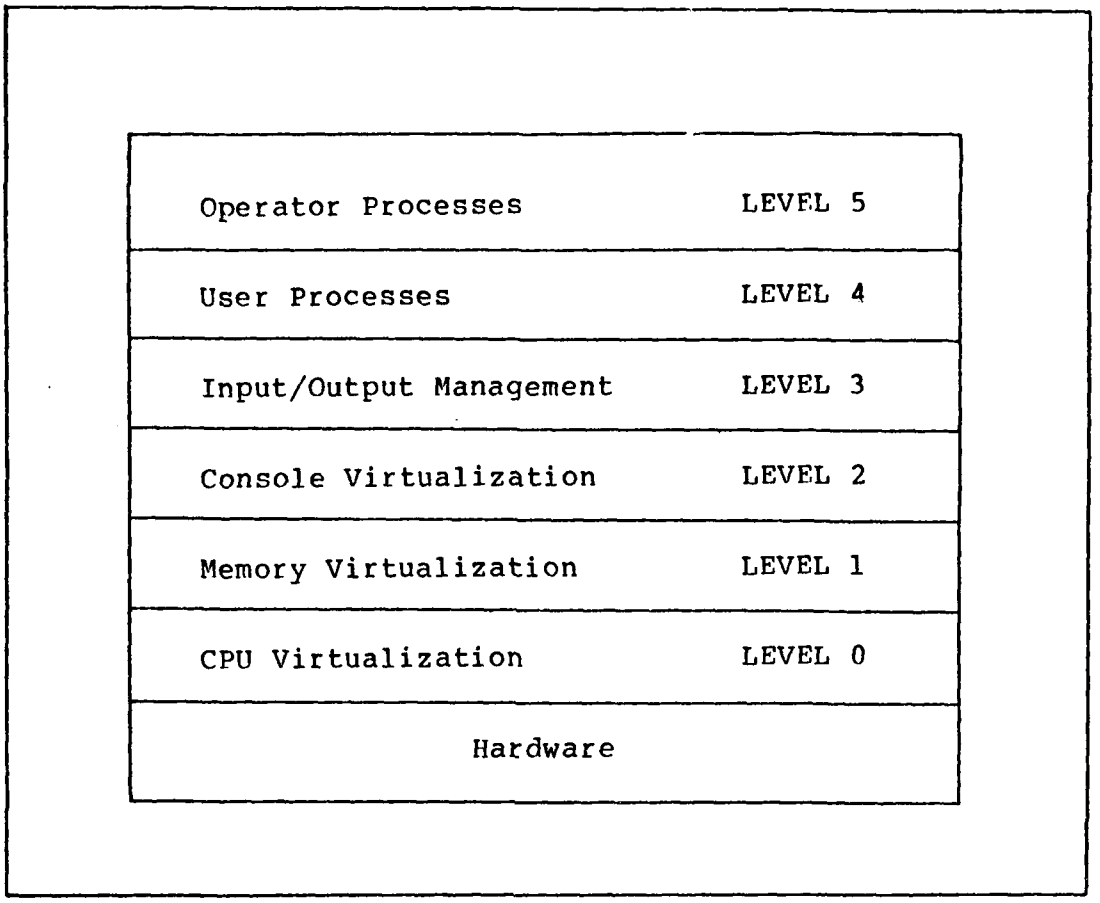


Figure 2. THE operating system hierarchy

methodology is attractive to system designers. By isolating operating system functions on an hierarchical level basis and establishing well-defined communication between levels, the complexity of design is considerably reduced. Dijkstra claims the system can be proven logically correct before implementation (Ref. 9: 342).

Another methodology, developed by Hansen, uses a bottom-up approach to design (Ref. 15). His design is concentrated on a multiprogramming nucleus that is general

enough to permit the construction of a variety of operating systems around the nucleus, as shown in Figure 3. The nucleus can be extended to new operating systems in an ordering fashion. Thus, the design is not limited to one application.

The ideas developed by Dijkstra and Hansen are classic examples of operating system design. Since their development, other techniques such as structured analysis (Ref. 57, 59) and its related software tools (i.e. information hiding, modularity, structured programming, data flow diagrams) have contributed significantly to operating system design (Ref. 5).

Hierarchical design, whether top-down like Dijkstra's approach or bottom-up like Hansen's approach, uses the

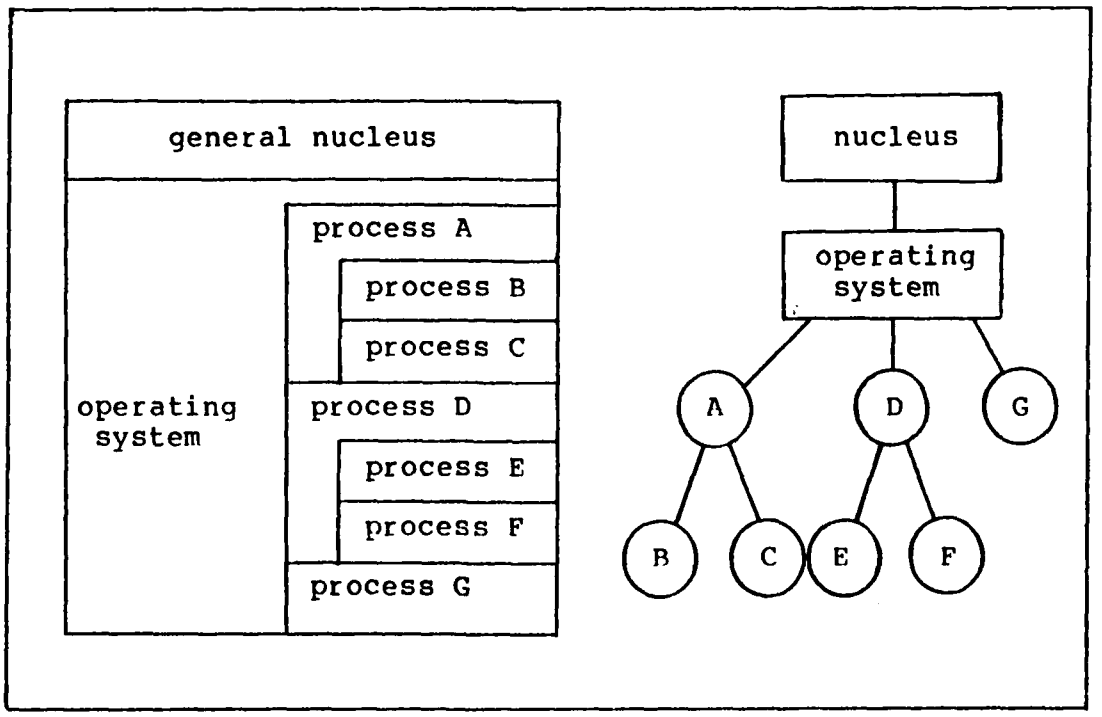


Figure 3. Hansen's multiprogramming nucleus.

concepts of partitioning, information hiding, and modularity to clarify the system at several levels. Each level becomes more refined until a level is reached where the degree of understanding is sufficient for the design to be easily understood and implemented.

The structured analysis approach (Ref. 57, 59) is similar because it uses graphic tools to represent levels of hierarchy and partitioning. Further, software tools, such as structured English and data dictionaries, allow a smooth transition from requirements to design to implementation. This permits an orderly approach rather than the designer being overwhelmed with a volume of details.

The approach Shaw suggests (Ref. 51: 107) is not unlike the structured analysis or hierarchical methods. He claims the design of operating systems is, however, not a rigid science. It is part science, part art, part engineering and part management. The five design steps he suggests are:

(1) Specify the virtual machine requirements.
(2) Describe the paths of processes through the system (determine what the system does before deciding how it does it).
(3) Determine the processes required to perform the tasks specified by 1 and 2; specify interaction and data structures.
(4) Specify allocation algorithms and stategies of the nucleus.
(5) Prove the correctness of the design and predict its behavior.

Step 5 is probably the most difficult to accomplish. However, the overall approach has several similarities to structured analysis.

Browne (Ref. 5) provides one of the best references and bibliographies on the intersection of operating system design methodology and software engineering techniques. Several design issues are discussed as well as more specific examples and research sources.

Language Considerations

Understandably, language issues are more a part of implementation than requirements definition or design. However, experiences with successful systems have indicated that language choice has a strong impact on the system early in the design (Ref. 39 and 54).

Many large computer systems spend the majority of their time executing applications programs. It is usually the case that these applications programs are written in a higher order language. However, the software systems that provide the basic support for these packages (i.e. compilers, operating systems, etc.) are often coded in assembly language. When program size or execution speed is more important than software development efficiency, assembly language is required. However, when programmer productivity is required, higher level languages are the answer even though they are less efficient in speed and storage requirements than assembly language.

At one time the use of assembly languages could be justified by expensive hardware and inexpensive software. Also, there may have been a lack of appropriate languages to support the programming environment. Today, there are a

number of higher order languages capable of expressing complex algorithms well and hardware is inexpensive while the price of software has soared.

Several examples exist of higher order languages being used where assembly language was once thought to be the best alternative. Two of the best known examples are UCSD Pascal (Ref. 39) and UNIX (Ref. 54) written in Pascal and the C language, respectively. The authors of UCSD Pascal claim Pascal was chosen as the implementation language because of its "ease and power" of implementation. The authors of UNIX preferred the use of a higher order language because the benefits derived far exceeded the costs. An operating system foundation developed by Intel Corporation called RMX was written in PLM (Ref. 24). The developers of RMX compare the use of a higher order language instead of assembly language to the hardware designer's use of LSI devices instead of TTL components.

The benefits of writing an operating system in a higher level language can exceed the cost of storage requirements and efficiency loss. Two cost benefits are less maintenance and reduced development time. It seems the best approach to the operating system is to develop the software in a higher order language. Then, if necessary, use assembly language to optimize the required portion of code to gain the speed and minimize the storage needed.

If portability is an issue, the use of a higher order language is essential. UNIX and UCSD Pascal have been

implemented on numerous architectures with only a small portion of the system rewrittten (Ref. 23, 39). Since assembly level languages are, for the most part, heavily dependent on the architecture of the machine, portability is limited to a certain class of processors if the entire system is coded in assembly level code. A good example of this is CPM which is limited to 8080 and Z80 type architectures.

PLM is a high order language which provides access to the microprocessor hardware. It supports absolute addressing, interrupt handling, direct port input/output, and a reentrant attribute for procedures. PLM-86 would be a good choice to implement the operating system for an 8086 based computer.

Conclusion

It is generally agreed that the operating system is the most logically complex software developed for a computer system. Multiprogramming and timesharing capabilities add considerably more to the complexity. In fact, multiprogramming introduces all the complexities of multiprocessing except resolution of race conditions (Ref. 5: 1046). Reentrant coding, deadlock, synchronization, mutual exclusion, sharing, scheduling and concurrency are some of the major issues confronting the operating system. Dealing which each of these is a design issue.

The reseach of Dijkstra, Hansen and others present tried and tested methods for operating system requirements

and design. Because of their work, the complexity of operating systems has been reduced considerably. However, it is important to use tneir efforts as paradigms and not as absolutes for design.

Structured analysis offers several advantages and similarities to early operating system design. It is based on a top-down approach utilizing graphic tools to express data flow and processes. This allows the complexity to be addressed abstractly at the higher levels. Each level is then partitioned into more detail until the refinement reaches a low enough level for easy understanding and implementation.

A high level language should be used to implement the design such as PLM. This should ease implementation, maintenance, debugging, and aid productivity.

Generally, the development should follow contemporary software engineering techniques. Earlier operating system designs can be used as paradigms. The entire design is an iterative process and is not a single vast sweep of design effort.

III. Functional Requirements

Introduction

This chapter will focus on the functional requirements of the timesharing operating system under development. Discussion will focus on current operating systems, such as UNIX, and why they are attractive from a functional viewpoint. However, the discussion will primarily be concerned with the man-machine interfaces, human engineering, and programming productivity. The discussion revolves around "user-oriented system behavior" or "user quality". These terms involve system characteristics such as ease of use, tolerance to user errors, minimum astonishment behavior, and minimization of error opportunities by the user. Each of these is important to the man-computer interface in a productive computer environment. They refer to how a system should behave to meet the user's needs and constraints. In German technical literature, "benulzerfreundlichkert" is widely used and literally translated as "user-friendliness" (Ref. 10: 270).

This is not a subject that can be covered easily. It is very important, yet little material is available that is well known and recognized. This may be because there are so many poor examples available and successful attempts are publicized too little. Appendix B contains support material for this chapter and further references for a more detailed study. Studies are revealed which indicate important

qualities for a particular class of users, as well as the physical characteristics desirable for interactive systems and appropriate dialogue styles.

Background

Several requirements can be specified for an operating system to fit a general environment. This aproach, however, can result in a vague and incomplete picture of a specific operating system. Different user requirements, machine architectures, cost, applications, and software are among the considerations affecting the design requirements of an operating system. The design of an "optimal and perfect" operating system is impractical because this presumes an exact knowledge of user needs, preferences, and experience.

However, several systems might be considered "standard" because of their widespread use and popularity. Their popularity is a result of their ablity to meet users' needs.

For example, CP/M is widely accepted as a single user operating system for small computers. As a result, software packages specifically for CP/M have been successfully marketed by several commercial vendors. This insures that each user of a minicomputer system has a "standard" to depend on with many software packages available (Ref. 61: ix).

UNIX, the operating system discussed in Chapter One, is quickly becoming a standard because of its versatility, adaptability to other hardware, and ease of use. Like CP/M, it also has hundreds of applications packages (Ref. 43).

Though CP/M and UNIX may be considered standards, they were developed for specific purposes. CP/M was developed specifically for use with the 8080 microprocessor in conjunction with the new floppy disk drives and 16K of main memory (Ref. 26: 222). UNIX initially was designed on a PDP-7 and was specifically meant for programming research. The fact that they have become industry standards is a compliment to their design. They did not set out to be standards but became popular by meeting needs of users and filling a gap between the user and an often hostile computing environment. UNIX has become so popular that "UNIX-like" systems for specific hardware configurations are being introduced rapidly, i.e. UniFLEX for the Motorola 68000 (Ref. 3: 7), Zeus for the Zilog Z8000 (Ref. 58 :120), Cromix for the Cromemco minicomputers (Ref. 3: 7), OMNIX for the Z80 (Ref. 3: 7), IDRIS (Ref. 40: 125) and LSX (Ref. 30: 2087) for the LSI-11. Two other transportable systems, Xenix (Ref. 14: 248) and Coherent (Ref. 3: 8) are intended to work on 16-bit microprocessors.

TENEX, developed in the early 1970's by BBN (Bolt Beranek and Newman, Inc.), was one of the first operating systems to incorporate "good human engineering" as a major design goal (Ref. 2: 136). An executive command language interpreter provided direct access to a large variety of commonly used system functions, and control and access to other subsystems and user programs. Command language forms were meant to be extremely versatile, adapting to the skill

and experience of the programmer. This creates a very productive environment for the programmer. The TENEX command language interpreter, FXEC, was designed with two primary requirements - ease of use and ease of learning. Thus, novices to experts can easily work with TENEX because of its versatile command language.

Another influential system, CTSS (Compatible Time-Sharing System), came into use at MIT in 1963. An interesting point about CTSS is that it influenced user/machine interface so radically that it was studied for the development of TENEX (Ref. 2: 136) and MULTICS (Multiplexed Information and Computing System) which was a cooperative effort between MIT, Bell Telephone Laboratories and General Electric (Ref. 38). Even the Popular UNIX system is considered a modern implementation of CTSS (Ref. 43: 1948).

Thus, a good case can be made for human engineering or "user friendliness" as an objective when developing the requirements of an operating system. Not only does it increase popularity of the system, but it also increases software productivity.

"User Friendliness"

The huge success of UNIX is largely attributable to its ease of use and user interface with the machine. As its inventor puts it, "Throughout, simplicity has been substituted for efficiency" (Ref. 54: 1932). This idea seems to be a general trend in operating systems. In the

past, efficiency has been squeezed out of the processor and memory, but often at the expense of the programmer's use of the machine. The Editor-in-Chief of a popular computer publication expressed the feeling of many software designers, "I'd buy an operating system any day that takes a long time to run a given program, but which makes me more productive by communicating with me in useful ways...the cost of a line of code is becoming astronomical" (Ref. 35: 6).

Up until the last few years, more opinion then knowledge has existed about what user-quality is in interactive systems. A widely accepted definition cannot be found. User-quality may be defined as a set of system properties which are relevent to man-computer interaction from the user point of view according to their sensory, cognitive and affective structures (Ref. 10: 270).

The user is typically unconcerned with the technical details of how services are programmed or produced, except perhaps when they fail to meet user requirements. Users are interested in the level of service delivered at the terminal, not the internal operation of delivery. However, knowledge of what constitutes good service presumes an understanding of user needs. The question is, what does the user need?

Users perceive the quality of interactive computer systems from different viewpoints. User-quality depends on the needs, preferences, problems and past experience of

users and can vary widely. It follows then that there can be no single measure of system quality. Therefore, "the" user-oriented system does not exist. For a group of users or for one user some system properties will be selected over others and some system characteristics will be judged undesirable by other users (see Apendex B).

When developing an interactive system, a frequent question involves the allocation of tasks to the computer and man. Usually, the question is resolved by allowing the computer to perform anything it can do better than the user. Assuming the current level of computer intelligence, this is a reasonable approach (Ref. 46: 852). But, the future may see computers performing the human role and taking over real life information processing tasks. What tasks are left for the human if the computer is allocated these tasks? It may be that the computer assumes all routine duties and humans are free for other more creative endeavors.

For many jobs, however, the computer may not be able to accurately simulate the human processor. The user will retain these tasks and the man-machine interface will always remain. If the total man-computer performance is to be maintained, the role filled by the user must be coherent with the computer. In future designs of man-computer interfaces, a strong attempt should be made to assume the user has a meaningful role and a friendly environment.

Command Language

The user access to a computer system and its various facilities is, in almost all cases, via a system command language. Probably no other feature is more important in deciding the individual's effectiveness in using the system than this aspect (Ref. 34: 512).

The command language structure may be thought of as a generalized finite state automaton where each state transition is associated with a condition and a response (Ref. 6: 362). The automaton can be represented by a state transition graph. For the environment to be controllable, the user must be capable of freely moving around in the graph. The user must be able to select a command and abort it when required. However, when an abort is selected, care should be taken when deciding to what state control should return. It is important that the user feel in control of the system and have adequate knowledge of the system to make control possible. As a minimum, it should be known where the user is, where the user has been, and where the user can go (Ref. 11: 344).

Contemporary computer scientists agree that computer systems should be more natural to use. The problem is, what is meant by "natural"? Fitter (Ref. 11) argues that while some users prefer computers to use plain English dialogue, it would be more desirable to issue instructions in computer language to humans. The point is, plain English is full of ambiguity and is unfit when precision is required. For the

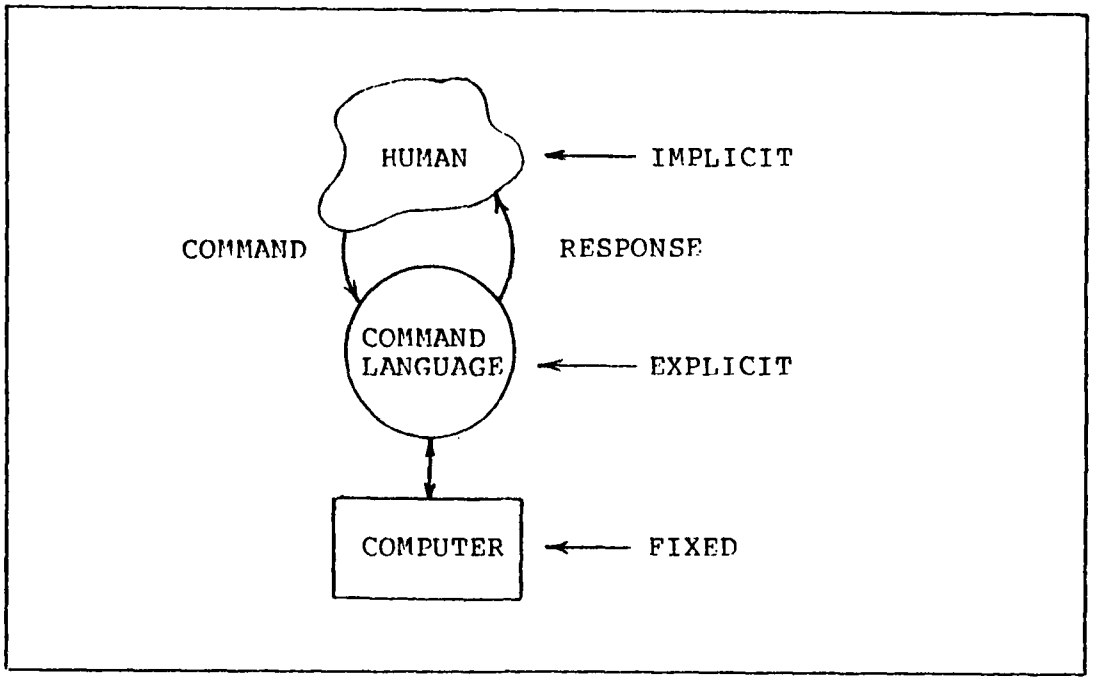


Figure 4. Command Language Interface

purpose of man-computer communication, a natural language is one that makes explicit, as indicated in figure 2, the knowledge and processes for which the human and computer share a commmon understanding (Ref. 11: 340).

It becomes the responsibility of the systems designer to provide a language structure which will make apparent to the user the procedures on which it is based and will not lead him to expect unrealistic powers of inference from the computer. It is required from the beginning to recognize the fact that a good command language will require complex programming. A reluctance to accept the overhead involved is one of the main factors contributing to the lack of good command languages available today (Ref. 25: 316).

Arguments

A command language must allow unambiguous specification

of what the user wishes accomplished. The information or arguments to be acted upon and the action to be performed must be specified. These can be specified by a number of methods (Ref. 56: 359). Arguments may be given explicitly, default values, implied by context, or abbreviated in an agreed form.

An argument specifies various options or alternatives for realizing the particular command. If a user does not specify certain arguments, the command language may automatically assign default values. For example:

```
SAVE     FILE1 DISK PROTECTED
(COMMAND)....(ARGUMENTS).....
```

specifies that FILE1 will be SAVEd. DISK will be the storage device and it is to be PROTECTED from other users. The default values may have been to store the file on tape and leave the file unprotected.

One alternative is to prompt the user with regard to missing arguments. However, the selection may become time consuming and complicated depending on the command and types of arguments involved. An attractive alternative is to assign default values automatically to missing arguments. This is one of the more powerful existing computer system concepts for achieving a user-oriented environment. Essentially, the use of defaults constitutes an agreement between user and the computer as to what a normal working value is for each default. Problems may still arise if the default values are unknown to the user or if the values are not easily changed. One solution may be for the computer

system to optionally display default values.

There are generally two methods for arguments to be formatted with commands. Positional format requires information to be in a fixed or absolute position within the string of arguments. Keyword format allows arguments to be given as a string permutation of special words indicating the argument type as well as its value. The positional format imposes the additional burden on the user of remembering positions. The keyword format is more user-oriented and studies show the positional format produces greater error rates (Ref. 34: 513).

Prompts

As mentioned earlier prompts may be used to cue the user when arguments are deleted from commands. In situations where it is undesirable to assign default values, the user should be prompted with some brief characters to indicate what is expected next and perhaps displaying the alternatives available for specifying or selecting the required information. Prompts may also be useful to act as cues in complex command chains to avoid confusion and prevent the chain of commands from becoming a maze. Above all, prompts should serve as a positive reinforcement that the computer is responding as required or expected - the equivalent of a human nod.

Some users consider it desirable to be able to select more or less terse prompts. More experienced users should be able to turn the prompts off if desired. However,

beginners have indicated that prompts are useful and usually desire prompts to be an obvious English mnemonic (Ref. 56: 363).

On Line Documentation

An interactive system should have the capability of providing help to a user when needed. Granted, on-line documentation is no substitute for written system documentation, but the middle of an interactive session is no place to study system manuals. Occasions may arise during on-line sessions when it is questionable what the system wants or what options are available.

A "help" facility can add flexibility to the system by allowing the novice or casual user the confidence and same capability as the more experienced user. A simple reply, such as a "?", to the computer should provide sufficient information for the user to continue the computing session. However, verbose messages should be avoided if possible.

One method of on-line documentation is to provide the user with a hierarchical classification tree which is traversed by selection of key words from menus. This is not an unreasonable approach and it may save the interactive user from digging in hard-copy manuals but at the same time, it can be time-consuming and digging in hard-copy may be the better alternative until the user is more experienced on the system.

Error Messages

The text of error messages is important and should be

as specific as possible to the problem. However, lengthly or verbose messages are best avoided for the experienced user who has made a simple error. It is a good policy to have a facility to supress long error messages if desired. A main theme throughout the command language facilities is to adjust the flow of interaction to the ability of the user.

Recovery

Recovery is the reinstatement of some past state of the computer system, usually after an error or system malfunction. There is little literature on the specfic procedures to take when recovery is necessary, but Miller (Ref. 34) presents four types of recovery situations with respect to the interactive user:

(1) user correction of data input
(2) user deletion of issued commands
(3) abort or abnormal exit from a program
(4) system crash

With respect to (1), most systems allow local editing or correction before transmission to the computer. This presents little or no problem.

In situation (2), a more complex procedure is involved depending on the command that has been issued. For example, if the wrong file has been inadvertently deleted the system may provide back-up copies of deleted files and recovery is achieved by the user requesting his back-up copy. Obviously, not all issued commands can be recovered with no loss. Nevertheless, the novice who is afraid of issuing

commands may find this beneficial. It helps ease the stress of issuing commands if it is known some may be withdrawn.

In situations (1) and (2) it is important to point out that it was up to the user to detect errors and initiate recovery action. However, situations (3) and (4) originate with the actions of the system. In both situations it is desirable to inform users with regard to what happened, why it happened, and how to recover.

Situation (3) typically occurs because the program controlling the user session has been asked to perform some illegal procedure and no error routine is available to test or handle that particular circumstance. Usually the user is given no more than a "job aborted" message. Preferably, the supervisory system will provide information on where the error occurred, what rule was violated, and what can be done to recover and continue processing.

There is usually little that can be done in situation (4). System crashes are abrupt, unscheduled terminations of system operation. It would be desirable to shift full responsibility for system crashes to the system itself by providing a separately configured and powered processor and media system. Following the recovery procedure, the system could automatically recreate the earlier system state, however, this could become quite extravagant.

Files System

Users ususally create a large number of text and program files. Operations on these files are numerous and

include merges, hard-copy outputs, editing, manipulations to other peripheral devices, and execution. Simple, direct, and consistent means should be provided for accomplishing all of these operations independent of the nature of the file, its structure or size. Also, the methods should be consistent throughout the system.

The file system is very visible to the user and its effectiveness contributes a great deal to the friendliness of the operating system. As a minimum the file system may:

(1) Allow creation and deletion of files.
(2) Perform automatic management of secondary memory space. The user should not have to be concerned with location of his file in secondary memory.
(3) Protect user files against system failures. Unless convinced of system reliability, user will be hesitant to use the file system.
(4) Allow reading and writing to files.
(5) Use symbolic names to reference files. Users should not be required to keep track of physical locations of files.
(6) Users should be able to share files among cooperating users and protect files as desired.

As an example of what has made UNIX so popular, consider the file system used by UNIX. It is a tree structure originating from a root directory, thus, a recursive structure (Ref. 14: 256). Names of files and subdirectories are contained in the root. Subdirectories contain names of other files and additional subdirectories, etc. A user is assigned a unique subdirectory when logging on as the current working directory. Full path names for files consist of a possibly null sequence of subdirectories separated by a slash beginning with either the root or a current working directory, and followed by the file name as

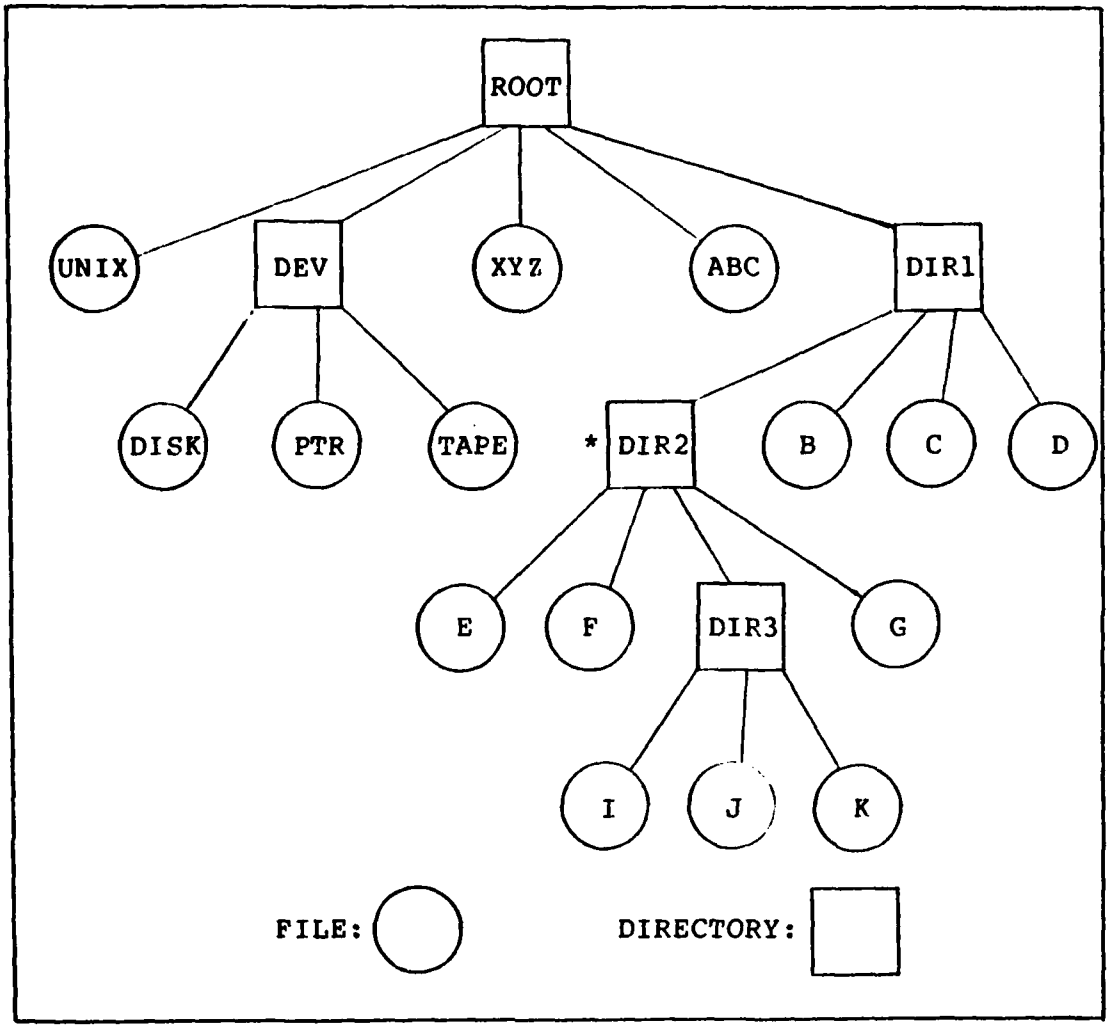


Figure 5. UNIX File System Implementation

Table 1. UNIX File Access Convention

From user start point * (DIR2):

File	can be accessed by:	alternative:
UNIX	../../UNIX	/UNIX
PTR	../../DEV/PTR	/DEV/PTR
C	../C	/DIR1/C
E	E	/DIR1/DIR2/E
K	DIR3/K	/DIR1/DIR2/DIR3/K

indicated in figure 3. (Ref. 4). Table 1 contains commands for files and directories in the file system. By convention, the file in each subdirectory called ".." refers to the parent directory. Therefore, the user has a concept of local and global files neatly organized into directory groupings.

Applications Packages

Applications packages and utilities are normally not part of the operating system and it is not intended to make them part of this design effort. However, an operating system, besides performing services for tasks as they execute, should provide the facilities to assist system users in the design and testing of new software. Applications packages can make a significant impact on user productivity. Operating systems are often characterized by the applications programs they provide. These facilities may include editors, debuggers, performance measurement, diagnostics, language translators, program loading, libraries, and input/output facilities (Ref. 62: 38).

If frequency of usage is intended to measure the importance of facilities, editing appears to be the most important facility provided by the computer system. An analysis was done on the interactive commands issued to a IBM TSS/360 which found that 75% of the commands were for editing. Programmers issued approximately 50% editing commands and users involved in text documents issued 80% editing commands (Ref. 34: 514).

Editors should have the ability to change, insert, and delete character strings. However, a well-designed editor should also be able to establish fields and move from field to field (perhaps via tab controls), have special commands for moving groups of lines from one position to another, provide a scheme for numbering lines, and string search facilities. One approach to editing, which is very user-oriented is the screen oriented editor which uses function keys to manipulate the input text or program. This is usually reserved for text editors only but program editors could also benefit from the screen oriented approach.

Response or Feedback

The computer should respond quickly or if that is not possible, the user should be provided with some feedback that the program or activity is operating. If the user's input causes a move through different states in the machine, the user should be informed immediately of the new state entered. Systems which allow the user to think one activity is being performed when the system is actually doing something else are extremely frustrating (Ref. 11: 345).

Although an immediate response is obviously desirable, there can be technical difficulties in producing one. In a multi-user system the problem is dependent on many factors. Overloading is a function of the user population and can be controlled more easily as the users become homogeneous and well defined. Complex scheduling algorithms may be required

to guarantee response times among fluctuating user characteristics.

Gaines and Facey (Ref. 13: 895) found that even under ideal circumstances, it is difficult to estimate response times even for the zero-computation interactive user. Also, varying response times without apparent cause were frustrating to the user. However, it is also pointed out that the response the user expects is related to the task requested. The interactive user may expect an immediate response or feedback after a simple command but the user is also willing to accept a longer wait period if a complex computation from the system was requested (Ref. 33).

Storage Size

Storage requirements are dependent on a number of factors. The number of users, size of user programs, application packages, and efficient memory management will all have a bearing on the storage needed.

Generally, the operating system should be required to manage a hierarchy of memory. Slower memory devices such as disks can be treated as input/output devices with sufficient storage capacity for each user. Faster direct access memory is more limited and more strictly managed by the operating system. A balance should be achieved between user needs, the number of users and what is available.

The operating system itself can be expected to utilize a significant portion of memory. For example, the UNIX system kernel occupies about 90K (Ref. 42: 1907) and

XENIX-8086 memory needs are about 82K (Ref. 27: 51).

Media Requirements

To facilitate flexibility and speed for retrieval and storage, media devices are required which allow rapid access to information. Space should be available in reasonable quantity and at a reasonable cost so the user does not feel restricted in what or how information is stored. This may ultimatly require a hierarchy of media, including removable media such as floppy disks. Irregardless of the type of media, the system should support variable types and numbers of devices without effecting system performance drastically. The operating system should create the abstraction of a virtual device. Given the proper device handlers to interface with the operating system, a variety of media is possible.

Security and privacy capabilities of media is also important. Media should be sharable when necessary. Yet, provisions should exist to allow access only to the author when desired.

Ease of Implementation

Implementation was briefly discussed in Chapter Two. To be implemented a system must be easily understood and well-designed. Ease of implementation is required if a system is ever to be realized. Otherwise, the most innovative ideas and proficient design is of little value.

Weinburg supports a top-down implementation (Ref. 57:

216) as do Yourdon and Constantine (Ref. 59). Their approach encourages the use of graphic methods, i.e. data flow diagrams and structure charts as development and design tools. Such graphic methods provide a modular picture of the entire system. The apportionment of the system can be visualized and implementation phases are more easily identified.

Usually, systems easily implemented using the above approach are also easily modified. A well designed system will need few modifications. However, as needs and users change, modifications may be desirable.

The language aspects of implementation were discussed in Chapter Two. An appropriate high order language can ease implementation considerably.

Ease of Learning

Throughout this chapter, user oriented topics such as command language, files management, system response, and system "friendliness" have been stressed. Hopefully, the end result is a system that is easy for the user to communicate and work with. The prime funtional requirement, ease of use, is a must for a high degree of productivity in an interactive system. For the novice, ease of learning is equally important and should remain a dominant theme throughout the design.

Conclusions

Much of the above discussion may present itself as

obvious requirements. Even though this may be the case, it remains a fact that few interactive computing systems score high when measured against some of these requirements. It has been suggested that this is because human factors such as "user-friendliness" are considered too late in the design process and are only considered an embellishment to the system (Ref. 11: 346).

IV. System Requirements

Introduction

This chapter addresses the system requirements necessary for the operating system under development. The functional requirements of Chapter Three will be translated into more specific system specifications. The first area will address the hardware requirements for a timesharing operating system. The software requirements are approached using methods discussed in Chapter Two, primarily structured analysis techniques as given by Weinberg (Ref. 57). The application of structured analysis is briefly discussed and its use is justified. The software requirements are developed from the user level, to the file management level, to the input/output manager, to the process scheduler, to the memory manager level, and finally to the nucleus.

Hardware Requirements

The following hardware requirements are based on several factors. First, any multiprogramming system which provides timesharing requires a specific type of hardware environment to operate in. Second, the functional requirements of Chapter Three imply certain hardware facilities to function as intended. Those same requirements may introduce constraints on the hardware. Further constraints are possibly imposed by the existing hardware or eliminated by state-of-the-art techniques.

While the hardware requirements reflect the desired hardware configuration, any shortcomings in the actual hardware must be addressed in software. In other words, if the hardware capability does not exist on the implementation computer, the requirement must be met with software.

Interrupt Hardware

Interrupts generally arrive at unpredictable rates and times with several interrupt events occurring simultaneously. Process switching, input/output devices, timers, and errors all create interrupts within the system. Both hardware and software priorities on interrupts are required because of the importance to the operating system of processes waiting for interrupt signals for awakening and because of the existence of timing constraints between an interrupt causing event and its processing. Preferably, a set of hardware priorities for a broad class of interrupts plus the ability to enable and disable interrupts permits the system to dynamically establish precedence relationships among interrupt causing events. Upon an interrupt occurance, the hardware should save the state of the CPU, determine the interrupt source, and initiate the appropriate routine to handle the interrupt.

Timer Mechanism

A timer must be available that is programmable and can produce interrupts at regular intervals to control timesharing. The timer is particularly important to multiprogramming and preemptive scheduling techniques.

Storage Protection Capability

It is important to have hardware facilities for protection of main memory. Protection should be sufficient to protect processes from malicious or erratic invasion of other processes. This is applicable to both system and user processes.

Direct Access Secondary Storage

System flexibility is increased by providing fast storage for library routines and system programs. Peripheral devices are kept busy by providing auxilary storage of input/output. User files must be stored in an orderly fashion on readily available media.

Structured Specification of Software Requirements

Structured Analysis is a top down, philosophical approach to each phase of the systems development life cycle, utilizing graphic tools and a structured methodology (Ref. 57: 320). Several techniques are available to engineer the software structure of operating systems (Ref. 5). Structured analysis was chosen because of its advantages over other methods and similarities with the methods used on successful operating systems as discussed in Chapter Two.

In structured analysis, the top level of the system, closest to the user, is addressed first in an abstract manner. Each succeeding lower level is addressed to refine the higher levels by partitioning the processes and data flows. This concept of partitioning allows a complex system

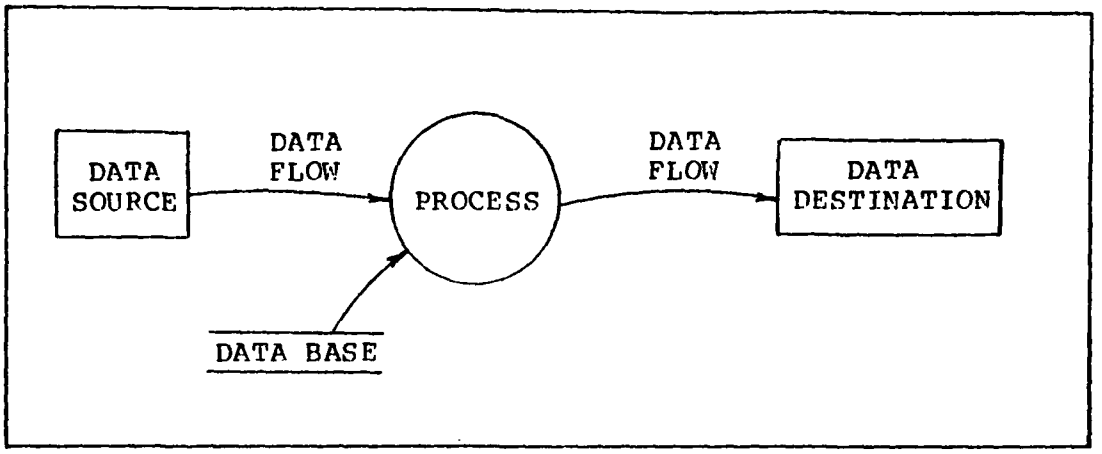


Figure 6. Data Flow Diagram Symbols

to be addressed one level at a time without confusing the operation with burdensome details.

The graphic tools used to represent the processes and data flows are indicated in figure 6. Rectangles represent the source and destination of data for a particular level. Circles represent the process involved and an arrow is an indication of data flow to and from the processes. Data bases are shown by straight lines. A data element is a data flow which cannot be partitioned further. Each data flow and process is partitioned from the user level to a low level representing data elements and low level processes. At this point the data flows, data bases and processes are defined and recorded in a dictionary.

The approach to structuring a hierarchical system will create layers which provide a set of functions dependent only on the layers within. Each layer can be regarded as implementing a virtual machine to the layers above. The final layer represents the virtual machine the user sees.

Table 7

Operating System Layers

Operating System Shell
File Management
Input/Output Management
Scheduling Management
Memory Management
System Nucleus

As indicated by figure 7, the operating system must provide all interaction between the user input and the user output. The interaction will include action by the other layers in the operating system invisible to the user. The operating system includes all layers as well as the hardware interfaces required to implement the functional requirements of the system. A "boot" process is necessary to initiate the system configuration.

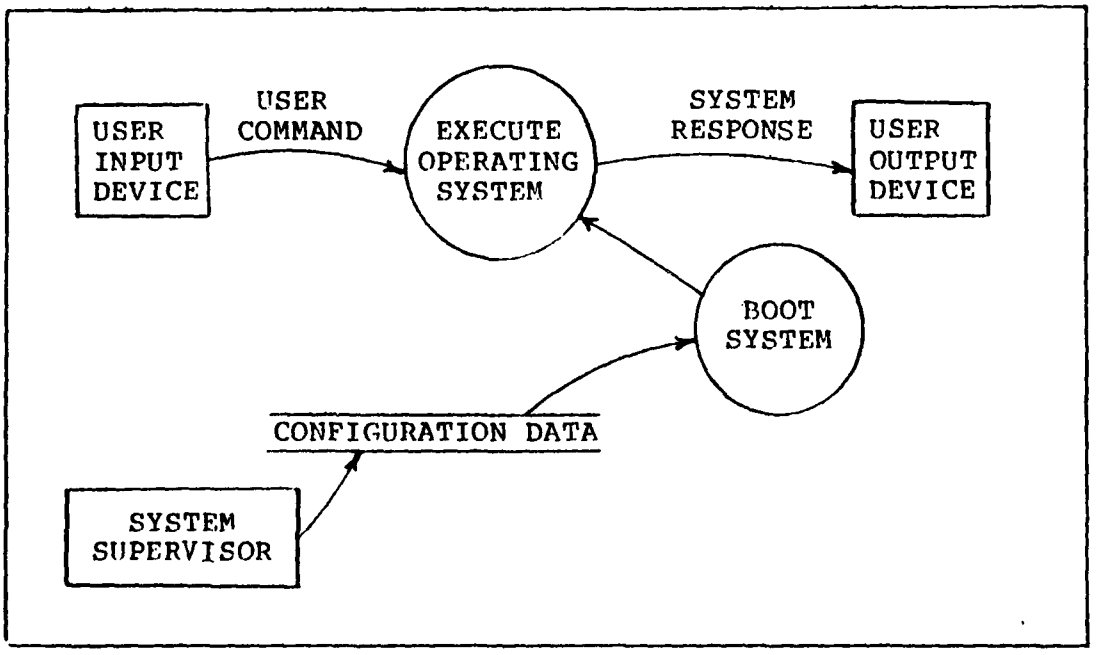


Figure 7. Operating System Context Diagram

Table 3

Operating System Shell
(Figure 8)

1. Determine Command Type

2. Execute System Command (Figure 9)
   - 2.1 Verify Authorization
   - 2.2 Provide System Menu
   - 2.3 Configure System

3. Determine Command

4. Execute Control Command (Figure 10)
   - 4.1 Determine Control Command
   - 4.2 Log-In User
   - 4.3 Log-Out User
   - 4.4 Execute Inquiry Command

5. Execute Help Command (Figure 11)
   - 5.1 Determine Help Required
   - 5.2 Provide Command Information
   - 5.3 Provide System Information

6. Execute User Command (Figure 12)
   - 6.1 Determine Command Conditions
   - 6.2 Prompt User
   - 6.3 Execute Command File

7. Respond to User

Operating System Diagram

The data flow diagram in Figure 8 indicates the software requirements for the operating system. Input commands must be tested (1) to determine the type, supervisor or user. Supervisor commands are reserved for the operating system supervisor and execute (2) processes which alter the configuration of the system, peripheral device parameter, or users eligible to use the system. If the command is determined to be a user command, three further possiblities cause another determination to be made (3). The user command may be a control command (4) which

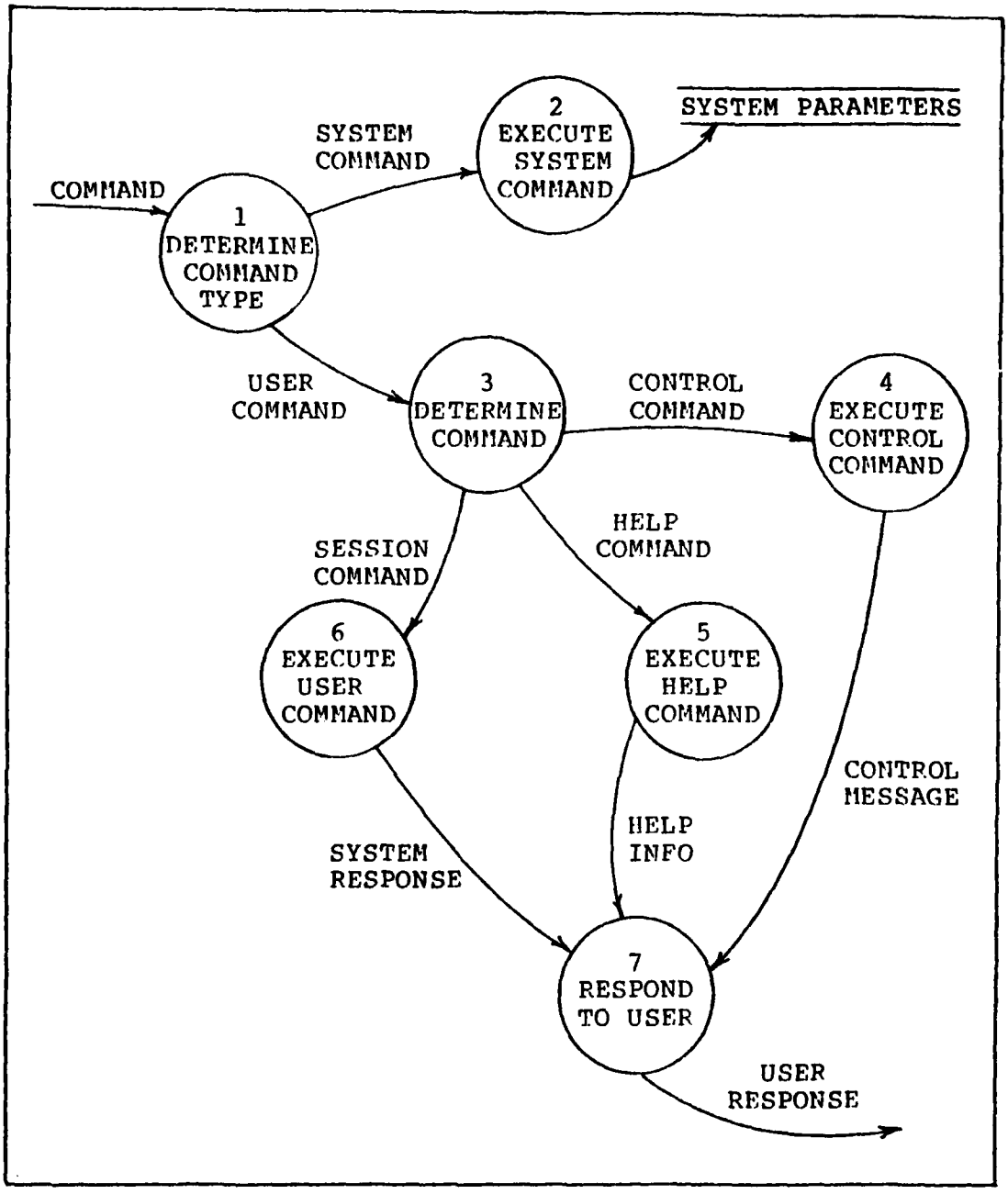


Figure 8. Operating System Shell Diagram

requires special attention by the system. Examples of control commands are log-in, log-out, or special system inquiry commands. A help command (5) will provide the user with assistance in determining the next possible action or

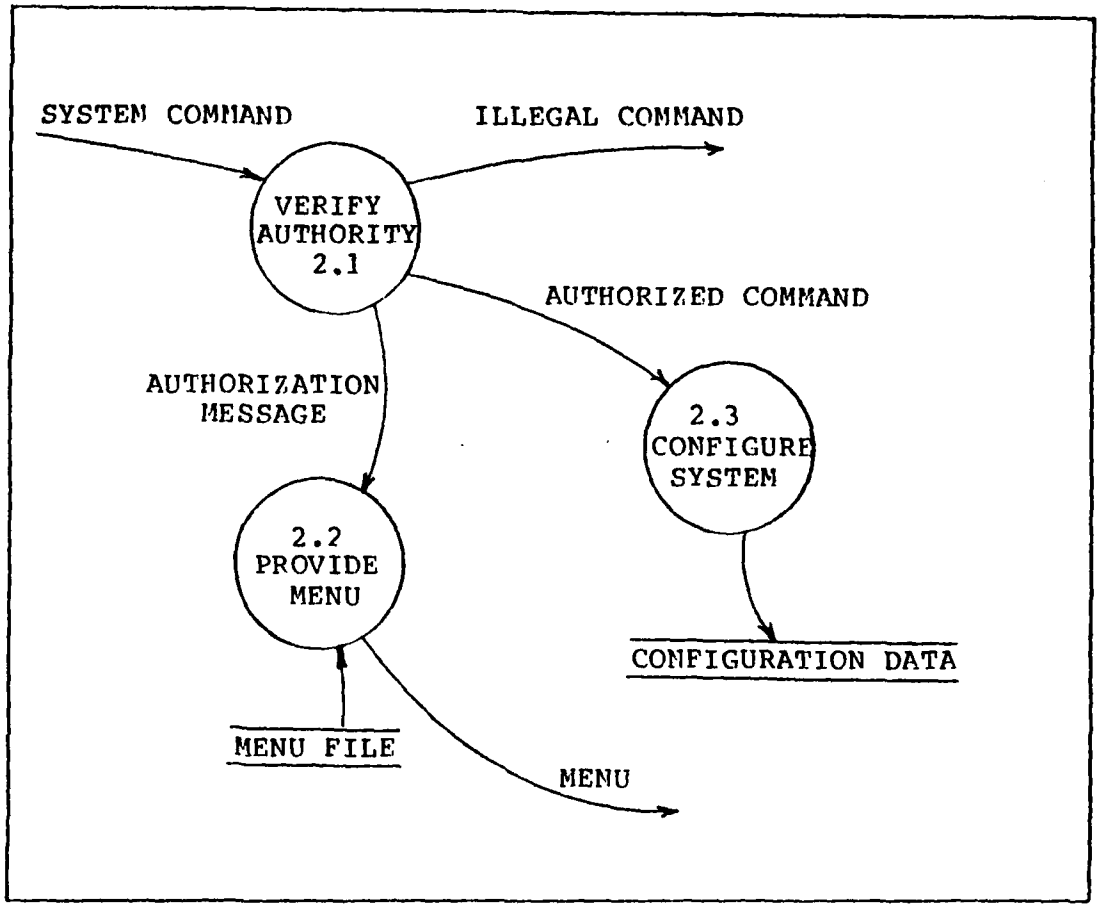


Figure 9. Execute System Command

command. Help provided will depend on the users state in the system. Commands to the operating system (6) are executed in a manner characteristic of each command. Since commands may require different parameters and arguments, each command is dependent on special circumstances and is treated as a general case here. Finally, each request by the user requires a response by the system (7). The system must keep the user informed of the user status as often as necessary.

System commands are commands reserved for the system

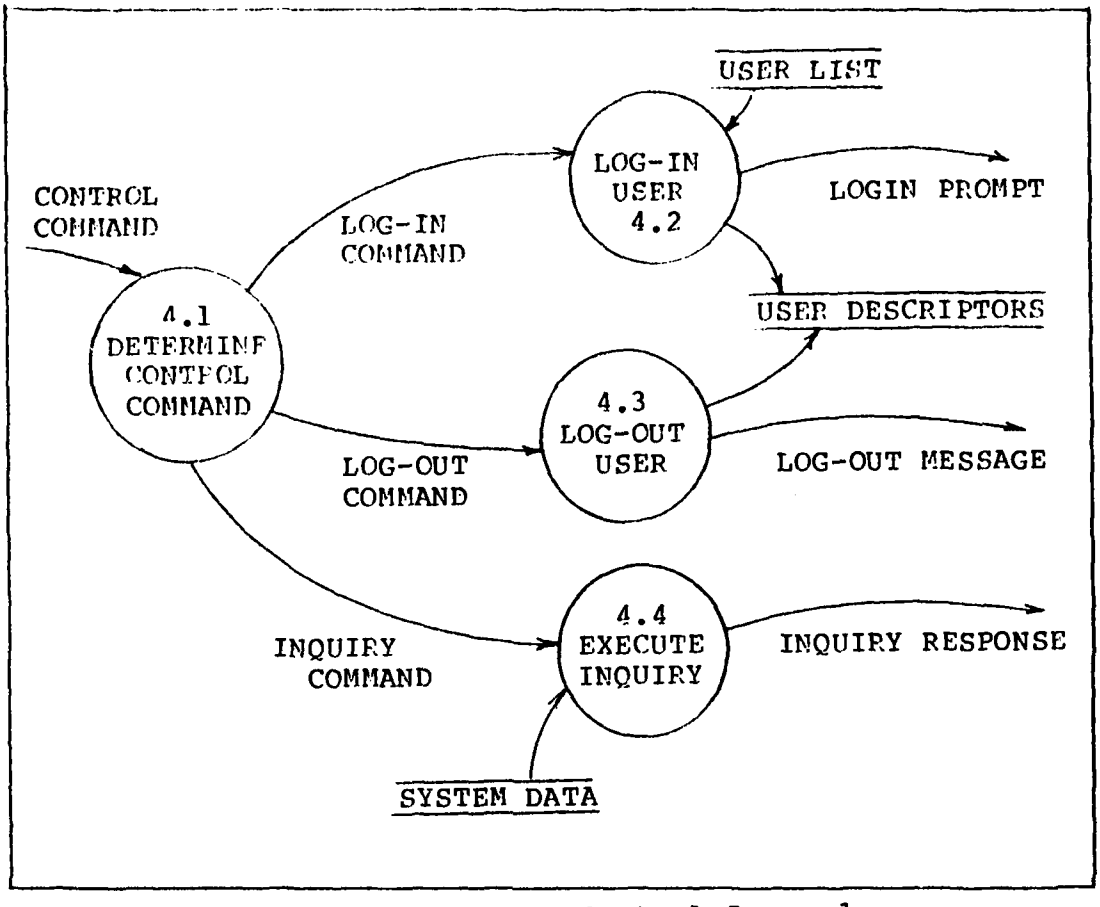


Figure 10. Execute Control Command

supervisor. In order to use a system command, the user's authority must be established by the system (2.1). Once authorization is given, the system provides a menu (2.2) of characteristics which may be altered to suit user needs. Commands are selected from the menu, issued to the system, and written out (2.3) to the data base used to configure the system.

Control commands are used to request special information from the system, enter or exit the system. A determination (4.1) must be made which of these commands is being requested. A log-in command (4.2) will cause a check

of the configuration data, more specifically, the user list. If the log-in attempt is legitimate, a user descriptor will be established. A log-out command will delete the user descriptor (4.3) and a log-out message is passed to the user. An inquiry command is similar to a help command with the exception that the user need not be logged-on the system to successfully execute it. Inquiries are executed (4.4) by accessing a system file containing general information such as who is authorized use, the name of the system manager, and system capabilities.

A help command is classified as a system information request or a command information request (5.1). A system information request will access a data base containing information on the general capabilities, characteristics, and configuration of the operating system (5.2). A command

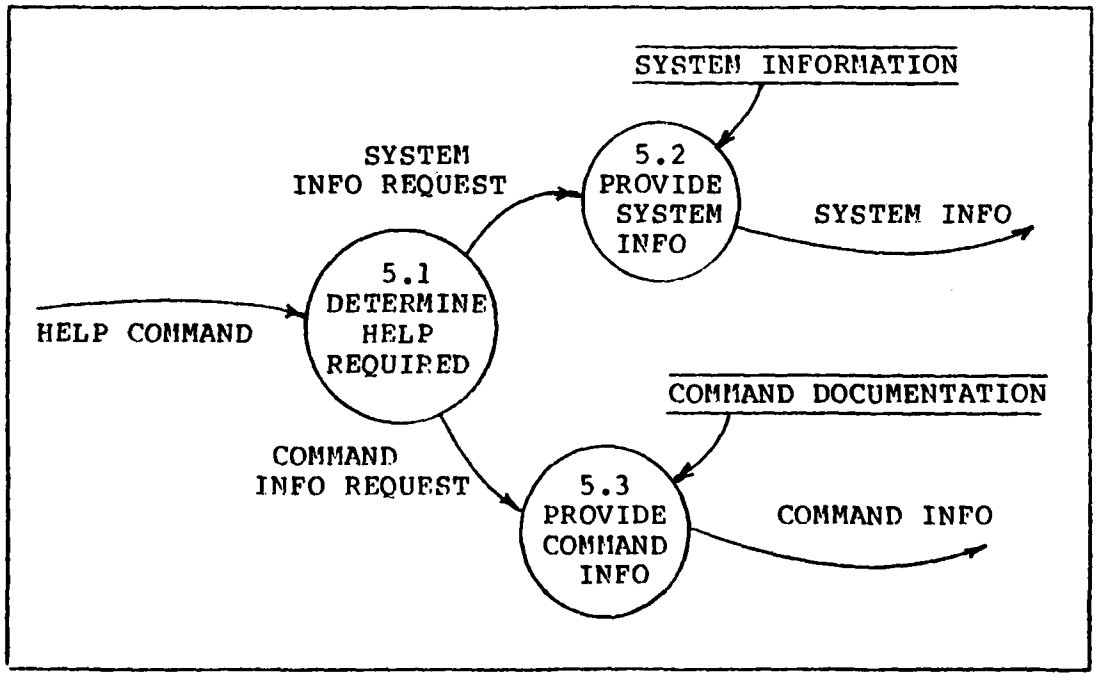


Figure 11. Execute Help Command

information request (5.3) will provide information on available commands and their usage.

There are generally two schools of thought on the issue on command languages in operating systems (Ref. 29: 51). The first is that a command language is not part of an operating system but should be a separate applications program which runs on the system. The second philosophy is that the command language is the final interface between the user and the machine, and is an additional built-in layer of the operating system. The approach here falls somewhere between the two arguments. The requirements as given provide an environment for a flexibile command language but do not specify commands, arguments, or execution. Thus, environment is part of the operating system and the command language is a separate package.

A session command will execute a command to the operating system. First, the command string is interpreted (6.1). Special conditions involving the command execution are examined (6.2). This information is contained in a command table and includes arguments or special characteristics peculiar to that command. Special conditions may include such capabilities as looping a command several times with different arguments. Any necessary information not provided by the user must be requested (6.3) to accomplish the command. Finally, the command is executed given the conditions and arguments (6.4).

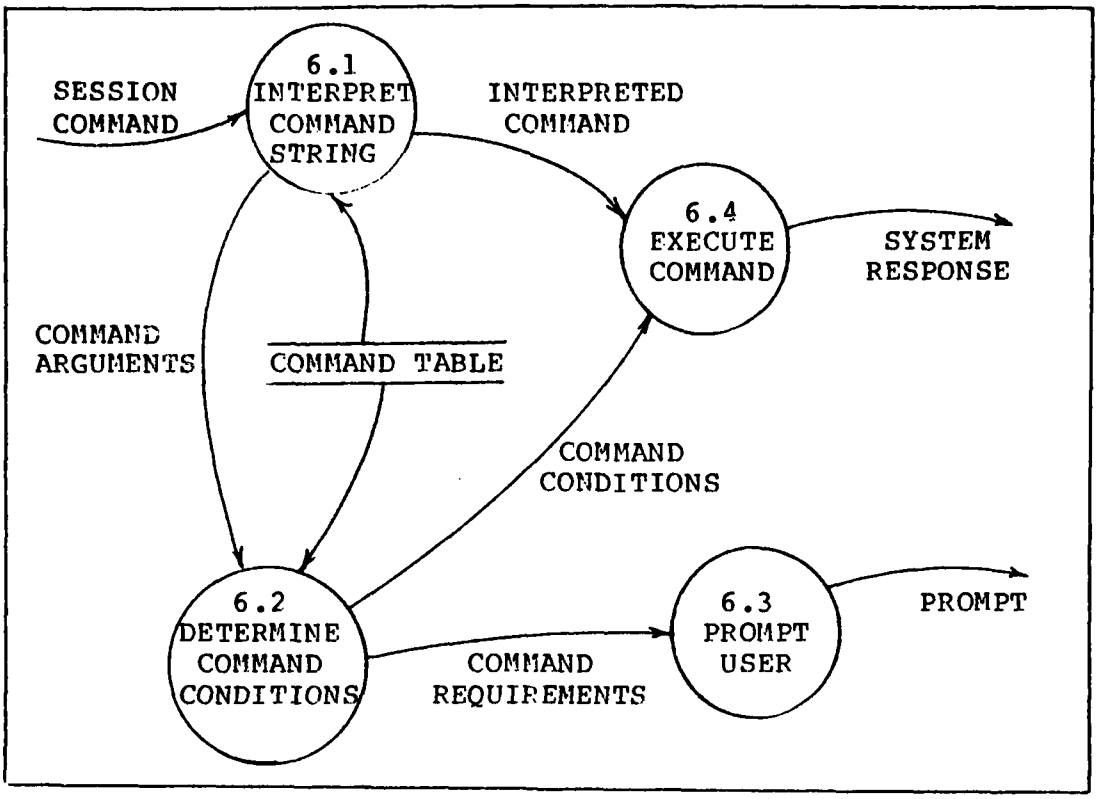


Figure 12. Execute User Command

Files Management

Files management is concerned with the storage and retrieval of information by the system in an efficient and well organized manner. This involves four system responsibilities (Ref. 28: 337). A record must be maintained on all information available in the file system. This will include the name, location, and access rights to all files. A determination must be made of how information is stored and who has the right to read, write and access the files in storage. Some files may be shared and others must be protected. File space must be allocated to the users and initial access rights and characteristics.

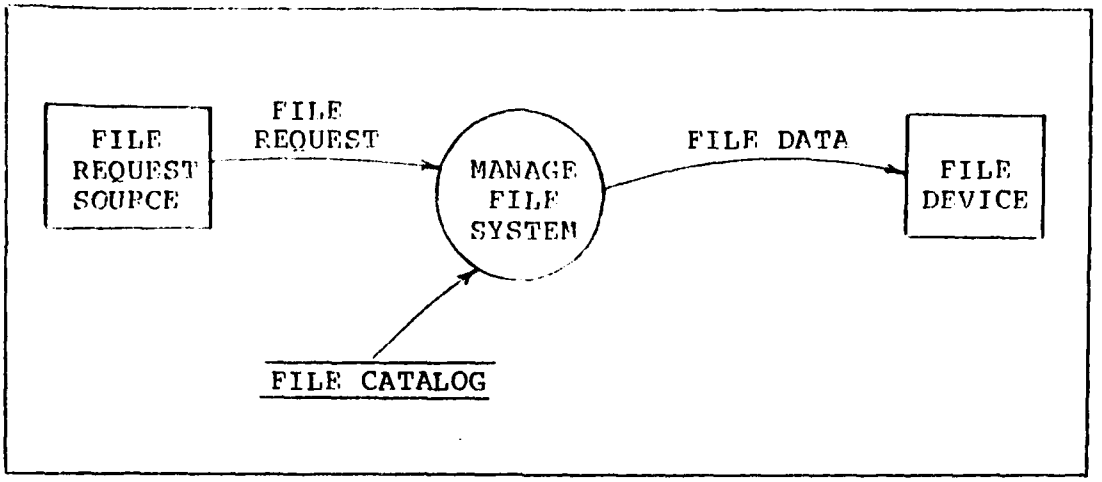


Figure 13. File Management Context Diagram

established. Also, deallocation must be accomplished when the user no longer needs the information.

This process is conceptually simple, yet is very important with regard to the user-machine interface. It is also the largest part of many operating systems and requires the most design and coding effort (Ref. 43: 244).

There are two benefits that motivate file storage. It is a great convenience to the user to be able to store information on-line. In a multi-user system it is impractical to expect the user to operate without on-line storage when the only input device available is a video terminal. Second, the sharing of information is desirable among users. Applications packages can be used by several programmers and data may be shared between several users. Storage is in the form of a logical unit, a "file", of arbitrary size. The file is separated into blocks of fixed length on the media. The requirement is to map a symbolic

Table 4

File Management
(Figure 14)

1. Determine Master Directory

2. Locate User File Directory

3. Execute Open File (Figure 15)
   - 3.1 Determine Mode
   - 3.2 Locate Directory Entry
   - 3.3 Allocate File Space (Figure 16)
     - 3.3.1 Determine Number of Blocks
     - 3.3.2 Identifiy Free Blocks
     - 3.3.3 Connect File Blocks
     - 3.3.4 Establish Access Rights
   - 3.4 Extract Directory Data
   - 3.5 Check Access Rights
   - 3.6 Execute Link File Routine (Figure 17)
     - 3.6.1 Determine User Directory
     - 3.6.2 Link Owner to User File
     - 3.6.3 Link User to Owner File
   - 3.7 Determine Physical File Location
   - 3.8 Create File Descriptor (Figure 18)
     - 3.8.1 Create Local File Block
     - 3.8.2 Test Central File Block
     - 3.8.3 Create Central File Block

4. Execute Close File (Figure 19)
   - 4.1 Delete Local File Descriptor
   - 4.2 Update Central File Block Status
   - 4.3 Delete Central File Block

name to a physical location in secondary storage. A file catalogue is needed to hold information on the names and associated locations of files. A master file catalogue holds the location for each user's files directory.

The master directory must be determined for the user making the request (1). Since a user may specify another directory as the master directory, and each of the files can be directories, this check must be made each time a user makes a file request. The master directory is fixed for system file requests. Once the master is determined, the

location of the user file must be determined (2). The user file directory contains all the information about a file that is necessary to fill the file request. The file is opened (3) by creating a file descriptor for input/output processing. After the file action is completed, the file is closed and must be re-opened before it can be read or written to again.

Opening the file consists primarily of looking up where the file is stored, on what device, and gathering the required information for the requested action. The first

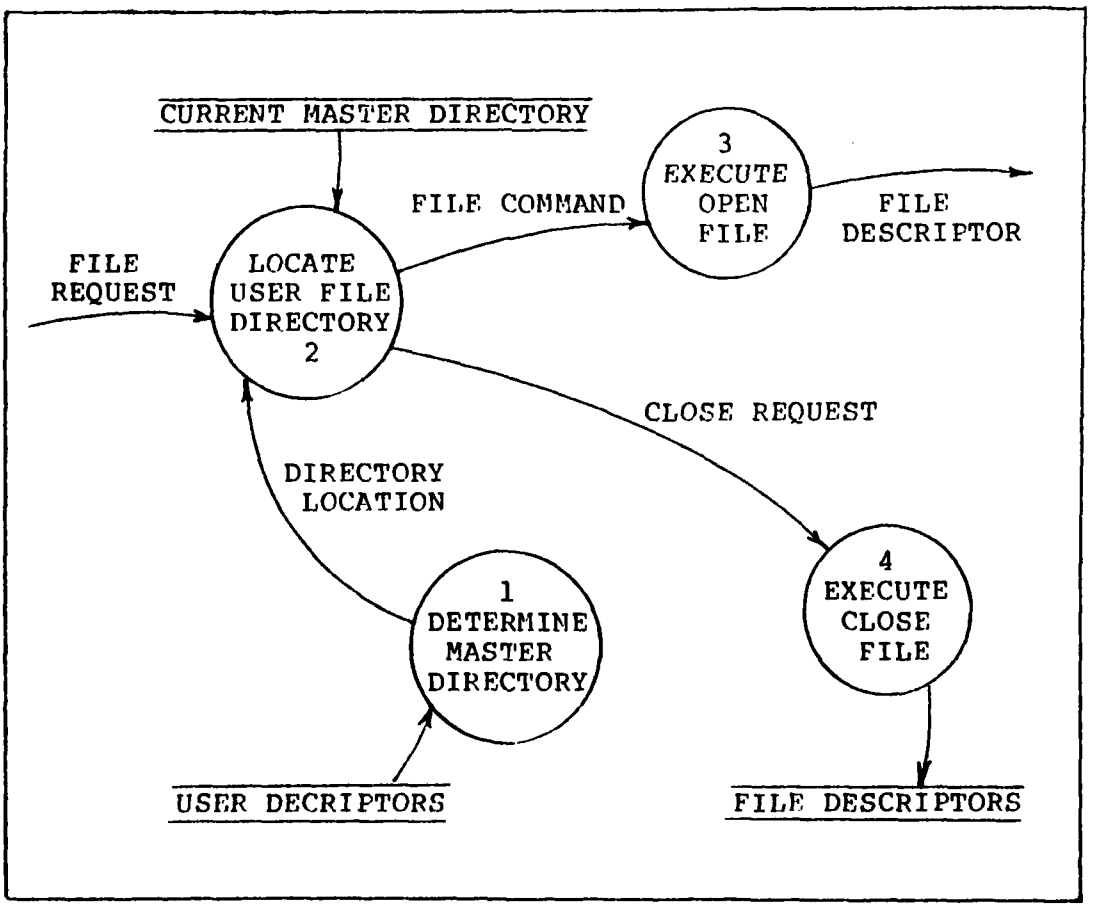


Figure 14. File Management Overview

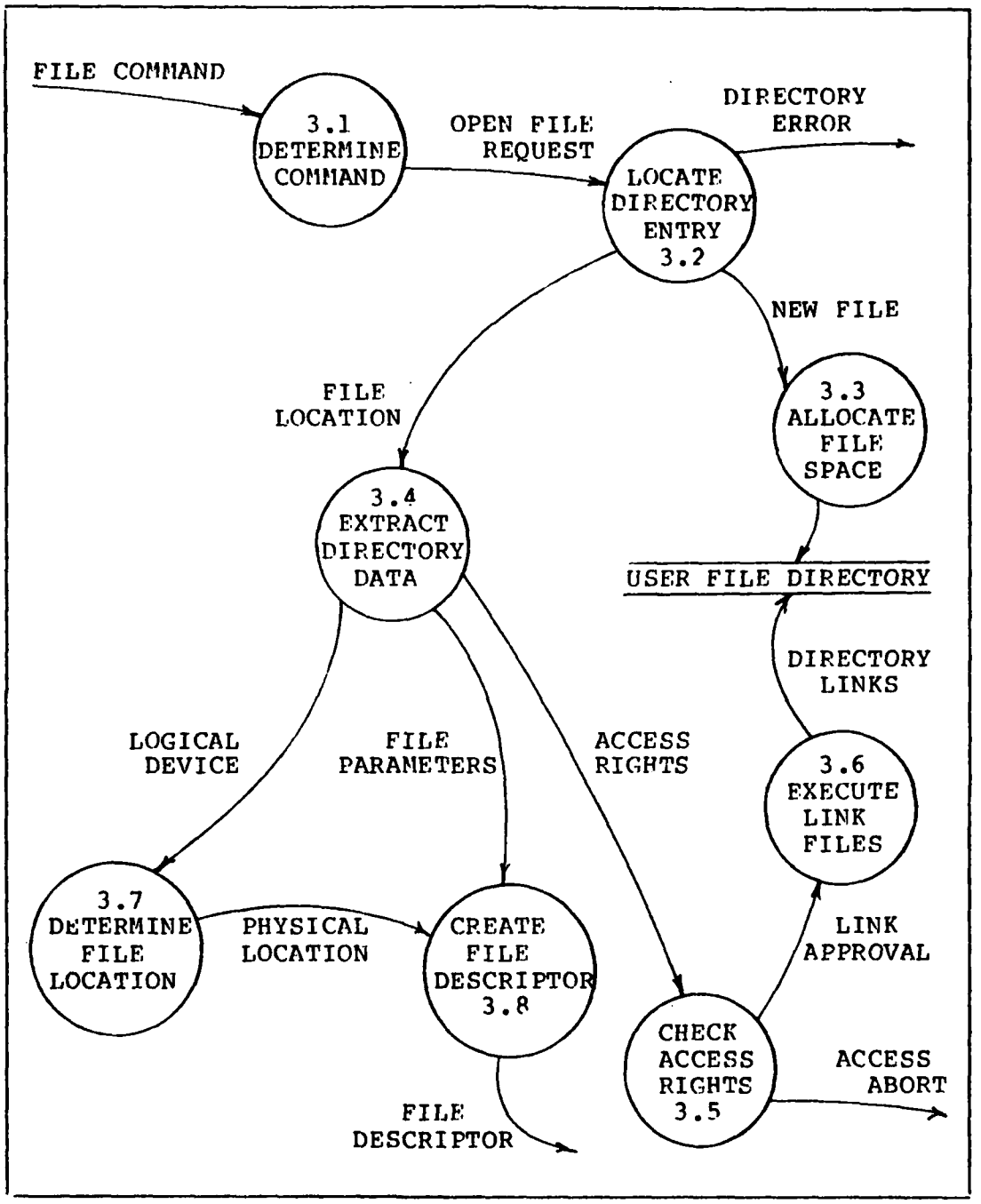


Figure 15. Execute Open File

requirement is to determine the mode of the operation (3.1) i.e. write or read. Next, the filename is located in the

directory (3.2) and the entry is inspected. If the directory entry is being created, new file space is allocated (3.3) and the necessary directory information is updated before the file is opened. Required information is read from the directory (3.4) such as location, access rights, and other file parameters. The access privileges are determined by comparing the requesting user's credentials with the recorded access rights in the file directory (3.5). Only the owner of the file can change the access rights. If they do not match the mode requested, the attempt to open the file will abort. If the request is to

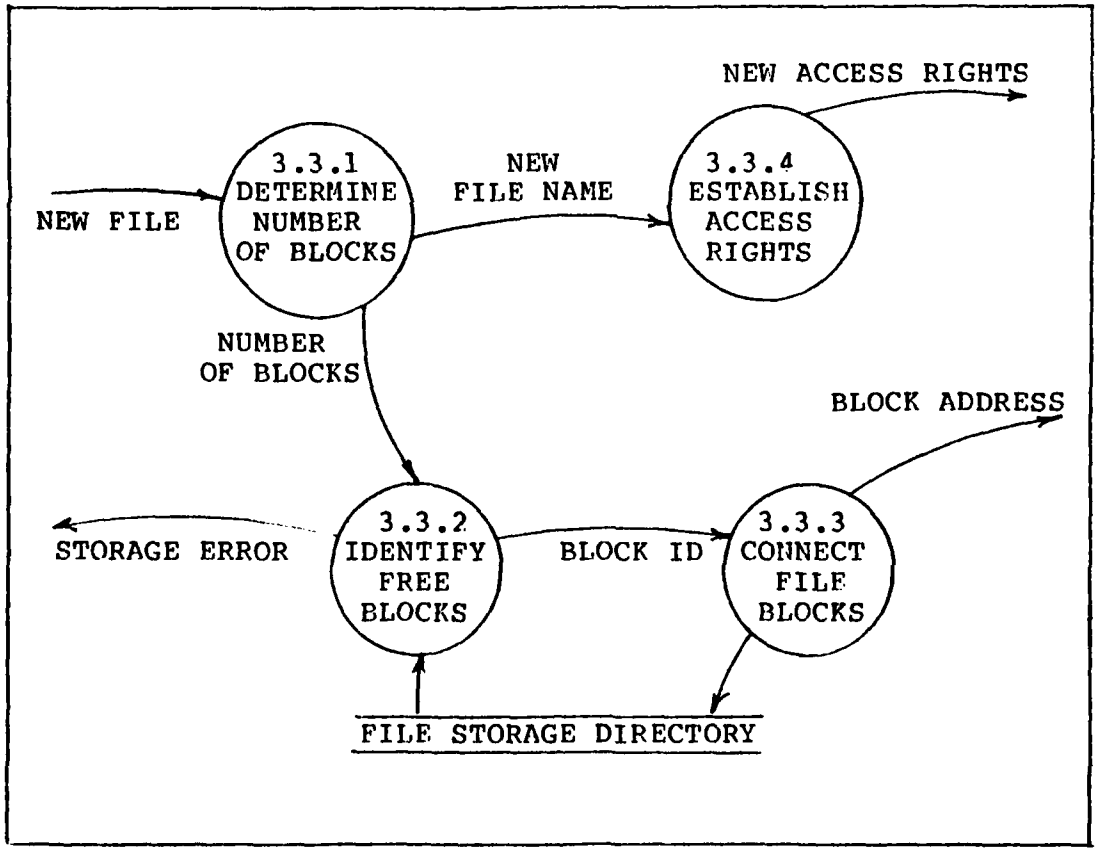


Figure 16. Allocate File Storage

link one user's file with another, (3.6) the links are created after access rights are established. The physical location and device must be determined (3.7) to initiate input/output. This information is passed to the procedure which creates the file descriptor (3.8).

Allocating file space requires the correct number of file blocks to be connected and associated with a specific filename. The number of blocks must be determined (3.3.1) and identified (3.3.2) in the file storage directory. Once identified as free, the blocks are connected (3.3.3) and associated with the specific file. If the file is new, the access rights must be established (3.3.4) by default or user

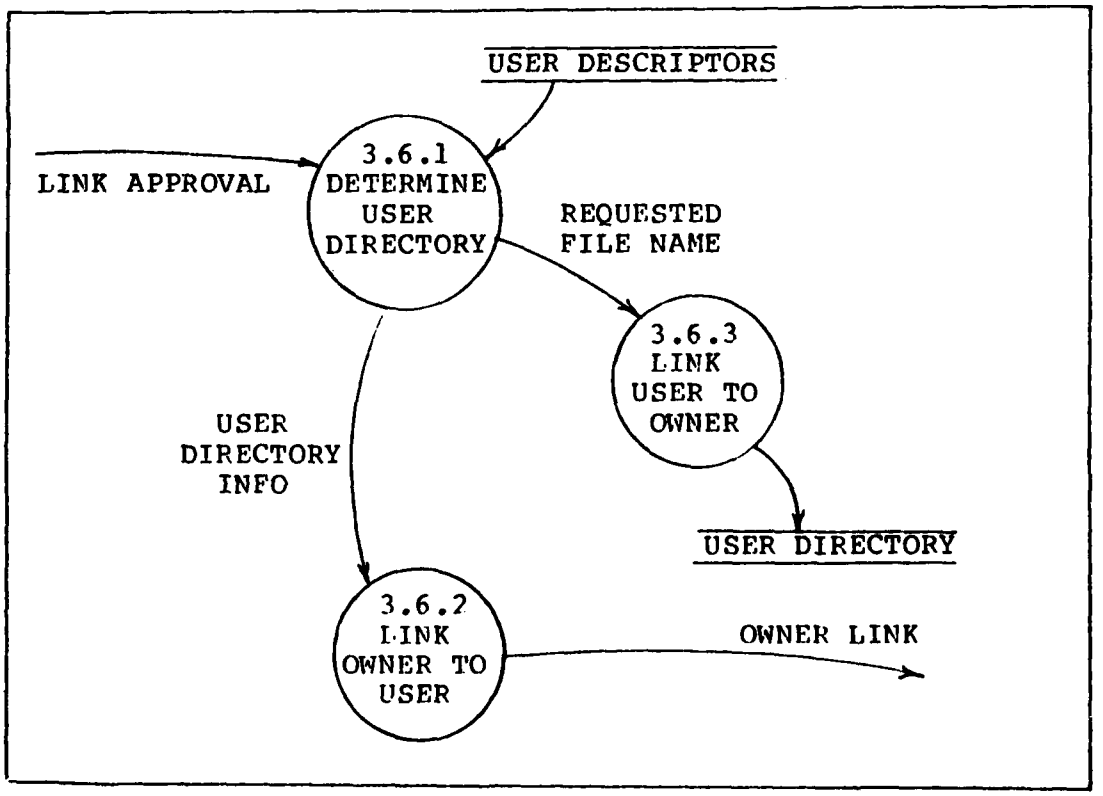


Figure 17. Execute Link Files

specification. The access rights are recorded in the user file directory.

Shared files must appear in both the owner's directory and the directory of the user sharing the file. Once access privileges are verified, the directory of the requester is determined (3.6.1) and a link is established (3.6.2). A link is also established in the opposite direction (3.6.3) for the owner to keep track of the users sharing the file. If the owner changes access rights or the file is deleted, the links must be destroyed. Only the owner can change the access rights to a file.

The file descriptor must be created for all

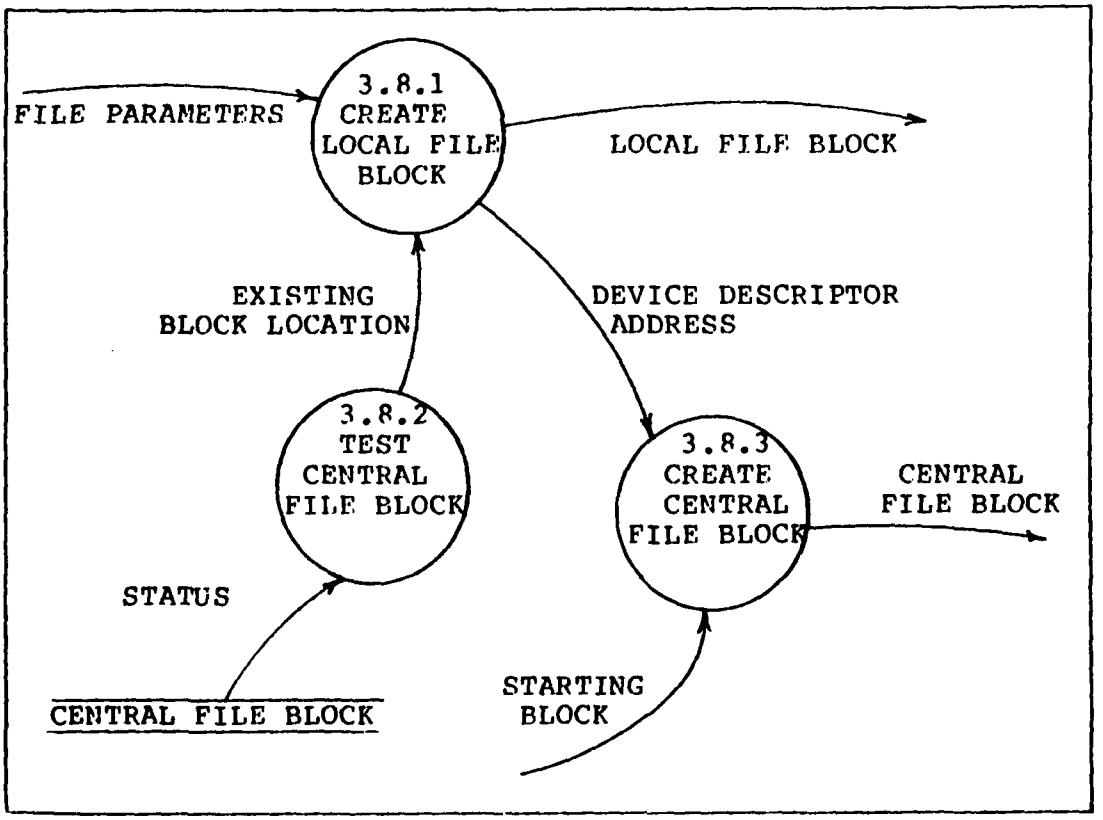


Figure 18. Create File Descriptor

input/output actions. The descriptor contains all the information about the file necessary for data transfers. The descriptor must consist of two structures. The local descriptor block is created (3.8.1) each time a process opens a file and all the local file descriptors for a particular file must be connected to a central descriptor which is associated with the appropriate device. A central file descriptor is created (3.8.3) for each file and is tested (3.8.2) each time a local file descriptor is created to determine if a file can be read or written to. If a central file descriptor is not available, a new one is

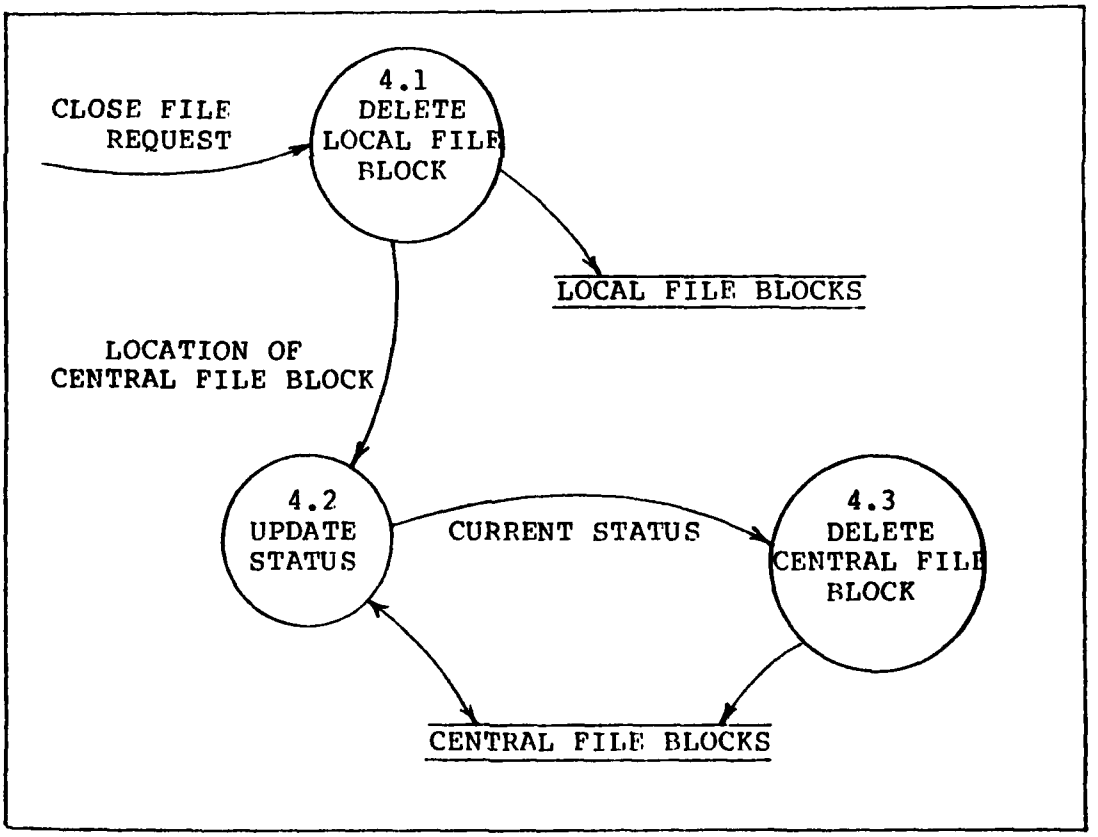


Figure 19. Close File

created. If one is available, its location is made known to the new local file descriptor. Two file descriptors are necessary to allow mutual exclusion to be applied to each data structure separately rather than an entire list of descriptors.

Closing a file is relatively simple. The local file descriptor is deleted (4.1) and the central file descriptor is notified (4.2). If no further processes require the central file descriptor, then it too is deleted (4.3).

Input/Output Management

The management of input/output devices is often difficult because of the variety of devices that must be handled. The characteristics and mode of operation of each device can differ drastically. The data may be transferred in different units such as blocks, records, bytes or words. The speed may also differ. A storage device may transfer a million characters a second and a printer may run no faster than 300 baud. The representation of data varies from one

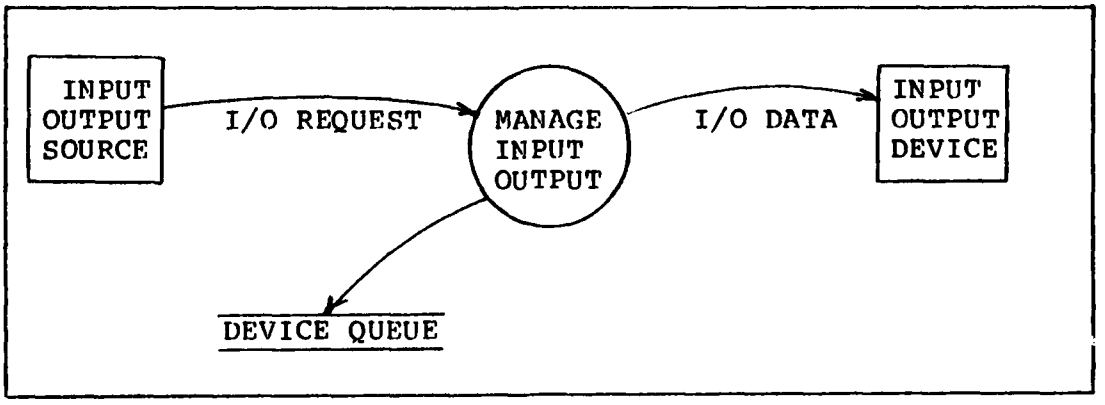


Figure 20. Input/Output Management Context Diagram

media to another. The type of operations also vary, such as outputing to a printer, inputing from a disk and rewinding a tape.

Ideally, the matter would be simplified considerably if all devices could be handled uniformly. Many operating systems achieve this by creating the abstraction of a virtual device. UCSD Pascal refers to input/output devices as "volumes" (Ref. 34) and MULTICS uses "files" (Ref. 33).

The input/output manager must support device and code independence for all devices. It should be unnecessary for a user to be required to know the character codes for the devices he wants to use. Further, the system should be able to accomplish input/output with any device and achieve the same result. It should not matter whether the output is transferred to a printer or a disk. It is desirable that input/output should be as device independent as possible.

Input/output devices often degrade performance by creating a bottleneck in the system. It is desirable to perform input/output as efficiently as possible and keep interrupt routines as short as possible.

The characteristics of each device should be associated with the device itself, not with the operating system input/output routines that manage them. The device routines can thus be treated similarly by the system. Special information needed for the operation of each device can be obtained from device tables or descriptors and may contain such information as the status of the device, translation

Table 5
Input/Output
(Figure 21)

1. Map Logical to Physical Device

2. Initiate Request (Figure 22)
   - 2.1 Assemble Request Block
   - 2.2 Notify Device Handler
   - 2.3 Add to Device Queue

3. Device Handler (Figure 23)
   - 3.1 Remove Request
   - 3.2 Delete from Queue
   - 3.3 Initiate Input/Output
   - 3.4 Notify Process

tables for the device, the current process using the device, or instructions to operate the device.

The input/output manager must map the device identifier to the physical device, check the validity of the parameters supplied to the device, and initiate the request for service. This must be a reentrant section to allow several processes to use it concurrently.

Mapping the device identifier to the physical device is accomplished by comparing the device parameter with a table of device descriptors (1). When the device has been identified, the parameters can be checked for consistency (4) with the information available in the device descriptor. The type of operation, rate of transfer, destination or origin must be checked. Initiating the request (2) requires assembling the parameters of the request into a data structure for the specific device. When the device handler (3) is notified of a pending request, it will check the service file.

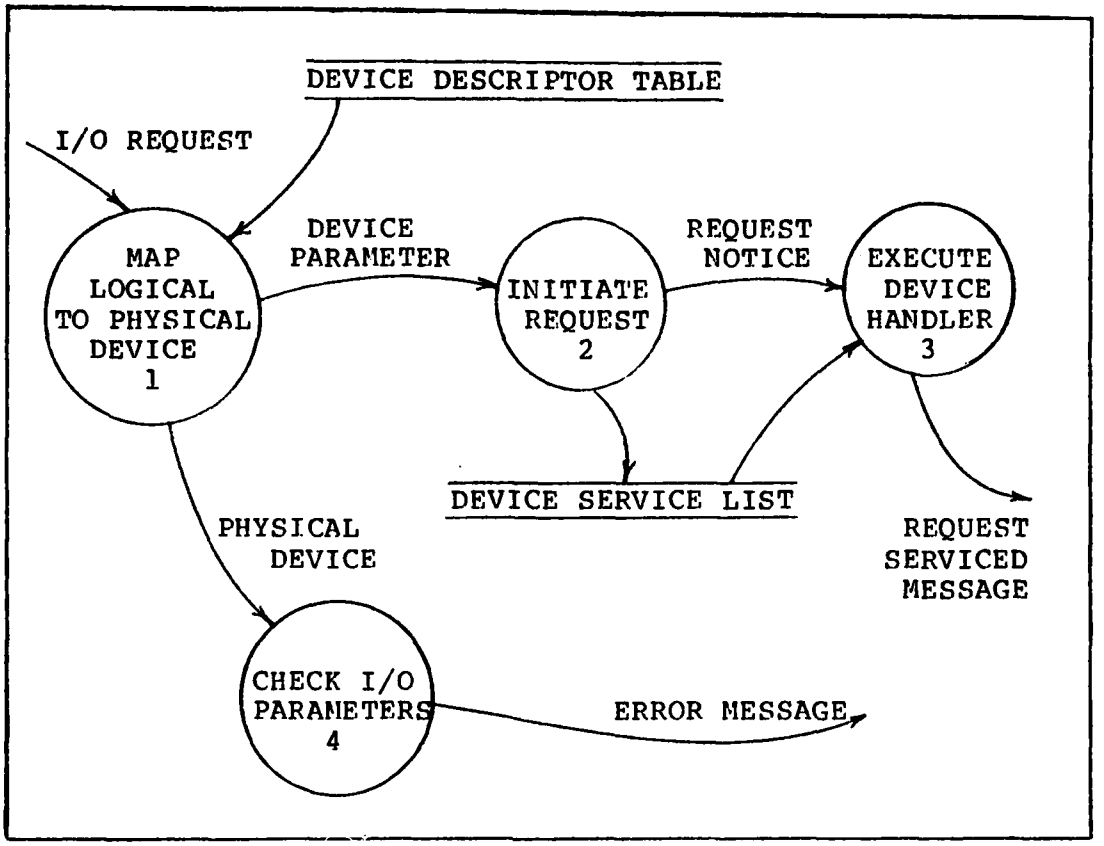


Figure 2: Input/Output Management Overview

Initiating the request consists of three processes. The request parameters must be formed into a block (2.1) to represent the information needed by the device handler. It is added to the service file (2.3) for the particular device which may have many such requests waiting to be serviced. The device handler is notified of each request (2.2). Each device handler may have many such requests waiting to be serviced.

Device Handlers

Each device must have its own device handler to deal with the peculiarities of each particular device. Also,

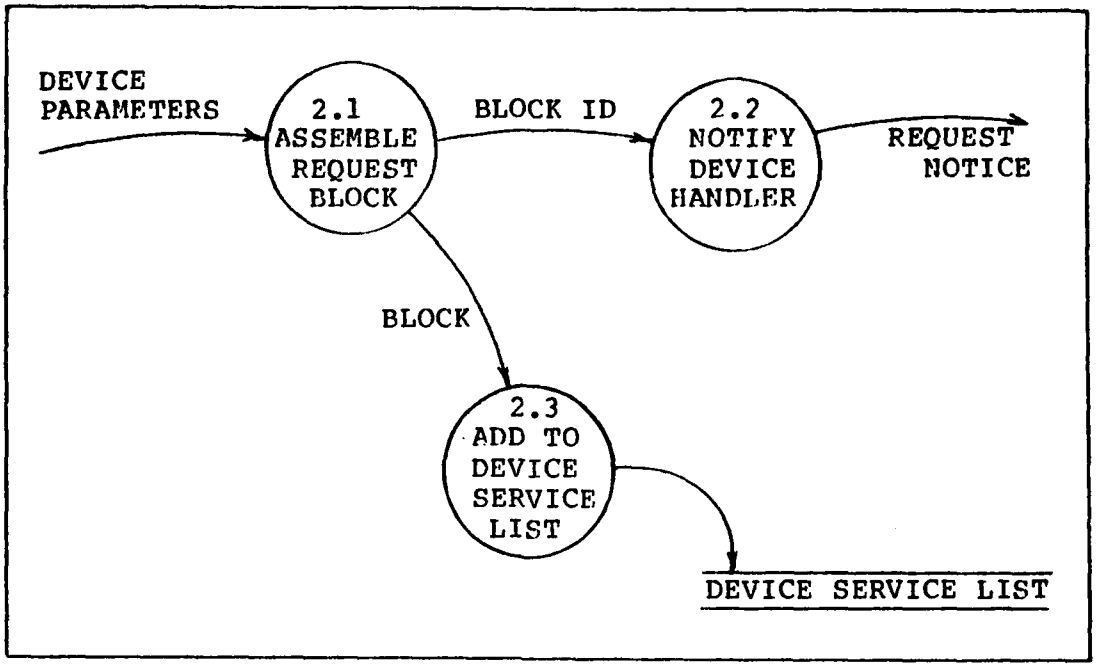


Figure 22. Initiate Input/Output Request

each device handler must have its own service file to maintain the blocks of information which represent service requests. The difference in devices is dealt with by the characteristics stored in each descriptor table for the particular device. Otherwise, the device handlers are similar and can share code segments.

Interprocess communication is required to notify the device handler when a process is waiting. This communication is required of the procedure that puts the request block in the corresponding device handler. Likewise, when the operation completes, the device handler is responsible for notifying the corresponding process that originated the request. All interprocess communication and interrupt handling is performed by the nucleus of the

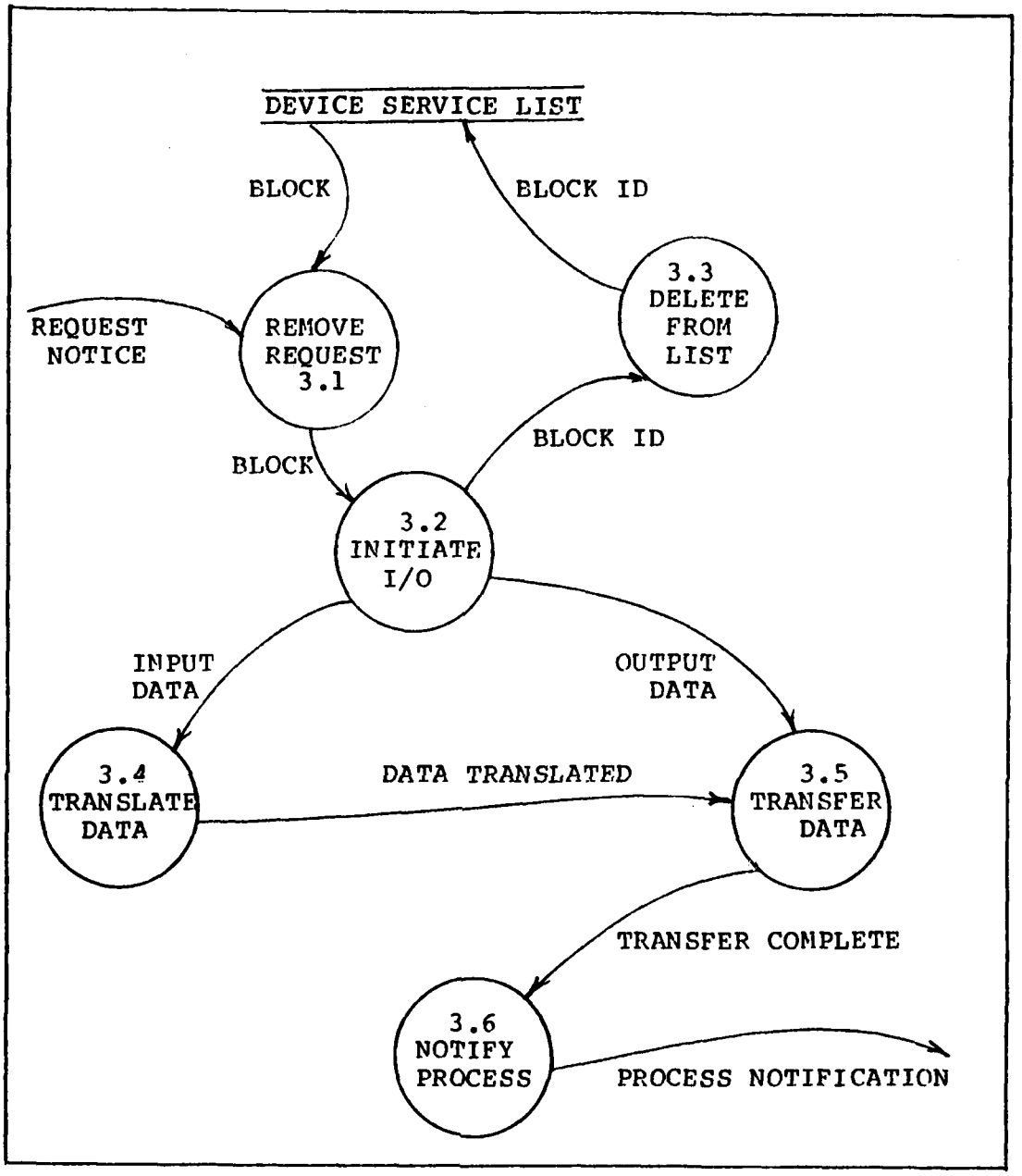


Figure 23. Execute Device Handler

operating system.

The device handler removes (3.1) a request block from the data structure maintaining requests and initiates (3.2) the corresponding input/output operation. This is the most

device dependent operation. The device descriptor is consulted for any information on the device or transfer charateristics. The data is translated (3.4), if necessary, and then transfered (3.5) as indicated by the request block. The request block is then deleted from the service file (3.3). When the operation completes, the device handler is responsible for notifying the corresponding process that originated the request (3.6).

Transfer of Files

The above requirements assume the use of sequential devices. If this is the case, the name or identifier of the peripheral device is sufficient information for a given transfer. However, if random access devices are to be able to transfer data, a facility must be provided to identify not only the device but also an area on the device to be transferred. Disks store information in areas ususally called files which have an associated unique name. Files are seldom located sequentially on the disk medium.

The transfer of files is similar to the transfer of other information with one exception. Additional data structures are required to store the information on the file, its location, and other needed information. This file descriptor is used in addition to the device descriptor when an input/output request is made. This will become part of the parameters which made up the request block.

Devices which support files are referred to as file devices and are identified differently than the other

input/output devices. The user requests the file rather than the device when desiring a transfer. This creates the file descriptor that is used for future tranfers. The file descriptor must include the device the file is stored on, the associated device descriptor, and the location of the file on the device.

The Scheduler

The scheduler has three primary duties. First, it must provide for the introduction of new processes. Second, the scheduler must assign priorities to determine their order of execution. The dispatcher selects the highest priority to run but has no input as to the priority. This minimizes the overhead of the dispatcher. Finally, the scheduler must implement the system allocation policies to maintain a system balance. No system resource should be over-committed or under-committed. The objective of the scheduler is to insure all processes obtain the system resources they need in a "reasonable" amount of time.

It is evident that the performance of the system is largely dependent on the scheduler and its policies. The scheduler should, therefore, have a high priority in the

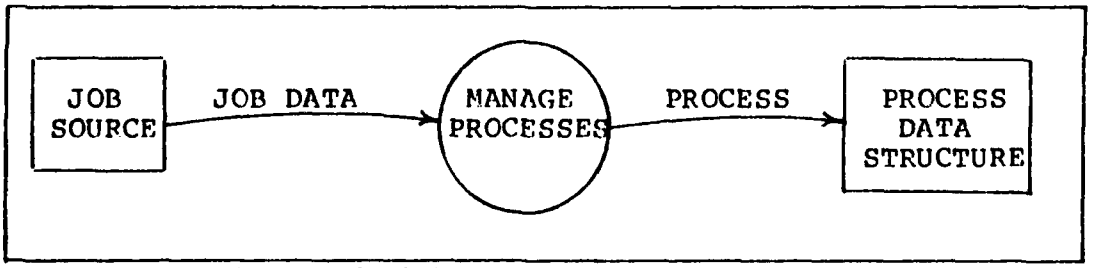


Figure 24. Schedule Management Context Diagram

Table 6

Scheduling Management
(Figure 25)

1. Create Process (Figure 26)
   - 1.1 Determine Process Image
   - 1.2 Initialize Process
   - 1.3 Assign Priority
   - 1.4 Schedule New Process

2. Execute Scheduler (Figure 27)
   - 2.1 Determine Process Status (Figure 28)
     - 2.1.1 Determine Process Status
     - 2.1.2 Change Status to Runnable
     - 2.1.3 Change Status to Unrunnable
     - 2.1.4 Change Status to Running
     - 2.1.5 Request Queue Action
   - 2.2 Determine Running Process (Figure 29)
     - 2.2.1 Determine Queue Priority
     - 2.2.2 Determine Process Status
     - 2.2.3 Determine Location
   - 2.3 Enter Process Queues (Figure 30)
     - 2.3.1 Select Queue Action
     - 2.3.2 Delete from Queue
     - 2.3.3 Determine Queue
     - 2.3.4 Add to Wait Queue
     - 2.3.5 Add to Ready Queue

3. Execute Memory Swap (Figure 31)
   - 3.1 Determine Memory Available
   - 3.2 Execute Swap-In
   - 3.3 Execute Swap-Out
   - 3.4 Initiate Swap I/O

system compared to other processes. The dispatcher will then choose the scheduler whenever it is runnable. The scheduler is runnable on at least four occasions. When a new process arrives, when a process terminates, when a process is blocked or unblocked, and when a process is preempted, the scheduler should become active.

Notice the similarities between the interrupt handler and the activities of the scheduler. Each will occur at intervals unpredictable to the system and likely cause

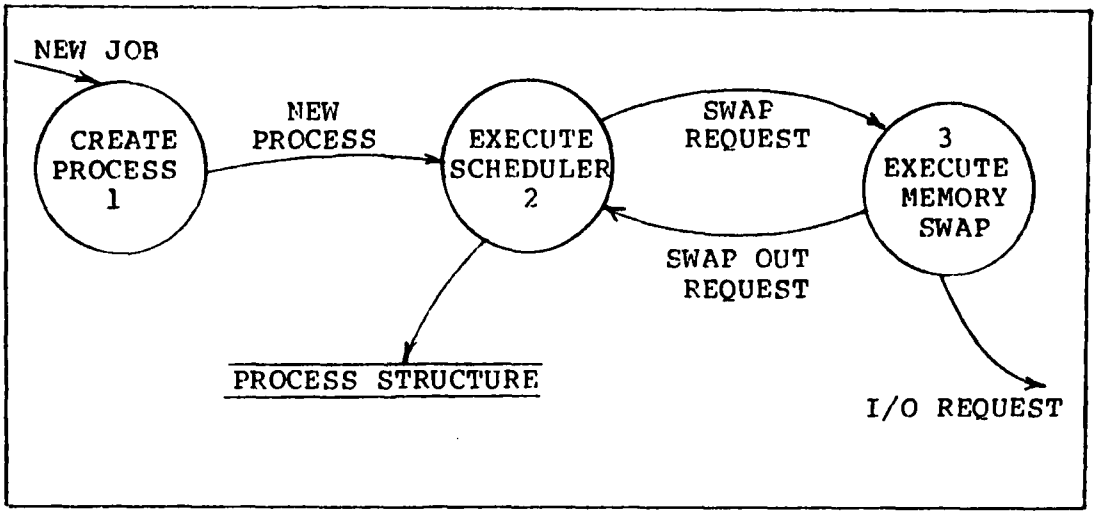


Figure 25. Schedule Management Overview

modifications to system behavior. The differences lie in the level of each service. The interrupt handler is a low level action affecting the status of processes and allocation of the processor. The scheduler is a high level action affecting the number of processes and their priority. While interrupts may occur every few milliseconds, a call to the scheduler may be made only every few seconds.

The three major operations of the scheduler are shown in figure 25. New processes are created by the scheduler (1) as they arrive. An attempt is made to enter the new process into memory and it is passed to the scheduling procedure to enter the processor queue. Each process must be prioritized to achieve the desirable system performance. This requires selecting the most important process (2) in the system that is runnable or can be made runnable by the scheduler. Processes may be swapped (3) into and out of

memory by the scheduler as demands change and memory becomes available.

Creating the process consists of establishing a process descriptor to contain the environment in which the process exists. The image of the process (1.1) contains identification of the process and its originator. If the process is new, all data associated with its running environment must be initialized (1.2). Priorities are assigned (1.3) based on some criteria established to optimize system response. At this point, the priorities are based on limited data about the process and may be updated by the scheduler at a later time.

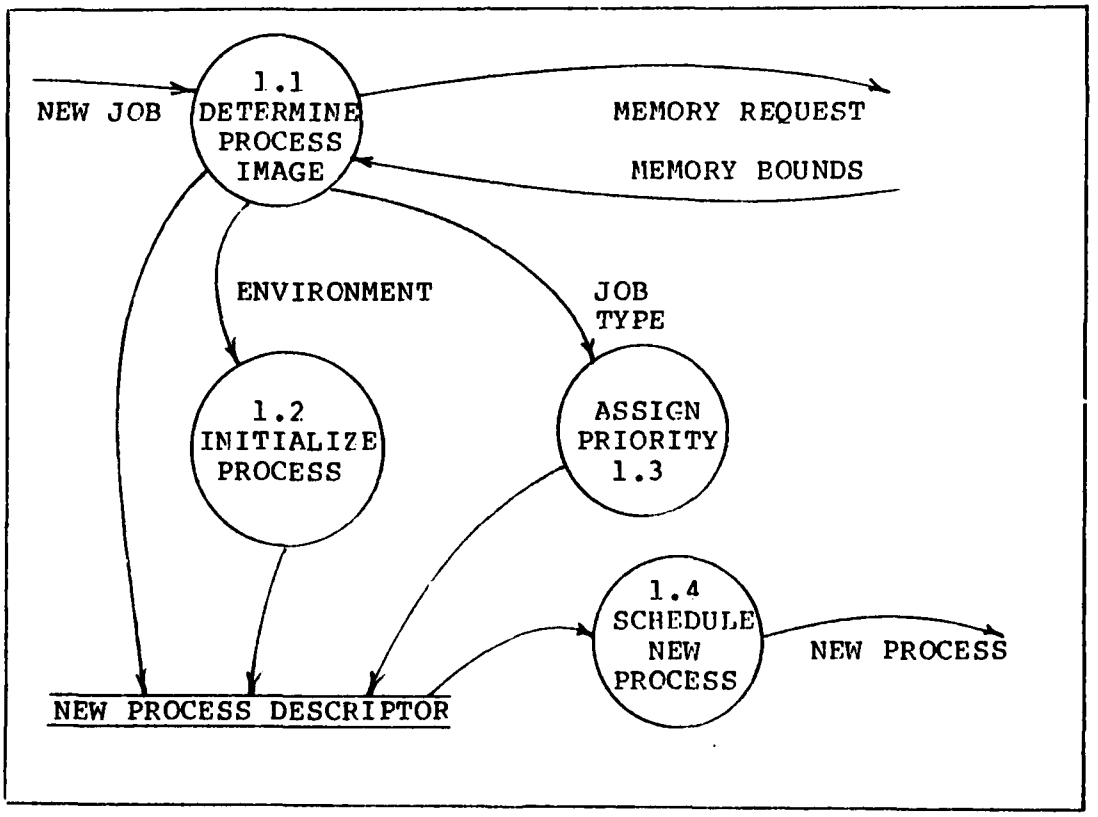


Figure 26. Create Process

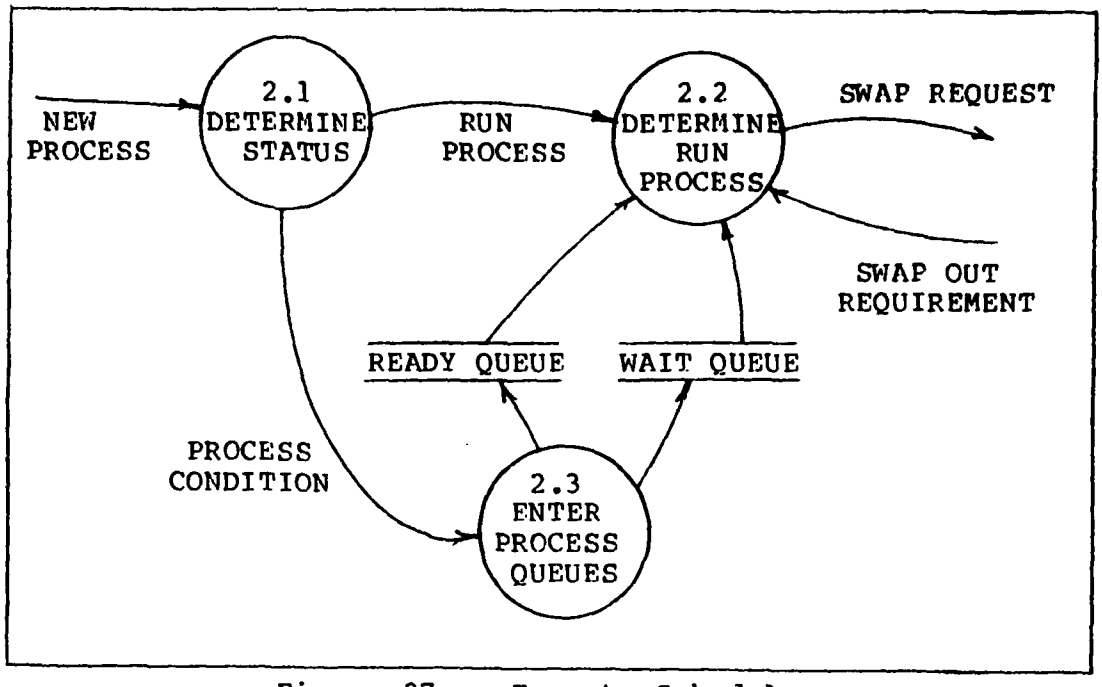


Figure 27. Execute Scheduler

The *scheduling procedure* has three primary requirements. The scheduler must determine the status of processes (2.1) and update the status when a process changes position in the processor data structure. A running process must be selected (2.2) from among the available processes in the processor data structures. This may involve moving processes around in the system to get the most important process running. Finally, the processor data structures must be manipulated to reflect the status of all processes in the system (2.3). This involves ordering the processor queues, determining which process to delete, and which to add in each queue.

Determining the status of processes requires the scheduler to be activated when a process is blocked,

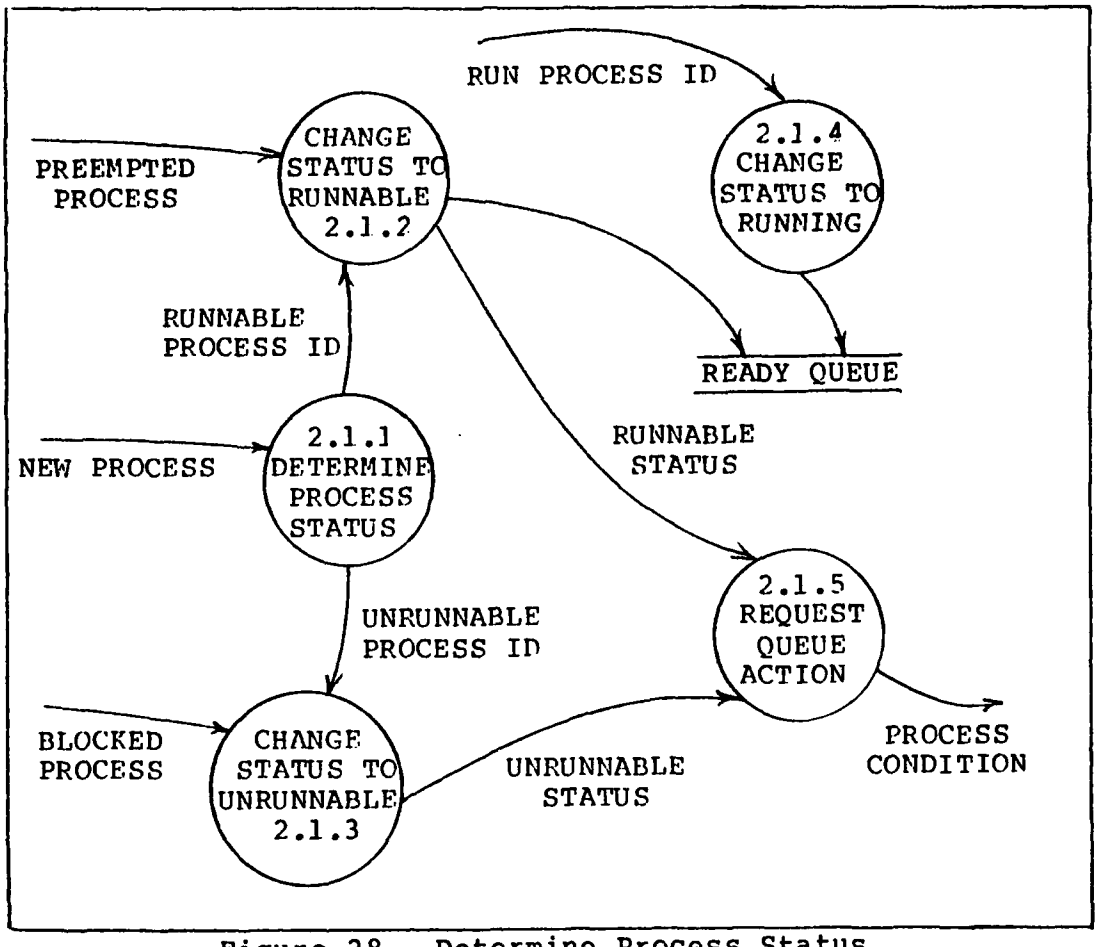


Figure 28. Determine Process Status

unblocked, terminated, preempted, or a new process enters the system. Process condition must checked to determine if it is currently in memory (2.1.1). Processes in memory or preempted processes are changed to a runnable status (2.1.2). Blocked, terminated or other processes not in memory are changed to an unrunnalbe status (2.1.3). The process most eligible to run is changed to running status (2.1.4) in the ready queue where the dispatcher can allocate it to the processor. All process changes that require a move in the processor data structure (2.1.5) are

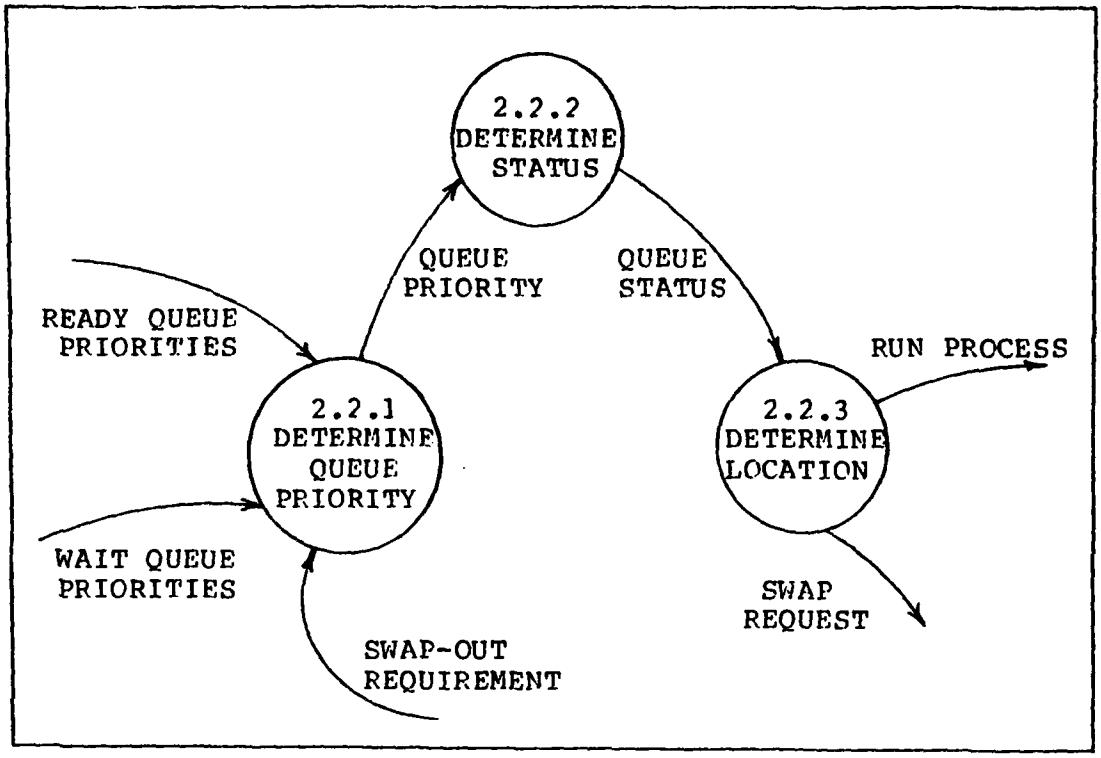


Figure 29. Determine Running Process

determined by the status conditions of the respective process.

Determinig the running process is accomplished by inspecting the processor queues for the highest priority. Both runnable and unrunnable processes are checked for priority (2.2.1). The highest priority process is inspected for status (2.2.2) to determine if it is runnable or can be made runnable. If runnable, the process is selected to be the running process and if a swap is necessary the swap-in is requested (2.2.3).

Entering the processor queues is based on the priority and status of the processes. A determination must be made

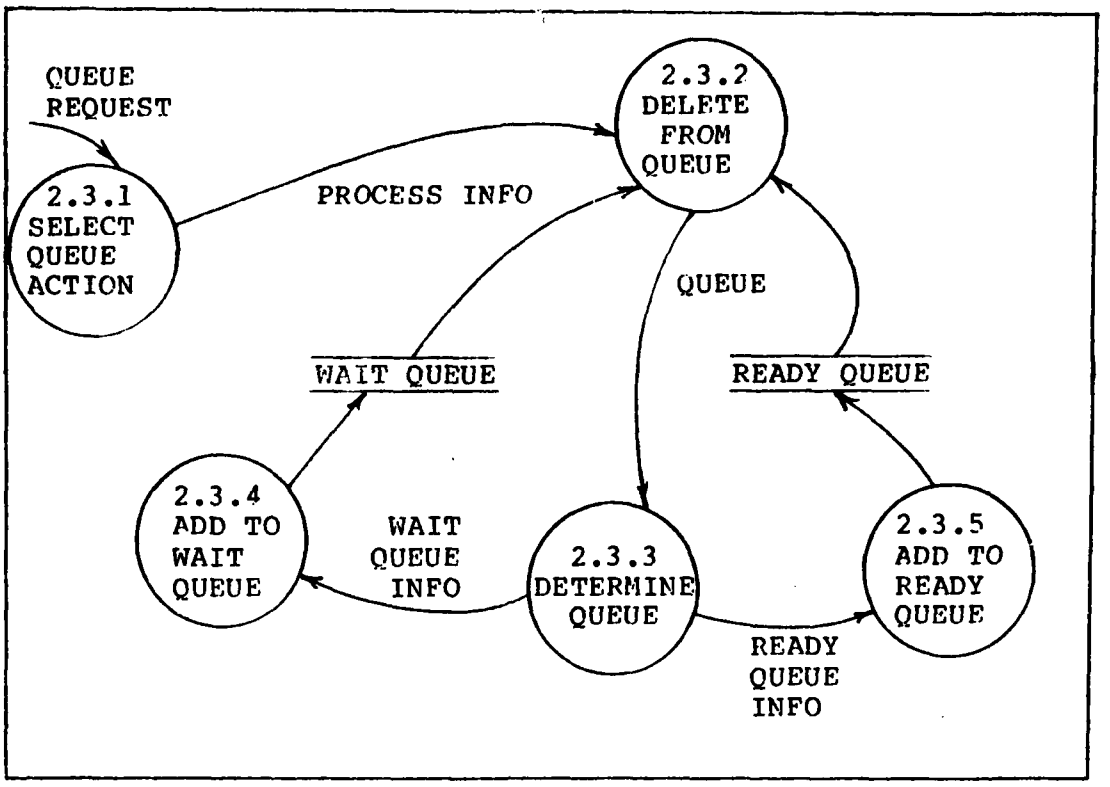


Figure 30. Enter Processor Queues

what action is required on the process and what queue is affected. (2.3.1). Processes moving from one queue structure to another must first be deleted (2.3.2) from the respective queue and a determination made what queue the process is to enter (2.3.3) based on status. Priorities for the wait queue for unrunnable processes (2.3.4) may vary from the processes entering (2.3.5) the ready queue for runnable processes.

The major data associated with a process is moved to and from main memory as needed. Since main memory is limited, and a process cannot execute unless it is in main memory, the scheduler must decide when to swap a process

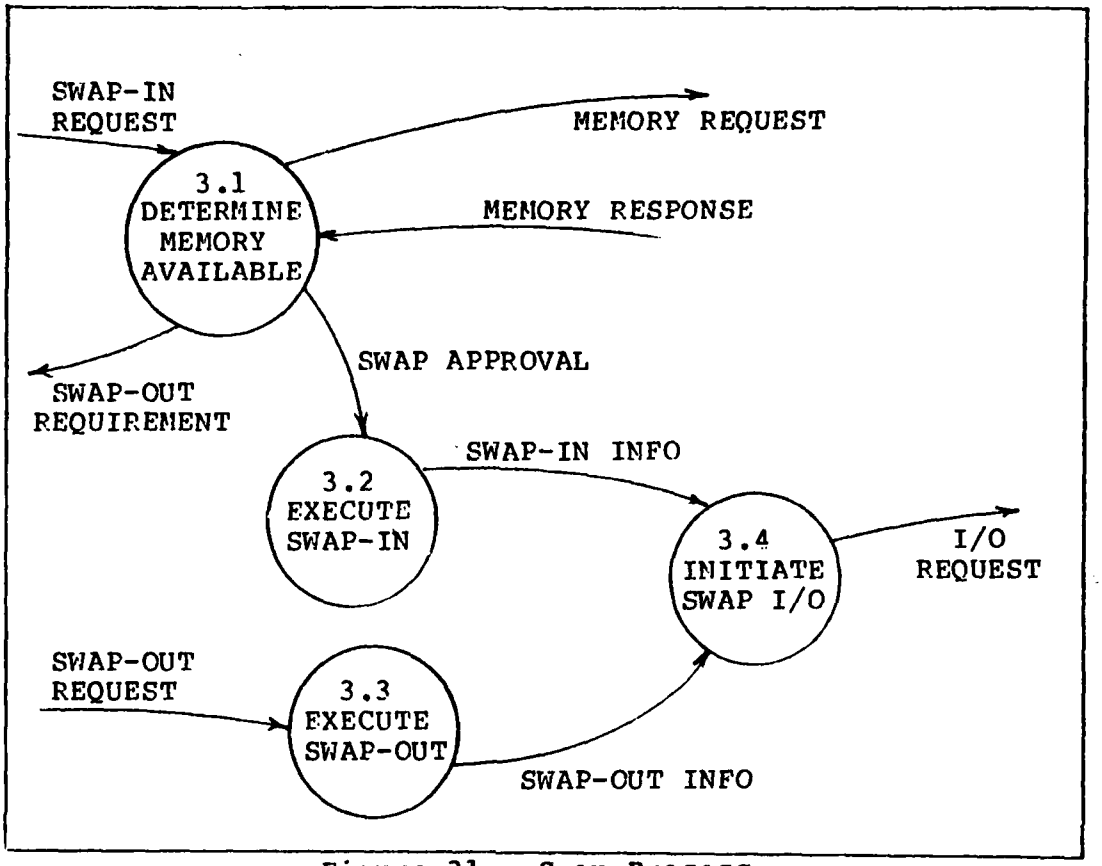


Figure 31. Swap Process

between secondary storage and main memory. When the scheduler selects a process to run which is not in memory it requests a swap-in. The swapping procedure determines the amount of memory needed to swap-in the process (3.1). A request is made to the memory manager. If enough memory is availble the swap-in is initiated (3.2). If main memory is full, the swap procedure informs the scheduler. The scheduler must decide what process to swap out to make room and informs the swap procedure to initiate a swap-out (3.3). Swapping is accomplished by activating an input/output request (3.4) with the associated information.

Memory Management

For mutiprogramming, it is necessary for more than one process to have access to the processor. This requires several independent programs to reside in main memory simultaneously, so the processor can be switched between them. The programs must not interfer with each other, must be removed when terminated, and new programs must be given access to available memory when required.

In a multiprogrammed computer it is impossible for the user to know where each process is in memory. This means the exact location of the user process is unknown, it cannot be written in terms of absolute memory addresses. If the memory allocated to a process remained fixed during its entire execution it would be possible to transform symbolic or relative addresses into absolute addresses at the time the program was loaded, but this is seldom the case. As processes run to termination the memory they use becomes free for other processes. They exist in a dynamic system. The system must be responsible for transforming the addresses used by the user into the actual addresses in

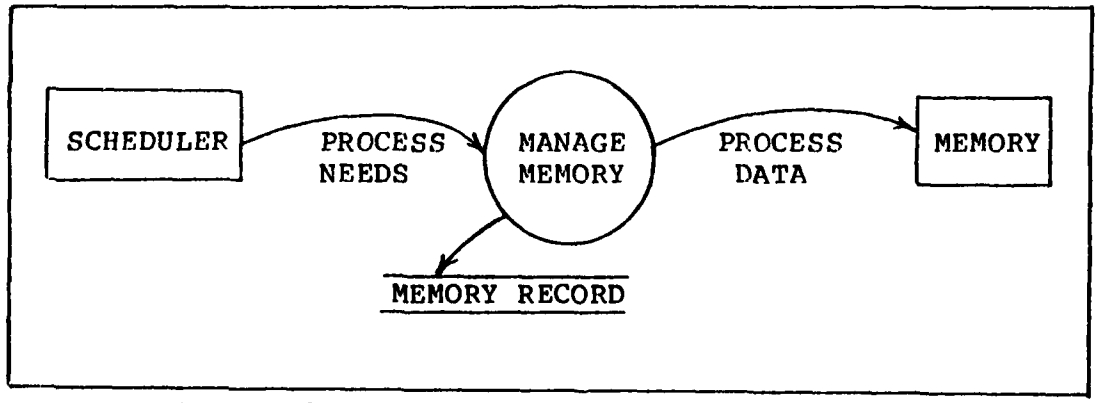


Figure 32. Memory Management ContextDiagram

Table 7

Memory Management
(Figure 33)

1. Determine Need

2. Select Area (Figure 34)
   2.1 Examine Free Area
   2.2 Compare Area with Size
   2.3 Update Free Area Table

3. Assign Area

4. Record Bounds of Process

5. Deallocate Memory (Figure 35)
   5.1 Match Entry with Process
   5.2 Update Memory Table
   5.3 Adjust Free Space Table

which the process is physically located.

Memory management involves the administering of primary memory to processes and data where it can be accessed by the processor. This generally requires four actions by the memory manager: the allocation of memory, record keeping involved with how the memory is allocated, a protection policy to prevent processes from interfering, and the deallocation of memory after the process terminates (Ref. 28: 105).

After the memory need is determined, an area is selected for the process in memory (2). Once an area is selected the process must be assigned that area (3) by recording the process, memory size, locations and bounds in a memory table. The process must also be aware of its own memory bounds by writing to the process descriptor of the requesting process (4). When the process terminates the memory used must be deallocated (5) and the memory table

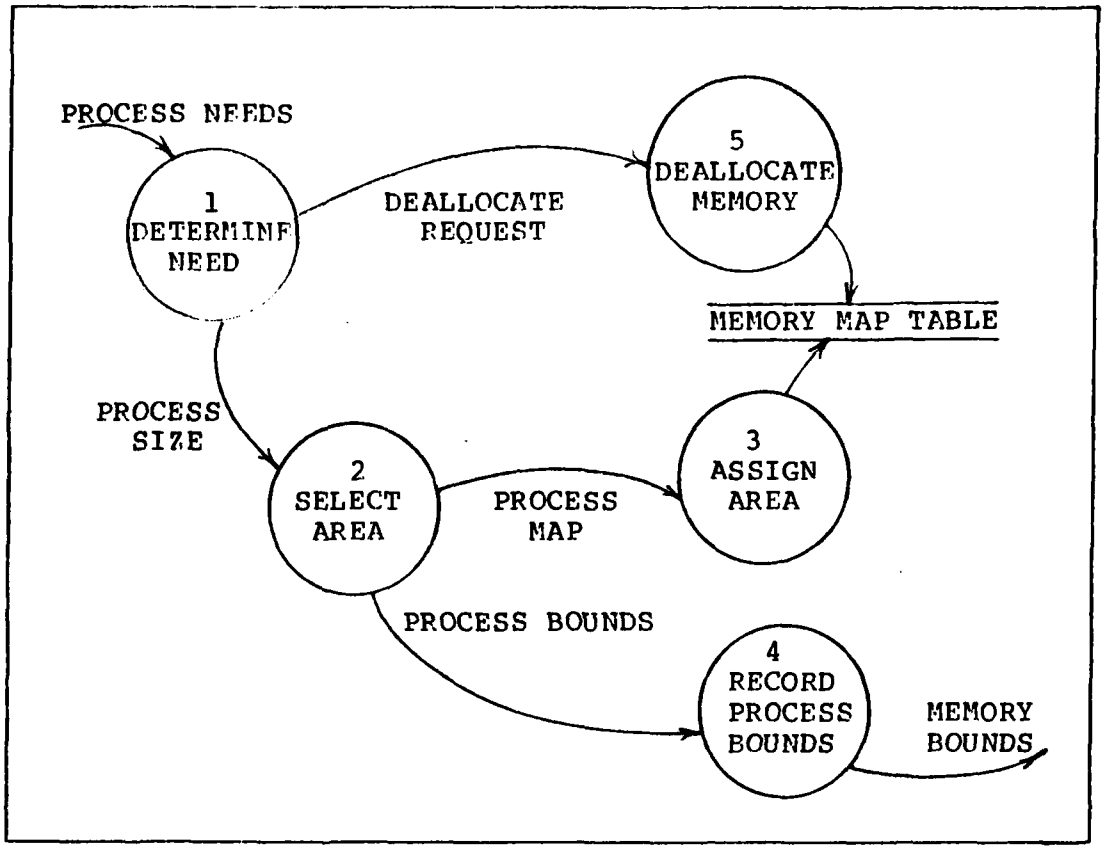


Figure 33. Memory Management

is updated.

Main memory is partitioned into blocks of memory. These blocks may be a fixed size or a variable size. Block size can be static and determined by the system or dynamic and vary with the size of the need. These are issues determined in design. The primary difference between these methods is that a program entering memory may be in a contiguous memory space or may be divided among several different noncontiguous blocks. However, in any case, a list of free memory space must be maintained. When a process is requesting memory, the free space is examined

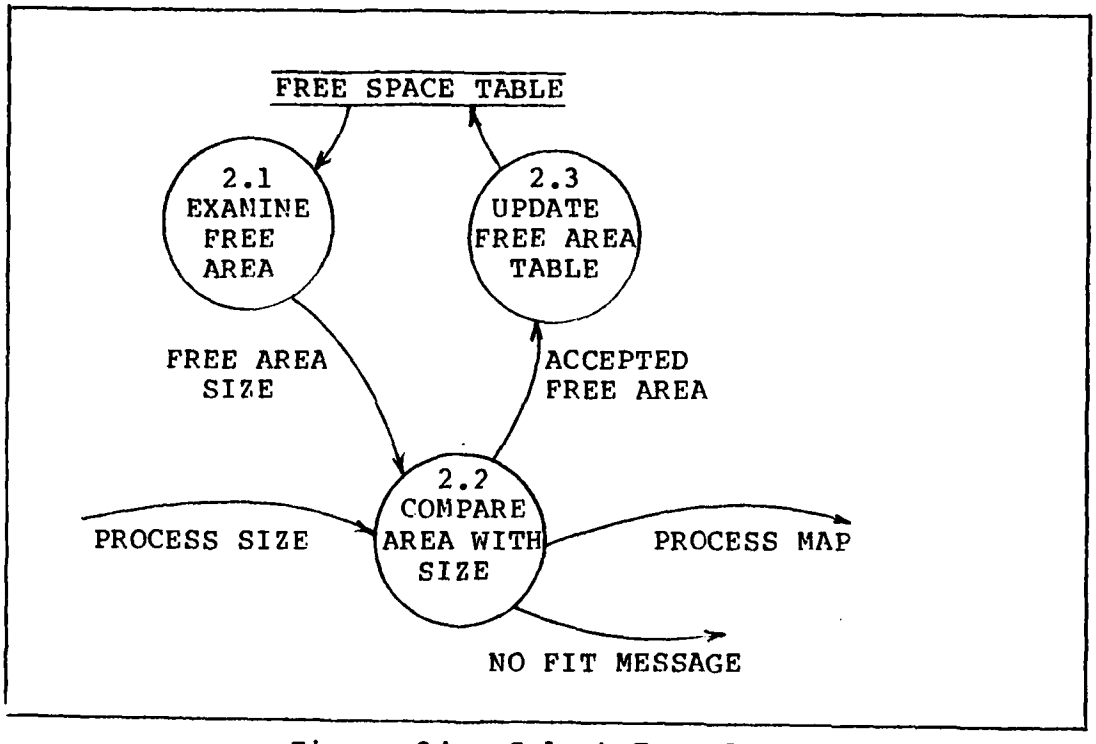


Figure 34. Select Free Area

(2.1) for a large enough space, either contiguous or noncontiguous, depending on the algorithm. Each free space is compared with the size needed (2.2). If a large enough area cannot be found, a response from the system is required. If a free area is identified, the free space must be updated (2.3) before the actual assignment is made to the memory table.

Deallocation simply involves matching a process with an entry in the memory table (5.1) and updating the corresponding entry (5.2). The area deallocated is added to the free area (5.3).

A great deal of overhead can be involved in the allocation of memory. Usually, the greater the memory

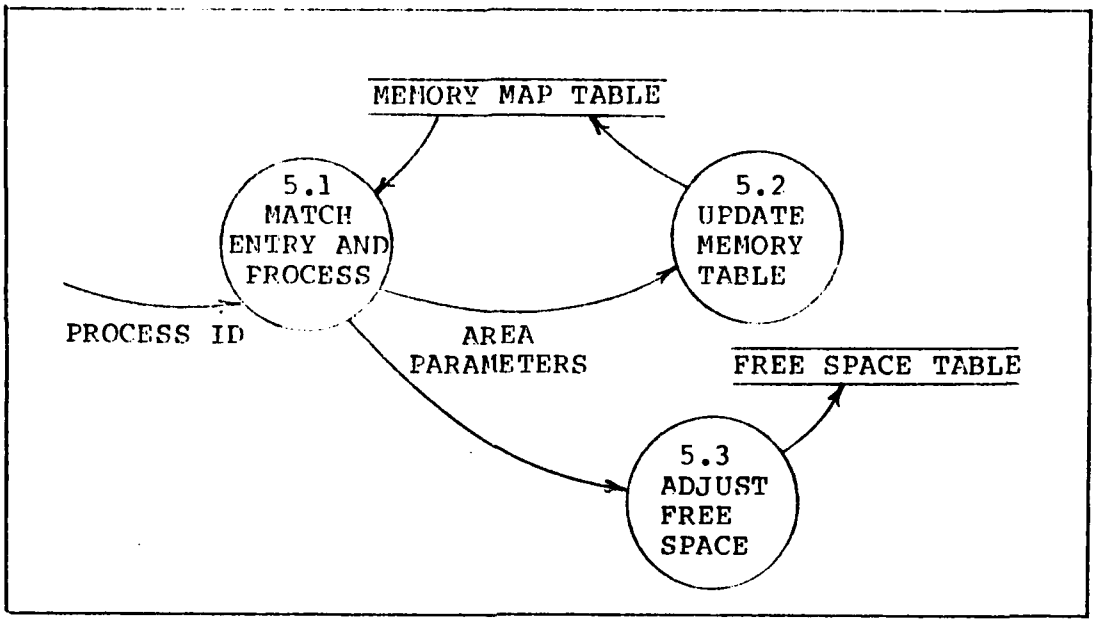


Figure 35. Deallocation Memory Space

utilization, the greater the overhead. The price paid for simplicity is that some memory space will be unused. Fragmentation is the condition resulting when leftover sections of memory are too small to fill allocation requests (Ref. 51: 69). There are many algorithms for dealing with fragmentation and memory management. Madnick (Ref. 28) presents one of the best comparisions between different schemes.

Nucleus Requirements

The facilities exist at the center of the system which interfaces directly with the hardware itself. This is the most machine dependent part of the system and is the major interface between the operating system and the machine. This inner-most layer is commonly referred to as the

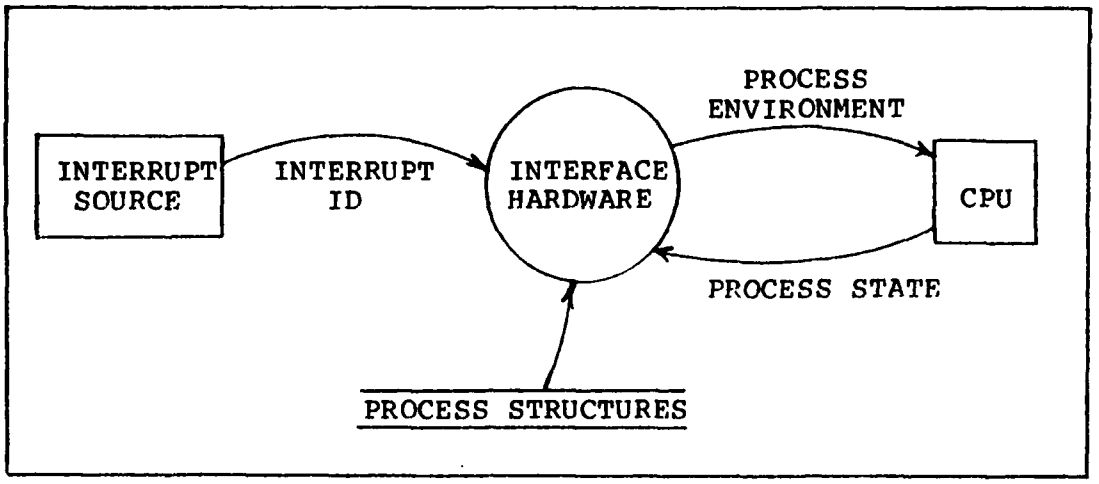


Figure 36. Nucleus Context Diagram

Table 8

Nucleus Composition
(Figure 37)

1. Dispatch Process (Figure 38)
   - 1.1 Test for Current Process
   - 1.2 Update Processor State
   - 1.3 Record Processor State

2. Interprocess Communication (Figure 39 and 40)
   - 2.1 Lock CPU
   - 2.2 Block Process
   - 2.3 Awaken Process
   - 2.4 Unlock CPU

3. Execute Interrupt (Figure 41 and 42)
   - 3.1 Save CPU State
   - 3.2 Identify Interrupt Source
   - 3.3 Determine Priority
   - 3.4 Disable Lower Priorities
   - 3.5 Determine Routine Location
   - 3.6 Service Interrupt
   - 3.7 Restore CPU State

"nucleus". The functions of a nucleus vary among designs. Because it is so hardware dependent, no clear definition exists for the explicit functions of a nucleus. For this effort, the nucleus will be required to handle interrupts, provide the mechanisms for interprocess

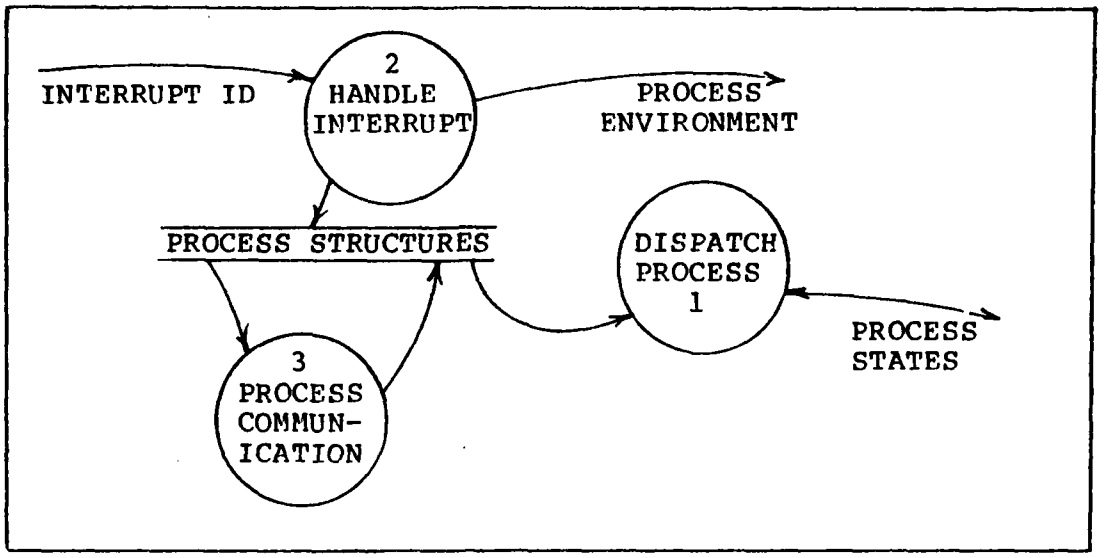


Figure 37. Nucleus Overview Diagram

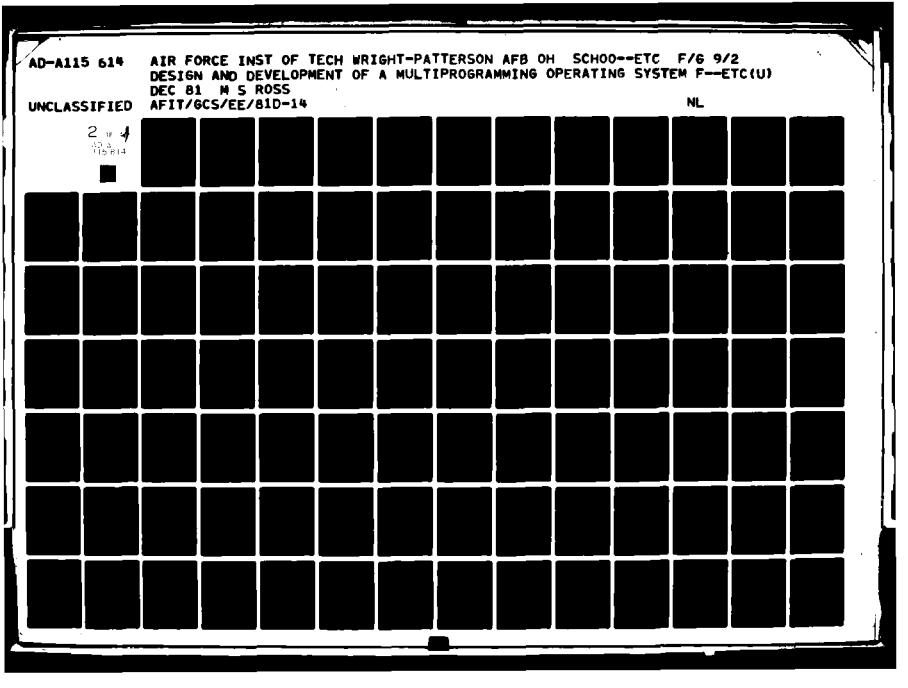
communications, and switch the processor between processes.

Dispatcher Requirements

The requirements of the dispatcher are to allocate the processor to the most eligible process. It is called whenever the currently executing process cannot continue to run or a change to the system has indicated the processor should run another process. This may be caused by a status change of the running process or an error condition. Actually, these are all interrupt conditions, thus, the dispatcher must be entered after all interrupts to determine the best process to allocate the processor to. Granted, the dispatcher may not need to be consulted after each interrupt, but in this manner each interrupt is treated uniformly and the overhead is justified.

The requirements of the dispatcher are relatively simple. It tests the current process on the processor to

AD-A115 614 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
DESIGN AND DEVELOPMENT OF A MULTIPROGRAMMING OPERATING SYSTEM F--ETC(U)
DEC 81 M S ROSS
UNCLASSIFIED AFIT/GCS/EE/81D-14 NL



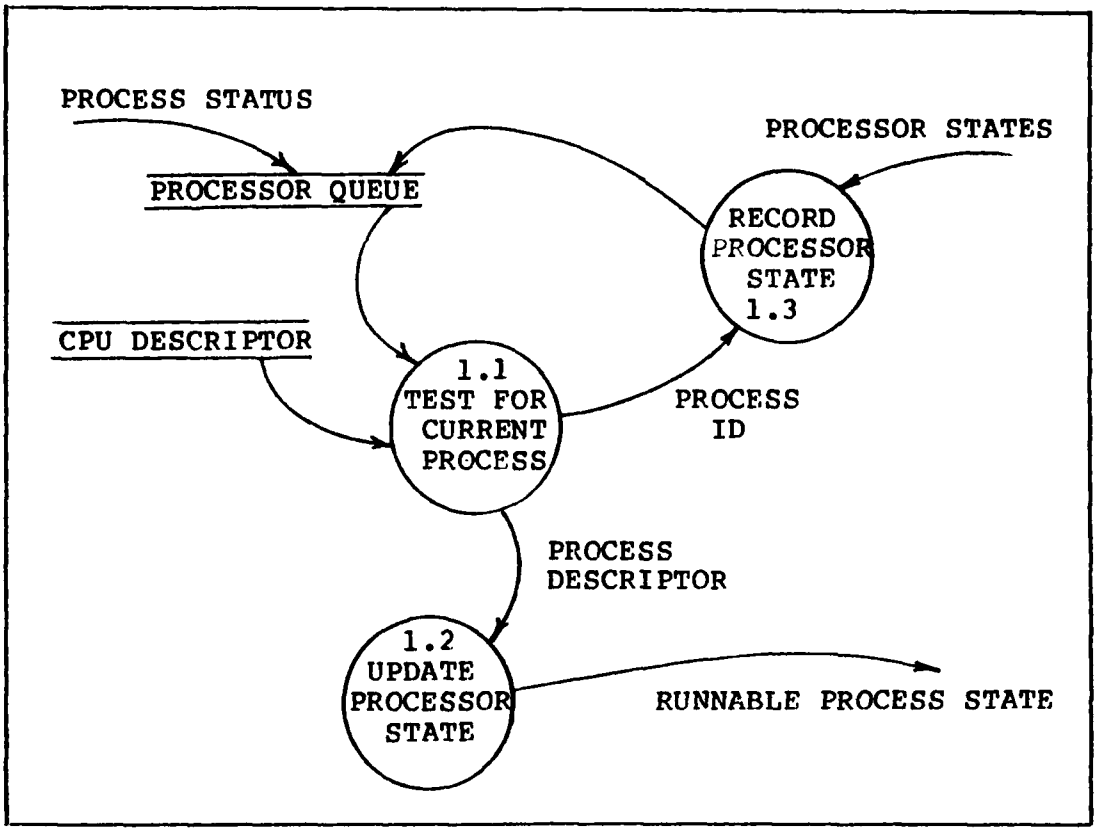


Figure 38. Dispatch Process

see if it is the highest priority available (1.1). If so, the dispatcher will return control to the system as stored before the interrupt was serviced (1.2). If not, the dispatcher will save the current system state associated with the current process (1.3) and retrieve the system state of the higher priority process. Next, control is transferred to accomodate the new running process (1.2).

The dispatcher does not determine priorities. It assumes each process has an assigned priority associated with it which was assigned by the scheduler at a higher level. The dispatcher must have access to the data structure the processes are stored in. If the processes are

stored in a queue fashion and ordered by priority, then the dispatcher is able to simply select the process from the front of the queue. This may or may not be the current running process.

Since an interrupt may effect the status of a process in the system, it is a requirement for each interrupt routine to update the status of the process. The routine must also place it in the processor queue according to the proper priority.

Interprocess Communication

Because a number of processes are operating concurrently in the system, a requirement exists for some means of communication between processes. This mechanism must be in the nucleus because all processes must have access to it. Also, in order for the mechanism to block or awaken a process it must have immediate access to the dispatcher and interrupt handler. It is necessary to block and awaken processes (2.2 and 2.3) for three reasons: deadlock avoidance, critical sections, and syncronization.

Deadlock is a condition in which processes are waiting indefinitely for events that will never occur (Ref. 16: 336). A deadlock condition involves the circular waiting of processes. Each process is waiting on the resource of another process. Since they are all waiting on another to resolve the condition, they are all unable to continue. Hansen (Ref. 16) gives a complete discussion of deadlock

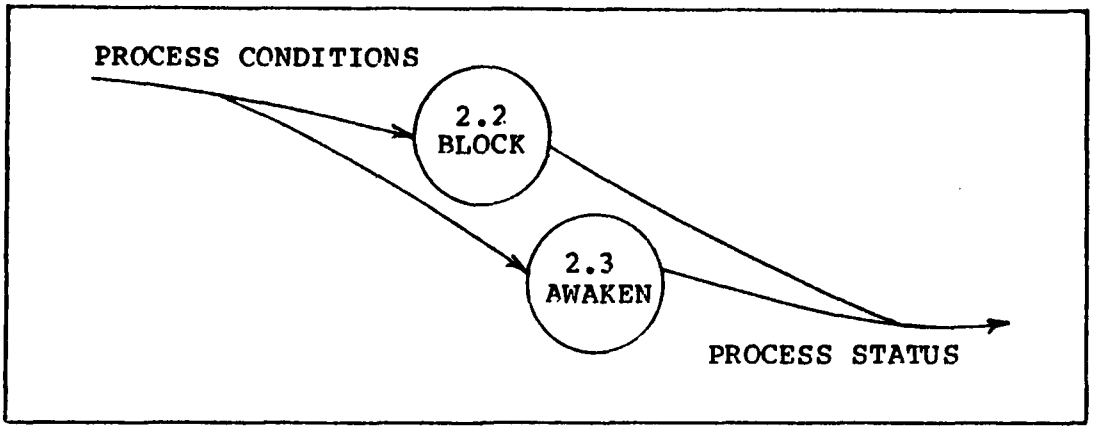


Figure 39. Interprocess Communication

avoidance as well as a thorough collection of literature sources.

Mutual exclusion (Ref. 43: 62) is the condition where certain processes are prevented from executing other processes. For example, when several processes asynchronously change the contents of a common data area, it is necessary to protect the data from simultaneous access and change by two or more processes. The solution is to prevent more than one process from entering the "critical section".

Syncronization is required among processes because certain processes cannot execute until others have occurred. Process syncronization introduces the concept of time and order among processes (Ref. 28: 247).

Depending on the data structure used, blocked processes may or may not be prioritized. Because processes may be blocked for a number of reasons it may be desirable to have different data structures for blocked processes. A

first-in-first-out arrangement may be sufficient for some applications and a prioritized queue for others.

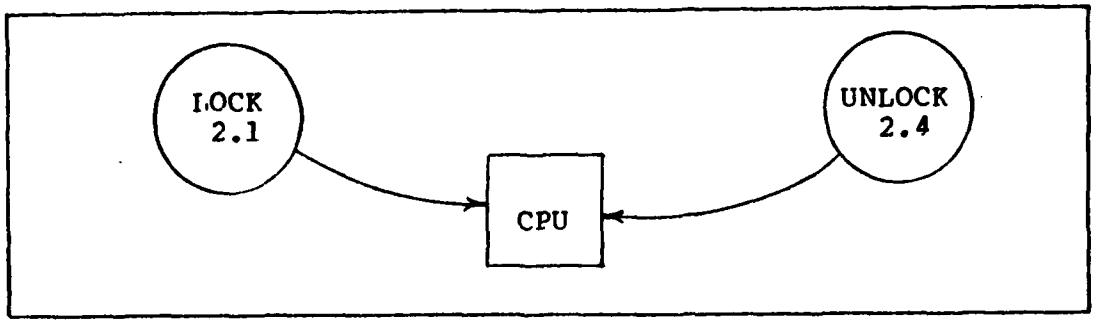


Figure 40. Lock and Unlock CPU

A lock and unlock procedure (2.1 and 2.4) must be provided to the interprocess communication mechanism to prevent more than one process from executing them at the same time. A process must not be able to lose the processor while executing interprocess communication. This will insure the block and awake mechanisms are indivisable operations.

Interrupt Handler

The interrupt handler is responsible for responding to all interrupts, both internal and external. Multiprogramming systems, especially timesharing systems, depend heavily on interrupt mechanisms for scheduling, input/output devices and system response to user requests. The interrupt handler must be able to identify the source of the interrupt and direct the processor to the correct routine to service the request. Generally, there are two categories of interrupts: internal, which are generated by

the system; and external, which are generated by peripheral devices. Their differences in handling are a design issue and will be discussed at a later point.

Many architectures provide a means for handling interrupts, their priorities, service routines, and identification. Because of the response desired in a timesharing system, and the overhead of handling interrupts, they are best implemented in hardware. However, any requirements which cannot be met by the hardware architecture must be implemented by software routines.

The first action by the interrupt handler is to save the system state at the time of the interrupt (3.1). After the interrupt is serviced (3.6) the system will return to this state (3.7) and continue as it was before the interrupt occurred. However, if the interrupt caused a condition to occur which altered the priorities of the system, another process may be selected to run after the interrupt. After the interrupt handler has saved the system state,

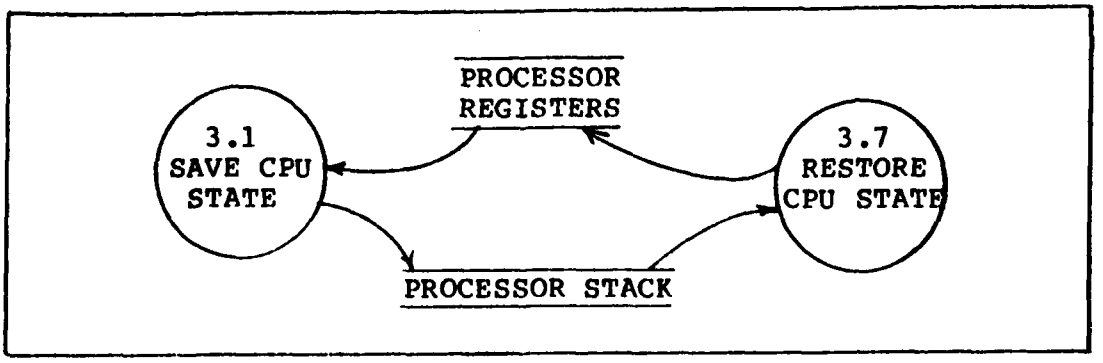


Figure 41. Save and Restore CPU State

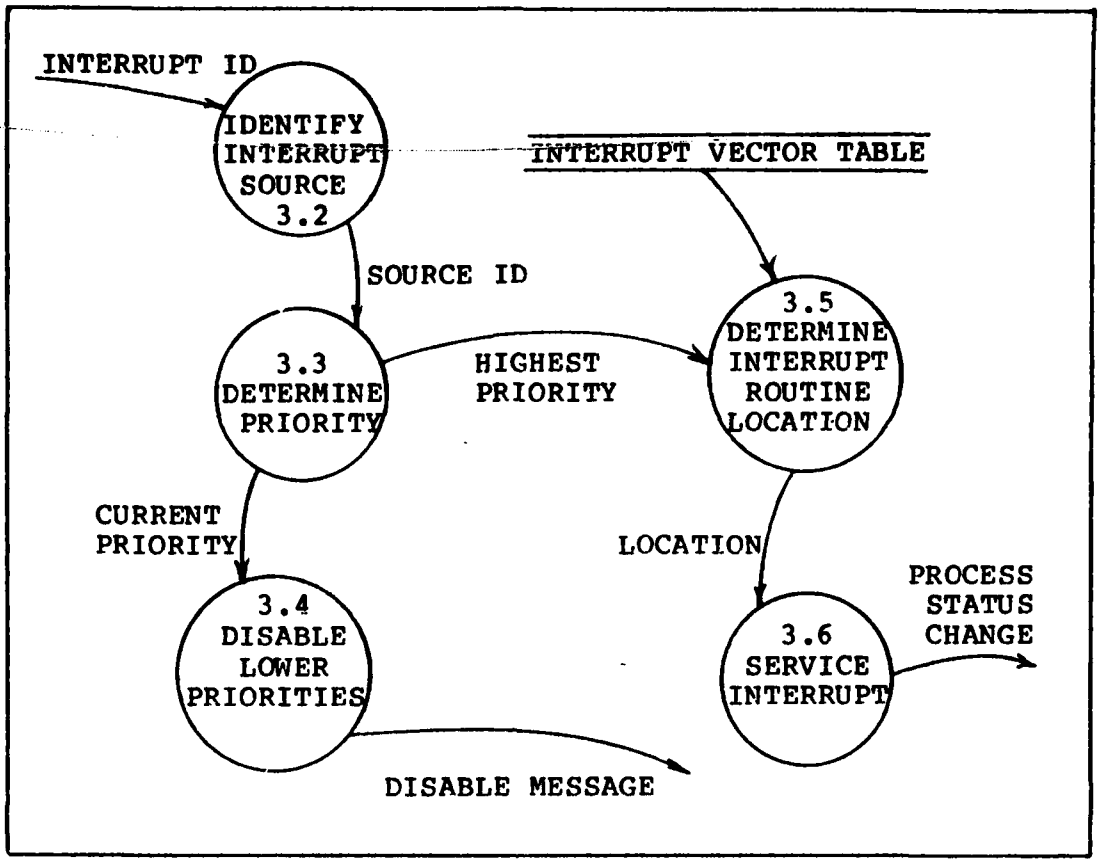


Figure 42. Interrupt Handler

the source of the interrupt must be determined (3.2). Because several interrupts may occur in a short span of time, a priority scheme must be organized to determine which interrupt should be serviced first (3.3). Depending on the processor architecture, this is most easily done in hardware. When a high order interrupt is being serviced, other interrupts may be ignored (3.4) or stored until the high priority interrupt is complete. High priority interrupts may yield to higher priority interrupts. Consideration must be given to the type, importance, and source of the interrupt when determining priorities. The

routines should be identified by mapping the interrupt to an address from a vector interrupt table (3.5).

Servicing the interrupt (3.6) is dependent on the type of interrupt involved. Each interrupt routine will perform a different task and may have a different effect on the system. For example, an interrupt to change running processes will alter the status of processes waiting for service and save the CPU status of the current process. However, it is important to keep the interrupt routines as short as possible since they must run in supervisor mode and run with interrupts disabled or partially disabled. Otherwise, system response may suffer.

It is likely that the occurrance of an interrupt will alter the state or status of some process. For this reason, the interrupt handler must have access to the process structures. As a result, the current process in the system may not be the best choice of all processes to run. Therefore, the dispatcher must be consulted after an interrupt occurs and the system is ready to run.

Summary

This chapter has addressed the hardware and software requirements of the timesharing operating system under development. The specifications have been involved and detailed in some areas and general in others. The use of data flow diagrams becomes a lengthly process in a large software effort such as this. However, the benefits derived from addressing the system at a high level of abstraction

makes the system easier to understand - an important funtional requirement. Structured analysis provides the partitioning and modularity required for a large system to be properly designed.

Appendix E provides the explanation of processes, data bases, and data flows in the form of a data dictionary (Ref. 57: 150) to complete the specification. The data dictionary defines terms of the data flow diagrams for the systems development. The data dictionary and data flows form the foundation for the design in Chapter Five.

V. The Operating System Design Specification

Introduction

Earlier chapters discussed the functional requirements and transformed them into system requirements by means of a structured specification. This chapter considers the structured software specification, hardware constraints, and background in operating systems to develop a design for the 8086 operating system.

The data flow diagrams from Chapter Four are transformed into module structure charts using structured design techniques as given by Weinberg (Ref. 57). By distinguishing data from control, a structure chart clearly indicates switches in the system. A structure chart also shows major loops and decisions in the system.

Hardware Design

One important hardware requirement is for interrupt handling. The 8086 has three types of interrupts: predefined which are functional within the 8086, user defined hardware interrupts, and software interrupts (Ref. 41: 8-30).

The CPU support card (see Appendix D) can handle up to sixty-four interrupt sources by cascading lines on the support card (Ref. 50: 6). The support card uses a 8259A programmable interrupt controller which is specifically designed for real time applications and controls priorities, vectoring, and cascading of interrupts (Ref. 21: B-107).

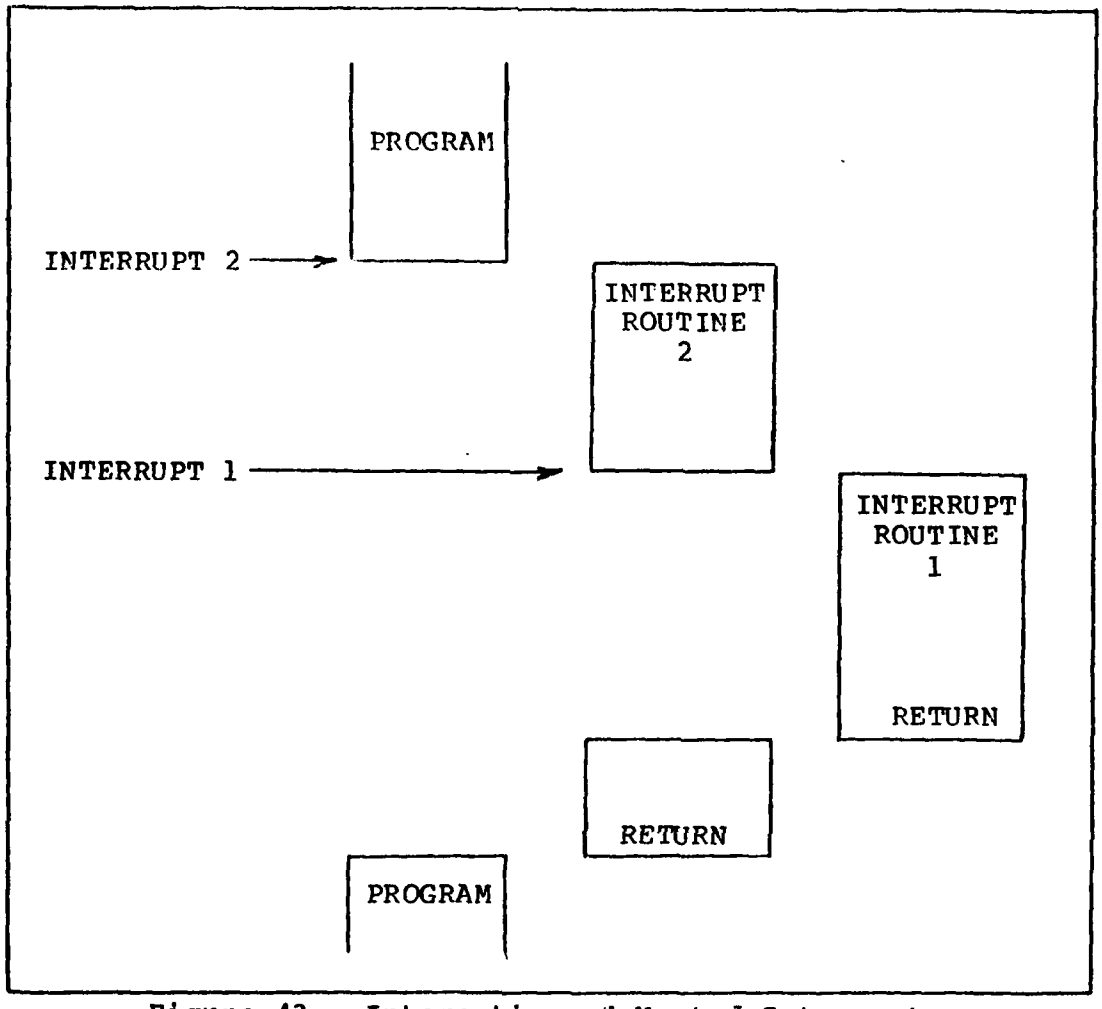


Figure 43. Interaction of Nested Interrupts

Figure 43 shows how interrupts are cascaded to allow higher priority events precedence. When an interrupt arrives, the execution program is suspended and the interrupt is vectored to an interrupt routine. Higher priority interrupts can execute by overriding lower interrupts. Interrupts are only acknowledged, howver, if the microprocessor has previously executed an enable interrupt instruction. This allows some degree of protection for critical sections of interrupt routines.

If an interrupt occurs while servicing a higher priority interrupt, it is recorded by the 8259A and serviced after the high priority interrupt. When allowing interrupts of several priorities to occur and interrupt one another, care must be taken to store the program registers of the interrupted process in different locations according to priority level of the interrupt received.

The 8086 requires each interrupt to have a number associated with it. The interrupt number indentifies the vector within the interrupt vector table. The interrupt number multiplied by four will give the address (absolute address) of the interrupt entry. Thus, the interrupt structure allows specification of the memory address for every interrupt service routine.

The lock and unlock mechanisms to provide indivisibility for the block and awaken mechanisms are best implemented by disabling and enabling interrupts. This will guarantee no process can lose control of block and awaken because there is no way to interrupt them. This will work only on a uniprocessor system however.

Memory Management Design

The addresses of processes are contained in memory based tables called process descriptors. The processor has the abliity to access these structures as the processes are activated. The descriptors contain all information necessary to describe the process to the system, particularly, the addressable environment of the process.

The 8086 computes memory addresses by summing the contents of a segment register and an effective memory address (Ref. 41:3-30). Segment registers are shifted left four bits and added to the effective address to get the actual address. The result is a twenty-bit address, or one megabyte capability. The 8086 address is composed of two addresses, the segment address and the effective address (referred to as the offset address). The segment register then contains the physical base address of the segment. The address space can then be viewed as a virtually unlimited number of segments up to one megabyte. However, the actual number of segments is limited by size and the physical addressability. The 8086 has four segment registers for immediate access to four types of segments without changing a segment register.

Figure 44 shows the method developed for memory protection and addressing. The addressable environment of the process is contained in the process descriptor as a base address, upper limit of the process, and address with the segment as an offset register. These values are loaded into the processor when a new process is activated. The base value is loaded into the offset register and the desired process address is loaded into the offset register. A comparision is made to determine if each address is within bounds for the process by checking the limit with the offset register.

The disadvantage of this method is the processor

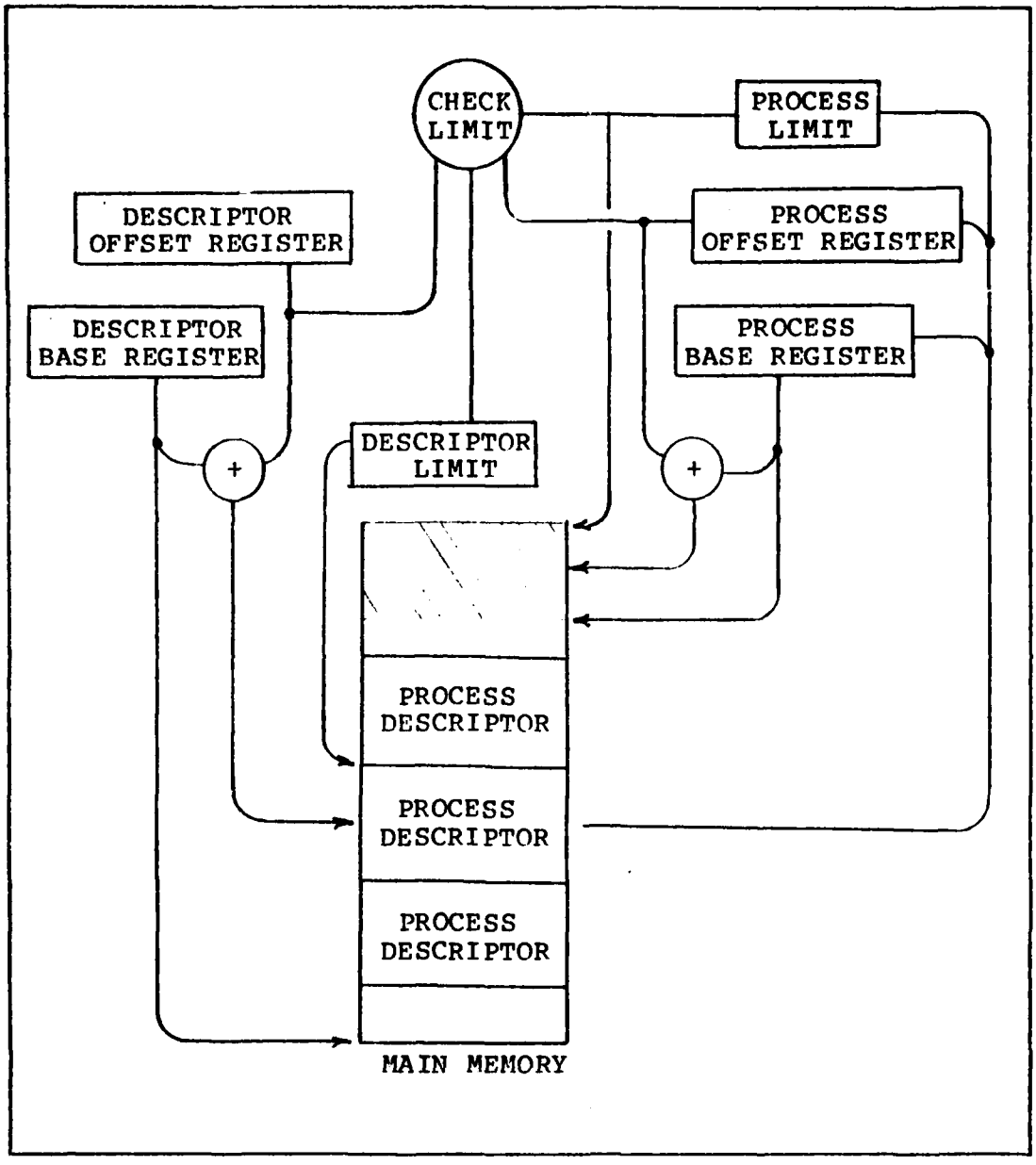


Figure 44. Memory Addressing and Protection

overhead of checking each process address. However, significant protection is the benefit. If additional overhead can be tolerated, further checks can be made in the process descriptor for process types, access rights and layers of access. A further extension of this method can

provide a method of virtual memory space (Ref. 32: 64).

Timing Mechanism

The support card has five programmable timers. One is dedicated to providing baud rates to the serial input/output ports. The other four are general purpose counters implemented by the 9513 system timing controller chip (Ref. 50: 24). The counting of the clocks is software controlled by programming the 9513. Commands available include ARM to enable the counter, LOAD to load the counter with the desired value, and DISARM to disable the counter.

The timing needed for preemptive scheduling can be provided by enabling the counter when a new process is dispatched to run. When the process finishes its timeslice, the counter issues an interrupt. A new process is selected to run and the counter is loaded with the time quantum and enabled again. If a process is interrupted by an external source while running, the counter can be disarmed and rearmed again when the running process continues if it is the same process.

Software Design Specification

The design of the software was accomplished by using the methods presented by Weinberg (Ref. 57: 134-167) which is based on the work of Yourdon and Constantine (Ref. 59). This involves the transformation of the data flow diagrams developed in Chapter Four to structure charts by means of transform centered design (Ref. 57: 176) and transaction

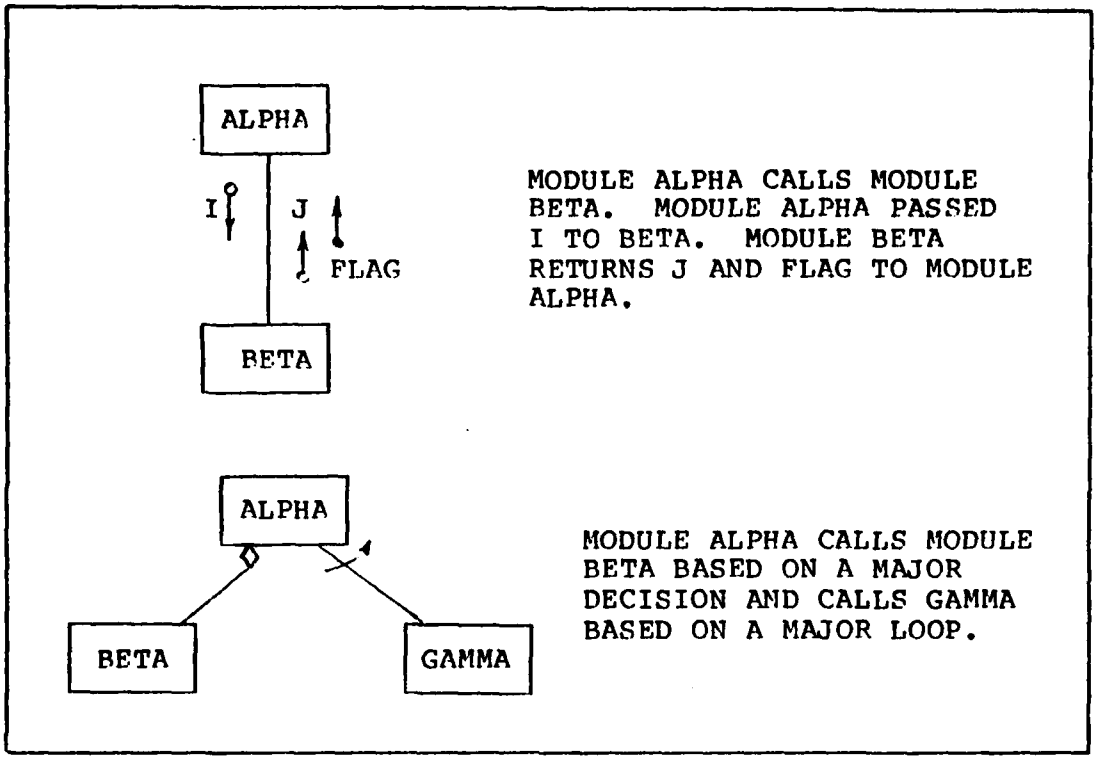


Figure 45. Structure Chart Notation

analysis (Ref. 57: 182). A structure chart indicates loops, decisions and switches in the system not indicated by data flow diagrams. Most of all, structure charts provide the design reviewer with a document that serves as the focus of design evaluation provided the modules, interfaces, and functions are rigorously defined in a data dictionary.

Structure Charts

Figure 45 indicates the symbols used in a structure chart. It is understood that the called module returns to the calling module. The called module may return control or data information. Control information is indicated by a solid circle and data information is represented by an open circle. This representation of the structure chart

illustrates two properties. First, the hierarchical modular structure is clearly indicated and second, the module function and communication in the structure is cleary shown.

The horizontal placement of module has no particular influence on the order of execution. This is important to note because two structures charts can represent the same data flow and yet physically appear entirely different.

Perhaps the most important function of the structure chart is that is facilitates change. Since it identifies input, outputs, processes and control very clearly, the modules affected by a desired change are easily identified. Thus, just as a data flow diagram indicates the system's logical design, the structure chart represents the physical design.

Structure chart symbology may include reference to pathological connections when one module references data defined in another module or when a module does not return to a calling module. This is undesirable and should be avoided if at all possible. Yourdon and Constantine discuss the treatment of pathological situations (Ref. 59: 235-249).

Transform Centered Design and Transaction Analysis

Transform centered design is a modular design strategy that specifies building a system around the data flow conception. This approach requires a top level module to call for processed logical input, and lower level modules to transform physical input into logical input and logical

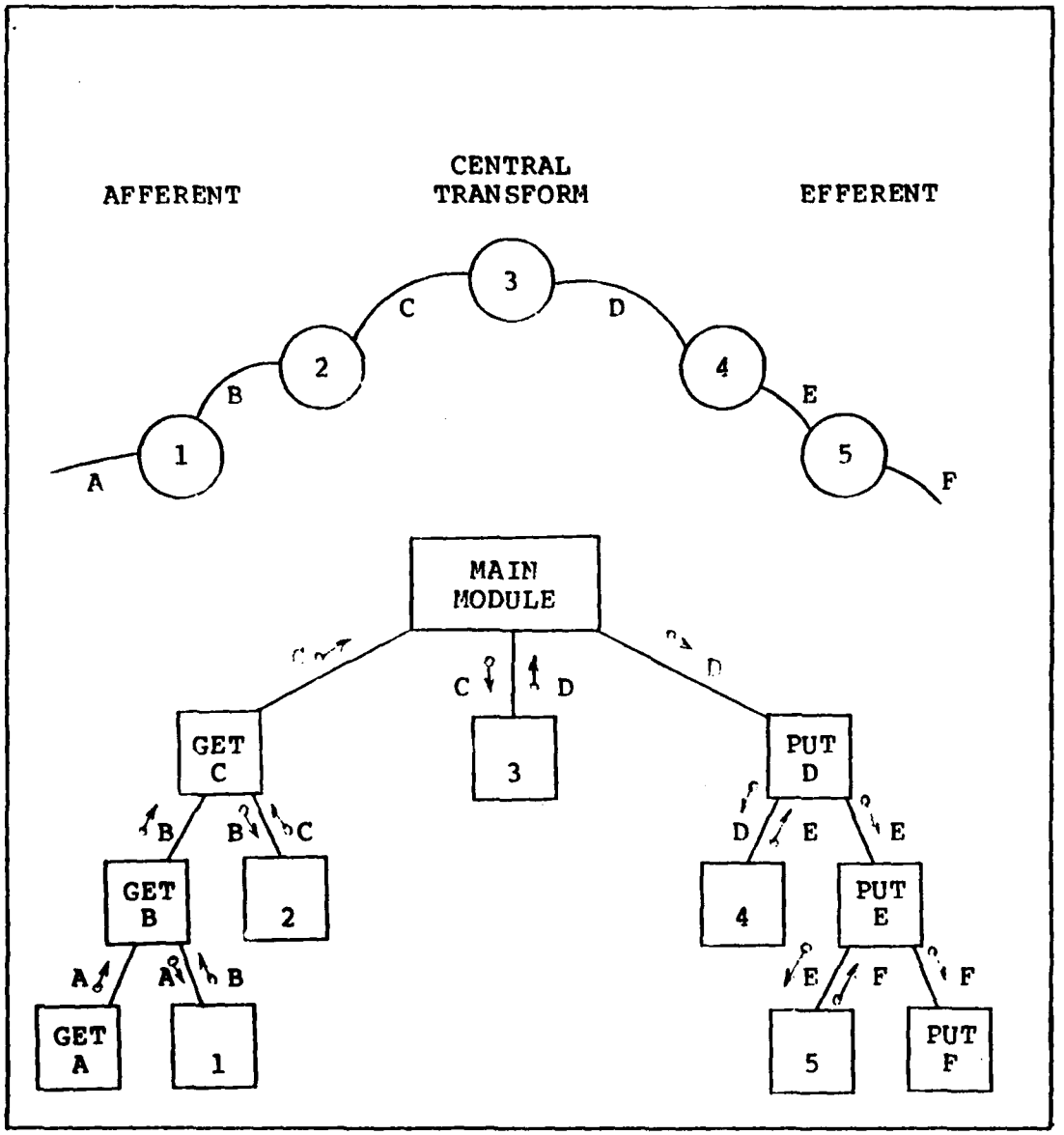


Figure 46. Transform Centered Design

output into physical output.

To develop structure charts from data flows using transform centered design, the major input and output data streams must be identifed. The input data stream is traced until it has reached a high abstract level where it is no longer considered input. This is the afferent branch. The

output data stream is traced backward until it can no longer be considered output. This is the efferent branch. The remaining portion of the data flow in the middle is the central transform. This process is called factoring. Factoring of the afferent branch, efferent branch, and central transforms is continued until the entire data flow is transformed to a structure chart. A simplified example is shown in figure 46.

Transaction analysis is another modular design strategy that builds a design around the concept of a transaction. A transaction is any element of data that sets off a sequence of actions. Data flow diagrams that fan out to do processing by transaction type strongly suggest transaction analysis. The resulting structure chart typically demonstrates a significant degree of fan-in and fan-out (Ref. 57: 185).

Both transaction analysis and transform centered design can be applied to the same data flow diagram if it demonstrates the properties of each technique. The resulting structure chart for this system provides a strong foundation from which subsequent design revision can be made judiciously. However, it should be kept in mind that the derived structure chart is not a final design effort. It is a first draft of a module design. Weinberg addresses additional design criteria to be considered such as cohesion, coupling, scope of effect, scope of control, and morphology to be considered to identify potential weaknesses

in a hierarchical structure (Ref. 57: 187-231).

The complete set of structure charts is located in Appendix F. Additional data dictionary entries were added to Appendix E to compensate the added data and processes derived when constructing the charts. Additional design decisions and policies are discussed in the following sections.

Interprocess Communications

The operations block and awaken are developed by using semaphores (Ref. 8). The choice of semaphores is based on the widespread use and understanding of semaphore implementation.

The block operation implies a process is blocked when a semaphore is zero and freed when an awaken operation increments the value to one. The easiest method of implementation is a semaphore queue. When a block operation is performed on a zero value semaphore, the process is added to a queue and is made unrunnable. Conversely, when an awaken operation is performed on a nonempty queue, a process is taken off the queue and made runnable.

Semaphore queues may be organized as first-in-first-out or prioritized. Different queue organizations may apply to each resource. The structure of the semaphore must contain three items, the semaphore integer, the type of queue organization, and a pointer to the queue.

As discussed earlier, block and awaken must be indivisible processes. If interrupted during execution

their values can easily be mistaken. On a single processor system, the easiest method of guaranteeing indivisibility is to disable interrupts. Since the execution of the semaphore mechanism is very short, interrupts will be disabled for a very short time.

Scheduling Management

The scheduler must decide which process gets priority in the processor queue. Assignment of priority to system processes is performed by the scheduler based on the expected response time. For example disk processes have higher priority than printer processes. However, it has been observed that system priorities have little impact on performance (Ref. 54: 1937). The user process priorities are lower than the lowest system priority. Thus, all system processes are selected to run before user processes. The user process is assigned a priority based on the ratio of processor time to real time used by the process and updated each time quantum. UNIX uses a similar ratio but updates every second.

Since interactive processes are characterized by a low ratio, interactive response is desirably low. A high priority process cannot dominate the processor because its priority will drop as more compute time is accumulated, thus producing a desirable feedback situation. Likewise, a low priority process will not be ignored because as real time increases its priority will rise.

The dispatcher overhead is minimized by selecting the

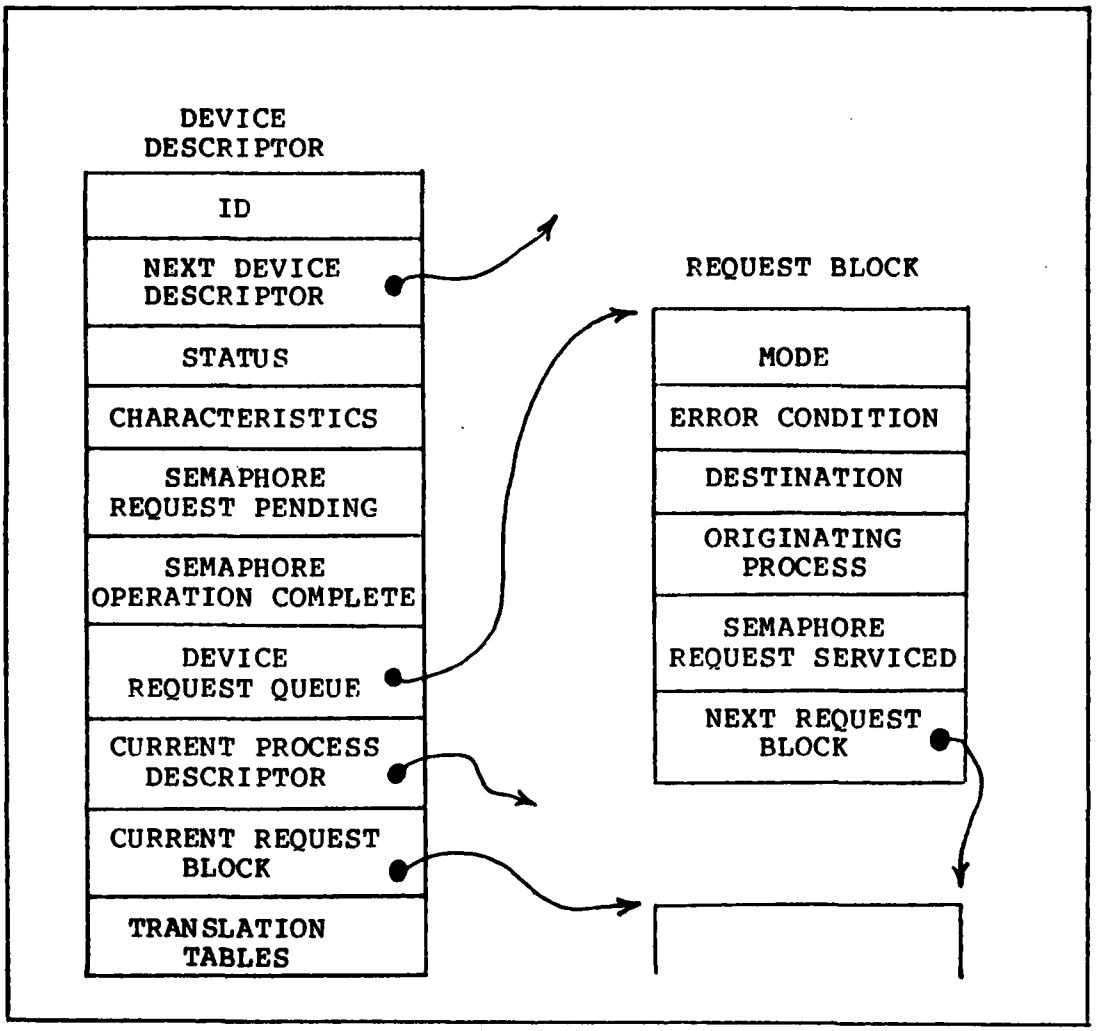


Figure 47. Input/output Data Structures.

front process in the ready queue. The scheduler performs all prioritizing and manipulation of the queue to place the higher priority job up-front.

Input/Output Management

As mentioned in the requirements for input/output management, a device handler is responsible for servicing requests on the queue and notifing the requesting process when a transfer is complete. Figure 47 indicates the data

structure diagram for the device descriptor and associated request blocks which are dealt with by the device descriptor.

A separate device descriptor exists for each device. Request blocks are added as processes request service from the device. The semaphore, request pending, is signaled by the procedure each time it puts a request block on the queue. If the queue is empty the semaphore will be zero. The semaphore, operation complete, is signaled by the interrupt routine after an interrupt is generated for the device.

Conclusions

This chapter has developed a concept of design from the requirements of earlier chapters. The hardware design concentrates on the interrupt structure, timing mechanisms, and memory addressing scheme. The techniques specified by Weinberg were used to transform the data flow diagrams into module structure charts to yield a software design based on the structured specification.

VI. Conclusions and Recommendations

This investigation has concerned the development of an operating system based on several objectives. First, an operating system must be friendly to the user, that is, a user must be able to communicate his needs easily to the computer and the computer must respond in a manner explicit to the user, whether novice or expert. Second, when deciding between simplicity and efficiency, simplicity should come out the winner. Third, ease of understanding should guide the system design and ease of use should guide the user interface.

To support user friendliness, detailed functional requirements of the man-machine interface have been presented. Existing systems have also been studied to examine existing user environments.

An attempt has been made to keep the development simple by using known tools such as semaphores, hierarchical levels, and common operating system structures. The complex nature of an operating system makes it difficult to simplify certain issues and as a result some over simplification may have resulted.

Ease of understanding is essential in a large software effort such as this. The structured specification was the most time-consuming portion of the research and the most productive as well. The development of the design progressed easily because a significant amount of partitioning was completed in the structured specification.

The techniques of structured analysis are highly recommended for any such effort. Ease of understanding impacts directly on ease of implementation.

In summary, existing operating systems have been studied, and functional requirements specified for a productive and friendly user environment. By using structured analysis techniques, the system requirements were specified and module structure charts were developed. The structured specification, though time consuming, produced an understandable development of the system requirements.

Certain assumptions were made during this investigation. It was assumed that interrupt routines existed to drive input/output devices. All other interrupt routines such as those to handle error conditions and preemptive scheduling were also assumed. Command routines were taken for granted. The position was taken that an environment is provided by the operating system which flexibly supports command routines. However, the command itself is part of an applications layer in the operating system and not an integral part of the system (Ref. 29: 47-55).

Recommendations

Several areas must still be addressed before implementation can take place. To simplify much of the design in this stage of development, many of the structured specifications were not detailed down to the lowest level. The partitioning of the design must continue before

implementation can take place.

Operating system commands must be developed in more detail. The interrupt rountines that have been assumed must be designed and interfaced.

No input/output buffering has been specified. If a process is performing repeated transfers on the same device it will repeatedly be suspended while the transfers take place. In order to avoid this overhead, input/output buffering should be developed.

No provision for spooling is currently designed. During periods of high demand some input/output devices may become heavily loaded and the processes may be forced to wait for release. Input/output spooling should be developed to spread the load on heavily used devices such as a printer.

The major advantage of modern timesharing systems is to provide a workbench, so to speak, for the programmer. To extend this research to the point where it may be considered a programmer's workbench the following features would be necessary:

(1) At least one programming language,
(2) Complete set of commands for running, testing, and modification of programs,
(3) A run-time interface that may intersect with the command language,
(4) Provisions for the sharing, maintenance, and protection of files, to include the ability to define sharing rights between groups of users,
(5) Provide unattended operation of any given program,
(6) The ability to pass arguments between commands,
(7) The capability to form working dialects for a specialized group of users.

Each of these can develop into a major research effort. Some provisions have already been made for 4, and several of the others have been alluded to.

A research effort involving a subject as detailed as operating system design becomes not only quite deep, but broad as well. It is suggested that future efforts by single reseachers be narrow topics. This approach should allow the depth of study necessary without the distraction of a wider subject area.

Bibliography

1. Bard, Y. "Performance Criteria and Measurement of a Time-Sharing System", IBM Systems Journal, 10: 193-216 (1971).

2. Bobrow, Daniel G. "TENEX, a Paged Time-Sharing System for the PDP-10," Communications of the ACM, 15: 135-143 (March 1972).

3. Boldyreff, Cornelia. "UNIX on a Mirco," Sigsmall Newsletter, 7: 7-8 (February 1981).

4. Bourne, S. R. "The UNIX Shell," Bell System Technical Journal, 57: 1971-1990 (July-August 1978).

5. Browne, James C. "The Interaction of Operating Systems and Software Engineering," Proceedings of the IEEE, 68: 1045-1049 (September 1980).

6. Cheriton, D. R. "Man-Machine Interface Design for Time-Sharing Systems," Proceedings ACM Conference, 362-380 (1976).

7. Coffman, Edward and Peter Denning. Operating System Theory, New Jersey: Prentice-Hall, 1973.

8. Dijkstra, E. W. "Cooperating Sequential Process," Programming Languages, Y. Genuys, Editor. New York: Academic Press, 1968.

9. Dijkstra, E. W. "The Structure of T.H.E. Multiprogramming System," Communications of the ACM, 11: 341-346 (May 1968).

10. Dzida, W., S. Herda and W. D. Itzfeldt. "User-Perceived Quality of Interactive Systems," IEEE Transactions on Software Engineering, 4: 270-276 (July 1978).

11. Fitter, M. "Towards More Natural Interactive Systems," International Journal of Man-Machine Studies, 11: 339-350 (January 1979).

12. Freeman, Peter. Software Systems Principles, Chicago: Science Research Associates, 1975.

13. Gaines, Brian R. and Pater V. Facey. "Some Experience in Interactive System Development," Proceedings of the IEEE, 6: 894-911 (June 1975).

14. Greenburg, Robert B. "The UNIX Operating System and the XENIX Standard Operating Environment," Byte, 6:

248-264 (June 1981).

15. Hansen, Brinch. "The Nucleus of a Multiprogramming System," Communications of the ACM, 13: 238-241 (April 1970).

16. Hansen, Brinch. Principles of Operating Systems, New Jersey: Prentice-Hall, 1973.

17. Hayes, Phil et al. "Breaking the Man-Machine Communication Barrier," Computer, 14: 19-27, (March 1981).

18. Hellerman, Herbert and T. F. Conroy. Computer System Performance, New York: McGraw Hill, 1975.

19. Hornig, J. J. "Process Structuring," ACM Computing Surveys, 5: 5-30 (March 1973).

20. Hsiao David K. Systems Programming - Concepts of Operating and Data Base Systems, Reading: Addison-Wesley, 1975.

21. Intel Corporation. The 8086 Family User's Manuel, Santa Clara: Intel Corporation, 1980.

22. Johnson, Jan. "Intellect on Demand," Datamation, 27: 73-78 (November 1981).

23. Johnson S. C. and D. M. Ritchie. "Portability of C Programs and the UNIX System," Bell System Technical Journal, 57: 2021-2048 (July-August 1978).

24. Kahn, Kevin C. "A Small-Scale Operating System Foundation for Microprocessor Applications," Proceedings of the IEEE, 66: 209-216 (February 1978).

25. Kennedy, T. C. S. "The Design of Interactive Procedures for Man-Machine Communication," International Journal of Man-Machine Studies, 6: 309-334 (1974).

26. Kindall, Gary. "CP/M: A Family of 8 and 16 bit Operating Systems," Byte, 6: 216-232 (June 1981).

27. Libes, Sol. "16-Bit Microcomputer Disk Operating Systems," Microsystems, 50-54 (July 1981).

28. Lister, A. M. Fundamentals of Operating Systems, New York: Springer-Verlag Inc., 1979.

29. Lorin, H. and H. M. Deitel. Operating Systems, Reading, Massachusetts: Addison-Wesley Publishing Company, 1981.

30. Lycklama, H. " Unix on a Microprocessor," Bell System Technical Journal, 57: 2087-2101 (July-August 1978).

31. Madnick, Stuart E. and John J. Donovan, Operating Systems, New York: McGraw-Hill, 1974.

32. Markowitz, R. and W. B. Pohlman. "The Evolution Path of the 8086 Microprocessor Architecture for Operating System Environments," Microprocessors in Military and Industrial Systems Workshop. 62-66. IEEE Computer Society, February 1980.

33. Martin, J., Design of Man-Computer Dialogues. New Jersey: Prentice-Hall, 1973.

34. Miller, L. A. and J. C. Thomas, "Behavioral Issues in the Use of Interactive Systems," International Journal of Man-Machine Studies, 9: 509-536 (March 1977).

35. Morgan, Chris. "The New 16-bit Operating Systems, or, The Search for Benutzerfreundlichkeit," Byte, 6: 6 (June 1981).

36. Muller, K. G. Specification and Design of Interactive Systems, SHAPE Technical Center, The Hague, (June 1980).

37. Norman, Donald A. "The Trouble with UNIX," Datamation, 27: 139-160 (November 1981).

38. Organick, E. I. The Multic System: An Examination of Its Structure. Cambridge: The MIT Press, 1972.

39. Overgaard, Mark. "UCSD Pascal: A Portable Software Environment for Small Computers," National Computer Conference, 1980, 49: 747-754, (1980).

40. Plauger, P. J. and M. S. Krieger. "UNIX-like Software Runs on Mini- and Microcomputers," Electronics, 54: 125-129 (March 24, 1981).

41. Rector, Russell and George Alexy. The 8086 Book, Berkeley: McGraw-Hill, 1980.

42. Ritchie, D. M. and K. Thompson. "The UNIX Time-Sharing System," The Bell System Technical Journal, 57: 1905-1929 (July-August 1978).

43. Ritchie, D. M. "UNIX Time-Sharing System: A Retrospective," Bell System Technical Journal, 57: 1947-1969 (July-August 1978).

44. Rosin, S. "Electric Computers: A Historical Survey,"

ACM Computing Surveys, 1: 7-36 (March 1969).

45. Rosin, T. F. "Supervisory and Moniter Systems," ACM Computing Surveys, 1: 37-54 (March 1969).

46. Rouse, William B. "Design of Man-Computer Interfaces for On-Line Interactive Systems," Proceedings of the IEEE, 63: 847-855 (June 1975).

47. Sackman, H. Man-Computer Problem Solving, "Experimental Evaluation of Time-Sharing and Batch Processing", New York: Auerbach, 1970.

48. Schorer, Peter. "Structure the Use: Notes on a Method for Designing Computing System Environments," Computer, 14: 77-86 (December 1981)

49. Schwabe, J.,M. J. Elmore and D. Miller. "The Impact of 16-Bit Microprocessors on Software Development," Computer Design, 20: 111-115 (June 1981).

50. Seattle Computer Products, Inc. "CPU Support Board - Instruction Manual, Model SCP300F," Revision F, Seattle: Seattle Computer Products, Inc., 1980.

51. Shaw, Alan C. The Logical Design of Operating Systems, New Jersey: Prentice-Hall, 1974.

52. Sherman, S. et. al. "Trace Driven Modeling and Analysis of CPU Scheduling in a Multiprogramming System," Communications of the ACM, 12: 1063-1069 (1972).

53. Tesler, Larry. "The Smalltalk Environment," Byte, 6: 90-147 (August 1981).

54. Thompsom, K. "UNIX Time-Sharing System: UNIX Implementation," Bell System Technical Journal, 57: 1931-1946 (July-August 1978).

55. Watson, Richard W. Timesharing System Design Concepts, New York: McGraw-Hill, 1970.

56. Watson, Richard William. "User Interface Design Issues for a Large Interactive System," National Computer Conference 1976, 45: 357-364 (June 1976).

57. Weinberg, Victor. Structured Analysis, New York: Yourdon Press, 1979.

58. Weiner, Bruce and Douglas Swartz. "Adapting Unix to a 16-bit Microcomputer," Electronics, 54: 120-129 (March 24, 1981).

59. Yourdon, Edward and Larry R. Constantine. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Englewood Cliffs, N. J.: Prentice Hall, Inc., 1978.

60. Yusko, Robert J. Development of a Multiprogramming System for the Intel 8086 Microprocessor," MS Thesis. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1981.

61. Zaks, Rodnay. The CP/M Handbook with MP/M. Berkeley: Sybex, Inc., 1980.

62. Zarella, John, Operating Systems Concepts and Principles. Suisun City, California: Microcomputer Applications, 1979.

Appendix A

Rationale for Timesharing and Multiprogramming

This appendix justifies the usage of timesharing and multiprogramming in modern computer systems. These two techniques are widely used and accepted but their advantages, disadvantages, and complexity are not as readily understood. Yet, both concepts have probably contributed more to the productivity and efficiency of computing than any other single technique.

Timesharing is the use of a computer system to support multiple users who view and interact with the system as if it was a dedicated system. The primary objective of timesharing is to provide fast, convenient, and economical man-machine interaction to several users (Ref. 18: 235).

Multiprogramming is the technique of concurrent execution of several programs by enabling the CPU to suspend the execution of one process to execute another and return to the suspended process at a later time. The primary objective of multiprogramming is to maximize the usage of the system resources while maintaining a good response time to the user (Ref. 20: 148).

Advantages of Timesharing

Timesharing is an interactive method of computing that is totally different from batch computer systems. They are not and should not be thought of as batch systems with interactive facilities added on. To appreciate this fact

the two methods should be compared.

The use of batch facilities and timesharing facilities can be viewed as computing in the passive and active modes respectfully. In the passive (batch) mode the user invokes the support of a computer center and submits his programs to the personnel, conforming to their requirements to receive service. The user has to predetermine all the job steps that are to be executed and has no access to the computer. He has no active role in the data processing operations for his own program. Turnaround time can be lengthly and the user has no control over it. The processing is totally off-line and remote from the user's normal working environment.

However, in the active mode (timesharing) the user has immediate access to the data processing system via a workstation in his normal working environment and he determines when processing takes place. All input is processed in his presence. The system interfaces directly with the user's task. In the interactive mode a user can adapt the algorithms and the workstation to suit the requirements of a given problem. Therefore, in addition to being a user of the system he can also be a producer of the system by creating his own software products.

It has been suggested to be truely interactive, a timesharing facility should exhibit three properties (Ref. 36: 24):

(1) Integration of user and computer capabilities by dialogue with free choice of user input and

deterministic response by computer.
(2) User has full control over processing sequence (proceduralization).
(3) User adapts system to his actual needs by adaptation of work station and by adaptation of software (actualization).

The most outstanding characteristic of a timesharing system may be that its user can adapt it permanently to actual problem solving situations and approaches. This is "actualization" (Ref. 36: 5). This may serve to adapt a system to new user classes. The user may discover that he can delegate more routine operations to the system, thus freeing him for more creativity. The user may wish to extend the system's capabilities so that it can be applied to new problems.

It should be pointed out that historically, all computer systems were dealt with interactively. Originally, computers were run by those who could start and stop them at will. It was only economics that led to the establishment of data computer centers and their passive mode of operation.

The User and Timesharing

Given the additional overhead and software complexity of a timesharing system, are the benefits worth the effort? Most interactive users agree. One study at MIT involved a number of students in a business class who were required to provide an optimal solution to a management problem using computer simulation techniques (Ref. 47). The class was divided into two groups. One group was to use an available

batch system and the other was given use of the timesharing facility. The computer facility provided information on the use of resources, the faculty gave its evaluation of the two groups solutions, and students completed questionaires to determine performance and user attitudes. The summary of the major results of the experiment are:

(1) The timesharing group had a lower man/effort cost, but higher computer cost. However, the total cost for both groups did not differ appreciably, even though the costs were distributed quite differently.
(2) A significantly better solution was achieved by the timesharing users.
(3) In the batch group, more than twice as many people did not arrive at a useful solution.
(4) More students prefered the timesharing system.

The conclusion is that timesharing can attribute to a better problem solution and is prefered by most users when compared to conventional batch systems. It is also interesting to note that costs, despite a different distribution, did not differ significantly between batch and timesharing use.

Effectiveness of Multiprogramming

The effectiveness of multiprogramming is usually measured by the amount of concurrent activity in the system (Ref. 12: 302). If all processes in the active multiprogramming mix are input/output bound, the processor is not effectively utilized. Likewise, if all processes are compute bound, the processor is utilized but not the input/output devices. The utilization of a specific resource is the fraction of time that the resource is busy.

The sum of the utilizations is an indication of the effectiveness of the multiprogramming. On most systems an attempt is made to maintain a mix of computation and input/output processes to balance the load on these resources. This is usually not feasibile, however, because the behavior of processes varies during execution.

The effectiveness of mutiprogramming as well as the responsiveness of timesharing systems depends ultimately on the performance of the scheduler. The definition of a scheduler is an algorithm that uniquely specifies which process is to receive service next by a resource (Ref. 18: 98). The scheduling algorithms must deal with a unknown set of processes whose behavior may change during execution (Ref. 12: 302). A system should be designed so that reasonable service is guaranteed irregardless of user input.

Each process uses a certain amount of CPU time before becoming blocked, waiting for an input/output request to be completed. A heuristic often applied to processor scheduling is to attempt to minimize the time until an execution interval ends and an input/output request is made. This can be viewed as an attempt to maintain as many processes doing input/output as possible. Since the input/output part of a system usually contains a number of asynchronous devices, this tends to increase the effectiveness of multiprogramming. If the execution interval were known, then a scheduling discipline could be selected for optimal performance. However, this is not

known because it requires knowledge of future performance of the process. A scheduling technique such as round-robin has the effect of giving preferential treatment to short service requests and is particularly effective when the service-time distribution has a variance much larger than the mean (Ref. 52: 1063).

Processor vs. I/O Device Speed

Many programming requirements require a large amount of input/output activity. Since input/output devices are dreadfully slow compared to CPUs, this encourages the use of multiprogramming to overlap computer bound processes with input/output bound processes. The case for multiprogramming is even more attractive due to the fact that the improvement in speed of input/output devices has not kept pace with the improvements in CPU speed. For example, disk improvements in transfer rate, access time, and rotational delay have been in the order of only 1.5 to 2 while the improvement of the internal processing speed of the CPU has been in the order of 5 to 10 (Ref. 20: 147).

The disparity of the hardware improvement in terms of the CPUs internal processing speed and the input/output transfer rate, and the users demand for faster response and direct access, strongly encourages the use of multiprogramming. The desired result will be: (1) reduced CPU wait time on input/ouput operations by running another process while the input/output process is running and (2) increased response time by proper scheduling techniques.

Multiprogrammed Influence on the User

In some circumstances it is beneficial for the user to be aware of the system characteristics of his computing environment. The user may be able to take advantage of techniques for saving time and space in the system if he is aware of operating system algorithms and the cost of each resource before he makes any tradeoffs.

The user should be aware of whether or not the host system is multiprogrammed. If not, it is likely the user has a large main store to work with and space is not a problem. In this case, a user may feel free to use more space to save CPU time. For a fixed task, main storage availability is inversely proportional to the time to accomplish the task. As more storage is available, the user

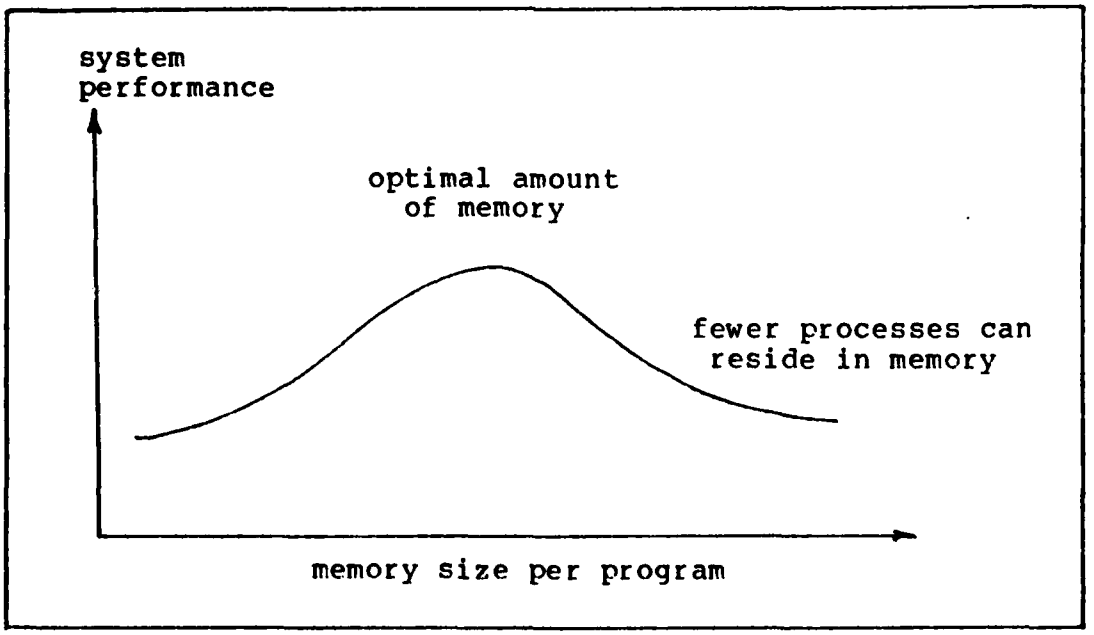


Figure A-1. Multiprogramming Performance vs. Memory Usage

can use more complex algorithms to execute the task more efficiently.

However, on a multiprogrammed system the main storage is divided among the users and less memory is available to each one. If one user increases his storage, the system cannot multiprogram as many processes and system performance is degraded. Figure A-1 indicates typical throughput for a multiprogrammed system with fixed main memory size. As the amount of storage for each job is increased, system performance is also increased. There comes a point, however, where the number of users is decreased and the mutiprogramming is decreased resulting in decreased performance. The optimal storage per user is at the peak of the performance curve, but this would vary with the mix of processes in the system (Ref. 31: 490).

In one study of costs on the tradeoff of input/output access versus processor time, it was found that 12 milliseconds of processor time equaled one input/output access. Or, if a user could spend less than 12 milliseconds of processor time and save one input/output, it would be to his advantage on this particular system (Ref. 1: 198).

Degree of Multiprogramming

Too much multiprogramming, too many processes competing for the processor, can have the opposite effect intended. Generally, the less time a processor spends waiting for input/output (more multiprogramming) the more utilized the CPU is. The implication is that increasing the degree of

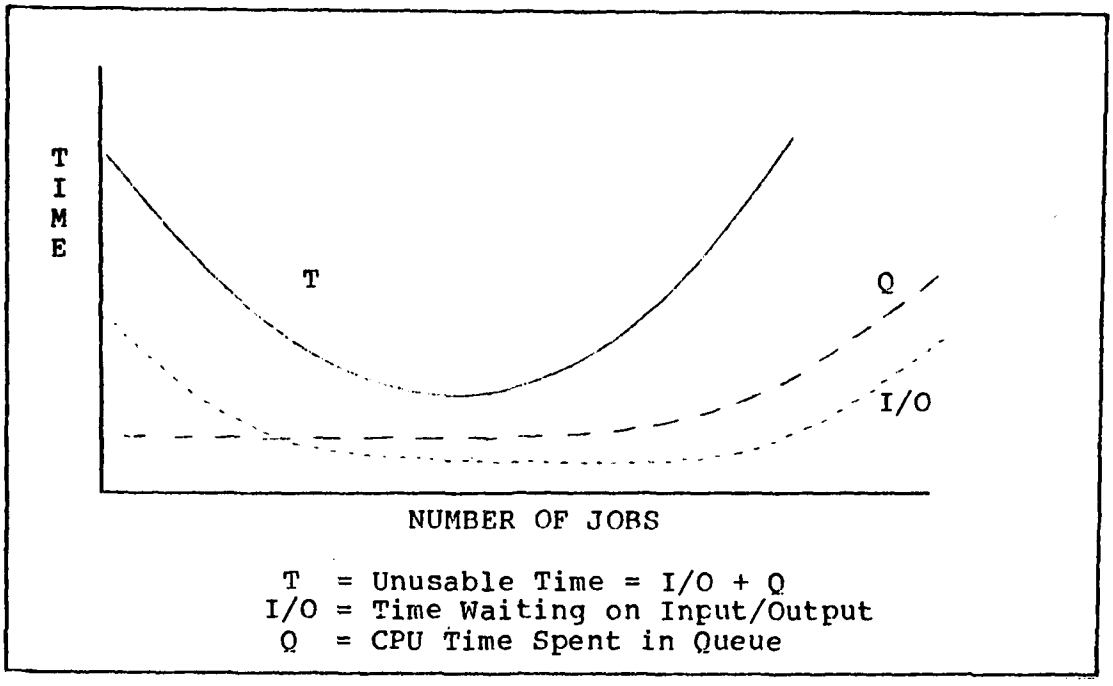


Figure A-2. Degreee of Multiprogramming

multiprogramming increases performance. This does not take into account the amount of overhead involved. The overhead is a result of two factors (Ref. 31: 485). Queuing input/output request requires a fixed amount of CPU time for each request. Second, limited input/output resources cause the device queues to back-up and the CPU becomes idle waiting for input/output devices to catch-up on requests.

Figure A-2 indicates the effect too much multiprogramming can have on CPU time (Ref. 31: 486). As indicated, a point is reached where increased multiprogramming has a negative effect on system performance. Even with fewer input/output requests, the switching time between processes will eventually cause poorer performance.

Conclusions

Timesharing is preferred because of its "conversational" ability to provide a quick response to user needs. Because several users can work on one system, software can easily be shared. The user's abiltiy to interact directly with the system generally increases productivity.

To provide quick response and adequate throughput, timesharing systems depend on multiprogramming and efficient scheduling techniques. Multiprogramming, when properly implemented, provides greater utilization of system resources and decreases turnaround time. However, the effectiveness of mutiprogramming depends on several factors including the degree of multiprogramming, amount of main memory, the memory allocation algorithms used, and scheduling techniques.

Appendix B

Man-Machine Interface Issues

This appendix is a correlation of several literature sources on user-friendliness and man-computer dialogue. Even though user-oriented systems are desirable, very little information exists to support what creates a friendly environment or makes a computer system easy to use.

Chapter Three focused on what functional requirements should be considered when designing a computer system to achieve user-friendliness. The material in this appendix supports and adds to those requirements.

What the User Perceives

The majority of work to improve support for interactive user has been done in the last decade. However, the level has not yet been reached where user-quality can be measured. Some researchers feel the ability to measure user-quality in interactive systems is essential to proper system design (Ref. 10: 270).

The research performed by Dzida, Herda, and Itzfeldt (Ref. 10) makes an attempt at measuring user-quality by estimating, with statistically nonoverlapping factors, how the user perceives the system. They believe their contribution is a step forward in actually measuring user quality by mathematical means. Their efforts are centered on two interests: (1) the expectations and desires of users, and (2) using a mathematical method which gives

empirical evidence about nonoverlapping of quality aspects.

The study is based on the response of 600 persons to a questionnaire which contained 100 system requirements based on user-quality. Factor analysis was performed using SPSS. The initial set of 100 system requirements was reduced to 57 requirements. Seven factors controlling 44 percent of variance of the data were extracted. Each factor is composed of a set of requirements belonging together, with factor loadings of at least 0.30 with smaller loadings indicating random correlations. The higher the loading the more important is a requirement (Ref. 10: 271). The seven factors were:

(1) Self-Descriptiveness
(2) User Control
(3) Ease of Learning
(4) Problem Adequate Usability
(5) Correspondence with User Expectations
(6) Flexibility and Task Handling
(7) Fault Tolerance

The factors are subject to some degree of subjective interpretation. It should also be considered that the factors and requirements were first formulated in German and then translated.

A partial list of the requirements under each factor are listed in table B-I. It is interesting to note that the more important requirements deal with the command language aspects of the computer rather than performance charactertics of the machine. It seems the users are more concerned with how they converse with the computer than how the computer functions. For this listing the requirements

Table B-I

Factor 1: Self-Descriptiveness

0.70 explain system requests to the user if and when necessary
0.68 supply explanations in different detail and different format upon user request
0.67 supply help features pertinent to any dialogue situation
0.60 enable transparency of dialogue organization and dialogue sequence at any time
0.52 explain each command and subcommand upon user request

Factor 2: User Control

0.60 admit interruptions of a task to start or resume another task
0.59 admit process canceling without detrimental side effects
0.59 allow abortion of particular dialogue stops or processes

Factor 3: Ease of Learning

0.63 make user manuals superfluous
0.61 facilitate the learning of system use without consulting manuals
0.57 be usable without special DP-knowledge
0.52 largely offer on-line forms for user input

Factor 4: Problem Adequate Usability

0.69 have a data management system that obviates as far as possible the need for the user to perform clerical or housekeeping activities
0.63 manage formatting, addressing, and memory organization without bothering the user
0.56 determine system decisions without consulting the user
0.50 accept free formatted command input

Factor 5: Correspondence with User Expectations

0.76 behave similarly in similar situations
0.71 request analogous user actions to similar tacks to be performed
0.65 offer minimum astonishment behavior towards the user

Factor 6: Flexibility in Task Handling

0.56 allow user to extend the command language
0.55 offer facilities for stacking tasks

Factor 7: Fault Tolerance

0.53 insist only on partial retyping if previous input was erroneous
0.52 tolerate typical typing errors
0.49 give error messages with correction hints

with a loading factor of 0.49 or higher were selected. The complete list is available in Ref. 10.

How Specific Users View the System

The study goes on to subdivide the users into groups based on their background experience in three areas:

(1) Interactive System Experience
(2) Mode of Operation
(3) Frequency of Use

The comparisons of weighted factors for different user groups is illustrated in figure B-1. It is interesting to note how the importance of factors is perceived in different ways by the separate user groups. Notice that only five

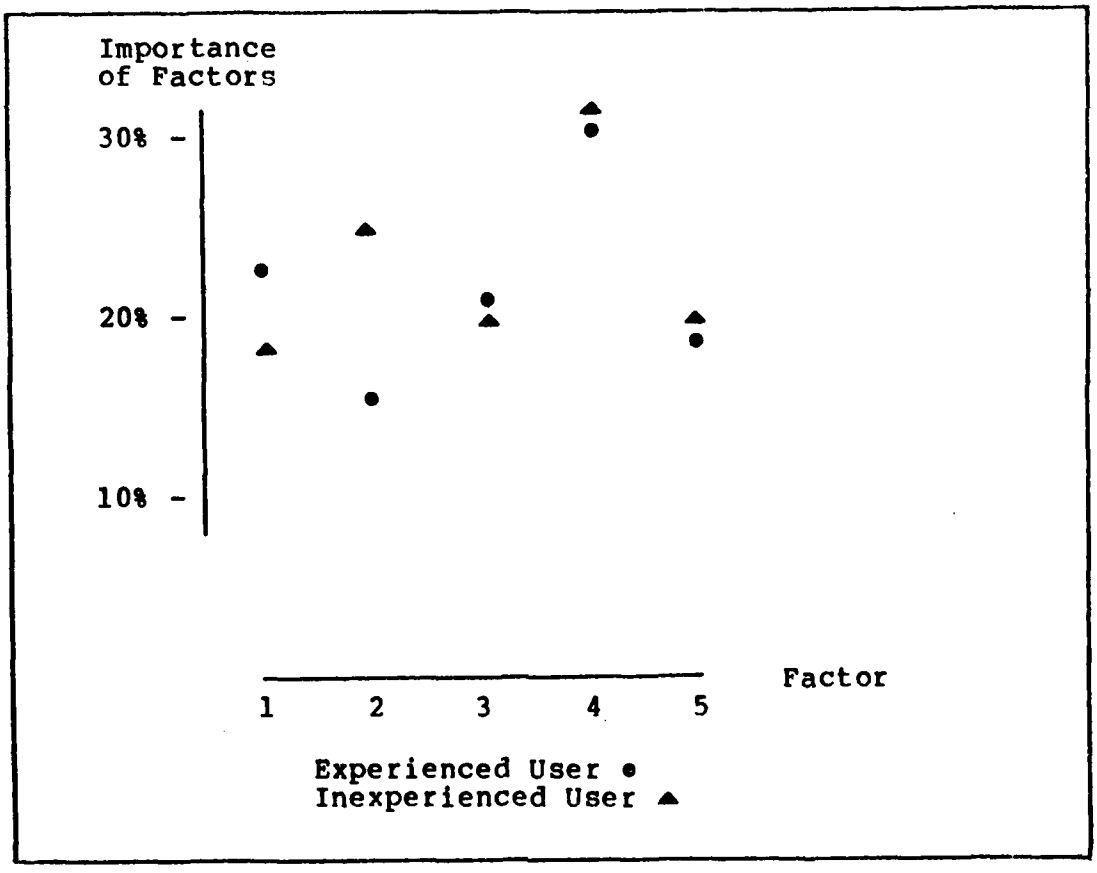


Figure B-1. Importance of User Factors

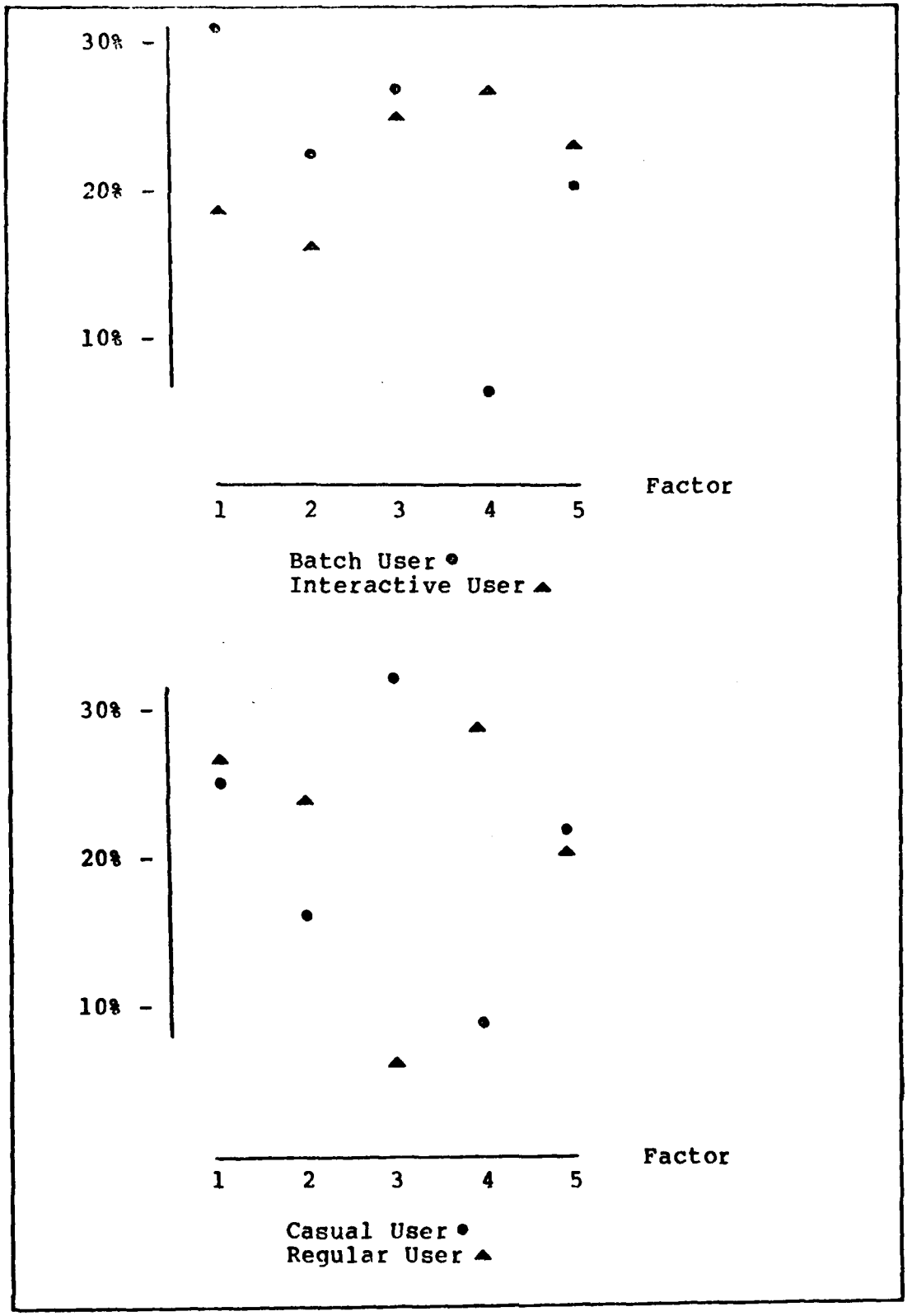


Figure B-1 continued.

factors are used because the authors found two of the factors to be questionably valid; i.e. fault tolerence and flexibility in task handling.

The authors readily admit their work does not lead to an absolute definition of user-quality. However, it is one of the very few studies in this area based on a mathematical analysis rather than speculation. Their transcript also contained one of the better bibliographies available on this subject.

Physical Characteristics of Interactive Devices

Rouse presents an interesting discussion on the physical devices used in the interactive system (Ref. 46). His work includes CRT displays, input devices, instrument scanning, and visual information processing. While the designer of an operating system has little control over the physical characteristics of the machine itself, the user's perception of the system can just as easily be influenced by the appearence of the system as how well the operating system performs. However, his paper tends to be too detailed in some respects, e.g. he specifies the optimal character matrix size for CRTs, the proper height to width ratio for characters, and preferred blink frequency for blinking cursors.

Miller also addresses the physical aspects of interactive computing and includes some suggestions to improve information transfer to the user (Ref. 34: 526). One idea involves the partitioning of the screen into

separate work areas. The possibilities include:

(1) main work area - 20 lines
(2) input preparation area - 1 to 2 lines
(3) system facility indicator - 1/2 line
(4) diagnostic area - 1 line
(5) fixed response area - 1 to 4 lines

The main work area would be for text, system menus, or a transaction record. Input preparation is for generating and editing the next input to the system. The system facility indicator is to indicate the system facility being used, such as a compiler or editor, as well as the characteristics of that facility such as compiler options. The diagnostics area could be used for error messages and conditions neccessary to recover. The fixed response area is for situations when a fixed set of responses are applicable.

USCD Pascal uses a partitioned screen to keep the user continuously informed about the state of the system and the options available in that state. A prompt line is maintained on the terminal screen listing the available options. The user selects an option by typing a single character command from the prompt line. Designers of USCD Pascal claim this feature allows a naive user to interact with the system easily and experienced users can ignor the prompt line unless needed (Ref. 39: 747). The use of a "prompt line" in this case is a combination of (3) and (5) above.

Interactive Dialogue

Most current literature agrees that the man-computer

dialogue basically differs in terms of two characteristics. First, whether the dialogue is directed by the user or the machine and second, whether the user has to determine the input from a choice of alternatives outlined by the system or the user is allowed to make a free response. The one which is the guide takes the initiative during the course of the exchange and also decides on a satisfactory termination point (Ref. 34: 523).

The two distinctions are independent of one another, therefore, four basic types of dialogues are possible. Each has its advantages.

(1) system guides - user has forced-choice
(2) system guides - user has free-response
(3) user guides - user has forced-choice
(4) user guides - user has free-response

In (1) the speed of the dialogue is increased due to the limited response of the user and the possiblity of error is also decreased. This method seems best suited to very structured dialogue or information gathering. For unstructured information gathering (2) is best suited. In (3) the user is at least somewhat knowledgeable about the possible requests he could make of the system. This dialogue style is often chosen for allowing a user to select desirable system alternatives. The maximum latitude for the user is provided the user by (4). It is most appropriate for experienced and confident users performing complex tasks. However, this situation also is the least structured and with current technology allows the maximum opportunity for errors between the user and the system. For variations

on these types of dialogues see Martin (Ref. 33)

Conclusion

A properly designed dialogue can enhance the productivity of a computer system by promoting active communication between the user and machine. The dialogue style should be tailored to the users and the situation.

In summary, studies suggest that users of practically all backgrounds prefer the computer operating system to concentrate on intrinsic qualities rather than technical ones. The user is concerned with the programming problem at hand, not the obstacles caused by a poor operating environment.

Appendix C

Computing System Environments

Introduction

A special interest in any operating system is the environment created for the user. This appendex discusses some of the issues concerned with the user environment, its involvement in productivity, and relationship to the operating system itself.

This appendix is related to the material in Appendex B, Man-Machine Interface Issues, but deals with the subject on a much higher level. Chapter Three introduced the concept of user friendliness and relates to the subject of functional requirements in the computer environment.

Creating a Graceful Computing Environment

It seems ironic that a computer able to display complex information and perform difficult calculations, compared to human capacity, has such a difficult time communicating with that same human being. However, most computing systems are not very good at communicating with the human user. Communication with computer systems is often time consuming and frustrating because the system does not understand the user and the misunderstanding cannot be made known to the user by the system.

As indicated in Chapter Three, the user represents an implicit interface to an explicit machine. Timesharing systems typically have very structured command languages

which operate contrary to human conversation. The computer concentrates on the specific command, whereas the human concentrates on the context of the conversation. Too often, a system will respond only to commands phrased exactly as required by a strict syntax for a particular computing system. Compare this to the communication environment used by humans. A syntax error made by one person talking to another is not totally rejected, but usually corrected automatically by the receiver and the conversation never breaks its pace. The receiver understands despite the syntactical error.

A study conducted at Carnegin-Mellon University (Ref. 17: 19) identified seven capabilites an interactive system should have to provide the communication needs of a human.

(1) Flexible Parsing
(2) Robust Communication
(3) Identification from Description
(4) Focus Tracking
(5) Natural Output
(6) Explanatory Facility
(7) Personalization

The seven capabilites are based on two assumptions. First, humans have basic conversational needs when they communicate with other people. Second, computers are hard to communicate with because they do not fill those needs. Flexible parsing refers to the ability to allow minor syntax errors and permit the user the opportunity to correct the error or choose alternates if the mistake presents different possible interpretations. Robust communication is the ability of a system to correct misapprehensions the user may

have and clearly define the understanding between the user and computer system. Without verbosity, the system must make clear to the user any assumption made during the conversation. Identification from description is a system attribute of recognizing objects known internally from a user description of the object. Focus tracking involves following the context of a user as the dialogue changes. The system should track the attention of the user even across large spans that occur during command sessions and return the focus back to the original context. Thus, a command can be broken out of, another can be executed and the user can return to the context of the first command. Natural output means the output should be appropriate and contain a sufficient amount of detail. An explanation facility is broken down into two categories, static and dynamic. Static explanation relates to the capabilities of the system. Dynamic explanation relates to what the system is doing, why, and explanation of past events. Personalization refers to the systems ability to sense the peculiarities of each user. For example, constant syntax errors should be pointed out and messages or responses should be adjusted to the user experience.

These attributes add up to what may be considered a "graceful" interactive language. However, this does not imply a natural input/output language. A command laguage interface that assumes an implicit input language and provides a tabular output can be made graceful as well.

Several means are available to communicate with the user and achieve the seven capabilities mentioned above. They are best used in combination and consist of:

(1) Animated Sequences
(2) Hand Drawn Sketches
(3) Voice Annotations
(4) Facsimile Images

Animated sequences may consist of a time-syncronized set of graphic displays such as cursor motions, highlighted text, and recorded speech. Hand-drawn Displays allow the system to control cursor position to select objects that were displayed earlier. This may involve the use of a joy stick or a mouse as used in Smalltalk (Ref. 53). Facsimile images refers to printed material entered into the system through a low cost optical scanner. Voice annotations are easily possible even for the most inexpensive computer as a result of developments in speech synthesis circuits. While some of these capabilities appear too elaborate for a normal interactive user, the potential of multimedia communication may be too great to completely ignore as a partial solution to solving the problems in the computing system environment.

The UNIX Computing Environment

UNIX was introduced in Chapter One and again in Chapter Two as an example of a friendly computing environment. Essentially, UNIX was written by programmers and for programers (Ref. 43). Every object in UNIX looks like a file, including all input/output channels. This makes it possible to for input/output to be taken from any program to

or from any file or input/output device without special planning by the programmer. In addition, UNIX provides a hierarchical file structure to allow users to position themselves anywhere in the data structure to work with near by files. The system keeps track of the current directory the user is referring to for each file command.

The "shell" of UNIX is the command interpreter. The shell can take input from a variety of sources. Input can come from a file which provides an easy means of invoking a frequently used set of commands. The commands are listed in the file and calling the file invokes the string of commands. This technique can produce a specialized command for a particular user. Each user can build a set of commands based on programming needs. The shell also allows iterations of commands, conditional operations, and passing arguments from the result of one command to the input of another.

UNIX provides communication channels called "pipes" which allow the output from one program to be easily directed to the input of another. Therefore, a sequence of programming modules can be strung togeter to do some task that would require special purpose programs in other system environments. UNIX is just the operating system and does not provide special purpose programs. Instead, it attempts to provide a set of basic software tools that can be strung together in flexible ways using input/output redirection, pipes, and shell programs to create a programmer's

workbench.

However, there are several problems with UNIX (Ref. 37). First, the language, functions, and syntax are inconsistent. Second, the syntax, command names, and formats seem to have little relationship to their functions. Third, the lack of interaction makes the state of the system hard to determine. The operation of the system is so hidden from the user that UNIX becomes recluse. Finally, learning the system can be very difficult for novices. There is a lack of sensible mnemonic structures.

UNIX is a powerful computing environment. However, as the author of "The Trouble with UNIX" (Ref. 37) observed, operating system designs should not be for the computer and not even for the designer, but for people.

Designing Environments

Not knowing how difficult a system is to use until after it is completed can be a costly experience. Usually, redesign and system modification are the result. Schorer (Ref. 48: 77) suggests three techniques for designing system environments.

(1) Designing a software system is designing a behavior field for the user. A sociology of interaction exists between the user and the system.
(2) The "use" of the system must be structured. Good environment does not necessarily result from structuring the system itself.
(3) The major issue is coordinating the "use" of programs, not writing programs.

The basic philosophy underlying Schorer's approach is process-oriented rather than object-oriented view of

software systems. The object-oriented view is characterized by the belief that the most important idea is the hardware or software itself, rather than its use. The use is determined after the hardware or software is completed. The process-oriented view is characterized by the designer who attempts to engineer the "use" of the system. The system cannot be described without describing its use.

Since the use of the system is primary, the largest class of intended users (LCIU) should be identified. The designer can then use "environmental consciousness" to continually observe and record the needs of the users and when in the environment the users have these needs. This reduces to what use is needed when. This may point to possible subenvironments.

This design method can be summarized by three points. First, define the minimun each user is expected to know (the LCIU). Second, define the major functions the users will perform in the environment. Third, evolve the environments and subenviroments by considering the designer as a user and test the system against members of the LCIU. Improve the system as needed. Essentially, the designer is building human factor research into the design itself.

Conclusion

The user environment is of primary importance to the operating system. Design of an operating system for a specific group of users must first satisfy the needs of the user environment. This environment should include a people

oriented command language and programming facilities as demonstrated by UNIX files and pipes for maximum productivity.

Appendix D

Hardware Configuration

The hardware configuration is relatively fixed based on available resources. The heart of the system, the Intel 8086 microprocessor, was mentioned in Chapter One as the first powerful 16-bit microprocessor. The CPU card, SCP 200B is configured to the IEEE-696 standard S100 bus as are all the cards being used. The CPU runs at 8 MHZ and has an onboard 24 MHZ clock. The 8086 allows three clock cycles for memory access. At 8 MHZ, 250 ns memory is required (Ref. 21).

The CPU Support Card, SCP 300F, has one parallel input, one paralled output, and one serial port. A time of day clock and two 16-bit wide timers are on the Support Card. A vectored interrupt controller provides fifteen levels of vectored interrupts expandable to sixty-four through slave controllers (Ref. 50: 6-22).

The Multiport Serial Card, SCP 400B, has four serial ports, four programmable channels, and four handshaking lines per channel. Band rates to 19,200 can be selected for either terminals or modems. The interrupt controller may be slaved to the CPU Support Card for fully vectored operation or used in the polled mode.

The system configuration is indicated in figure 3. The original system will support four users but additional hardware may be added to support more. The current memory is 48K but more may be added to support an increased load of

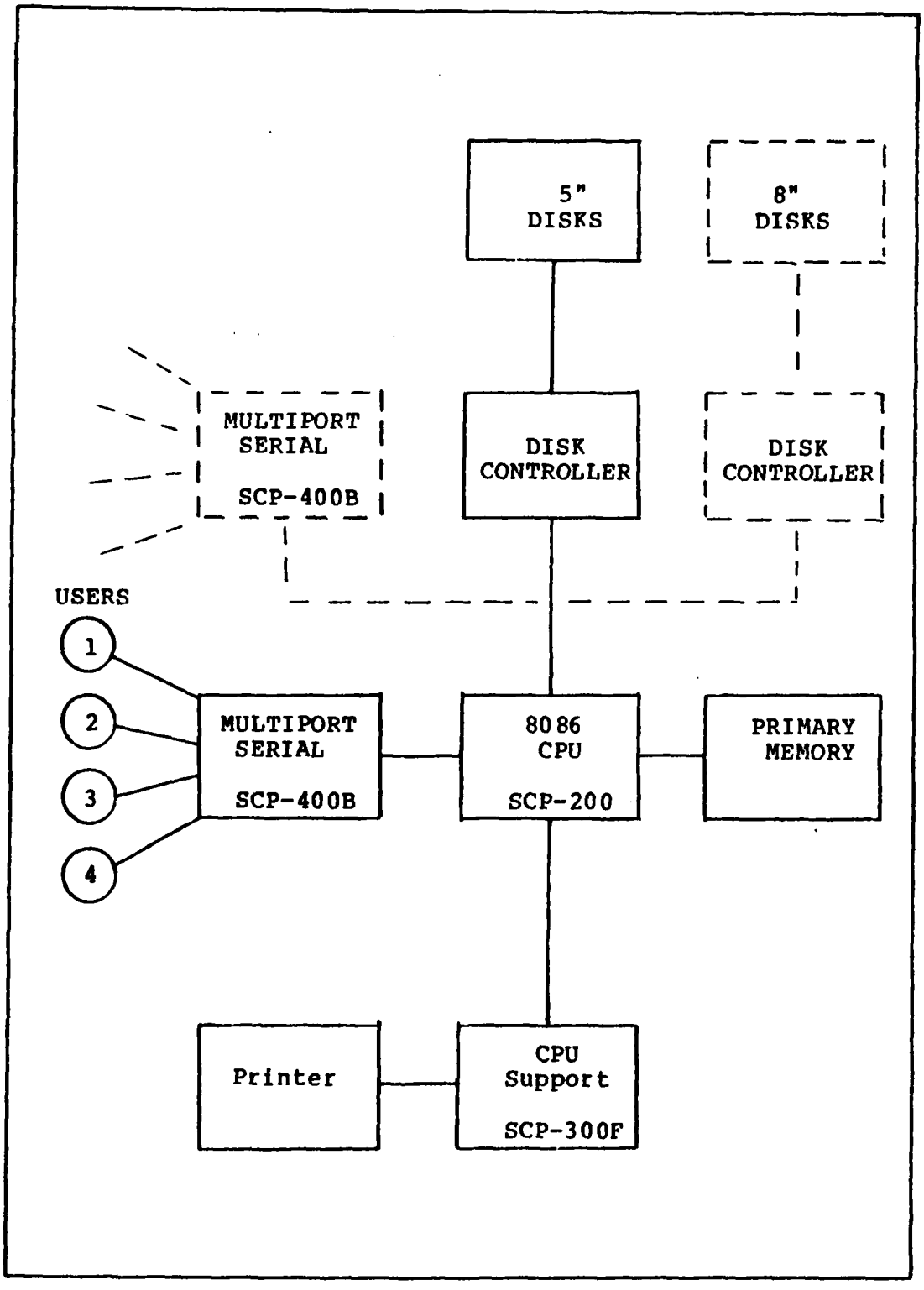


Figure C-1. Hardware Configuration

processes. The number of peripheral devices are not limited to those indicated by figure 3, but are currently limited to the hardware available.

Appendix E

Structured Specification

This appendix contains the structured specification for the operating system under development. Figures 6-42, the data flow diagrams, are reproduced and the data dictionary follows. The data dictionary contains the process specfications, file definitions and data dictionaries for each layer of the operating system.

Notation for the data dictionary:

=	means	is composed of
+	means	AND
[ ]	means	choose one of (exclusive OR)
< >	means	at least oneof (inclusive OR)
( )	means	optional
{ }	means	interations of

Index

Structured Specification Definitions

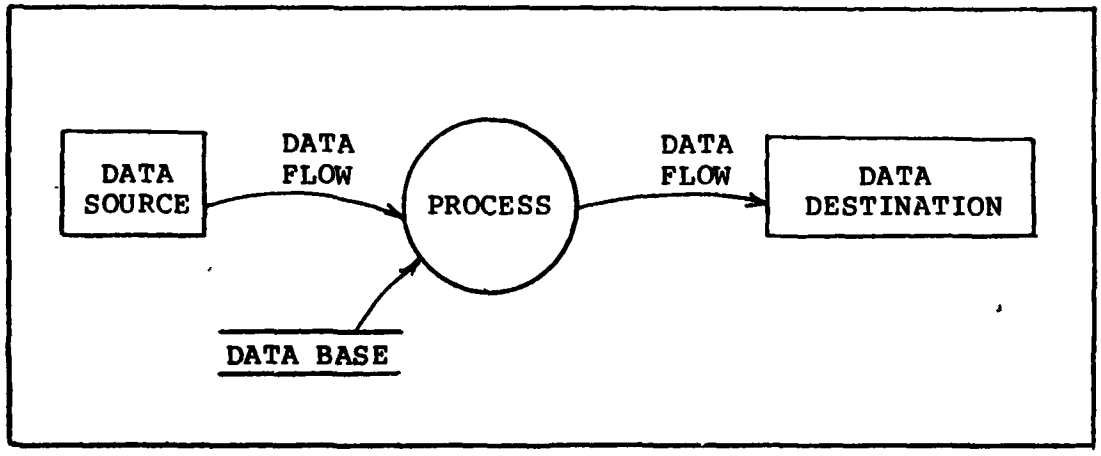


Figure 6. Data Flow Diagram Symbols

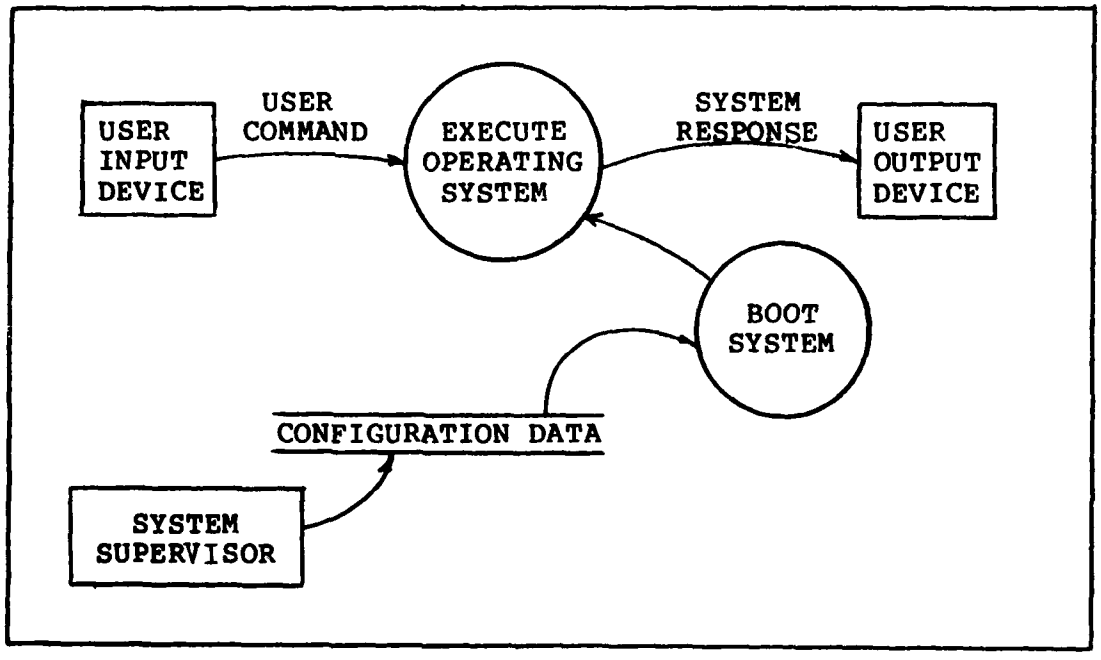


Figure 7. Operating System Context Diagram

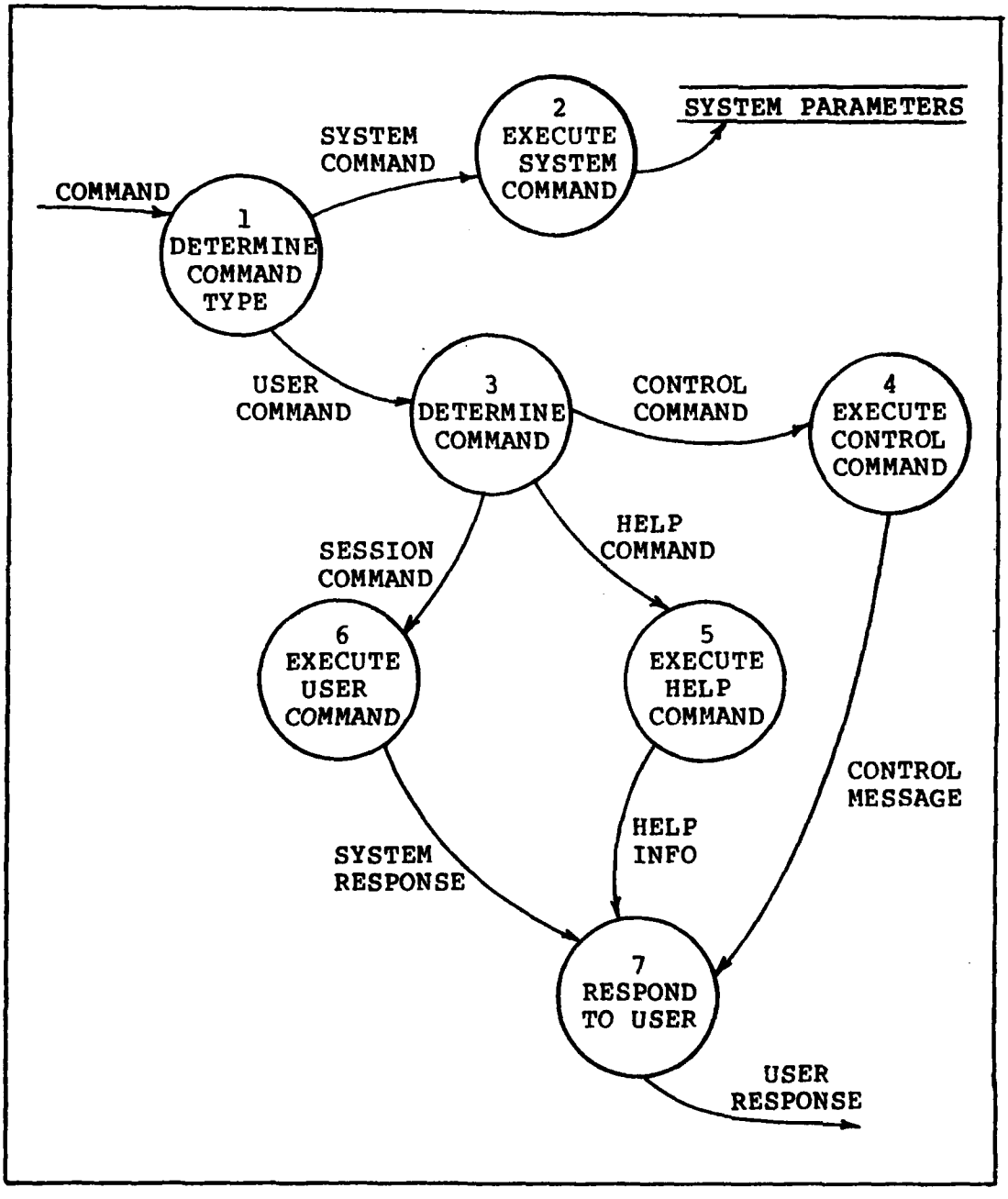


Figure 8. Operating System Shell Diagram

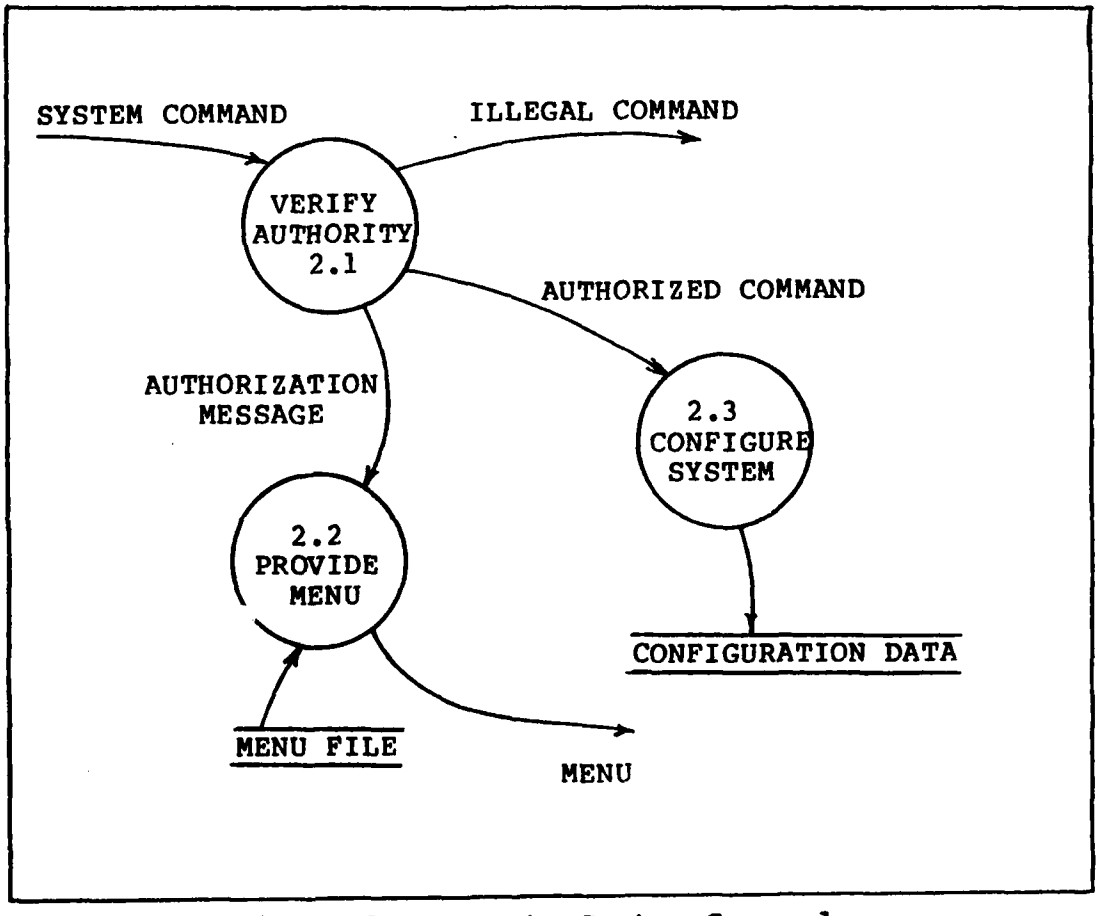


Figure 9. Execute System Command

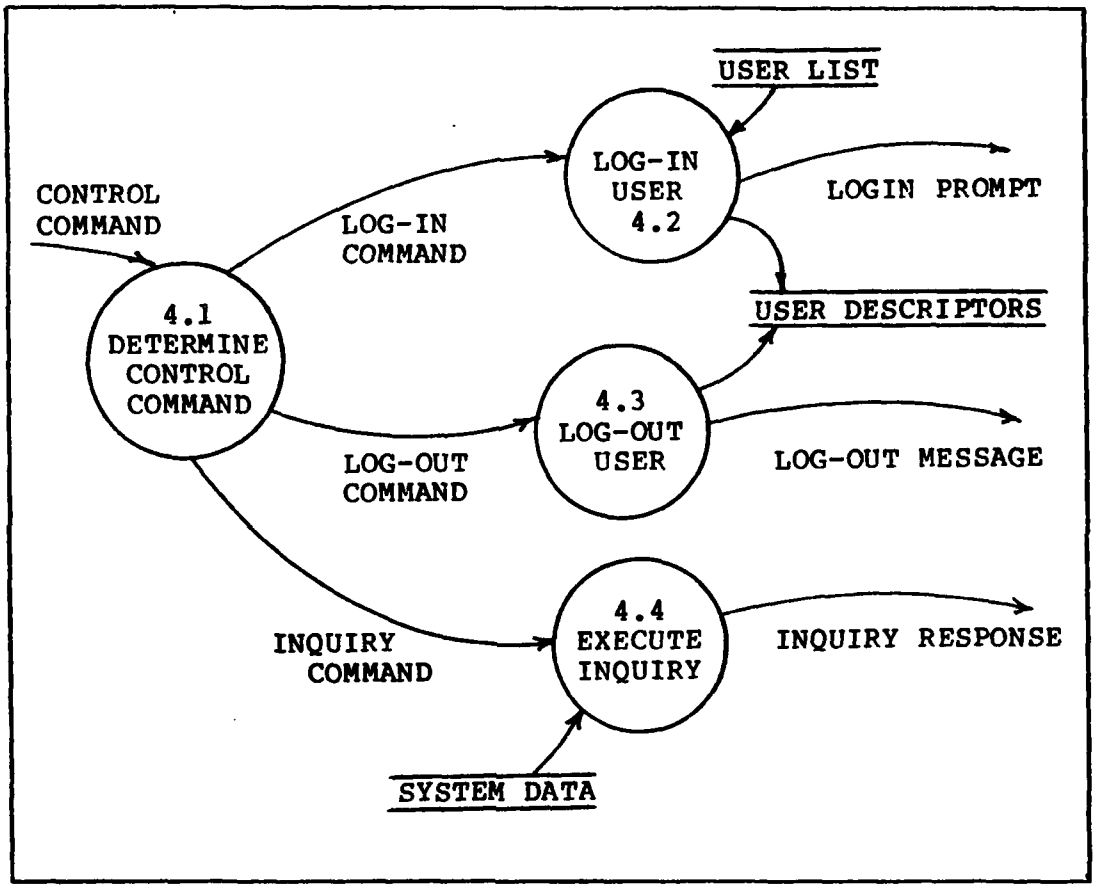


Figure 10. Execute Control Command

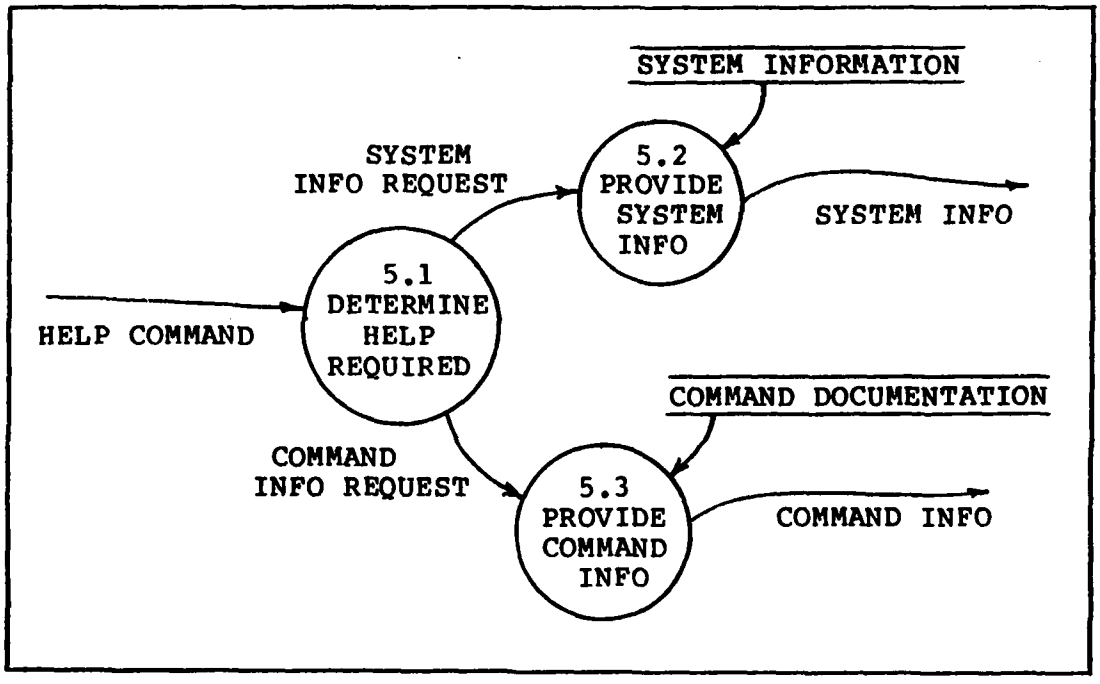


Figure 11. Execute Help Command

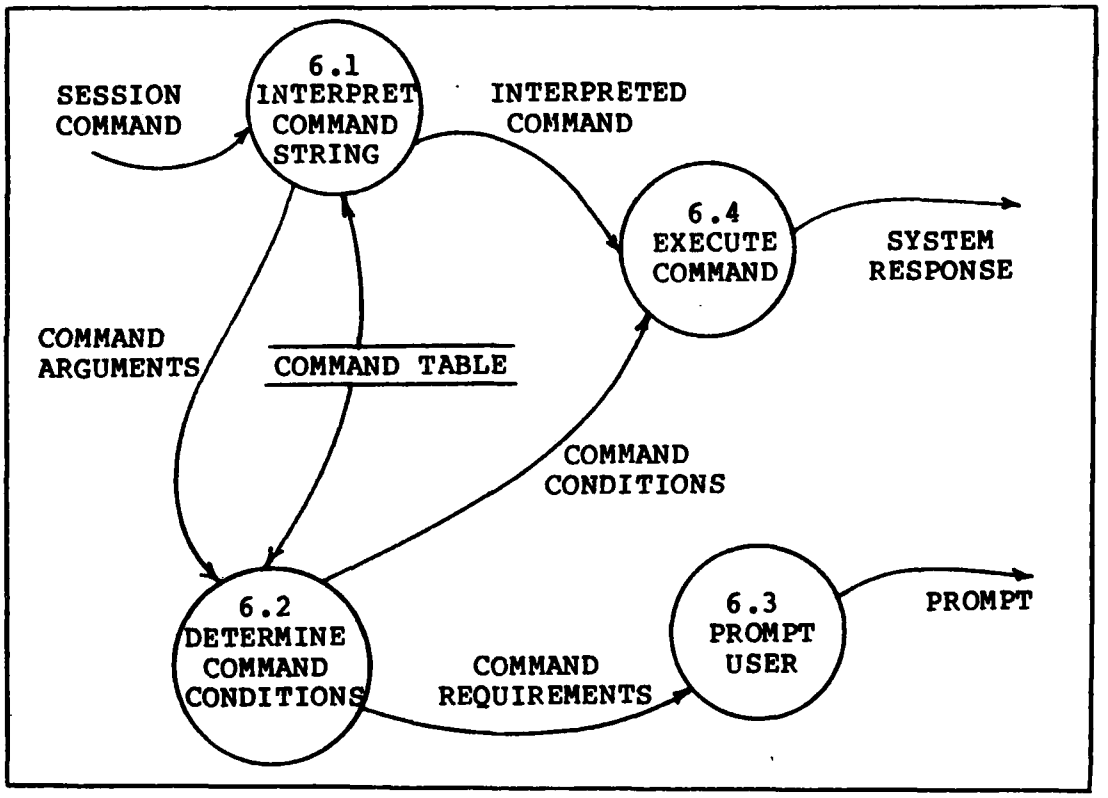


Figure 12. Execute User Command

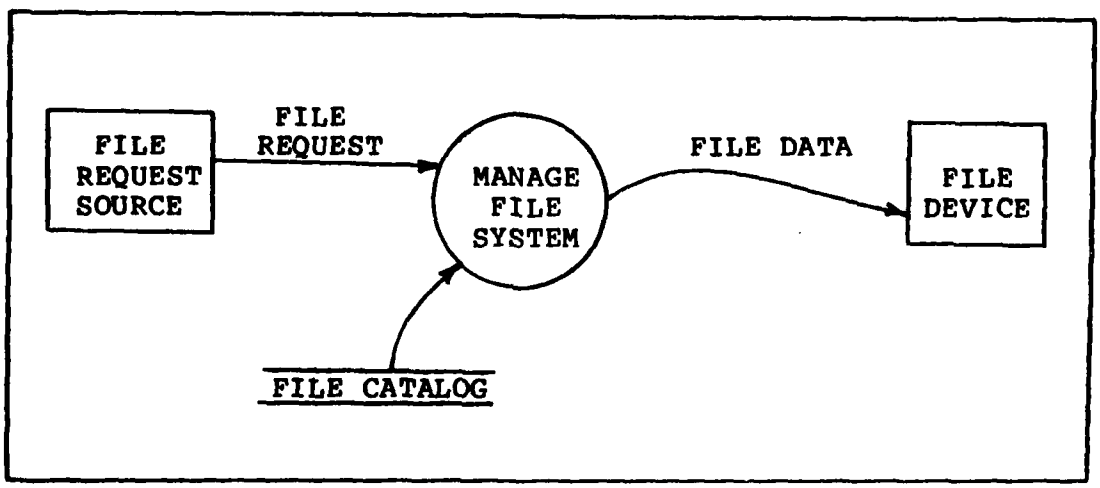


Figure 13. File Management Context Diagram

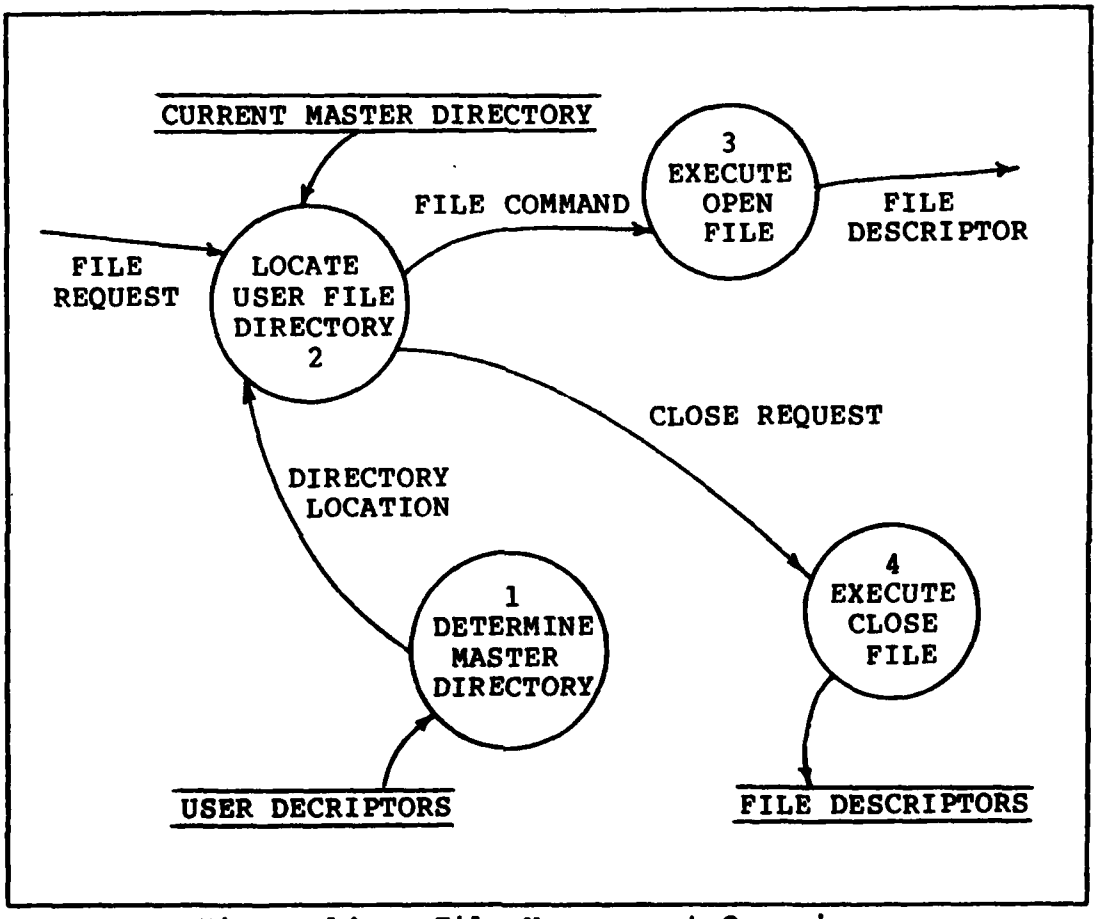


Figure 14. File Management Overview

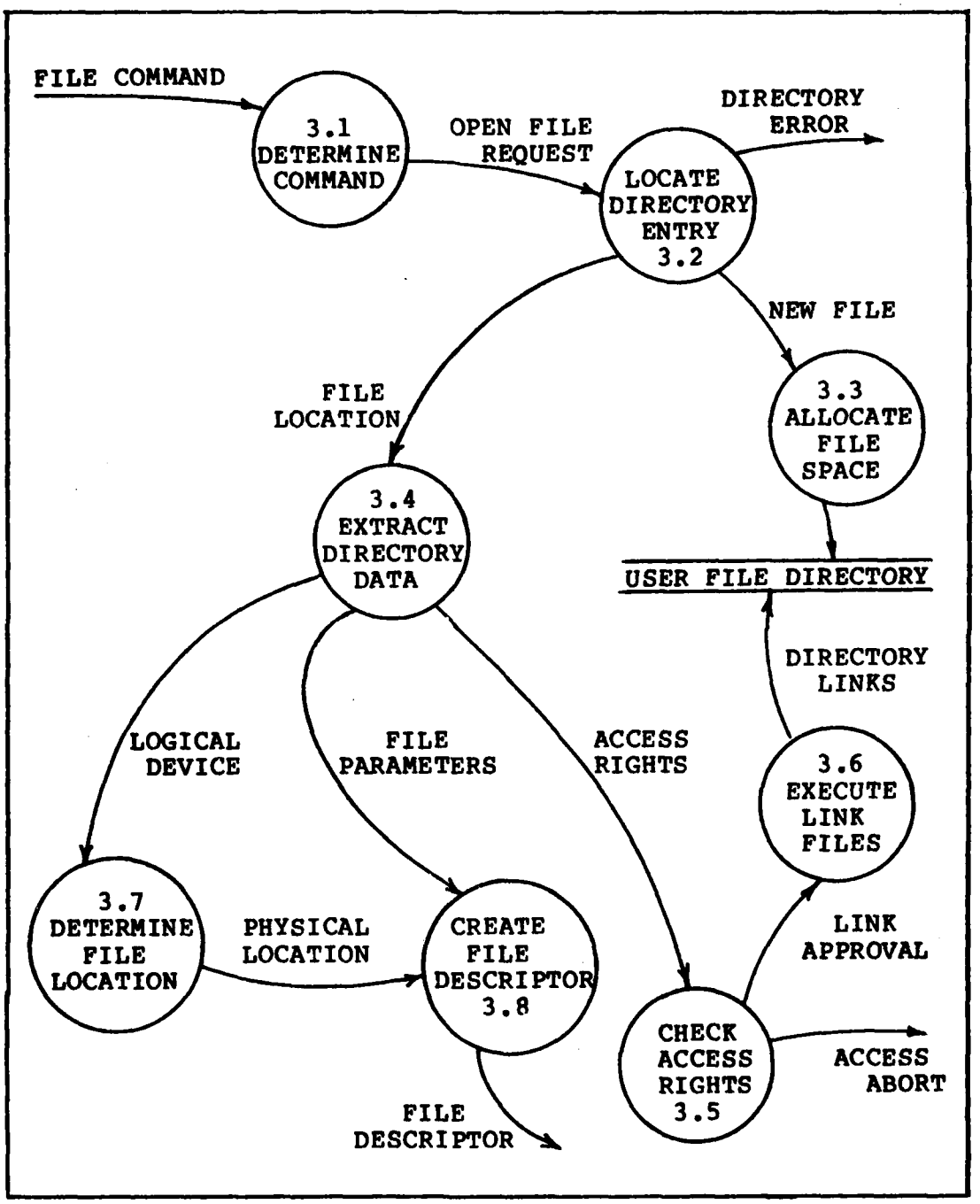


Figure 15. Execute Open File

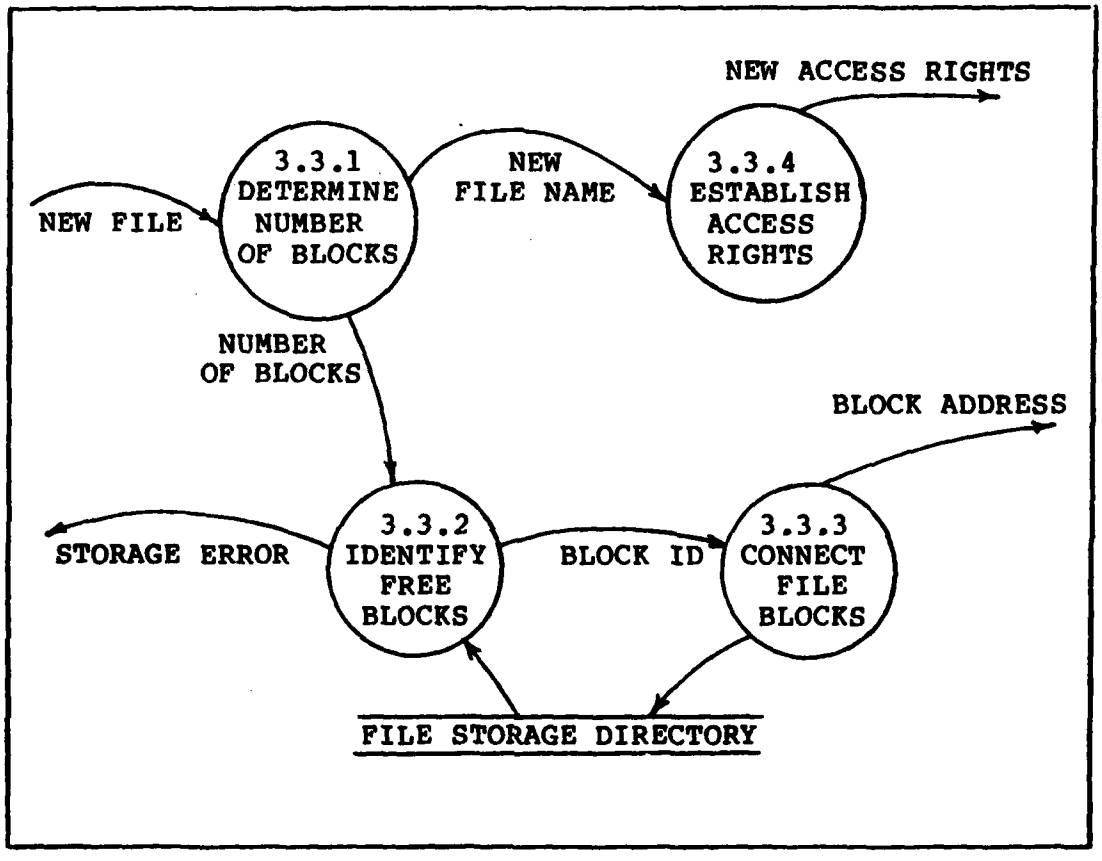


Figure 16. Allocate File Storage

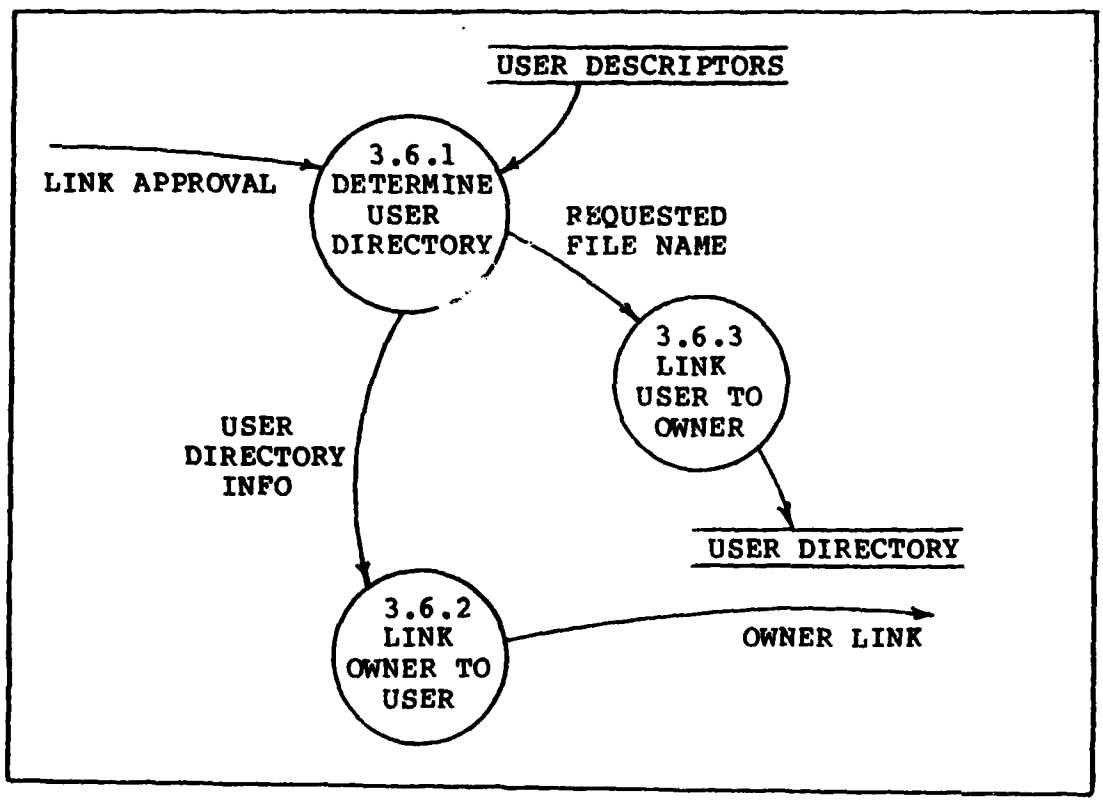


Figure 17. Execute Link Files

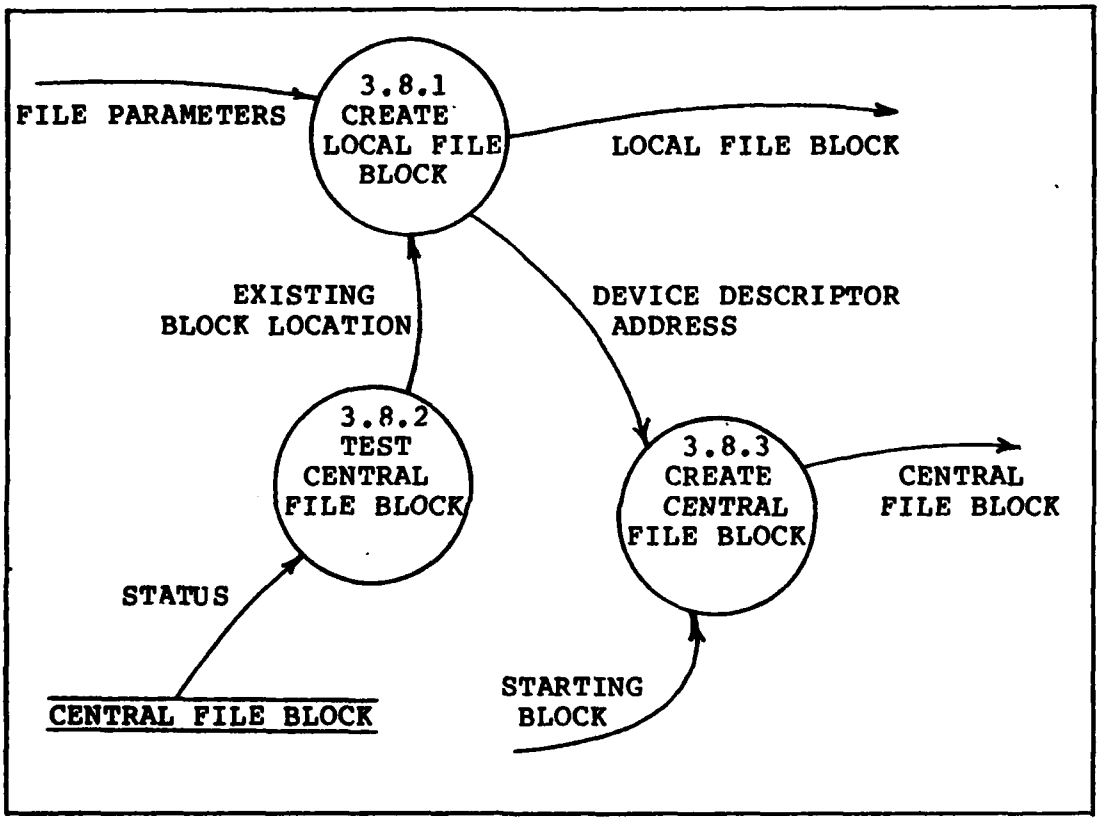


Figure 18. Create File Descriptor

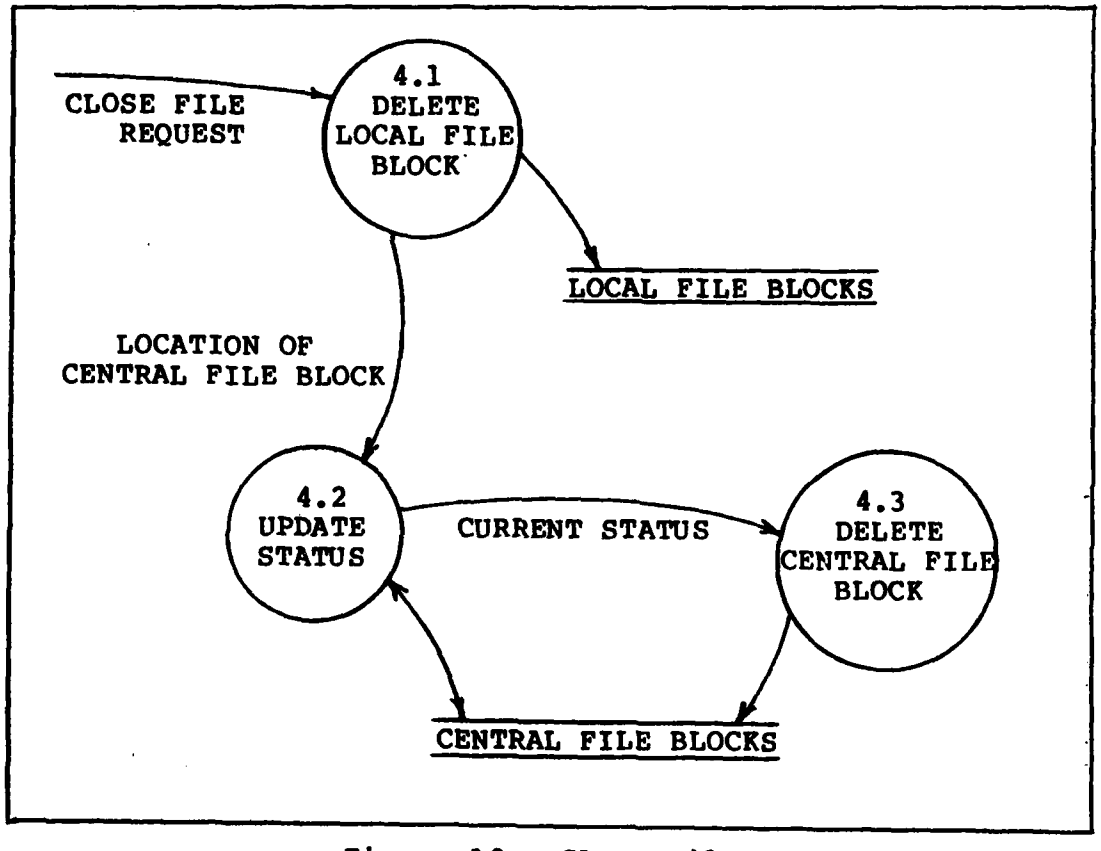


Figure 19. Close File

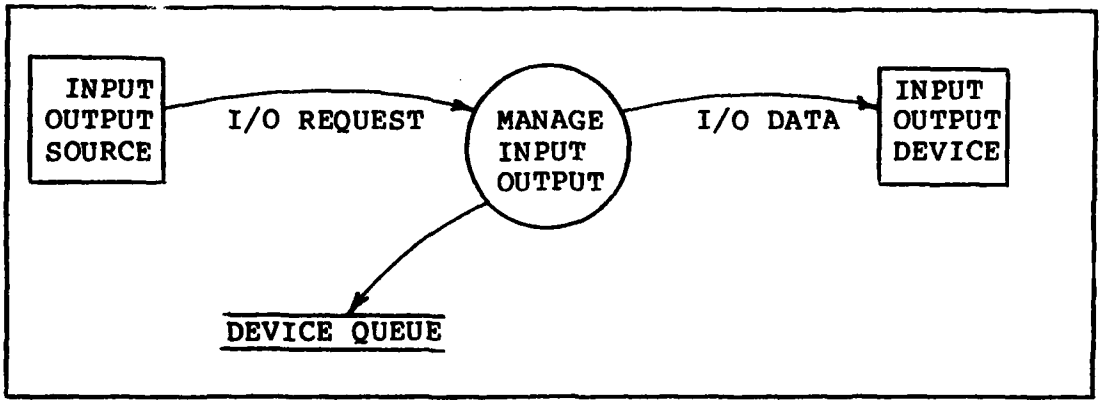


Figure 20. Input/Output Management Context Diagram

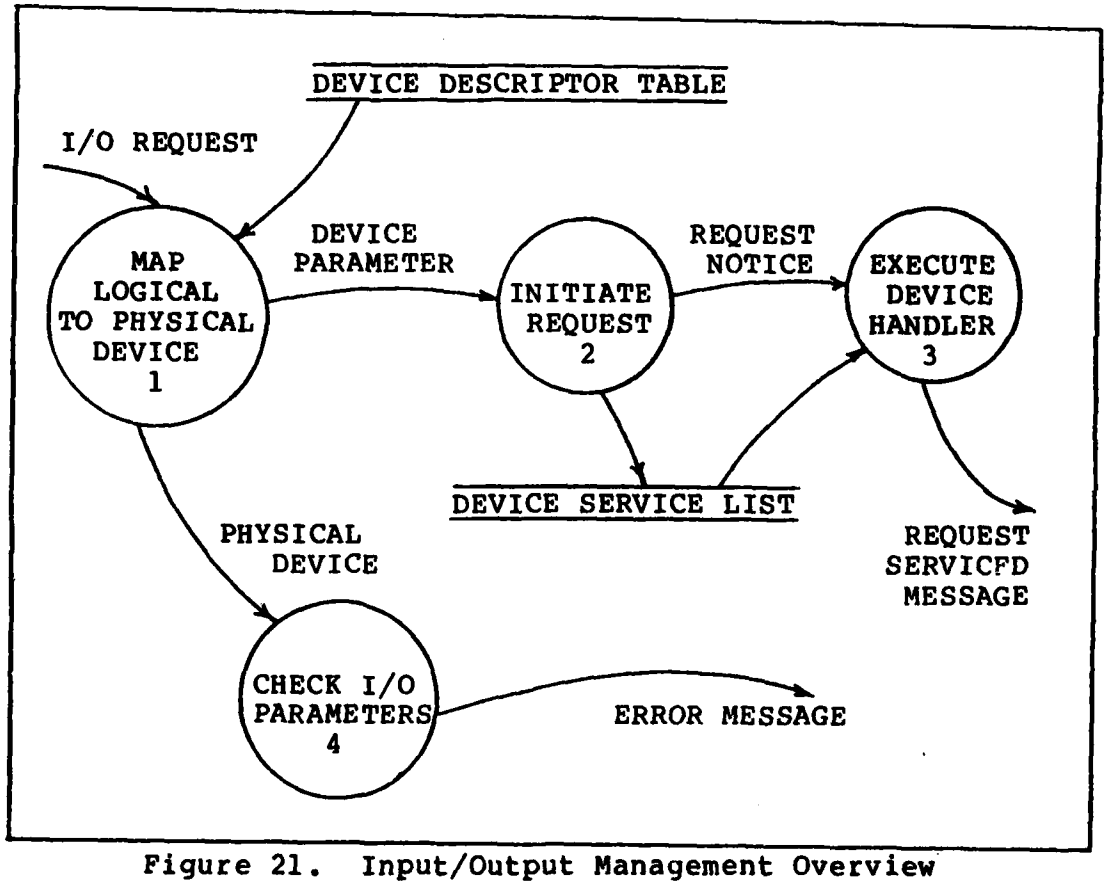


Figure 21. Input/Output Management Overview

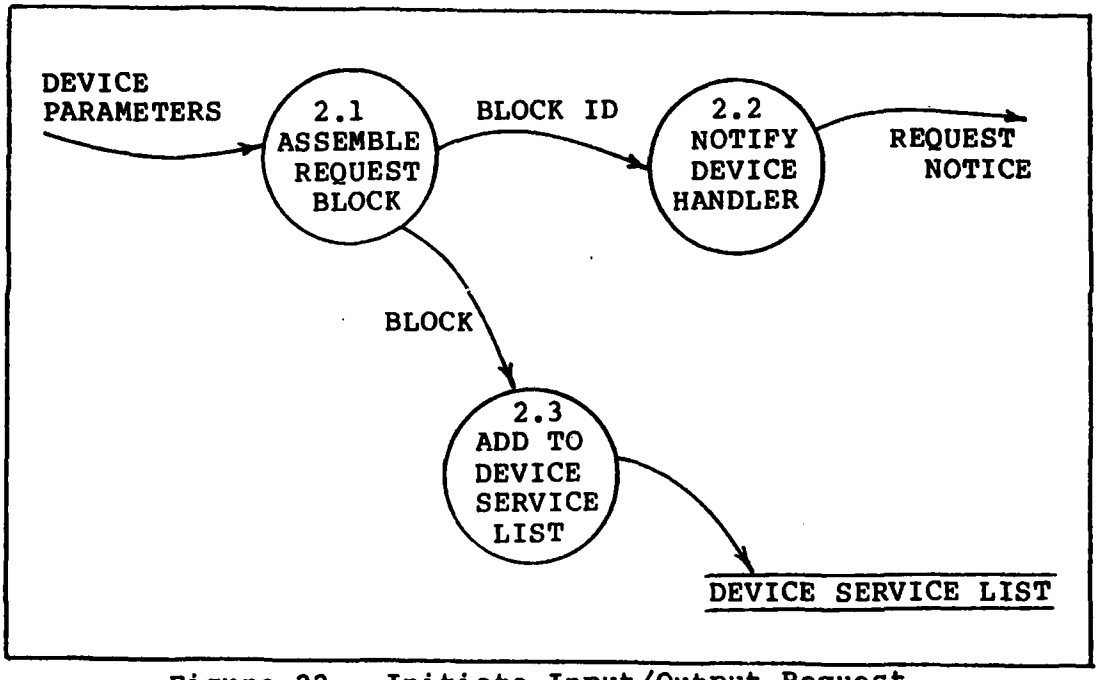


Figure 22. Initiate Input/Output Request

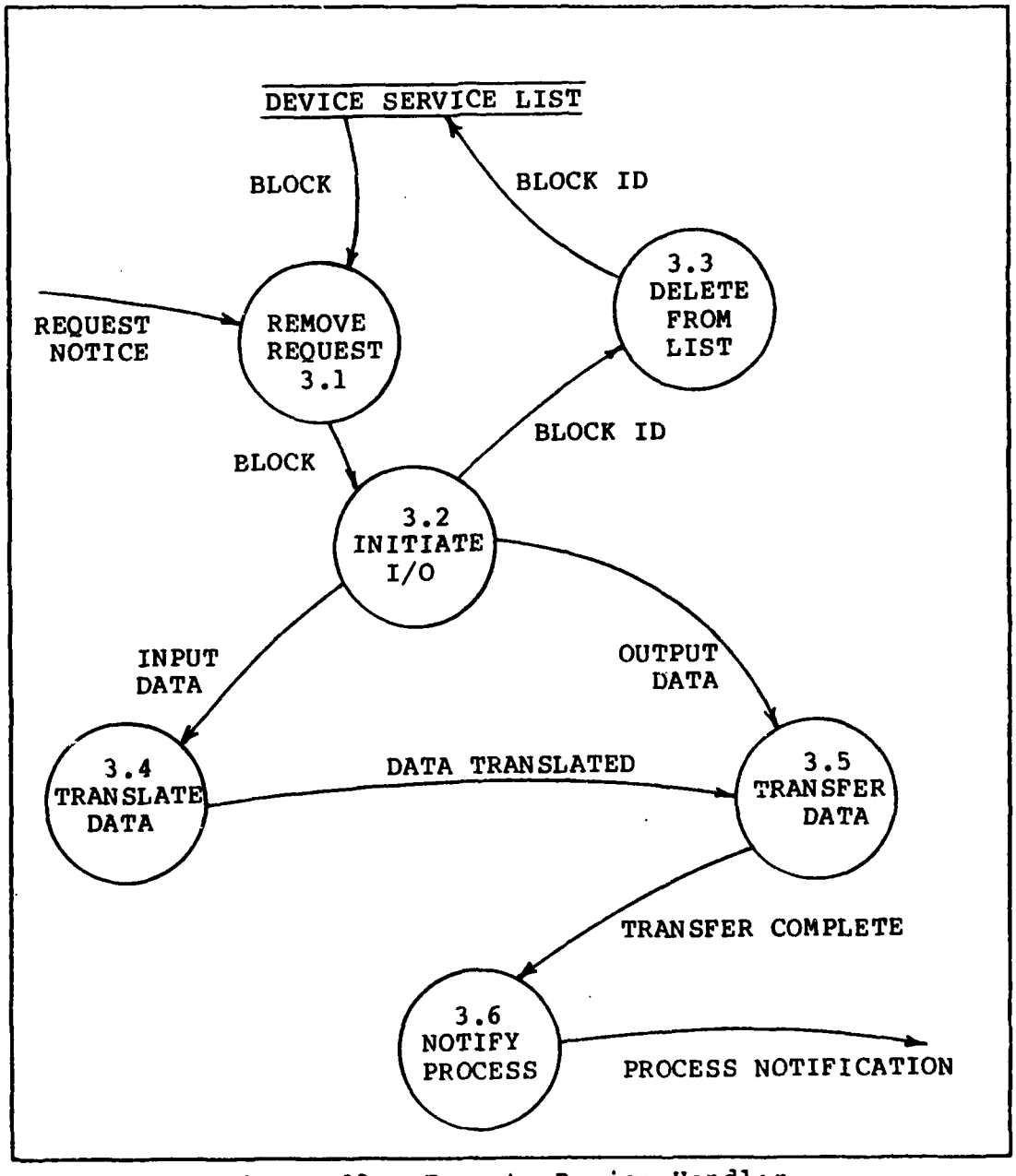


Figure 23. Execute Device Handler

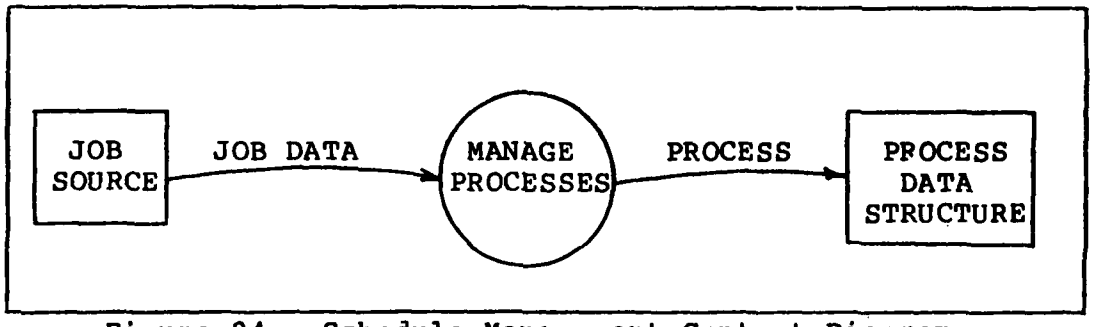


Figure 24. Schedule Management Context Diagram

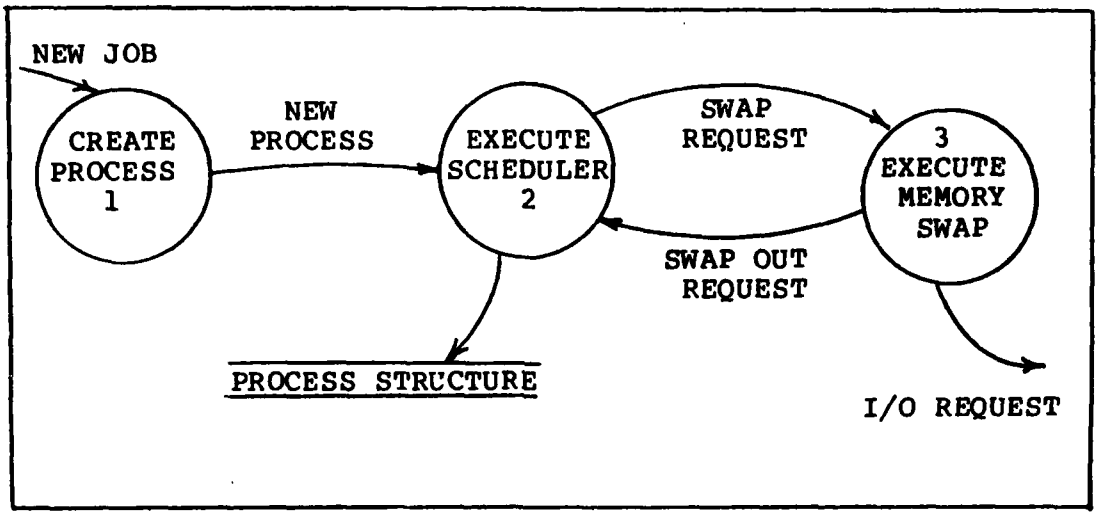


Figure 25. Schedule Management Overview

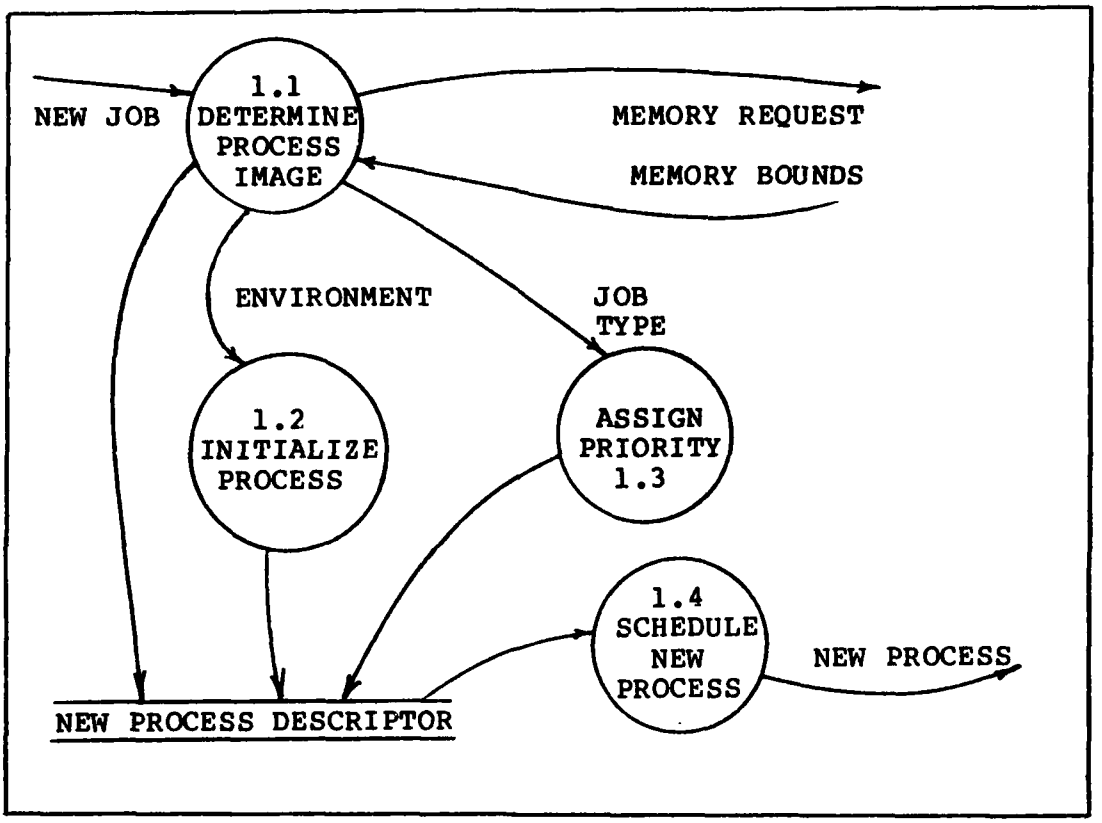


Figure 26. Create Process

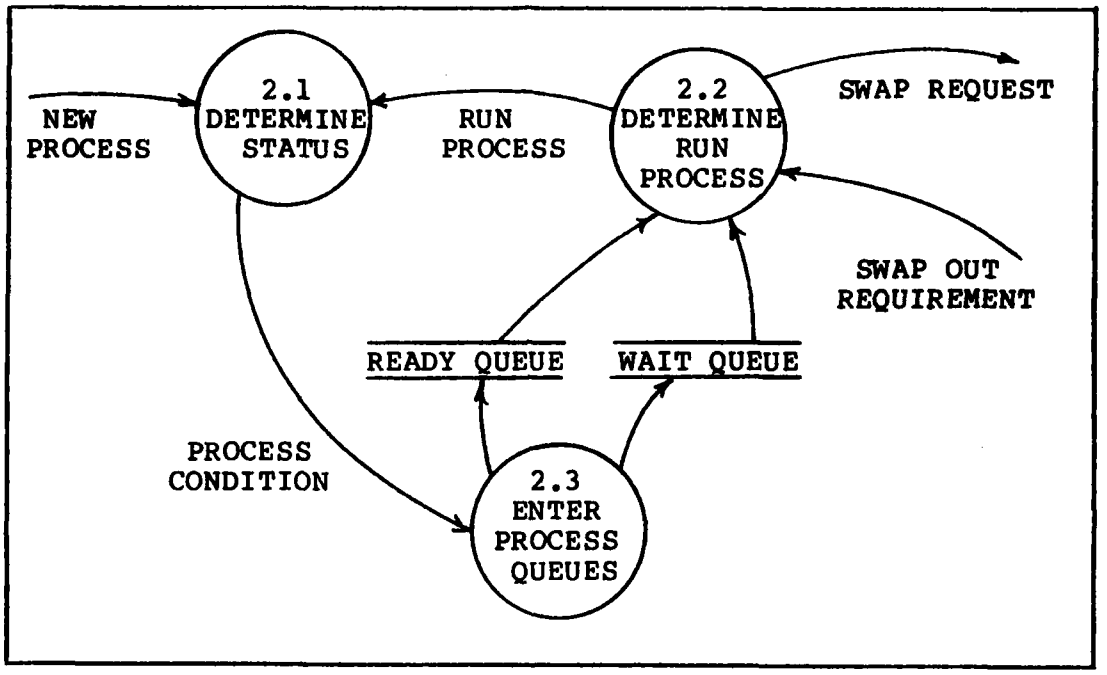


Figure 27. Execute Scheduler

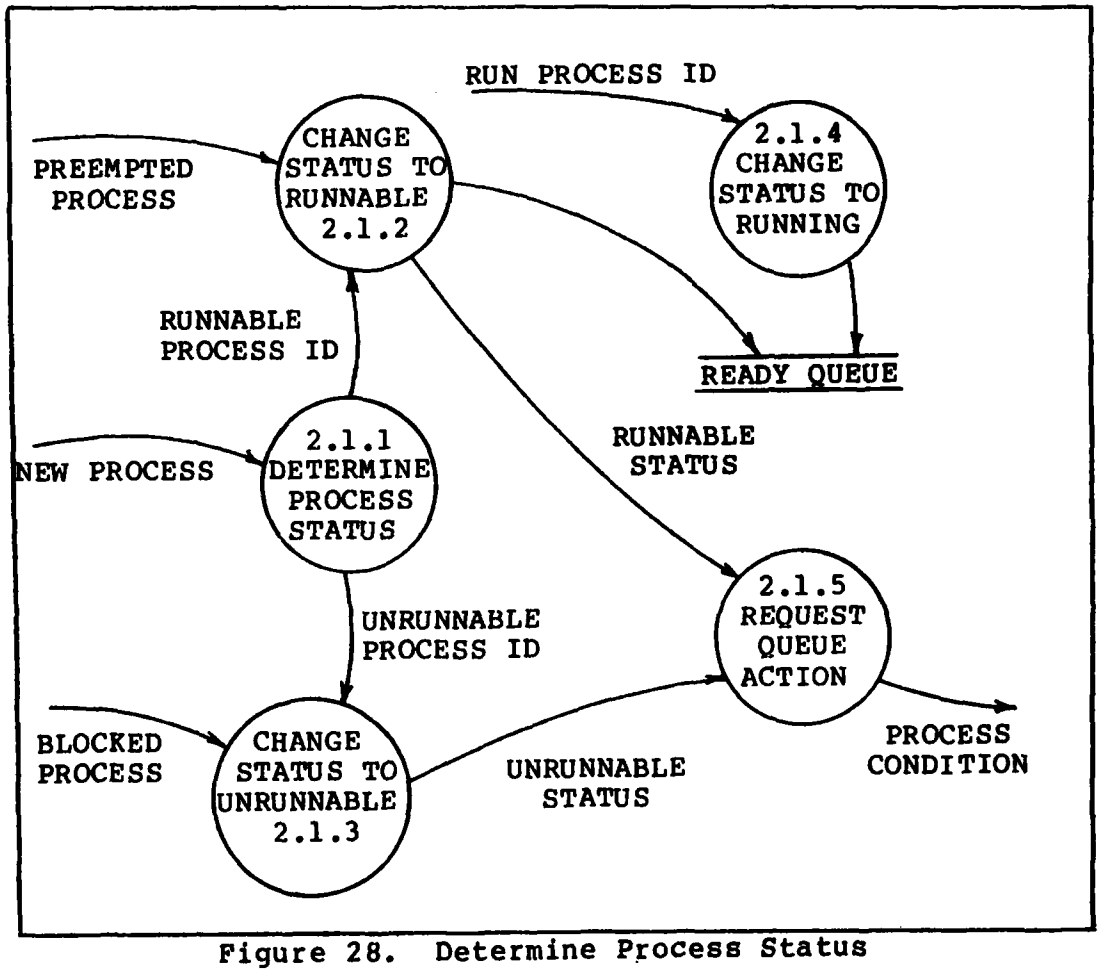


Figure 28. Determine Process Status

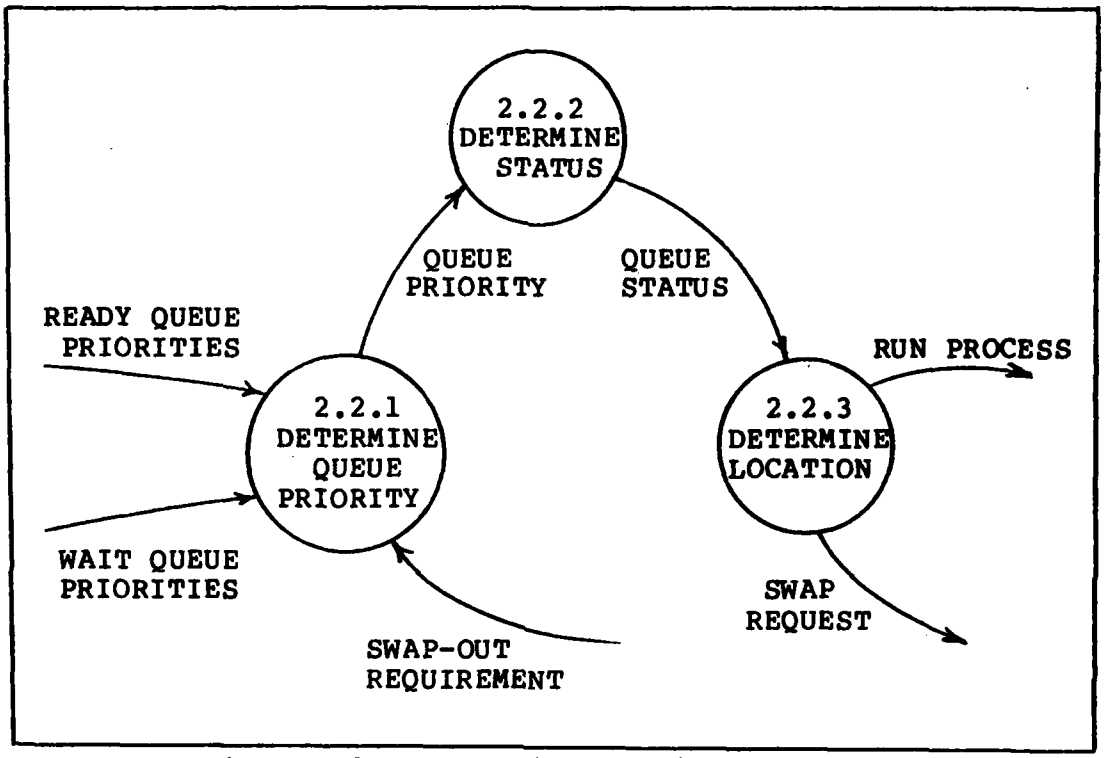


Figure 29. Determine Running Process

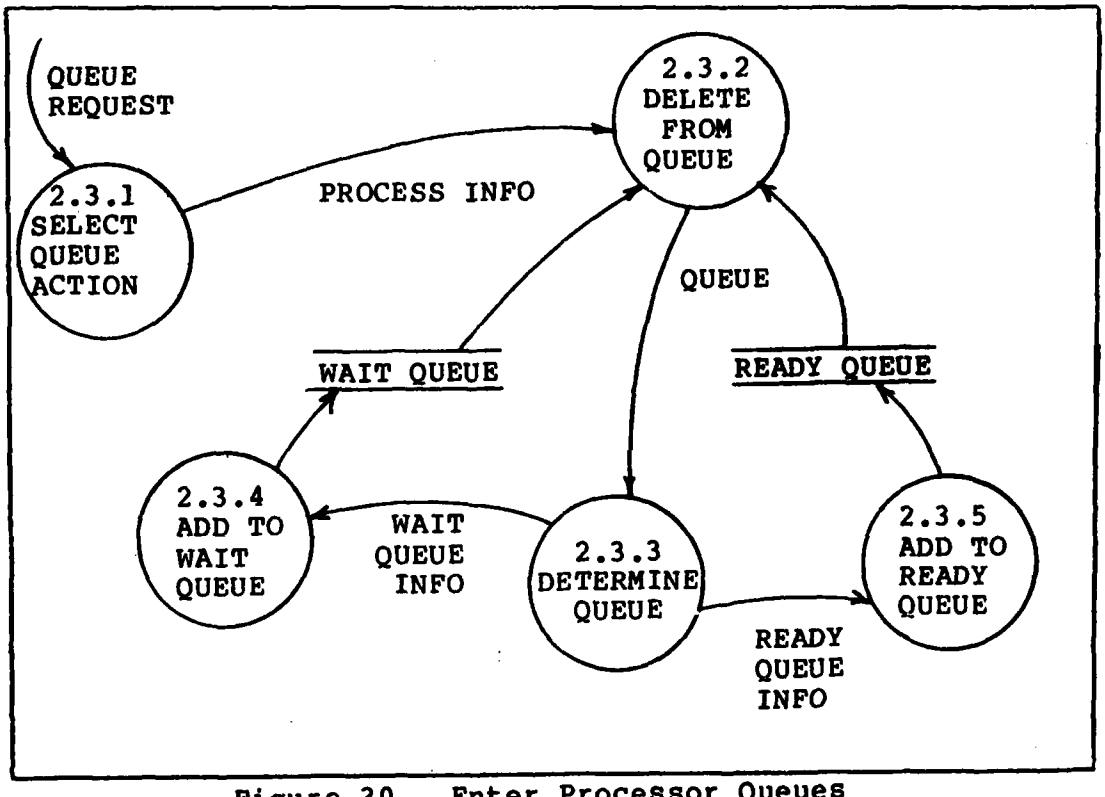


Figure 30. Enter Processor Queues

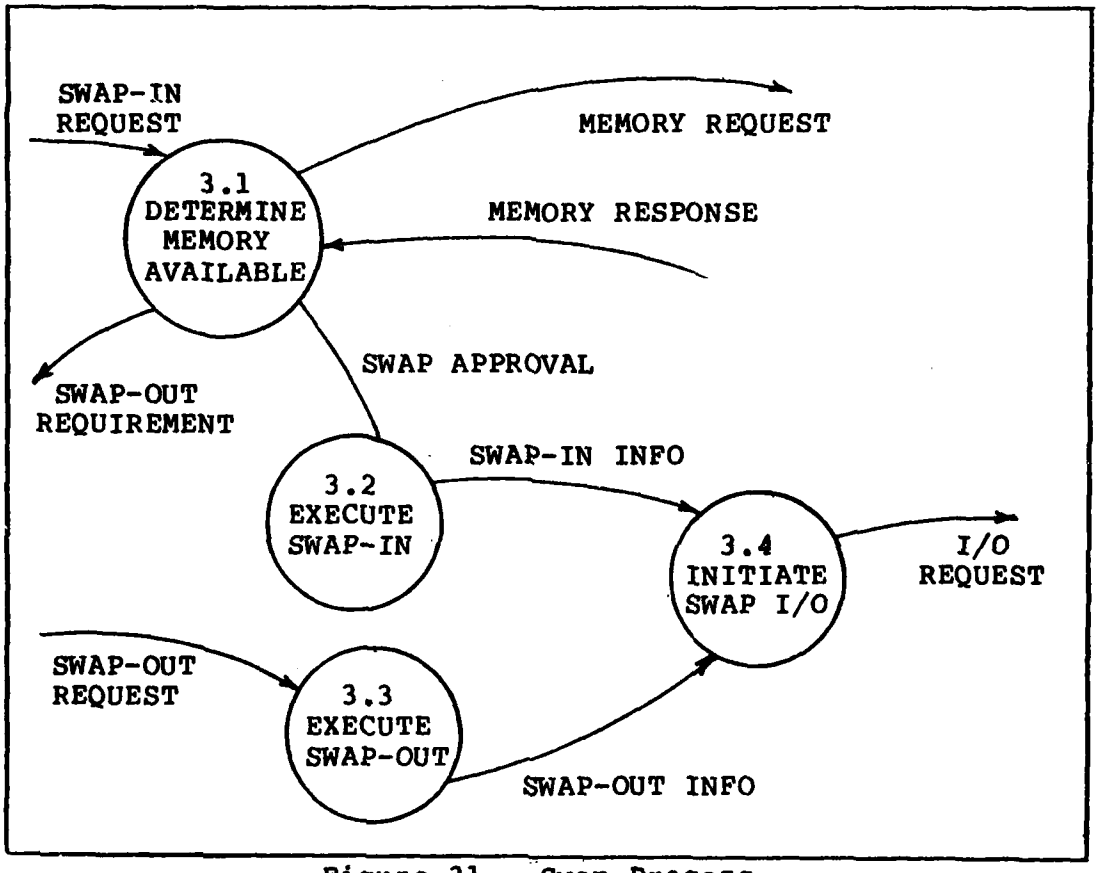


Figure 31. Swap Process

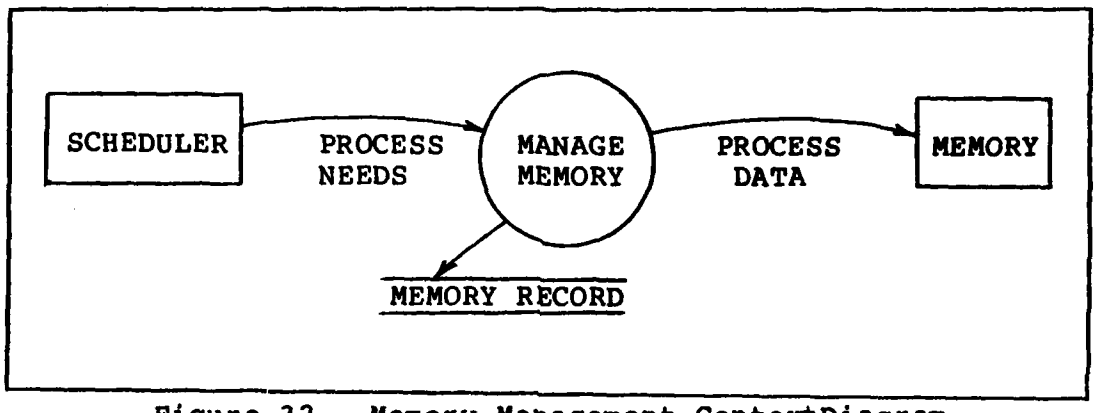


Figure 32. Memory Management ContextDiagram

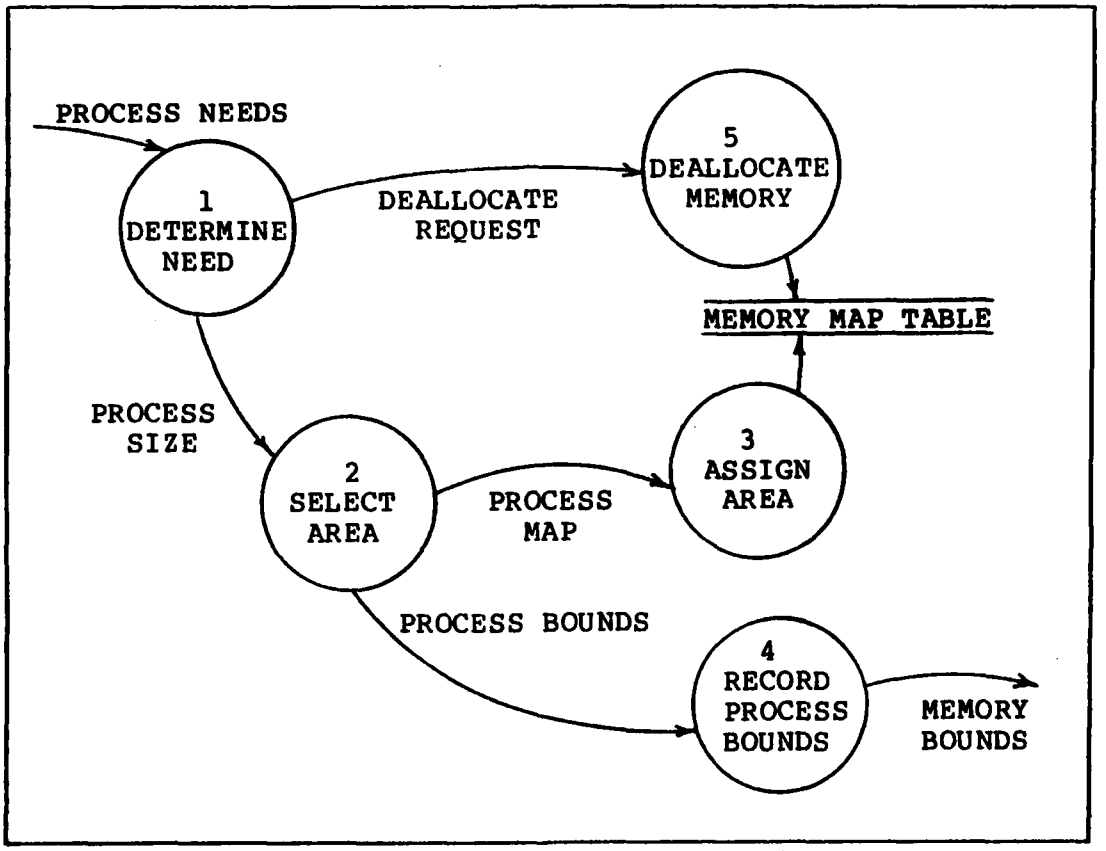


Figure 33. Memory Management

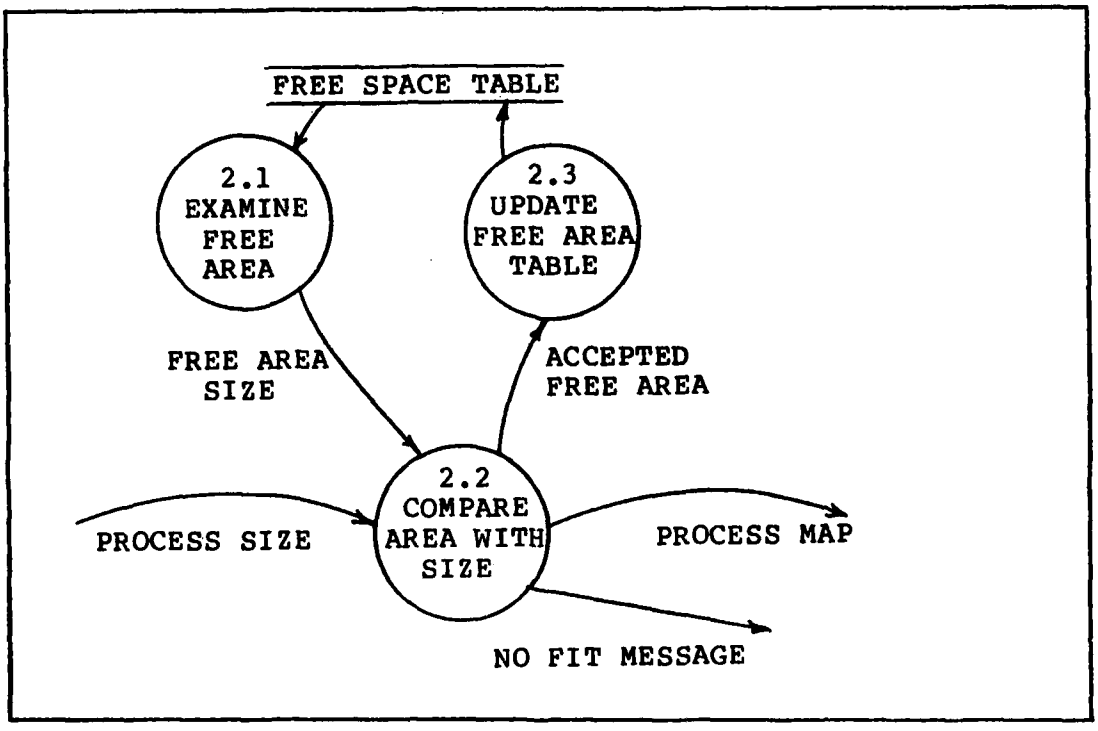


Figure 34. Select Free Area

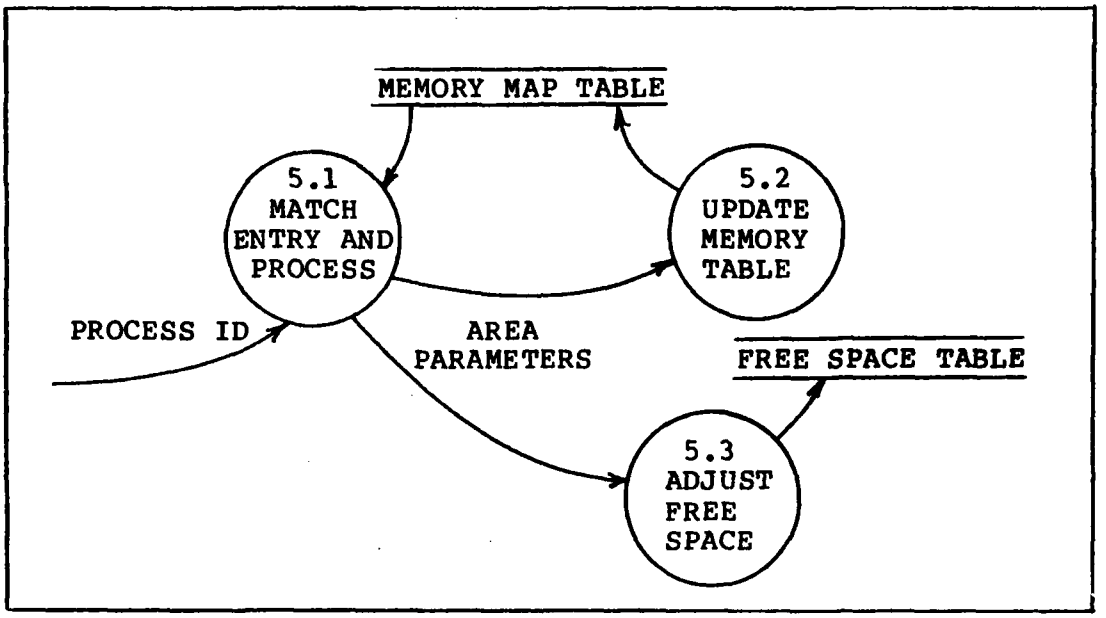


Figure 35. Deallocation Memory Space

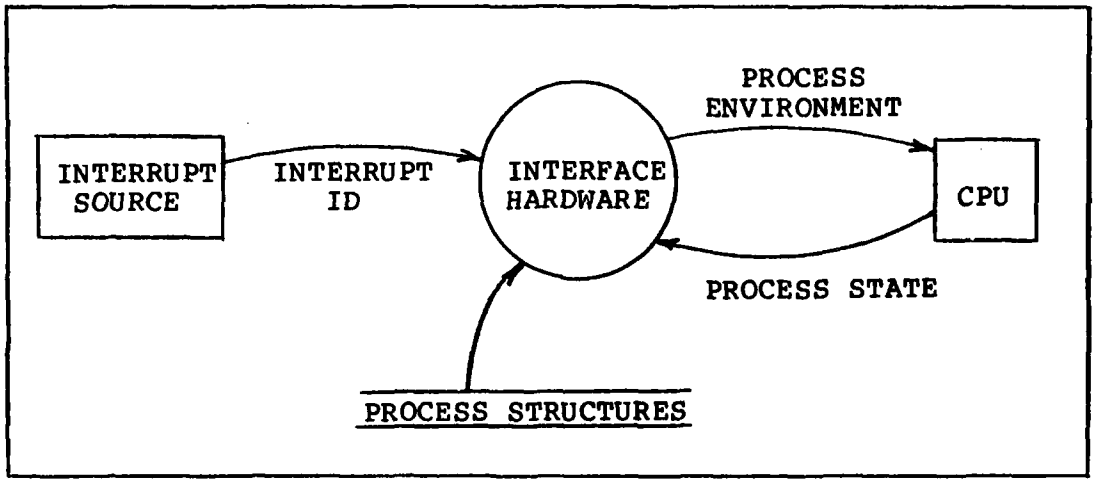
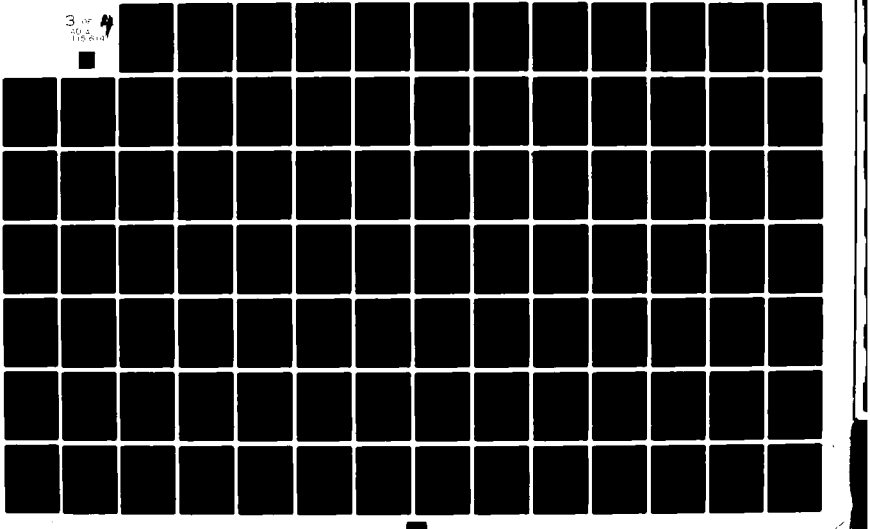


Figure 36. Nucleus Context Diagram

AD-A115 614 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
DESIGN AND DEVELOPMENT OF A MULTIPROGRAMMING OPERATING SYSTEM F--ETC(U)
DEC 81 M S ROSS
UNCLASSIFIED AFIT/GCS/EE/81D-14 NL



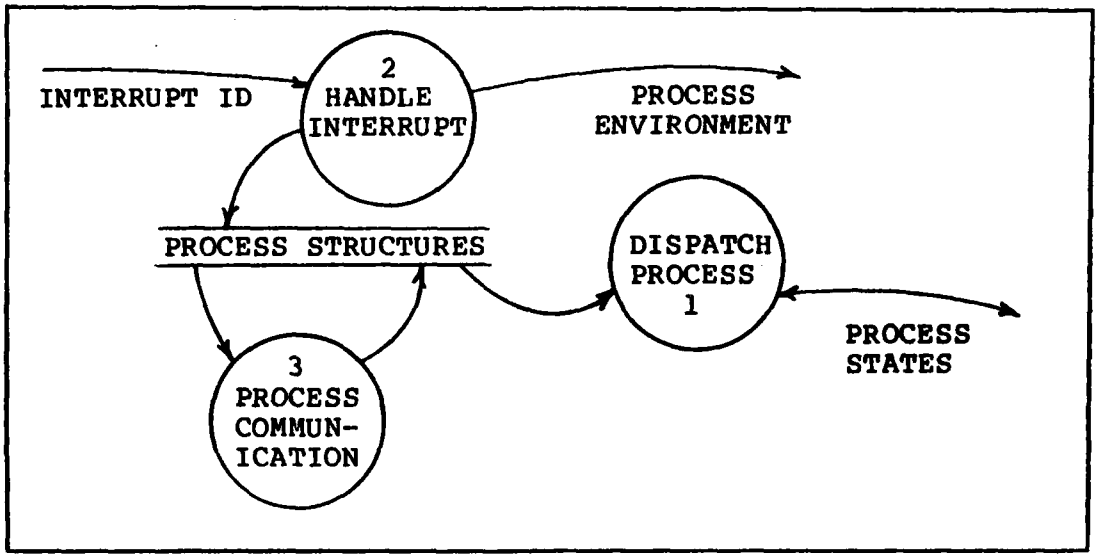


Figure 37. Nucleus Overview Diagram

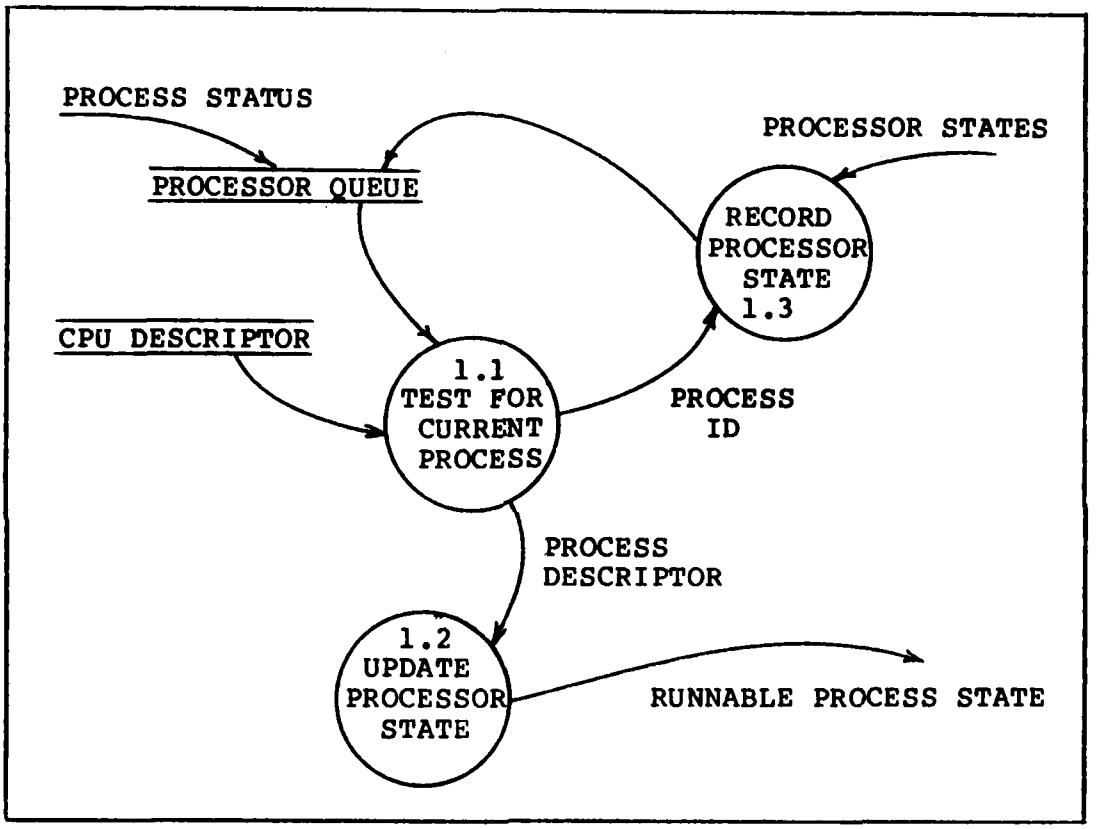


Figure 38. Dispatch Process

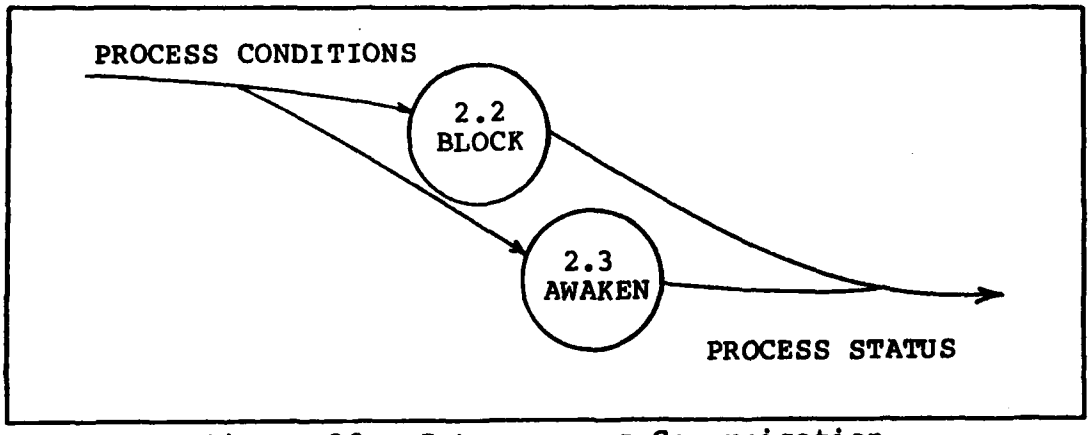


Figure 39. Interprocess Communication

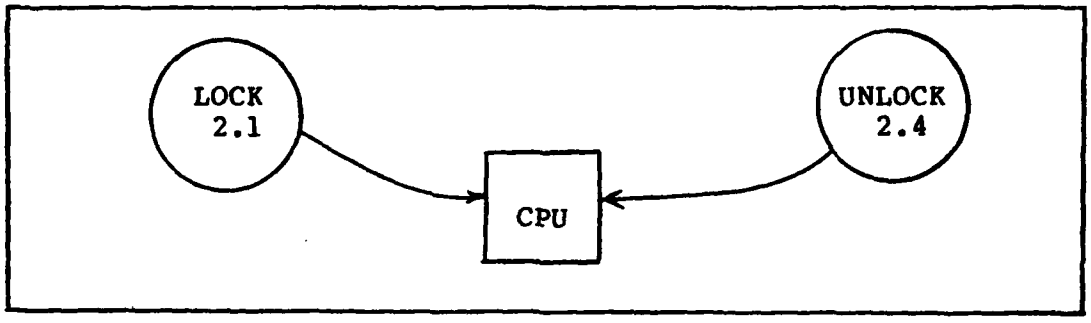


Figure 40. Lock and Unlock CPU

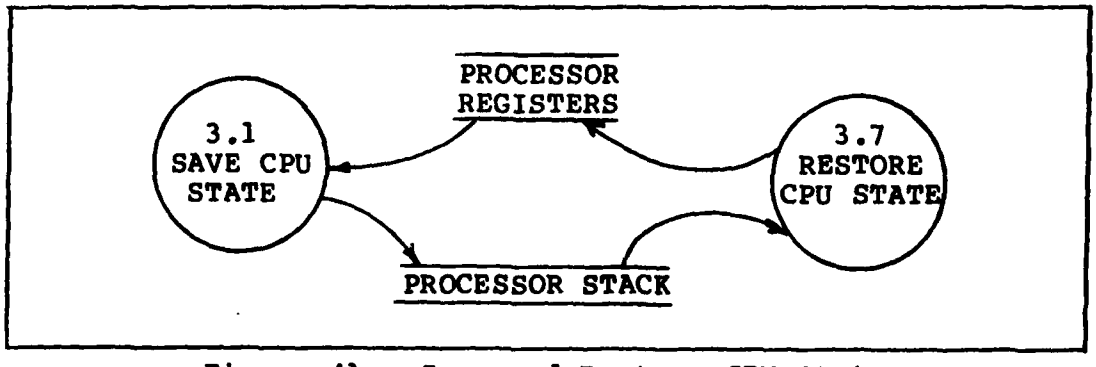


Figure 41. Save and Restore CPU State

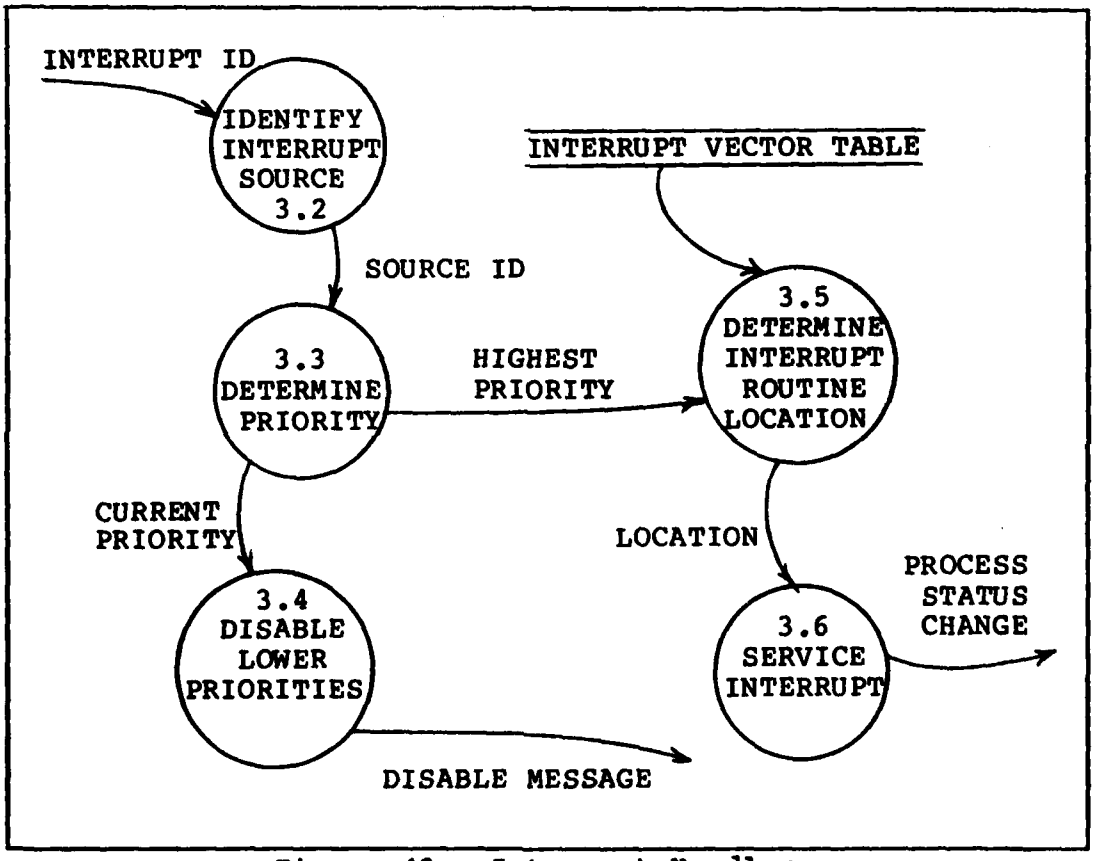


Figure 42. Interrupt Handler

DATA DICTIONARY
FOR OPERATING SYSTEM SHELL

```
DATA ELEMENT NAME:  AUTHORIZATION-MESSAGE
ALIASES:            SUPERVISOR-ID
COMPOSITION:        AUTHORIZATION-MESSAGE = UNIQUE INTEGER
                          ASSIGNED TO SYSTEM-SUPERVISOR
                          BY THE OPERATING SYSTEM
SOURCE:             VERIFY AUTHORITY (2.1)
DESTINATION:        PROVIDE MENU (2.2)
COMMENTS:           EXECUTE SYSTEM COMMAND LEVEL


DATAFLOW NAME:      AUTHORIZED-COMMAND
ALIASES:            NONE
COMPOSITION:        AUTHORIZED-COMMAND = SYSTEM-COMMAND
                          BY A VERIFIED SYSTEM-SUPERVISOR
SOURCE:             VERIFY AUTHORITY (2.1)
DESTINATION:        CONFIGURE SYSTEM (2.3)
COMMENTS:           EXECUTE SYSTEM COMMAND LEVEL


DATAFLOW NAME:      COMMAND
ALIASES:            NONE
COMPOSITION:        COMMAND = USER-COMMAND
                            | SYSTEM-COMMAND
SOURCE:             USER
DESTINATION:        DETERMINE COMMAND TYPE (1)
COMMENTS:           OPERATING SYSTEM SHELL


DATA ELEMENT NAME:  COMMAND-LOCATION
ALIASES:            COMMAND-ADDRESS
COMPOSITION:        ADDRESS OF COMMAND IN MEMORY
COMMENTS:           USED IN THE FILE: COMMAND-TABLE


DATAFLOW NAME:      COMMAND-ARGUMENTS
ALIASES:            NONE
COMPOSITION:        COMMAND-ARGUMENTS = {CHARACTERS
                                      | FILE-NAME}
SOURCE:             INTERPRET COMMAND STRING (6.1)
DESTINATION:        DETERMINE COMMAND CONDITIONS (6.2)
COMMENTS:           EXECUTE USER COMMAND LEVEL.
                    AS REQUIRED BY COMMAND ENTRY IN
                    COMMAND-TABLE
```

DATAFLOW NAME: COMMAND-CONDITIONS
ALIASES: NONE
COMPOSITION: COMMAND-CONDITIONS = {CHARACTER}
SOURCE: DETERMINE COMMAND CONDITIONS (6.2)
DESTINATION: EXECUTE COMMAND (6.4)
COMMENTS: EXECUTE USER COMMAND LEVEL
AS SPECIFIED BY COMMAND ENTRY IN
COMMAND-TABLE

DATA ELEMENT NAME: COMMAND-INFO
ALIASES: COMMAND-INFORMATION
COMPOSITION: SEE ALIASE
SOURCE: PROVIDE COMMAND INFO (5.3)
DESTINATION: RESPOND TO USER (7)
COMMENTS: EXECUTE HELP COMMAND LEVEL

DATAFLOW NAME: COMMAND-INFO-REQUEST
ALIASES: NONE
COMPOSITION: COMMAND-INFO-REQUEST = ? + COMMAND
SOURCE: DETERMINE HELP REQUIRED (5.1)
DESTINATION: PROVIDE COMMAND INFO (5.3)
COMMENTS: EXECUTE HELP COMMAND LEVEL

DATA ELEMENT NAME: COMMAND-INFORMATION
ALIASES: COMMAND-INFO
COMPOSITION: TEXT CONTAINING OPERATING INSTRUCTIONS
AND CAPABILITIES FOR EACH COMMAND.
SOURCE: PROVIDE COMMAND INFO (5.3)
DESTINATION: RESPOND TO USER (7)
COMMENTS: USED IN COMMAND-DOCUMENTATION FILE

DATAFLOW NAME: COMMAND-REQUIREMENTS
ALIASES: NONE
COMPOSITION: COMMAND-REQUIREMENTS = REQUIRED-
ARGUMENTS - COMMAND-ARGUMENTS
SOURCE: DETERMINE COMMAND CONDITIONS (6.2)
DESTINATION: PROMPT USER (6.3)
COMMENTS: EXECUTE USER COMMAND LEVEL.
REQUIRED-ARGUMENTS = MINIMUM SET OF
COMMAND-ARGUMENTS TO EXECUTE COMMAND.

```
DATAFLOW NAME:      CONTROL-COMMAND
ALIASES:            NONE
COMPOSITION:        CONTROL-COMMAND = LOG-IN-COMMAND
                                    | LOG-OUT-COMMAND
                                    | INQUIRY-COMMAND
SOURCE:             DETERMINE COMMAND (3)
DESTINATION:        EXECUTE CONTROL COMMAND (4)
COMMENTS:           OPERATING SYSTEM SHELL LEVEL


DATAFLOW NAME:      CONTROL-MESSAGE
ALIASES:            NONE
COMPOSITION:        CONTROL-MESSAGE = LOG-IN-PROMPT
                                    | LOG-OUT-MESSAGE
                                    | INQUIRY-RESPONSE
SOURCE:             EXECUTE CONTROL COMMAND (4)
DESTINATION:        RESPOND TO USER (7)
COMMENTS:           OPERATING SYSTEM SHELL LEVEL


DATAFLOW NAME:      HELP-COMMAND
ALIASES:            NONE
COMPOSITION:        HELP-COMMAND = SYSTEM-INFO-REQUEST
                                 | COMMAND-INFO-REQUEST
SOURCE:             DETERMINE COMMAND (4)
DESTINATION:        EXECUTE HELP COMMAND (5)
COMMENTS:           OPERATING SYSTEM SHELL LEVEL


DATAFLOW NAME:      HELP-INFO
ALIASES:            NONE
COMPOSITION:        HELP-INFO = SYSTEM-INFO
                              | COMMAND-INFO
SOURCE:             EXECUTE HELP COMMAND (5)
DESTINATION:        RESPOND TO USER (7)
COMMENTS:           OPERATING SYSTEM SHELL LEVEL


DATA ELEMENT NAME:  INQUIRY-COMMAND
ALIASES:            NONE
COMPOSITION:        INQUIRY-COMMAND = SPECIAL CHARACTER
SOURCE:             DETERMINE CONTROL COMMAND (4.1)
DESTINATION:        EXECUTE INQUIRY (4.4)
COMMENTS:           EXECUTE CONTROL COMMAND LEVEL
```

DATAFLOW NAME: INQUIRY-RESPONSE
ALIASES: NONE
COMPOSITION: INQUIRY-RESPONSE = USER-LIST
| SYSTEM-DOCUMENTATION
| PERIPHERAL-CHARACTERISTICS
SOURCE: EXECUTE INQUIRY (4.4)
DESTINATION: RESPOND TO USER (7)
COMMENTS: EXECUTE CONTROL COMMAND

DATAFLOW NAME: INTERPRETED-COMMAND
ALIASES: NONE
COMPOSITION: INTERPRETED-COMMAND = COMMAND
+ COMMAND-LOCATION
SOURCE: INTERPRET COMMAND STRING (6.1)
DESTINATION: EXECUTE COMMAND (6.4)
COMMENTS: EXECUTE USER COMMAND

DATAFLOW NAME: LOG-IN-COMMAND
ALIASES: NONE
COMPOSITION: LOG-IN-COMMAND = "LOGIN"
+ USER-NAME
+ USER-ACCOUNT-NUMBER
SOURCE: DETERMINE CONTROL COMMAND (4.1)
DESTINATION: LOG-OUT USER (4.3)
COMMENTS: EXECUTE CONTROL COMMAND LEVEL

DATAFLOW NAME: LOG-IN-PROMPT
ALIASES: NONE
COMPOSITION: LOG-IN-PROMPT = CURRENT DATE
+ CURRENT TIME
+ VERIFICATION OF LOGIN
SOURCE: LOG-IN USER (4.2)
DESTINATION: RESPOND TO USER (7)
COMMENTS: EXECUTE CONROL COMMAND LEVEL.
MAY INCLUDE ANY SPECIAL INFORMATION
AS DETERMINED TO SYSTEM-SUPERVISOR.

DATAFLOW NAME: LOG-OUT-COMMAND
ALIASES: NONE
COMPOSITION: LOG-OUT-COMMAND = "LOGOUT"
SOURCE: DETERMINE CONTROL COMMAND (4.1)
DESTINATION: LOG-OUT USER (4.3)
COMMENTS: EXECUTE CONTROL COMMAND LEVEL

```
DATAFLOW NAME:      LOG-OUT-MESSAGE
ALIASES:            NONE
COMPOSITION:        LOG-OUT-MESSAGE = CURRENT TIME
                                    + USER-NAME "LOGGED OUT"
                                    + RUNNING PROCESSES
SOURCE:             LOG-OUT USER (4.3)
DESTINATION:        RESPOND TO USER (7)
COMMENTS:           EXECUTE CONTROL COMMAND LEVEL.
                    RUNNING PROCESSES ARE THOSE USER
                    PROGRAMS STILL IN EXECUTION AT
                    LOGOUT TIME.


DATAFLOW NAME:      PERIPHIAL-CHARACTERISTICS
ALIASES:            NONE
COMPOSITION:        ALL INFORMATION REQUIRED TO INTERFACE
                    THE OPERATING SYSTEM TO INPUT/OUTPUT
                    DEVICES (BAUD RATE, STEP RATE, PORT
                    NUMBER, ETC).
COMMENTS:           USED IN SYSTEM-PARAMETER FILE.


DATAFLOW NAME:      PROCESS-CONDITIONS
ALIASES:            NONE
COMPOSITION:        PROCESS-CONDITIONS = PROCESS-STATUS
COMMENTS:           PROCESS-STATUS IS AN ELEMENT OF AN
                    ACTIVE PROCESS-DESCRIPTOR.
                    PROCESS-STATUS = READY | WAITING
                                   | RUNNING


DATAFLOW NAME:      PROMPT
ALIASES:            NONE
COMPOSITION:        PROMPT = COMMAND-REQUIREMENTS + "?"
SOURCE:             PROMPT USER (6.3)
DESTINATION:        RESPOND TO USER (7)
COMMENTS:           OPERATING SYSTEM SHELL LEVEL


DATAFLOW NAME:      SESSION-COMMAND
ALIASES:            NONE
COMPOSITION:        SESSION-COMMAND = ENTRY IN COMMAND-TABLE
SOURCE:             DETERMINE COMMAND (3)
DESTINATION:        INTERPRET COMMAND STRING (6.1)
COMMENTS:           OPERATING SYSTEM SHELL LEVEL
```

```
DATAFLOW NAME:      SYSTEM-COMMAND
ALIASES:            NONE
COMPOSITION:        SYSTEM-COMMAND = ENTRY FROM MENU-FILE
SOURCE:             DETERMINE COMMAND TYPE (1)
DESTINATION:        VERIFY AUTHORITY (2.1)
COMMENTS:           OPERATING SYSTEM SHELL LEVEL


DATAFLOW NAME:      SYSTEM-DOCUMENTATION
ALIASES:            NONE
COMPOSITION:        TEXT FILE CONTAINING INFORMATION ON
                    SYSTEM OPERATION AND CAPABILITIES
                    ON BOTH HARDWARE AND SOFTWARE
SOURCE:
DESTINATION:
COMMENTS:           USED IN CONFIGURATION-DATA, SYSTEM-
                    INFORMATION, AND SYSTEM-DATA FILES.


DATAFLOW NAME:      SYSTEM-INFO
ALIASES:            NONE
COMPOSITION:        SYSTEM-INFO = USER-LIST
                                | COMMAND-DOCUMENTATION
                                | PERIPHERAL-CHARACTERISTICS
                                | SYSTEM-DOCUMENTATION
SOURCE:             PROVIDE SYSTEM INFO (5.2)
DESTINATION:        RESPOND TO USER (7)
COMMENTS:           EXECUTE HELP COMMAND LEVEL


DATAFLOW NAME:      SYSTEM-INFO-REQUEST
ALIASES:            NONE
COMPOSITION:        SYSTEM-INFO-REQUEST = USER + "?"
                          | COMMAND + "?"
                          | DEVICE + "?"
                          | SYSTEM + "?"
SOURCE:             DETERMINE HELP REQUIRED (5.1)
DESTINATION:        PROVIDE  YSTEM INFO (5.2)
COMMENTS:           EXECUTE HELP COMMAND LEVEL


DATAFLOW NAME:      SYSTEM-RESPONSE
ALIASES:            NONE
COMPOSITION:        SYSTEM-RESPONCE = PROMPT
                                    | COMMAND-PROMPT
SOURCE:             EXECUTE COMMAND (6.4)
DESTINATION:        RESPOND TO USER (7)
COMMENTS:           EXECUTE USER COMMAND LEVEL.
                    COMMAND-PROMPT IS A RESPONSE FROM
                    A SPECIFIC COMMAND'S EXECUTION.
```

```
DATAFLOW NAME:      USER-COMMAND
ALIASES:            NONE
COMPOSITION:        USER-COMMAND = SESSION-COMMAND
                                 | HELP-COMMAND
                                 | CONTROL-COMMAND
SOURCE:             DETERMINE COMMAND TYPE (1)
DESTINATION:        DETERMINE COMMAND (3)
COMMENTS:           OPERATING SYSTEM SHELL LEVEL


DATAFLOW NAME:      USER-AUTHORIZATION
ALIASES:            NONE
COMPOSITION:        USER-AUTHORIZATION = USER-ID
                                       + USER-NAME
                                       + USER-ACCOUNT
COMMENTS:           MUST BE CONTAINED IN USER-LIST FILE


DATAFLOW NAME:      USER-RESPONSE
ALIASES:            NONE
COMPOSITION:        USER-RESPONSE = HELP-INFO
                                  | SYSTEM-RESPONSE
                                  | CONRTOL-MESSAGE
SOURCE:             RESPOND TO USER (7)
DESTINATION:        USER
COMMENTS:           OPERATING SYSTEM SHELL LEVEL
```

FILE DEFINITIONS
OPERATING SYSTEM SHELL

```
FILE OR DATABASE NAME:  COMMAND-DOCUMENTATION
ALIASES:                NONE
COMPOSITION:            COMMAND-DOCUMENTATION =
                            {COMMAND
                             + COMMAND-INFORMATION}
ORGANIZATION:
COMMENTS:


FILE OR DATABASE NAME:  COMMAND-TABLE
ALIASES:                NONE
COMPOSITION:            COMMAND-TABLE = {COMMAND
                                        + {ARGUMENTS}
                                        + (COMMANDS-CONDITIONS)
                                        + COMMAND-LOCATION}
ORGANIZATION:           SEQUENTIAL BY COMMAND
COMMENTS:               EXECUTE USER COMMAND LEVEL


FILE OR DATABASE NAME:  CONFIGURATION-DATA
ALIASES:                NONE
COMPOSITION:            CONFIGURATION-DATA = USER-LIST
                                + USER-AUTHIZATION
                                + COMMAND-TABLE
                                + COMMAND-DOCUMENTATION
                                + PERIPHERAL-CHARACTERISTICS
                                + SYSTEM-DOCUMENTATION
ORGANIZATION:           TABULAR BY CATAGORY
COMMENTS:               OPERATING SYSTEM SHELL LEVEL


FILE OR DATABASE NAME:  MENU-FILE
ALIASES:                NONE
COMPOSITION:            MENU-FILE = CATAGORIES OF
                                    CONFIGURATION-DATA
ORGANIZATION:           BY CATAGORY
COMMENTS:               EXECUTE SYSTEM COMMAND LEVEL
```

```
FILE OR DATABASE NAME:  SYSTEM-INFORMATION
ALIASES:                NONE
COMPOSITION:            SYSTEM-INFORMATION = USER-LIST
                                + COMMAND-DOCUMENTATION
                                + PERIPHERAL-CHARACTERISTICS
                                + SYSTEM-DOCUMENTATION
ORGANIZATION:           BY CATAGORY
COMMENTS:               EXECUTE HELP-COMMAND LEVEL


FILE OR DATABASE NAME:  SYSTEM-DATA
ALIASES:                NONE
COMPOSITION:            SYSTEM-DATA = USER-LIST
                                + SYSTEM-DOCUMENTATION
                                + PERIPHERAL-CHARACTERISTICS
ORGANIZATION:           BY CATAGORY
COMMENTS:               EXECUTE CONTROL-COMMAND LEVEL


FILE OR DATABASE NAME:  SYSTEM-PARAMETERS
ALIASES:                NONE
COMPOSITION:            SYSTEM-PARAMETERS = MENU-FILE
                                          + CONFIGURATION-DATA
ORGANIZATION:           BY CATAGORY
COMMENTS:               OPERATING SYSTEM SHELL LEVEL


FILE OR DATABASE NAME:  USER-DESCRIPTOR
ALIASES:                NONE
COMPOSITION:            USER-DESCRIPTOR = USER-ID
                                        + USER-NAME
                                        + USER-FILE-DIRECTORY
                                        + ACCOUNT-NUMBER
ORGANIZATION:           RECORD
COMMENTS:               EXECUTE CONTROL COMMAND


FILE OR DATABASE NAME:  USER-LIST
ALIASES:                NONE
COMPOSITION:            USER-LIST = {USER-NAME}
ORGANIZATION:           SEQENTIAL
COMMENTS:               SYSTEM-DATA, SYSTEM-INFORMATION,
                        AND CONFIGURATION-DATA
```

PROCESS DESCRIPTION
OPERATING SYSTEM SHELL

```
PROCESS NAME:          DETERMINE COMMAND TYPE
PROCESS NUMBER:        1
PROCESS DESCRIPTION:
If COMMAND is a SYSTEM-COMMAND and USER is SYSTEM-SUPERVISOR
   then COMMAND is a  SYSTEM-COMMAND
If COMMAND is a SYSTEM-COMMAND and USER is not SYSTEM-
   SUPERVISOR
   then re-prompt USER for COMMAND
else COMMAND is USER-COMMAND
```

```
PROCESS NAME:          VERIFY AUTHORITY
PROCESS NUMBER:        2.1
PROCESS DESCRIPTION:
Compare USER-ID with SUPERVISOR-ID.
If USER-ID not SUPERVISOR-ID
   then reject COMMAND.
else PROVIDE MENU.
```

```
PROCESS NAME:          PROVIDE MENU
PROCESS NUMBER:        2.2
PROCESS DESCRIPTION:
Read MENU-FILE and send MENU to USER
```

```
PROCESS NAME:          CONFIGURE SYSTEM
PROCESS NUMBER:        2.3
PROCESS DESCRIPTION:
Manipulate CONFIGURATION-DATA as requested by the COMMAND.
Change USER-LIST, system resoures or peripheral data.
```

```
PROCESS NAME:          DETERMINE COMMAND
PROCESS NUMBER:        3
PROCESS DESCRIPTION:
Case USER-COMMAND of:
   LOG-IN or LOG-OUT or INQUIRY
      then USER-COMMAND = CONTROL-COMMAND
   HELP or ?
      then USER-COMMAND = HELP-COMMAND
else USER-COMMAND = SESSION-COMMAND
```

```
PROCESS NAME:          DETERMINE CONTROL COMMAND
PROCESS NUMBER:        4.1
PROCESS DESCRIPTION:
Case CONTROL-COMMAND of:
   LOG-IN-COMMAND
     then log USER into system
   LOG-OUT-COMMAND
     then log USER out of system
   INQUIRY-COMMAND
     then provide SYSTEM-DATA


PROCESS NAME:          LOG-IN USER
PROCESS NUMBER:        4.2
PROCESS DESCRIPTION:
If USER-AUTHORIZATION not in LOG-IN-COMMAND
   then prompt USER for USER-AUTHORIZATION
Examine the USER-LIST for USER-AUTHORIZATION
If USER-AUTHORIZATION is valid and no USER-DESCRIPTOR exists
   then establish a cooresponding USER-DESCRIPTOR
If USER-AUTHORIZATION is valid and USER-DESCRIPTOR exists
   then provide LOG-IN-PROMPT containing PROCESS-CONDITIONS


PROCESS NAME:          LOG-OUT USER
PROCESS NUMBER:        4.3
PROCESS DESCRIPTION:
If USER-ID in contained in USER-DESCRIPTORS
   then check PROCESS-CONDITIONS
If PROCESS-CONDITIONS are terminated
   then remove USER-DESCRIPTOR


PROCESS NAME:          EXECUTE INQUIRY
PROCESS NUMBER:        4.4
PROCESS DESCRIPTION:
Provide SYSTEM-DATA via INQUIRY-RESPONSE


PROCESS NAME:          DETERMINE HELP REQUIRED
PROCESS NUMBER:        5.1
PROCESS DESCRIPTION:
If HELP-COMMAND contains SESSION-COMMAND
   then PROVIDE COMMAND-INFO
else provide SYSTEM-INFO
```

```
PROCESS NAME:          PROVIDE SYSTEM INFO
PROCESS NUMBER:        5.2
PROCESS DESCRIPTION:
Provide SYSTEM-MENU via SYSTEM-INFO
Read selection from SYSTEM-MENU
Read SYSTEM-INFORMATION and respond via SYSTEM-INFO


PROCESS NAME:          PROVIDE COMMAND INFO
PROCESS NUMBER:        5.3
PROCESS DESCRIPTION:
Output COMMAND-DOCUMENTATION for associated COMMAND


PROCESS NAME:          INTERPRET COMMAND STRING
PROCESS NUMBER:        6.1
PROCESS DESCRIPTION:
If SESSION-COMMAND is contained in COMMAND-TABLE
   then route INTERPRETED-COMMAND to EXECUTE COMMAND
   route all other data to DETERMINE COMMAND CONDITIONS


PROCESS NAME:          DETERMINE COMMAND CONDITIONS
PROCESS NUMBER:        6.2
PROCESS DESCRIPTION:
If COMMAND-TABLE requires more COMMAND-ARGUMENTS
   then PROMPT USER
If COMMAND-CONDITIONS are in COMMAND-ARGUMENTS
   then route COMMAND-CONDITIONS to EXECUTE COMMAND


PROCESS NAME:          PROMPT USER
PROCESS NUMBER:        6.3
PROCESS DESCRIPTION:
Send COMMAND-REQUIREMENTS to USER via PROMPT
```

PROCESS NAME: EXECUTE COMMAND
PROCESS NUMBER: 6.4
PROCESS DESCRIPTION:
Create a process to run the INTERPRETED-COMMAND
given the COMMAND-CONDITIONS as arguments.
Inform the user of progress via SYSTEM-RESPONSE.

PROCESS NAME: RESPOND TO USER
PROCESS NUMBER: 7
PROCESS DESCRIPTION:
Send: HELP-INFO or
CONTROL-MESSAGE or
SYSTEM-RESPONSE
responce to proper USER-ID

DATA DICTIONARY
FOR FILE MANAGEMENT LEVEL

DATAFLOW NAME: ACCESS-ABORT
ALIASES: NONE
COMPOSITION: MESSAGE INDICATING VIOLATION OF FILE ACCESS PRIVILEGES
SOURCE: CHECK ACCESS RIGHTS (3.5)
DESTINATION: SOURCE OF FILE-REQUEST
COMMENTS: EXECUTE OPEN FILE LEVEL

DATAFLOW NAME: ACCESS-RIGHTS
ALIASES: NONE
COMPOSITION: ACCESS-RIGHTS = PARTNER-LIST
+ PROTECTION-KEY
SOURCE: EXTRACT DIRECTORY DATA (3.4)
DESTINATION: CHECK ACCESS RIGHTS (3.5)
COMMENTS: EXECUTE OPEN FILE LEVEL

DATAFLOW NAME: BLOCK-ADDRESS
ALIASES: NONE
COMPOSITION: BLOCK-ADDRESS = FILE SIZE
+ FIRST BLOCK-ID
+ LAST BLOCK-ID OF A NEW-FILE
SOURCE: CONNECT FILE BLOCKS (3.3.3)
DESTINATION: USER-FILE-DIRECTORY
COMMENTS: ALLOCATE FILE STORAGE LEVEL

DATA ELEMENT NAME: BLOCK-ID
ALIASES: NONE
COMPOSITION: BLOCK-ID = INTEGER UNIQUE TO FILE-BLOCK
SOURCE: IDENTIFY FREE BLOCKS (3.3.2)
DESTINATION: CONNECT FILE BLOCKS (3.3.3)
COMMENTS: ALLOCATE FILE STORAGE LEVEL

DATAFLOW NAME: CENTRAL-FILE-BLOCK
ALIASES: NONE
COMPOSITION: SEE FILE DEFINITIONS
SOURCE: CREATE CENTRAL-FILE-BLOCK (3.8.3)
DESTINATION: DEVICE-DESCRIPTOR
COMMENTS: CREATE FILE DESCRIPTOR LEVEL

DATAFLOW NAME: CLOSE-FILE-REQUEST
ALIASES: NONE
COMPOSITION: CLOSE-FILE-REQUEST = FILE-NAME
SOURCE: LOCATE USER FILE DIRECTORY (2)
DESTINATION: DELETE LOCAL FILE BLOCK (4.1)
COMMENTS: CLOSE FILE LEVEL

DATAFLOW NAME: CURRENT-STATUS
ALIASES: NONE
COMPOSITION: CURRENT-STATUS = USE-COUNT
SOURCE: UPDATE STATUS (4.2)
DESTINATION: DELETE CENTRAL-FILE-BLOCK (4.3)
COMMENTS: CLOSE FILE LEVEL

DATA ELEMENT NAME: DEVICE-DESCRIPTOR-ADDRESS
ALIASES: NONE
COMPOSITION: DEVICE-DESCRIPTOR-ADDRESS =
POINTER TO A DEVICE-DESCRIPTOR
SOURCE: CREATE LOCAL-FILE-BLOCK (3.8.1)
DESTINATION: CREATE CENTRAL-FILE-BLOCK (3.8.3)
COMMENTS: CREATE FILE DESCRIPTOR LEVEL

DATAFLOW NAME: DIRECTORY-ERROR
ALIASES: NONE
COMPOSITION: MESSAGE INDICATION FILE-NAME IS NOT
LOCATED IN THE USER-DIRECTORY
SOURCE: LOCATE USER DIRECTORY (3.2)
DESTINATION: SOURCE OF FILE REQUEST
COMMENTS: EXECUTE OPEN FILE LEVEL

DATAFLOW NAME: DIRECTORY-LINKS
ALIASES: NONE
COMPOSITION: POINTER TO FILE-NAME FROM USER-DIRECTORY
AND USER-DIRECTORY TO FILE-NAME
SOURCE: EXECUTE LINK FILES (3.6)
DESTINATION: USER-FILE-DIRECTORY
COMMENTS: EXECUTE OPEN FILE LEVEL

DATA ELEMENT NAME: DIRECTORY-LOCATION
ALIASES: NONE
COMPOSITION: ADDRESS OF MASTER-FILE-DIRECTORY
SOURCE: DETERMINE MASTER DIRECTORY (1)
DESTINATION: LOCATE USER FILE DIRECTORY (2)
COMMENTS: FILE MANAGEMENT OVERVIEW LEVEL

```
DATAFLOW NAME:      EXISTING-BLOCK-LOCATION
ALIASES:            NONE
COMPOSITION:        EXISTING-BLOCK-LOCATION = ADDRESS
                          OF ASSOCIATED CENTRAL-FILE-BLOCK
SOURCE:             TEST CENTRAL-FILE-BLOCK (3.8.2)
DESTINATION:        CREATE LOCAL-FILE-BLOCK (3.8.1)
COMMENTS:           CREATE FILE DESCRIPTOR LEVEL


DATAFLOW NAME:      FILE-COMMAND
ALIASES:            NONE
COMPOSITION:        FILE-COMMAND = USER-ID
                                 + OWNER-ID
                                 + {FILE-NAME}
                                 + OPERATION
                                 + USER-FILE-DIRECTORY-
                                   LOCATION
SOURCE:             LOCATE USER FILE DIRECTORY (2)
DESTINATION:        EXECUTE OPEN FILE (3)
COMMENTS:           FILE MANAGEMENT LEVEL AND
                    EXECUTE OPEN FILE LEVEL
                    USER-ID AND OWNER-ID MAY BE THE SAME


DATA ELEMENT NAME:  FILE-DATA
ALIASES:            NONE
COMPOSITION:        ANY BIT PATTERN PASSED TO A FILE DEVICE
SOURCE:             INPUT/OUTPUT DEVICE
DESTINATION:        FILE DEVICE
                    DEVICE
COMMENTS:           CONTEXT LEVEL


DATAFLOW NAME:      FILE-DESCRIPTOR
ALIASES:            NONE
COMPOSITION:        FILE-DESCRIPTOR = LOCAL-FILE-BLOCK
                                    + CENTRAL-FILE-BLOCK
SOURCE:             EXECUTE OPEN FILE (3)
DESTINATION:        DEVICE DESCRIPTOR
COMMENTS:           FILE-MANAGEMENT-LEVEL


DATAFLOW NAME:      FILE-LOCATION
ALIASES:            NONE
COMPOSITION:        FILE-LOCATION = FILE-NAME
                                  + USER-FILE-DIRECTORY-
                                    LOCATION
SOURCE:             LOCATE DIRECTORY ENTRY (3.2)
DESTINATION:        EXTRACT DIRECTORY DATA (3.4)
COMMENTS:           EXECUTE OPEN FILE LEVEL
```

```
DATA ELEMENT NAME:   FILE-NAME
ALIASES:             NONE
COMPOSITION:         FILE-NAME = {CHARACTERS}
SOURCE:              DETERMINE FILE LOCATION (3.7)
DESTINATION:         CREATE CENTRAL-FILE-BLOCK (3.8.3)
COMMENTS:            CREATE FILE DESCRIPTOR


DATAFLOW NAME:       FILE-PARAMETERS
ALIASES:             NONE
COMPOSITION:         FILE-PARAMETERS = FILE-NAME
                                       + STARTING-BLOCK
                                       + OPERATION
SOURCE:              EXTRACT DIRECTORY DATA (3.4)
DESTINATION:         CREATE FILE DESCRIPTORS (3.8)
COMMENTS:            EXECUTE OPEN FILE LEVEL


DATAFLOW NAME:       FILE-REQUEST
ALIASES:             NONE
COMPOSITION:         FILE-REQUEST = USER-ID
                                  + OWNER-ID
                                  + {FILE-NAME}
                                  + OPERATION
SOURCE:              USER COMMAND OR SYSTEM PROCESS
DESTINATION:         LOCATE USER FILE DIRECTORY (2)
COMMENTS:            USER-ID MAY BE SAME AS OWNER-ID


DATAFLOW NAME:       LINK-APPROVAL
ALIASES:             NONE
COMPOSITION:         LINK-APPROVAL = PROTECTION-KEY
                                   + PARTNER
SOURCE:              CHECK ACCESS RIGHTS (3.5)
DESTINATION:         EXECUTE LINK FILES (3.6)
COMMENTS:            EXECUTE OPEN FILE LEVEL
                     EXECUTE LINK FILES LEVEL


DATAFLOW NAME:       LOCAL-FILE-BLOCK
ALIASES:             NONE
COMPOSITION:         SEE LOCAL-FILE-BLOCK IN FILE OR DATA
                     BASE DICTIONARY
SOURCE:              CREATE LOCAL-FILE-BLOCK (3.8.1)
DESTINATION:         LOCAL-FILE-BLOCK CHAIN
COMMENTS:            CREATE FILE DESCRIPTOR LEVEL
```

DATA ELEMENT NAME: LOCATION-OF-CENTRAL-FILE-BLOCK
ALIASES: NONE
COMPOSITION: LOCATION-OF-CENTRAL-FILE-BLOCK =
POINTER MAINTAINED BY LOCAL-FILE-BLOCK
SOURCE: DELETE LOCAL-FILE-BLOCK (4.1)
DESTINATION: UPDATE STATUS (4.2)
COMMENTS: CLOSE FILE LEVEL

DATA ELEMENT NAME: LOGICAL-DEVICE
ALIASES: NONE
COMPOSITION: DISK OR DISK DRIVE DESIGNATION
SOURCE: EXTRACT DIRECTORY DATA (3.4)
DESTINATION: DETERMINE FILE LOCATION (3.7)
COMMENTS: EXECUTE OPEN FILE LEVEL

DATA ELEMENT NAME: MODE
ALIASES: OPERATION
COMPOSITION: SEE ALIASE
SOURCE: SEE ALIASE
DESTINATION: SEE ALIASE
COMMENTS: SEE ALIASE

DATAFLOW NAME: NEW-ACCESS-RIGHTS
ALIASES: NONE
COMPOSITION: NEW-ACCESS-RIGHTS = FILE-NAME
+ ACCESS-PRIVILEGES
SOURCE: ESTABLISH ACCESS RIGHTS (3.3.4)
DESTINATION: USER-FILE-DIRECTORY
COMMENTS: EXECUTE OPEN FILE LEVEL
ALLOCATE FILE STORAGE LEVEL

DATAFLOW NAME: NEW-FILE
ALIASES: NONE
COMPOSITION: NEW-FILE = FILE-NAME + USER-ID
+ (FILE-SIZE)
SOURCE: LOCATE DIRECTORY ENTRY (3.2)
DESTINATION: ALLOCATE FILE STORAGE (3.3)
COMMENTS: EXECUTE OPEN FILE LEVEL

DATAFLOW NAME: NEW-FILE-NAME
ALIASES: FILE-NAME
COMPOSITION: NEW-FILE-NAME = FILE-NAME
SOURCE: DETERMINE NUMBER OF BLOCKS (3.3.1)
DESTINATION: ESTABLISH ACCESS RIGHTS (3.3.4)
COMMENTS: EXECUTE OPEN FILE LEVEL
ALLOCATE FILE STORAGE LEVEL

DATA ELEMENT NAME: NUMBER-OF-BLOCKS
ALIASES: NONE
COMPOSITION: INTEGER
SOURCE: DETERMINE NUMBER OF BLOCKS (3.3.1)
DESTINATION: IDENTIFY FREE BLOCKS (3.3.2)
COMMENTS: ALLOCATE FILE STORAGE LEVEL

DATAFLOW NAME: OPEN-FILE-REQUEST
ALIASES: NONE
COMPOSITION: OPEN-FILE-REQUEST = FILE-NAME
+ USER-ID
+ OWNER-ID
+ [WRITE, READ,
DELETE, ADD]
+ USER-FILE-
DIRECTORY-LOCATION

SOURCE: DETERMINE COMMAND (3.1)
DESTINATION: LOCATE DIRECTORY ENTRY (3.2)
COMMENTS: EXECUTE OPEN FILE LEVEL

DATA ELEMENT NAME: OPERATION
ALIASES: NONE
COMPOSITION: OPERATION = [WRITE, READ, ADD,
DELETE]
COMMENTS: OPERATION IS AN ARGUMENT TO A FILE-
COMMAND OR FILE-REQUEST.
CREATE IS A SPECIAL CASE OF WRITE

DATA ELEMENT NAME: OWNER-LINK
ALIASES: NONE
COMPOSITION: POINTER TO A FILE IN A USER-FILE-
DIRECTORY
SOURCE: LINK OWNER TO USER (3.6.2)
DESTINATION: USER-FILE-DIRECTORY
COMMENTS: EXECUTE LINK FILES LEVEL

DATAFLOW NAME: PARTNER
ALIASES: NONE
COMPOSITION: PARTNER = USER-NAME
COMMENTS: USED IN USER-FILE-DIRECTORY AND PARTNER-LIST TO INDICATE ACCESS PRIVILEGES.

DATAFLOW NAME: PARTNER-LIST
ALIASES: NONE
COMPOSITION: PARTNER-LIST = {PARTNER
+ PROTECTION-KEY}
| {PARTNER}
+ PROTECTION-KEY
COMMENTS: CONTAINED IN THE USER-FILE-DIRECTORY FOR EACH FILE.

DATAFLOW NAME: PHYSICAL-LOCATION
ALIASES: NONE
COMPOSITION: PHYSICAL-LOCATION = DEVICE-DESCRIPTOR
+ FILE BLOCK LOCATION
SOURCE: DETERMINE FILE LOCATION (3.7)
DESTINATION: CREATE FILE DESCRIPTOR (3.8)
COMMENTS: EXECUTE OPEN FILE LEVEL

DATA ELEMENT NAME: PROTECTION-KEY
ALIASES: NONE
COMPOSITION: PROTECTION-KEY = [N,E,R,A,P,D]
COMMENTS: N = NO ACCESS
E = EXECUTE
R = READ
A = APPEND
P = PROTECTION ALTERATION
D = DELETE
N IS ASSIGNED AS A PROTECTION KEY BY DEFAULT. OTHERS MUST BE EXPLICITLY REQUESTED. PROTECTION KEYS ARE APPLIED TO THE PARTNER-LIST, NOT THE FILE OWNER.

DATAFLOW NAME: REQUESTED-FILE-NAME
ALIASES: FILE-NAME
COMPOSITION: SEE ALIASE
SOURCE: DETERMINE USER DIRECTORY (3.6.1)
DESTINATION: LINK USER TO OWNER (3.6.3)
COMMENTS: EXECUTE LINK FILES LEVEL

```
DATAFLOW NAME:        STARTING-BLOCK
ALIASES:              NONE
COMPOSITION:          STARTING-BLOCK = FIRST-FILE-BLOCK
                                     + DEVICE-NAME
SOURCE:               DETERMINE FILE LOCATION (3.7)
DESTINATION:          CREATE FILE DESCRIPTOR (3.8)
COMMENTS:             CREATE FILE DESCRIPTOR LEVEL


DATAFLOW NAME:        STATUS
ALIASES:              NONE
COMPOSITION:          STATUS = WRITE-BIT
                             + USE-COUNT
SOURCE:               CENTRAL-FILE-BLOCK
DESTINATION:          TEST CENTRAL-FILE-BLOCK (3.8.2)
COMMENTS:             CREATE FILE DESCRIPTOR LEVEL


DATA ELEMENT NAME:    STORAGE-ERROR
ALIASES:              NONE
COMPOSITION:          MESSAGE INDICATING INSUFFICIENT STORAGE
                      AVAIALBE FOR FILE-NAME.
SOURCE:               IDENTIFY FREE BLOCKS (3.3.2)
DESTINATION:          SOURCE OF FILE-REQUEST
COMMENTS:             ALLOCATE FILE STORAGE LEVEL


DATA ELEMENT NAME:    USE-COUNT
ALIASES:              NONE
COMPOSITION:          INTEGER
COMMENTS:             USED IN CENTRAL-FILE-DESCRIPTOR


DATAFLOW NAME:        USER-DIRECTORY-INFO
ALIASES:              NONE
COMPOSITION:          USER-DIRECTORY-INFO = USER-DIRECTORY
                                            LOCATION
                                          + FILE-NAME
SOURCE:               DETERMINE USER DIRECTORY (3.6.1)
DESTINATION:          LINK OWNER TO USER (3.6.2)
COMMENTS:             EXECUTE LINK FILES LEVEL


DATA ELEMENT NAME:    USER-FILE-DIRECTORY-LOCATION
ALIASES:              NONE
COMPOSITION:          LOGICAL POINTER TO A USER-FILE-
                      DIRECTORY
COMMENTS:             NONE
```

DATA ELEMENT NAME: USER-ID
ALIASES: NONE
COMPOSITION: INTEGER UNIQUE TO EACH USER.
COMMENTS: ASSIGNED AND KNOWN ONLY BY THE SYSTEM ASSOCIATED WITH USER-NAME.

DATA ELEMENT NAME: USER-NAME
ALIASES: NONE
COMPOSITION: CHARACTER STRING UNIQUE TO EACH USER.
COMMENTS: USED BY USER FOR SYSTEM IDENTIFICATION. SYSTEM ASSOCIATES USER-NAME WITH USER-ID.

DATA ELEMENT NAME: WRITE-BIT
ALIASES: NONE
COMPOSITION: INTEGER
COMMENTS: USED IN CENTRAL-FILE-DESCRIPTOR

FILE DEFINITIONS
FILE MANAGEMENT LEVEL

```
FILE OR DATABASE NAME:   CENTRAL-FILE-BLOCK
ALIASES:                 NONE
COMPOSITION:             CENTRAL-FILE-BLOCK =
                              FILE-NAME
                              + FIRST-BLOCK-ADDRESS
                              + USE-COUNT
                              + WRITE-BIT
                              + NEXT-CENTRAL-FILE-BLOCK
                              + DEVICE-DESCRIPTOR-ADDRESS
ORGANIZATION:            RECORD
COMMENTS:                CREATE-FILE-DESCRIPTOR LEVEL


FILE OR DATABASE NAME:   CURRENT-MASTER-DIRECTORY
ALIASES:                 NONE
COMPOSITION:             CURRENT-MASTER-DIRECTORY=
                              MASTER-FILE-DIRECTORY
                              | USER-FILE-DIRECTORY
ORGANIZATION:            SEE MASTER-FILE-DIRECTORY | USER-
                         FILE-DIRECTORY
COMMENTS:                FILE MANAGEMENT OVERVIEW


FILE OR DATABASE NAME:   FILE-CATALOGUE
ALIASES:                 NONE
COMPOSITION:             FILE-CATALOGUE =
ORGANIZATION:                 MASTER-FILE-DIRECTORY
                              + {USER-FILE-DIRECTORY}
COMMENTS:                FILE MANAGEMENT CONTEXT DIAGRAM


FILE OR DATABASE NAME:   FILE-DESCRIPTORS
ALIASES:                 NONE
COMPOSITION:             FILE-DESCRIPTOR =
                                  LOCAL-FILE-BLOCK
                                  + CENTRAL-FILE-BLOCK
ORGANIZATION:            RECORD ATTACH TO DEVICE-DESCRIPTORS
COMMENTS:                FILE MANAGEMENT OVERVIEW
```

```
FILE OR DATABASE NAME:  FILE-STORAGE-DIRECTORY
ALIASES:                NONE
COMPOSITION:            FILE-STORAGE-DIRECTORY =
                                    {FILE-NAME
                                    + USER-ID
                                    + FIRST-BLOCK
                                    + {FILE-BLOCK
                                    + NEXT-BLOCK}
                                    + LAST-BLOCK}
ORGANIZATION:           SEQUENTIAL BY FILE-NAME AND USER-ID
COMMENTS:               ALLOCATE FILE STORAGE LEVEL


FILE OR DATABASE NAME:  LOCAL-FILE-BLOCK
ALIASES:                NONE
COMPOSITION:            LOCAL-FILE-BLOCK =
                             ACCESS-MODE
                             + NEXT-BLOCK-LOCATION
                             + CENTRAL-FILE-BLOCK-LOCATION
ORGANIZATION:           RECORD
COMMENTS:               CREATE-FILE-DESCRIPTOR LEVEL


FILE OR DATABASE NAME:  MASTER-FILE-DIRECTORY
ALIASES:                NONE
COMPOSITION:            MASTER-FILE-DIRECTORY =
                             {USER-ID
                             + USER-FILE-DIRECTORY-LOCATION}
ORGANIZATION:           SEQUENTIAL BY USER-NAME
COMMENTS:


FILE OR DATABASE NAME:  USER-DESCRIPTOR
ALIASES:                NONE
COMPOSITION:            USER-DESCRIPTOR = USER-ID
                                        + USER-NAME
                                        + CURRENT-DIRECTORY
                                        + ACCOUNT-NUMBER
ORGANIZATION:           RECORD
COMMENTS:               FILE MANAGEMENT OVERVIEW
```

```
FILE OR DATABASE NAME:   USER-FILE-DIRECTORY
ALIASES:                 NONE
COMPOSITION:             USER-FILE-DIRECTORY =
                              {FILE-NAME
                              + FILE-LOCATION
                              + FILE-TYPE
                              + PARTNER-LIST
                              + PROTECTION-KEY
                              + FILE-LENGTH
                              + LINKS-TO-OTHER-FILES
                              + LAST-UPDATE}
ORGANIZATION:            SEQUENTIAL BY FILE-NAME
```

PROCESS DESCRIPTION
FOR FILE MANAGEMENT LEVEL

PROCESS NAME: DETERMINE MASTER DIRECTORY
PROCESS NUMBER: 1
PROCESS DESCRIPTION:
If the USER's CURRENT-DIRECTORY is MASTER-DIRECTORY
then determine location of USER's-FILE-DIRECTORY

PROCESS NAME: LOCATE USER FILE DIRECTORY
PROCESS NUMBER: 2
PROCESS DESCRIPTION:
If USER's-CURRENT-DIRECTORY is not the MASTER-DIRECTORY
then determine USER-FILE-DIRECTORY from CURRENT-
DIRECTORY entry in USER-DESCRIPTOR.
determine OWNER of USER-FILE-DIRECTORY

PROCESS NAME: DETERMINE COMMAND
PROCESS NUMBER: 3.1
PROCESS DESCRIPTION:
If ADD-COMMAND
then open only the FILE-DIRECTORIES
and EXECUTE-LINK-FILES (3.6)
else open FILE-NAME

PROCESS NAME: LOCATE DIRECTORY ENTRY
PROCESS NUMBER: 3.2
PROCESS DESCRIPTION:
SEARCH USER-FILE-DIRECTORY FOR LOCATION OF FILE-NAME.
If OPERATION is WRITE or CREATE
then pass NEW-FILE to ALLOCATE FILE SPACE (3.3).
If FILE-NAME is not in DIRECTORY
thne indicate DIRECTORY-ERROR

PROCESS NAME: EXTRACT DIRECTORY DATA
PROCESS NUMBER: 3.4
PROCESS DESCRIPTION:
From the FILE-LOCATION, Read: FILE-NAME
ACCESS-PRIVILEGES
PARTNER-LIST
LOGICAL LOCATION OF FILE
ADDRESS OF DEVICE DESCRIPTOR

PROCESS NAME: CHECK ACCESS RIGHTS
PROCESS NUMBER: 3.5
PROCESS DESCRIPTION:
Compare the USER-ID against the PARTNER-LIST.
Compare the OPERATION against the ACCESS-PRIVILEGES

PROCESS NAME: DETERMINE FILE LOCATION
PROCESS NUMBER: 3.7
PROCESS DESCRIPTION:
Compute the PHYSICAL-LOCATION from the LOGICAL-LOCATION

PROCESS NAME: DETERMINE NUMBER OF BLOCKS
PROCESS NUMBER: 3.3.1
PROCESS DESCRIPTION:
Calculate the NUMBER-OF-FILE-BLOCKS needed from FILE-SIZE.
NUMBER-OF-BLOCKS = (FILE-SIZE / BLOCK SIZE) rounded up.

PROCESS NAME: IDENTIFY FREE BLOCKS
PROCESS NUMBER: 3.3.2
PROCESS DESCRIPTION:
Examine the FILE-STORAGE-DIRECTORY for enough free BLOCKS
to store NEW-FILE.
If NUMBER-OF-BLOCKS are not available
then return a STORAGE-ERROR to source of NEW-FILE

PROCESS NAME: CONNECT FREE BLOCKS
PROCESS NUMBER: 3.3.3
PROCESS DESCRIPTION:
Link the identified FREE-BLOCKS together.
Remove the identified BLOCKS from the FREE LIST.
Record the STARTING- and ENDING-BLOCK-ADDRESS of NEW-FILE.

PROCESS NAME: DETERMINE USER DIRECTORY
PROCESS NUMBER: 3.6.1
PROCESS DESCRIPTION:
Examine the USER-DESCRIPTOR to determine the location of
the USER-DIRECTORY.

```
PROCESS NAME:          LINK OWNER TO USER
PROCESS NUMBER:        3.6.2
PROCESS DESCRIPTION:
Establish a pointer from the OWNER's-FILE TO THE USER's-
FILE-DIRECTORY in the OWNER's-DIRECTORY.


PROCESS NAME:          LINK USER TO OWNER
PROCESS NUMBER:        3.6.3
PROCESS DESCRIPTION:
Establish a pointer from the USER's-DIRECTORY to the OWNER's-
FILE in the USER's-DIRECTORY.


PROCESS NAME:          CREATE LOCAL-FILE-BLOCK
PROCESS NUMBER:        3.8.1
PROCESS DESCRIPTION:
Record:  CENTRAL-FILE-BLOCK-LOCATION
         OPERATION.
Executed each time a process opens a file.


PROCESS NAME:          TEST CENTRAL-FILE-BLOCK
PROCESS NUMBER:        3.8.2
PROCESS DESCRIPTION:
If USE-COUNT is set
   then FILE can be opened for READing only.
If WRITE-BIT is set
   then FILE cannot be opened.
else open file by CREATING FILE-DESCRIPTOR


PROCESS NAME:          CREATE CENTRAL-FILE-BLOCK
PROCESS NUMBER:        3.8.3
PROCESS DESCRIPTION:
Record:  FILE-NAME
         STARTING-BLOCK
         USE-COUNT
         WRITE-BIT
         DEVICE-DESCRIPTOR-ADDRESS
Link to DEVICE-DESCRIPTOR with other CENTRAL-FILE-BLOCKS.
One CENTRAL-FILE-BLOCK for each open FILE.
```

PROCESS NAME: DELETE LOCAL FILE BLOCK
PROCESS NUMBER: 4.1
PROCESS DESCRIPTION:
Locate and delete LOCAL-FILE-BLOCK associated with FILE-NAME.

PROCESS NAME: UPDATE STATUS
PROCESS NUMBER: 4.2
PROCESS DESCRIPTION:
Decrement USE-COUNT in CENTRAL-FILE-BLOCK associated with FILE-NAME.
Read current USE-COUNT.

PROCESS NAME: DELETE CENTRAL FILE BLOCK
PROCESS NUMBER: 4.3
PROCESS DESCRIPTION:
If USE-COUNT is not set after UPDATE STATUS (4.2)
   then delete CENTRAL-FILE-BLOCK

DATA DICTIONARY
FOR INPUT/OUPUT MANAGEMENT

DATA FLOW NAME:	BLOCK
ALIASES:	REQUEST-BLOCK
COMPOSITION:	BLOCK = DESTINATION + OPERATION-MODE + ORIGINATING PROCESS + ERROR ADDRESS + BLOCK-ID
SOURCE:	ASSEMBLE REQUEST BLOCK (2.1) REMOVE REQUEST (3.1)
DESTINATION:	ADD TO DEVICE SERVICE QUEUE (2.3) INITIATE I/O (3.2)
COMMENTS:	INITIATE INPUT/OUTPUT REQUEST LEVEL, EXECUTE DEVICE HANDLER LEVEL.

DATA ELEMENT NAME:	BLOCK-ID
ALIASES:	NONE
COMPOSITION:	BLOCK-ID = INTEGER
SOURCE:	ASSEMBLE REQUEST BLOCK (2.1) DELETE FROM LIST (3.3)
DESTINATION:	NOTIFY DEVICE HANDLER (2.2) INITIATE INPUT/OUTPUT (3.2)
COMMENTS:	INITIATE INPUT/OUTPUT REQUEST LEVEL, EXECUTE DEVICE HANDLER LEVEL.

DATA ELEMENT NAME:	DATA-TRANSLATED
ALIASES:	NONE
COMPOSITION:	DATA-TRANSLATED = DATA READY FOR TRANSFER TO DESTINATION
SOURCE:	TRANSLATE DATA (3.4)
DESTINATION:	TRANSFER DATA (3.5)
COMMENTS:	EXECUTE DEVICE HANDLER LEVEL

DATA FLOW NAME:	DEVICE-PARAMETERS
ALIASES:	NONE
COMPOSITION:	DEVICE-PARAMETERS = DEVICE-ID + DEVICE-STATUS + LOCATION OF TRANSLATION TABLE + DEVICE-CHARACTERISTICS
SOURCE:	MAP LOGICAL TO PHYSICAL DEVICE (1)
DESTINATION:	ASSEMBLE REQUEST BLOCK (2.1)
COMMENTS:	INPUT/OUTPUT MANAGEMENT OVERVIEW LEVEL

DATA ELEMENT NAME: ERROR-MESSAGE
ALIASES: NONE
COMPOSITION: ERROR-MESSAGE = DATA PLANTED BY
INITIATE I/O (3.4)
SOURCE: CHECK I/O PARAMETERS (4)
DESTINATION: INPUT/OUTPUT SOURCE
COMMENTS: INPUT/OUTPUT MANAGEMENT OVERVIEW LEVEL

DATA ELEMENT NAME: INPUT-DATA
ALIASES: NONE
COMPOSITION: INPUT-DATA = DATA TO BE TRANSLATED
SOURCE: INITIATE INPUT/OUTPUT (3.2)
DESTINATION: TRNSFER DATA (3.5)
COMMENTS: EXECUTE DEVICE HANDLER LEVEL

DATA FLOW NAME: I/O REQUEST
ALIASES: NONE
COMPOSITION: I/O REQUEST = DESTINATION
+ SOURCE
+ OPERATION-MODE
SOURCE: INPUT/OUTPUT REQUEST SOURCE
DESTINATION: MAP LOGICAL TO PHYSICAL DEVICE (1)
COMMENTS: INPUT/OUTPUT MANAGEMENT OVERVIEW LEVEL

DATA FLOW NAME: OPERATION-MODE
ALIASES: NONE
COMPOSITION: OPERATION-MODE = INPUT | OUTPUT
COMMENTS: USED IN THE REQUEST-BLOCK TO INDICATE
MODE OF TRANSFER DESIRED. ALSO USED AS
A CHARACTERISTIC OF A DEVICE.

DATA ELEMENT NAME: OUTPUT-DATA
ALIASES: NONE
COMPOSITION: OUTPUT-DATA = DATA READY TO BY TRANSFERED
SOURCE: INITIATE I/O (3.2)
DESTINATION: TRANSLATE DATA (3.4)
COMMENTS: EXECUTE DEVICE HANDLER LEVEL

DATA FLOW NAME: PHYSICAL-DEVICE
ALIASES: NONE
COMPOSITION: PHYSICAL-DEVICE = PERIPHERAL-ID
SOURCE: MAP LOGICAL TO PHYSICAL (1)
DESTINATION: CHECK I/O PARAMETERS (4)
COMMENTS: INPUT/OUTPUT MANAGEMENT OVERVIEW LEVEL

DATA ELEMENT NAME: PROCESS-NOTIFICATION
ALIASES: NONE
COMPOSITION: PROCESS-NOTIFICATION = MESSAGE OF COMPLETION TO ORIGINATING PROCESS
SOURCE: NOTIFY PROCESS (3.6)
DESTINATION: ORIGINATING PROCESS
COMMENTS: EXECUTE DEVICE HANDLER. ORIGINATING PROCESS IS INDICATED BY THE ENTRY IN THE REQUEST-BLOCK

DATA ELEMENT NAME: REQUEST-NOTICE
ALIASES: NONE
COMPOSITION: REQUEST-NOTICE = MESSAGE TO DEVICE-HANDLER ABOUT NEW I/O REQUEST
SOURCE: NOTIFY DEVICE HANDLER (2.2)
DESTINATION: REMOVE REQUEST (3.1)
COMMENTS: INITIATE INPUT/OUTPUT REQUEST LEVEL TO EXECUTE DEVICE HANDLER LEVEL

DATA ELEMENT NAME: REQUEST-SERVICED-MESSAGE
ALIASES: PROCESS NOTIFICATION
COMPOSITION: SEE ALIASE
SOURCE: EXECUTE DEVICE HANDLER (3)
DESTINATION: SEE ALIASE
COMMENTS: INPUT/OUTPUT MANAGEMENT OVERVIEW LEVEL

DATA FLOW NAME: TRANSFER-COMPLETE
ALIASES: NONE
COMPOSITION: TRANSFER-COMPLETE = MESSAGE INDICATING DATA TRANSFER IS FINISHED
SOURCE: TRANSFER DATA (3.5)
DESTINATION: NOTIFY PROCESS (3.6)
COMMENTS: EXECUTE DEVICE HANDLER LEVEL

FILE DEFINITIONS
FOR INPUT/OUTPUT MANAGEMENT

```
FILE OR DATABASE NAME:  DEVICE-DESCRIPTOR
ALIASES:                NONE
COMPOSITION:            DEVICE-DESCRIPTOR = DEVICE-ID
                            + LOCATION OF TRANSLATION-TABLE
                            + CURRENT STATUS
                            + CURRENT USER-PROCESS
                            + DEVICE-SERVICE-LIST
                            + CURRENT REQUEST-BLOCK
ORGANIZATION:           RECORD
COMMENTS:               INPUT/OUTPUT MANAGEMENT OVERVIEW


FILE OR DATABASE NAME:  DEVICE-DESCRIPTOR-TABLE
ALIASES:                NONE
COMPOSITION:            DEVICE-DESCRIPTOR-TABLE =
                                    {DEVICE-DESCRIPTOR}
ORGANIZATION:           SEQUENTIAL BY DEVICE
COMMENTS:               INPUT/OUTPUT MANAGEMENT OVERVIEW


FILE OR DATABASE NAME:  DEVICE-SERVICE-LIST
ALIASES:                NONE
COMPOSITION:            DEVICE-SERVICE-LIST = {REQUEST-BLOCK}
ORGANIZATION:           LINKED LIST
COMMENTS:               INITIATE INPUT/OUTPUT REQUEST LEVEL
                        EXECUTE DEVICE HANDLER LEVEL


FILE OR DATABASE NAME:  REQUEST-BLOCK
ALIASES:                NONE
COMPOSITION:            REQUEST-BLOCK = DESTINATION
                                      + OPERATION-MODE
                                      + ORIGINATING-PROCESS
                                      + ERROR-ADDRESS
ORGANIZATION:           SEQENTIAL, BY DESTINATION
COMMENTS:               REQUEST-BLOCK IS LINKED TO THE
                        CORRESPONDING DEVICE-DESCRIPTOR.
```

PROCESS DESCRIPTIONS
FOR INPUT/OUTPUT MANAGEMENT

PROCESS NAME: MAP LOGICAL TO PHYSICAL DEVICE
PROCESS NUMBER: 1
PROCESS DESCRIPTION:
Read LOGICAL-DEVICE from I/O-REQUEST. Match LOGICAL-DEVICE with entry in DEVICE-DESCRIPTOR-TABLE and read PHYSICAL-DEVICE

PROCESS NAME: ASSEMBLE REQUEST BLOCK
PROCESS NUMBER: 2.1
PROCESS DESCRIPTION:
Create REQUEST-BLOCK from DEVICE-PARAMETERS

PROCESS NAME: NOTIFY DEVICE HANDLER
PROCESS NUMBER: 2.2
PROCESS DESCRIPTION:
Use INTERPROCESS-COMMUNICATION to notify the DEVICE-HANDLER a new REQUEST-BLOCK is in the DEVICE-SERVICE-LIST

PROCESS NAME: ADD TO DEVICE QUEUE
PROCESS NUMBER: 2.3
PROCESS DESCRIPTION:
Insert the REQUEST-BLOCK in the DEVICE-SERVICE-LIST for the appropriate DEVICE-DESCRIPTOR.

PROCESS NAME: REMOVE REQUEST
PROCESS NUMBER: 3.1
PROCESS DESCRIPTION:
Select the highest priority REQUEST-BLOCK from the DEVICE-SERVICE-LIST.

PROCESS NAME: INITIATE I/O
PROCESS NUMBER: 3.2
PROCESS DESCRIPTION:
Read the DEVICE-DESCRIPTOR to determine requirements for I/O and initiate the I/O action.

PROCESS NAME: DELETE FROM LIST
PROCESS NUMBER: 3.3
PROCESS DESCRIPTION:
Delete the REQUEST-BLOCK from the DEVICE-SERVICE-LIST after INITIATE I/O (3.2) has completed.

PROCESS NAME: TRANSLATE DATA
PROCESS NUMBER: 3.4
PROCESS DESCRIPTION:
Translate INPUT-DATA according to the TRANSLATION-TABLE in the DEVICE-DESCRIPTOR

PROCESS NAME: TRANSFER DATA
PROCESS NUMBER: 3.5
PROCESS DESCRIPTION: Move DATA to DESTINATION

PROCESS NAME: NOTIFY PROCESS
PROCESS NUMBER: 3.6
PROCESS DESCRIPTION:
Signal the PROCESS-DESCRIPTOR of the current process when TRANSFER-DATA is complete via INTERPROCESS-COMMUNICATIONS.

PROCESS NAME: CHECK I/O PARAMETERS
PROCESS NUMBER: 4
PROCESS DESCRIPTION:
Check REQUEST-BLOCK against DEVICE-DESCRIPTOR
Check: OPERATION-MODE against CHARACTERISTICS
DESTINATION against OPERATION-MODE
TRANSFER-RATE
QUANITY OF DATA

DATA DICTIONARY
FOR SCHEDULE MANAGEMENT

DATA ELEMENT NAME: BLOCKED-PROCESS
ALIASES: NONE
COMPOSITION: BLOCKED-PROCESS = PROCESS-ID
SOURCE: SCHEDULE NEW PROCESS (1.5)
DESTINATION: CHANGE STATUS TO UNRUNNABLE (2.1.3)
COMMENTS: DETERMINE PROCESS STATUS LEVEL

DATA FLOW NAME: ENVIRONMENT
ALIASES: NONE
COMPOSITION: ENVIRONMENT = PROCESSOR REGISTERS
+ PSW
SOURCE: DETERMINE MEMORY BOUNDS (1.1)
DESTINATION: INITIALIZE PROCESS (1.3)
COMMENTS: CREATE PROCESS LEVEL

DATA FLOW NAME: I/O-REQUEST
ALIASES: NONE
COMPOSITION: I/O-REQUEST = MEMORY-BOUNDS
+ PROCESS LOCATION
SOURCE: INITIATE SWAP I/O (3.4)
DESTINATION: INPUT/OUTPUT MANAGER
COMMENTS: SWAP PROCESS LEVEL

DATA FLOW NAME: JOB-TYPE
ALIASES: NONE
COMPOSITION: JOB-TYPE = USER | SYSTEM
SOURCE: DETERMINE MEMORY BOUNDS (1.1)
DESTINATION: ASSIGN PRIORITY (1.4)
COMMENTS: CREATE PROCESS LEVEL

DATA FLOW NAME: MEMORY-BOUNDS
ALIASES: NONE
COMPOSITION: MEMORY-BOUNDS = STARTING-ADDRESS
+ ENDING-ADDRESS
SOURCE: MEMORY MANAGER
DESTINATION: DETERMINE MEMORY BOUNDS (1.1)
DETERMINE MEMORY AVAILABLE (3.1)
COMMENTS: CREATE PROCESS LEVEL.
SWAP PROCESS LEVEL.

```
DATA FLOW NAME:     MEMORY-REQUEST
ALIASES:            NONE
COMPOSITION:        MEMORY-REQUEST = PROCESS-ID
SOURCE:             DETERMINE MEMORY BOUNDS (1.1)
                    DETERMINE MEMORY AVAILABLE (3.1)
DESTINATION:        MEMORY MANAGER
COMMENTS:           CREATE PROCESS LEVEL.
                    SWAP PROCESS LEVEL.


DATA FLOW NAME:     MEMORY-RESPONSE
ALIASES:            NONE
COMPOSITION:        MEMORY-RESPONSE = MEMORY-BOUNDS
                                    | NO-FIT-MESSAGE
SOURCE:             USER OR SYSTEM REQUEST
DESTINATION:        DETERMINE MEMORY BOUNDS (1.1)
COMMENTS:           CREATE PROCESS LEVEL


DATA FLOW NAME:     NEW-JOB
ALIASES:            NONE
COMPOSITION:        NEW-JOB = PROGRAM-LOCATION
                            + USER-ID
SOURCE:             USER OR SYSTEM REQUEST
DESTINATION:        DETERMINE MEMORY BOUNDS (1.1)
COMMENTS:           CREATE PROCESS LEVEL


DATA FLOW NAME:     NEW-PROCESS
ALIASES:            NONE
COMPOSITION:        NEW-PROCESS = PROCESS-DESCRIPTOR
SOURCE:             SCHEDULE NEW PROCESS (1.5)
DESTINATION:        DETERMINE PROCESS STATUS (2.1.1)
COMMENTS:           DETERMINE PROCESS STATUS LEVEL


DATA FLOW NAME:     PREEMPTED-PROCESS
ALIASES:            NONE
COMPOSITION:        PREEMPTED-PROCESS = PROCESS-ID
SOURCE:             SCHEDULE NEW PROCESS (1.5)
DESTINATION:        CHANGE STATUS RUNNABLE (2.1.2)
COMMENTS:           DETERMINE PROCESS STATUS LEVEL.
                    PREEMPTED BY TIMER INTERRUPT ROUTINE.
                    STATUS CHANGES FROM RUNNING TO RUNNABLE
```

```
DATA FLOW NAME:     PROCESS-CONDITION
ALIASES:            QUEUE-REQUEST
COMPOSITION:        PROCESS-CONDITION = PROCESS-ID
                                      + [ READY-Q | WAIT-Q ]
SOURCE:             REQUEST QUEUE ACTION (2.1.5)
DESTINATION:        SELECT QUEUE ACTION (2.3.1)
COMMENTS:           DETERMINE PROCESS STATUS LEVEL.
                    READY-Q AND WAIT-Q ARE INDICATIONS OF
                    WHERE THE PROCESS MUST BE LOCATED.


DATA FLOW NAME:     PROCESS-IMAGE
ALIASES:            ENVIRONMENT
COMPOSITION:        PROCESS-IMAGE = PROCESSOR REGISTERS
                                  + PSW
COMMENTS:           AN ELEMENT OF PROCESS-DESCRIPTOR


DATA FLOW NAME:     PROCESS-INFO
ALIASES:            NONE
COMPOSITION:        PROCESS-INFO = PROCESS-ID
                                 + [WAIT-Q | READY-Q]
SOURCE:             SELECT QUEUE ACTION (2.3.1)
DESTINATION:        DELETE FROM QUEUE (2.3.2)
COMMENTS:           ENTER PROCESSOR QUEUE LEVEL


DATA FLOW NAME:     QUEUE
ALIASES:            NONE
COMPOSITION:        QUEUE = PROCESS-ID
                          + [WAIT-Q | READY-Q]
SOURCE:             DELETE FROM QUEUE (2.3.2)
DESTINATION:        DETERMINE QUEUE (2.3.3)
COMMENTS:           ENTER PROCESSOR QUEUE LEVEL


DATA FLOW NAME:     QUEUE-PRIORITY
ALIASES:            NONE
COMPOSITION:        QUEUE-PRIORITY = HIGHEST PRIORITY IN
                                     THE READY-QUEUE
                                   + HIGHEST PRIORITY IN
                                     THE WAIT-QUEUE
SOURCE:             DETERMINE QUEUE PRIORITY (2.2.1)
DESTINATION:        DETERMINE STATUS (2.2.2)
COMMENTS:           DETERMINE RUNNING PROCESS
```

DATA FLOW NAME: QUEUE-REQUEST
ALIASES: PROCESS-CONDITION
COMPOSITION: SEE ALIASE
SOURCE: REQUEST QUEUE ACTION (2.1.5)
DESTINATION: SELECT QUEUE ACTION (2.3.1)
COMMENTS: ENTER PROCESSOR QUEUE LEVEL

DATA FLOW NAME: QUEUE-STATUS
ALIASES: NONE
COMPOSITION: QUEUE-STATUS = PROCESS-ID IN WAIT-QUEUE
| PROCESS-ID IN READY-QUEUE
SOURCE: DETERMINE STATUS (2.2.2)
DESTINATION: DETERMINE LOCATION (2.2.3)
COMMENTS: DETERMINE RUNNING PROCESS LEVEL

DATA ELEMENT NAME: READY-Q
ALIASES: NONE
COMPOSITION: READY-Q = INDICATION OF WHERE A PROCESS IS MOVING TO
COMMENTS: USED IN THE ENTER PROCESSOR QUEUE LEVEL AND THE DETERMINE PROCESS STATUS LEVEL. CONVERSE IS WAIT-Q.

DATA FLOW NAME: READY-QUEUE-INFO
ALIASES: NONE
COMPOSITION: READY-QUEUE-INFO = PROCESS-ID
SOURCE: DETERMINE QUEUE (2.3.3)
DESTINATION: ADD TO QUEUE (2.3.5)
COMMENTS: ENTER PROCESSOR QUEUE LEVEL

DATA ELEMENT NAME: READY-QUEUE-PRIORITIES
ALIASES: NONE
COMPOSITION: READY-QUEUE-PRIORITIES = HIGHEST PRIORITY IN THE READY-QUEUE
SOURCE: READY-QUEUE
DESTINATION: DETERMINE QUEUE PRIORITY (2.2.1)
COMMENTS: DETERMINE RUNNING PROCESS LEVEL

```
DATA FLOW NAME:     RUN-PROCESS
ALIASES:            RUN-PROCESS-ID
COMPOSITION:        RUN-PROCESS = PROCESS-ID
SOURCE:             DETERMINE LOCATION (2.2.3)
DESTINATION:        CHANGE STATUS TO RUNNING (2.1.4)
COMMENTS:           DETERMINE RUNNING PROCESS LEVEL.
                    HIGHEST RUNNABLE PROCESS IN SYSTEM


DATA FLOW NAME:     RUN-PROCESS-ID
ALIASES:            RUN-PROCESS
COMPOSITION:        SEE ALIASE
SOURCE:             SEE ALIASE
DESTINATION:        SEE ALIASE
COMMENTS:           DETERMINE PROCESS STATUS LEVEL


DATA FLOW NAME:     RUNNABLE-PROCESS-ID
ALIASES:            NONE
COMPOSITION:        RUNNABLE-PROCESS-ID = PROCESS-ID
SOURCE:             DETERMINE PROCESS STATUS (2.1.1)
DESTINATION:        CHANGE STATUS TO RUNNABLE (2.1.2)
COMMENTS:           DETERMINE PROCESS STATUS LEVEL


DATA FLOW NAME:     RUNNABLE-STATUS
ALIASES:            NONE
COMPOSITION:        RUNNABLE-STATUS = PROCESS-ID + READY-Q
SOURCE:             CHANGE STATUS TO RUNNABLE (2.1.2)
DESTINATION:        REQUEST QUEUE ACTION (2.1.5)
COMMENTS:           DETERMINE PROCESS STATUS LEVEL


DATA FLOW NAME:     STATUS
ALIASES:            NONE
COMPOSITION:        STATUS = RUNNING
                           | RUNNABLE
                           | UNRUNNABLE
SOURCE:             SCHEDULER
DESTINATION:        PROCESS-DESCRIPTOR
COMMENTS:           AN ELEMENT OF A PROCESS-DESCRIPTOR
```

DATA FLOW NAME: SWAP-APPROVAL
ALIASES: NONE
COMPOSITION: SWAP-APPROVAL = MEMORY-RESPONSE
+ PROCESS-ID
SOURCE: DETERMINE MEMORY AVAILABLE (3.1)
DESTINATION: EXECUTE SWAP-IN (3.2)
COMMENTS: SWAP PROCESS LEVEL

DATA FLOW NAME: SWAP-IN-INFO
ALIASES: NONE
COMPOSITION: SWAP-IN-INFO = SWAP-APPROVAL
+ PROCESS-LOCATION
SOURCE: EXECUTE SWAP IN (3.2)
DESTINATION: INITIATE SWAP I/O (3.4)
COMMENTS: SWAP PROCESS LEVEL
REQUIRED TO INITIATE I/O REQUEST

DATA FLOW NAME: SWAP-IN-REQUEST
ALIASES: NONE
COMPOSITION: SWAP-IN-REQUEST = PROCESS-ID
+ PROCESS-LOCATION
SOURCE: DETERMINE LOCATION (2.2.3)
DESTINATION: DETERMINE MEMORY AVAILABLE (3.1)
COMMENTS: SWAP PROCESS LEVEL

DATA FLOW NAME: SWAP-OUT-INFO
ALIASES: NONE
COMPOSITION: SWAP-OUT-INFO = PROCESS-ID
+ MEMORY-ADDRESS
SOURCE: EXECUTE SWAP OUT (3.3)
DESTINATION: INITIATE SWAP I/O (3.4)
COMMENTS: SWAP PROCESS LEVEL.
REQUIRED TO INITIATE I/O REQUEST.

DATA FLOW NAME: SWAP-OUT-REQUEST
ALIASES: NONE
COMPOSITION: SWAP-OUT-REQUEST = PROCESS-ID
SOURCE: DETERMINE LOCATION (2.2.3)
DESTINATION: EXECUTE SWAP OUT (3.3)
COMMENTS: SWAP PROCESS LEVEL

DATA FLOW NAME: SWAP-OUT-REQUIREMENT
ALIASES: NONE
COMPOSITION: SWAP-OUT-REQUIREMENT = NO-FIT-MESSAGE
SOURCE: DETERMINE MEMORY AVAILABLE (3.1)
DESTINATION: DETERMINE QUEUE PRIORITY (2.2.1)
COMMENTS: SWAP PROCESS LEVEL.
INDICATES ROOM MUST BE MADE IN MEMORY
BEFORE A SWAP-IN CAN OCCUR.

DATA FLOW NAME: SWAP-REQUEST
ALIASES: NONE
COMPOSITION: SWAP-REQUEST = SWAP-IN-REQUEST
| SWAP-OUT-REQUEST
SOURCE: DETERMINE LOCATION (2.2.3)
DESTINATION: EXECUTE MEMORY SWAP (3)
COMMENTS: DETERMINE RUNNING PROCESS

DATA ELEMEMT NAME: WAIT-Q
ALIASES: NONE
COMPOSITION: WAIT-Q = INDICATION OF WHERE A PROCESS
IS MOVING TO
COMMENTS: USED IN THE ENTER PROCESSOR QUEUE LEVEL
AND THE DETERMINE PROCESS STATUS LEVEL.
CONVERSE IS READY-Q.

DATA FLOW NAME: WAIT-QUEUE-INFO
ALIASES: NONE
COMPOSITION: WAIT-QUEUE-INFO = PROCESS-ID
SOURCE: DETERMINE QUEUE (2.3.3)
DESTINATION: ADD TO WAIT QUEUE (2.3.4)
COMMENTS: ENTER PROCESSOR QUEUE LEVEL

DATA FLOW NAME: WAIT-QUEUE-PRIORITIES
ALIASES: NONE
COMPOSITION: WAIT-QUEUE-PRIORITIES = HIGHEST PRIORITY
IN THE WAIT-QUEUE
SOURCE: WAIT-QUEUE
DESTINATION: DETERMINE QUEUE PRIORITY (2.2.1)
COMMENTS: DETERMINE RUNNING PROCESS LEVEL

DATA FLOW NAME: UNRUNNABLE-PROCESS-ID
ALIASES: NONE
COMPOSITION: UNRUNNABLE-PROCESS-ID = PROCESS-ID
SOURCE: DETERMINE PROCESS STATUS (2.1.1)
DESTINATION: CHANGE STATUS TO UNRUNNABLE (2.1.3)
COMMENTS: DETERMINE PROCESS STATUS LEVEL

DATA FLOW NAME: UNRUNNABLE-STATUS
ALIASES: NONE
COMPOSITION: UNRUNNABLE-STATUS = PROCESS-ID + WAIT-Q
SOURCE: CHANGE STATUS TO UNRUNNABLE (2.1.3)
DESTINATION: REQUEST QUEUE ACTION (2.1.5)
COMMENTS: DETERMINE PROCESS STATUS

FILE DEFINITIONS
FOR SCHEDULE MANAGEMENT

FILE OR DATABASE NAME: NEW-PROCESS-DESCRIPTOR
ALIASES: NONE
COMPOSITION: NEW-PROCESS-DESCRIPTOR = PROCESS-DESCRIPTOR
ORGANIZATION: RECORD
COMMENTS: CREATE PROCESS LEVEL

FILE OR DATABASE NAME: PROCESS-DESCRIPTOR
ALIASES: PROCESS
COMPOSITION: PROCESS-DESCRIPTOR = USER-ID
+ PROCESS-ID
+ PRIORITY
+ STATUS
+ MEMORY-BOUNDS
+ PROCESS-IMAGE
+ POINTER-TO-NEXT-PROCESS
ORGANIZATION: RECORD
COMMENTS: ELEMENT OF PROCESS-STRUCTURE

FILE OR DATABASE NAME: PROCESS-STRUCTURE
ALIASES: PROCESSOR-QUEUE
COMPOSITION: PROCESS-STRUCTURE = READY-QUEUE
+ WAIT-QUEUE
ORGANIZATION:
COMMENTS: SCHEDULE MANAGEMENT OVERVIEW LEVEL

FILE OR DATABASE NAME: READY-QUEUE
ALIASES: NONE
COMPOSITION: READY-QUEUE = {PROCESS-DESCRIPTOR}
ORGANIZATION: LINKED LIST
COMMENTS: EXECUTE SCHEDULER LEVEL
ENTER PROCESSOR QUEUE LEVEL

FILE OR DATABASE NAME: WAIT-QUEUE
ALIASES: NONE
COMPOSITION: WAIT-QUEUE = {PROCESS-DESCRIPTOR}
ORGANIZATION: LINKED LIST
COMMENTS: EXECUTE SCHEDULER LEVEL
ENTER PROCESSOR QUEUE LEVEL

PROCESS DESCRIPTIONS
FOR SCHEDULE MANAGEMENT

```
PROCESS NAME:          DETERMINE PROCESS IMAGE
PROCESS NUMBER:        1.1
PROCESS DESCRIPTION:
Request memory for NEW-PROCESS
If memory is available
   then record MEMORY-BOUNDS in PROCESS-DESCRIPTOR and
   make STATUS = RUNNABLE
else make STATUS = UNNRUNNABLE
```

```
PROCESS NAME:          INITIALIZE PROCESS
PROCESS NUMBER:        1.2
PROCESS DESCRIPTION:
If PROCESS is a NEW-PROCESS
  then set PSW and PROCESSOR-REGISTERS to nil
```

```
PROCESS NAME:          ASSIGN PRIORITY
PROCESS NUMBER:        1.3
PROCESS DESCRIPTION:
If PROCESS is a SYSTEM-PROCESS
   then set PRIORITY high
else set PRIORITY low
```

```
PROCESS NAME:          SCHEDULE
PROCESS NUMBER:        1.4
PROCESS DESCRIPTION:
Interrupt SCHEDULER to introduce NEW-PROCESS
Extract NEW-PROCESS-DESCRIPTOR and send to SCHEDULER
```

```
PROCESS NAME:          DETERMINE PROCESS STATUS
PROCESS NUMBER:        2.1.1
PROCESS DESCRIPTION:
If NEW-PROCESS is in memory
   then PROCESS is RUNNABLE
If NEW-PROCESS is not in memory
   then PROCESS is UNRUNNABLE
```

```
PROCESS NAME:          CHANGE TO RUNNABLE
PROCESS NUMBER:        2.1.2
PROCESS DESCRIPTION:
If PROCESS was preempted by a timer interrupt
   then change STATUS from RUNNING to RUNNABLE
If PROCESS is new
   then STATUS = RUNNABLE


PROCESS NAME:          CHANGE TO UNRUNNABLE
PROCESS NUMBER:        2.1.3
PROCESS DESCRIPTION:
If PROCESS is BLOCKED-PROCESS
   then STATUS = UNRUNNABLE and record UNRUNNABLE-STATUS
If PROCESS is NEW-PROCESS
   then STATUS = UNRUNNABLE and UNRUNNABLE-STATUS = NO-
   MEMORY


PROCESS NAME:          CHANGE TO RUNNING
PROCESS NUMBER:        2.1.4
PROCESS DESCRIPTION:
CHANGE the PROCESS-DESCRIPTOR in READY-QUEUE associated
with PROCESS-ID to indicate STATUS = RUNNING.


PROCESS NAME:          DETERMINE QUEUE CHANGES
PROCESS NUMBER:        2.1.5
PROCESS DESCRIPTION:
If PROCESS-STATUS = RUNNABLE
   then PROCESS-DESCRIPTOR must be in READY-QUEUE
If PROCESS-STATUS = UNRUNNABLE
   then PROCESS-DESCRIPTOR must be in WAIT-QUEUE


PROCESS NAME:          DETERMINE QUEUE PRIORITY
PROCESS NUMBER:        2.2.1
PROCESS DESCRIPTION:
Read the highest PRIORITY in the READY-QUEUE
Read the highest PRIORITY in the WAIT-QUEUE blocked
for memory
```

```
PROCESS NAME:          DETERMINE STATUS
PROCESS NUMBER:        2.2.2
PROCESS DESCRIPTION:
If a WAIT-QUEUE PROCESS has the highest of the two priorities
   then it is selected
else the highest in the READY-QUEUE is selected


PROCESS NAME:          DETERMINE LOCATION
PROCESS NUMBER:        2.2.3
PROCESS DESCRIPTION:
If a WAIT-QUEUE PROCESS was selected
   then make a SWAP-IN-REQUEST
If a READY-QUEUE PROCESS was selected
   then it is the RUN-PROCESS


PROCESS NAME:          SELECT QUEUE ACTIONS
PROCESS NUMBER:        2.3.1
PROCESS DESCRIPTION:
If QUEUE-REQUEST = WAIT-Q
   then delete from READY-QUEUE and add to WAIT-QUEUE
If QUEUE-REQUEST = READY-Q
   then delete from WAIT-QUEUE and add to READY-QUEUE
If QUEUE-REQUEST = terminate PROCESS
   then delete from PROCESS-STRUCTURE


PROCESS NAME:          DELETE FROM QUEUE STRUCTURE
PROCESS NUMBER:        2.3.2
PROCESS DESCRIPTION:
If PROCESS-INFO not equal to terminate
   then QUEUE = PROCESS-INFO
Delete PROCESS from PROCESS-STRUCTURE


PROCESS NAME:          DETERMINE QUEUE ENTRY
PROCESS NUMBER:        2.3.3
PROCESS DESCRIPTION:
If QUEUE contains WAIT-Q
   then add to WAIT-QUEUE
IF QUEUE contains READY-Q
   then add to READY-QUEUE
```

```
PROCESS NAME:          ADD TO WAIT QUEUE
PROCESS NUMBER:        2.3.4
PROCESS DESCRIPTION:
Examine PROCESS-PRIORITY of PROCESS-ID
Insert in WAIT-QUEUE by priority


PROCESS NAME:          ADD TO READY QUEUE
PROCESS NUMBER:        2.3.5
PROCESS DESCRIPTION:
Examine PROCESS-PRIORITY of PROCESS-ID
Insert in READY-QUEUE by priority


PROCESS NAME:          DETERMINE MEMORY
PROCESS NUMBER:        3.1
PROCESS DESCRIPTION:
Request memory for the process to swap-in
If MEMORY-RESPONSE is negative
   then issue a SWAP-OUT-REQUIREMENT
If MEMORY-RESPONSE is positive
   then issue a SWAP-APPROVAL


PROCESS NAME:          EXECUTE SWAP-IN
PROCESS NUMBER:        3.2
PROCESS DESCRIPTION:
Determine location of the process to swap-in


PROCESS NAME:          EXECUTE SWAP-OUT
PROCESS NUMBER:        3.3
PROCESS DESCRIPTION:
Determine location of the process to swap-out
```

PROCESS NAME: INITIATE I/O
PROCESS NUMBER: 3.4
PROCESS DESCRIPTION:
Issue I/O-REQUEST to the INPUT/OUTPUT MANAGER

DATA DICTIONARY
FOR MEMORY MANAGEMENT

DATA ELEMENT NAME: ACCEPTED-FREE-AREA
ALIASES: NONE
COMPOSITION: ACCEPTED-FREE-AREA = MEMORY-SIZE
- PROCESS-SIZE
SOURCE: COMPARE AREA WITH SIZE (2.2)
DESTINATION: UPDATE FREE AREA TABLE (2.3)
COMMENTS: SELECT FREE AREA LEVEL

DATA FLOW NAME: AREA-PARAMETERS
ALIASES: NONE
COMPOSITION: AREA-PARAMETERS = PROCESS-ID
+ MEMORY-SIZE
+ MEMORY-LOCATION
SOURCE: MATCH ENTRY AND PROCESS (5.1)
DESTINATION: ADJUST FREE SPACE (5.3)
UPDATE MEMORY TABLE (5.2)
COMMENTS: DEALLOCATE MEMORY SPACE LEVEL

DATA FLOW NAME: FREE-AREA-SIZE
ALIASES: NONE
COMPOSITION: FREE-AREA-SIZE = MEMORY-SIZE
SOURCE: EXAMINE FREE AREA (2.1)
DESTINATION: COMPARE AREA WITH SIZE (2.2)
COMMENTS: SELECT FREE AREA LEVEL

DATA FLOW NAME: DEALLOCATE-REQUEST
ALIASES: PROCESS-ID
COMPOSITION: DEALLOCATE-REQUEST = PROCESS-I D
SOURCE: DETERMINE NEED (1)
DESTINATION: DEALLOCATE MEMORY (5)
COMMENTS: MEMORY MANAGEMENT LEVEL

DATA FLOW NAME: MEMORY-BOUNDS
ALIASES: NONE
COMPOSITION: MEMORY-BOUNDS = STARTING-ADDRESS
+ ENDING-ADDRESS
SOURCE: RECORD PROCESS BOUNDS (4)
DESTINATION: MEMORY REQUEST SOURCE
COMMENTS: MEMORY MANAGEMENT LEVEL

```
DATA FLOW NAME:      NO-FIT-MESSAGE
ALIASES:             NONE
COMPOSITION:         NO-FIT-MESSAGE = INDICATION OF LACK
                                      OF MEMORY FOR PROCESS-SIZE
SOURCE:              COMPARE AREA WITH SIZE (2.2)
DESTINATION:         RECORD PROCESS BOUNDS (4)
COMMENTS:            SELECT FREE AREA LEVEL


DATA FLOW NAME:      PROCESS-BOUNDS
ALIASES:             NONE
COMPOSITION:         PROCESS-BOUNDS = NO-FIT-MESSAGE
                                    | MEMORY-BOUNDS
SOURCE:              SELECT AREA (2)
DESTINATION:         RECORD PROCSS BOUNDS (4)
COMMENTS:            MEMORY MANAGEMENT LEVEL


DATA ELEMENT NAME:   PROCESS-ID
ALIASES:             DEALLOCATE-REQUEST
COMPOSITION:         PROCESS-ID = UNIQUE ID ASSIGNED BY SYSTEM
SOURCE:              DETERMINE NEED (1)
DESTINATION:         MATCH ENTRY AND PROCESS (5.1)
COMMENTS:            DEALLOCATE MEMORY SPACE LEVEL


DATA FLOW NAME:      PROCESS-MAP
ALIASES:             NONE
COMPOSITION:         PROCESS-MAP = PROCESS-ID
                                 + MEMORY-LOCATION
                                 + MEMORY-SIZE
SOURCE:              SELECT AREA (2)
DESTINATION:         ASSIGN AREA (3)
COMMENTS:            MEMORY MANAGEMENT LEVEL


DATA FLOW NAME:      PROCESS-NEEDS
ALIASES:             NONE
COMPOSITION:         PROCESS-NEEDS = DEALLOCATE-REQUEST
                                   | PROCESS-SIZE
SOURCE:              SCHEDULER LEVEL
DESTINATION:         DETERMINE NEED (1)
COMMENTS:            MEMORY MANAGEMENT LEVEL
```

DATA FLOW NAME:	PROCESS-SIZE
ALIASES:	NONE
COMPOSITION:	PROCESS-SIZE = AMOUNT OF MEMORY REQUIRED
SOURCE:	DETERMINE NEED (1)
DESTINATION:	SELECT AREA (2)
COMMENTS:	MEMORY MANAGEMENT LEVEL

FILE DEFINITIONS
FOR MEMORY MANAGER

FILE OR DATABASE NAME:	FREE-SPACE-TABLE
ALIASES:	NONE
COMPOSITION:	FREE-SPACE-TABLE = {MEMORY-SIZE + MEMORY-LOCATION + MEMORY-STATUS}
ORGANIZATION:	TABULAR BY SIZE
COMMENTS:	DEALLOCATE MEMORY SPACE LEVEL

FILE OR DATABASE NAME:	MEMORY-MAP-TABLE
ALIASES:	NONE
COMPOSITION:	MEMORY-MAP-TABLE = {PROCESS-ID + MEMORY-SIZE + MEMORY-LOCATION + MEMORY-STATUS}
ORGANIZATION:	TABULAR BY PROCESS-ID
COMMENTS:	MEMORY MANAGEMENT LEVEL, *DEALLOCATE MEMORY SPACE LEVEL*

PROCESS DESCRIPTIONS
FOR MEMORY MANAGEMENT

PROCESS NAME: DETERMINE NEED
PROCESS NUMBER: 1
PROCESS DESCRIPTION:
If PROCESS-NEEDS contain MEMORY-SIZE and PROCESS-ID
then MEMORY-SIZE = PROCESS-SIZE
If PROCESS-NEEDS contain only PROCESS-ID
then DEALLOCATE MEMORY of PROCESS-ID

PROCESS NAME: ASSIGN AREA
PROCESS NUMBER: 3
PROCESS DESCRIPTION:
Assign the process an entry in the MEMORY-MAP-TABLE

PROCESS NAME: RECORD PROCESS BOUNDS
PROCESS NUMBER: 4
PROCESS DESCRIPTION:
Write the starting address and ending address to the
PROCESS-DESCRIPTOR of the associated process

PROCESS NAME: EXAMINE FREE AREA
PROCESS NUMBER: 2.1
PROCESS DESCRIPTION:
Select a free area from the FREE-SPACE-TABLE and
determine its size.

PROCESS NAME: COMPARE AREA WITH SIZE
PROCESS NUMBER: 2.2
PROCESS DESCRIPTION:
If PROCESS-SIZE is less than or equal to FREE-AREA-SIZE
then accept the FREE-AREA
else reject the FREE-AREA

PROCESS NAME: UPDATE FREE AREA TABLE
PROCESS NUMBER: 2.3
PROCESS DESCRIPTION:
Remove the ACCEPTED-FREE-AREA from the FREE-AREA-TABLE

PROCESS NAME: MATCH ENTRY AND PROCESS
PROCESS NUMBER: 5.1
PROCESS DESCRIPTION:
Search MEMORY-MAP-TABLE for PROCESS-ID
AREA-PARAMETERS = MEMORY-SIZE + MEMORY-LOCATION of entry

PROCESS NAME: UPDATE MEMORY TABLE
PROCESS NUMBER: 5.2
PROCESS DESCRIPTION:
Change MEMORY-STATUS to AVAILABLE
Delete PROCESS-ID entry from MEMORY-MAP-TABLE

PROCESS NAME: ADJUST FREE SPACE
PROCESS NUMBER: 5.3
PROCESS DESCRIPTION:
Add FREE-SPACE to FREE-SPACE-TABLE.
Merge with adjacent areas.

DATA DICTIONARY
FOR THE NUCLEUS

DATA ELEMENT NAME: CURRENT-PRIORITY
ALIASES: NONE
COMPOSITION: CURRENT-PRIORITY = INTERRUPT-PRIORITY
SOURCE: DETERMINE PRIORITY (3.3)
DESTINATION: DISABLE LOWER PRIORITIES (3.4)
COMMENTS: INTERRUPT HANDLER LEVEL
CURRENT INTERRUPT EXECUTING

DATA ELEMENT NAME: DISABLE-MESSAGE
ALIASES: NONE
COMPOSITION: DISABLE-MESSAGE = CURRENT-PRIORITY
SOURCE: DISABLE LOWER PRIORITIES (3.4)
DESTINATION: PROCESSOR
COMMENTS: INTERRUPT HANDLER LEVEL

DATA ELEMENT NAME: HIGHEST-PRIORITY
ALIASES: NONE
COMPOSITION: HIGHEST-PRIORITY = INTERRUPT-PRIORITY
SOURCE: DETERMINE PRIORITY (3.3)
DESTINATION: DETERMINE INTERRUPT ROUTINE LOCATION (3.5)
COMMENTS: INTERRUPT HANDLER LEVEL

DATA ELEMENT NAME: INTERRUPT-ID
ALIASES: NONE
COMPOSITION: INTERRUPT-ID = INTERRUPT-NUMBER
SOURCE: INTERRUPT SOURCE (SINK)
DESTINATION: IDENTIFY INTERRUPT SOURCE (3.2)
COMMENTS: INTERUPT HANDLER LEVEL

DATA FLOW NAME: LOCATION
ALIASES: NONE
COMPOSITION: LOCATION = INTERRUPT-ROUTINE-ADDRESS
SOURCE: DETERMINE INTERRUPT ROUTINE LOCATION (3.5)
DESTINATION: SERVICE INTERRUPT (3.6)
COMMENTS: INTERRUPT HANDLER LEVEL

DATA FLOW NAME: PROCESS-CONDITIONS
ALIASES: NONE
COMPOSITION: PROCESS-CONDITIONS = PROCESS LOCATION
SOURCE: INTERRUPT ROUTINE
DESTINATION: BLOCK (2.2) OR AWAKEN (2.3)
COMMENTS: INTERPROCESS COMMUNICATION LEVEL

DATA FLOW NAME: PROCESS-DESCRIPTOR
ALIASES: NONE
COMPOSITION: PROCESS-DESCRIPTOR = PROCESS-ID
+ PROCESS-IMAGE
+ MEMORY-BOUNDS
+ USER-ID
+ STATUS
SOURCE: TEST FOR CURRENT PROCESS (1.1)
DESTINATION: UPDATE PROCESSOR STATE (1.2)
COMMENTS: DISPATCH PROCESS LEVEL

DATA FLOW NAME: PROCESS-ENVIRONMENT
ALIASES: PROCESS-IMAGE
COMPOSITION: PROCESS-ENVIRONMENT = PROCESSOR-
REGISTERS + PSW
SOURCE: HANDLE INTERRUPT (2)
DESTINATION: PROCESSOR
COMMENTS: NUCLEUS OVERVIEW LEVEL

DATA FLOW NAME: PROCESS-ID
ALIASES: NONE
COMPOSITION: PROCESS-ID = UNIQUE ID SET BY SYSTEM
SOURCE: TEST FOR CURRENT PROCESS (1.1)
DESTINATION: RECORD PROCESSOR STATE (1.3)
COMMENTS: DISPATCH PROCESS LEVEL

DATA FLOW NAME: PROCESS-STATES
ALIASES: PROCESS-IMAGE
COMPOSITION: PROCESS-STATES = PROCESSOR-REGISTERS
+ PSW
SOURCE: PROCESSOR
DESTINATION: RECORD PROCESSOR STATES (1.3)
COMMENTS: DISPATCH PROCESS LEVEL

DATA FLOW NAME:	PROCESS-STATUS
ALIASES:	NONE
COMPOSITION:	PROCESS-STATUS = RUNNABLE | UNRUNNABLE
SOURCE:	BLOCK (2.2) OR AWAKEN (2.3)
DESTINATION:	PROCESS-STRUCTURES
COMMENTS:	INTERPROCESS COMMUNICATION LEVEL. RESULTS OF BLOCK OR AWAKEN.

DATA FLOW NAME:	PROCESS-STATUS-CHANGE
ALIASES:	NONE
COMPOSITION:	PROCESS-STATUS-CHANGE = RUNNING | RUNNABLE | UNRUNNABLE
SOURCE:	SERVICE INTERRUPT SERVICE (3.6)
DESTINATION:	PROCESS-STRUCTURES
COMMENTS:	INTERRUPT HANDLER LEVEL

DATA FLOW NAME:	RUNNABLE-PROCESS-STATE
ALIASES:	PROCESSOR-STATES
COMPOSITION:	SEE ALIASE
SOURCE:	UPDATE PROCESSOR STATE (1.2)
DESTINATION:	PROCESSOR
COMMENTS:	DISPATCH PROCESS LEVEL

FILE DEFINITIONS
FOR THE NUCLEUS

```
FILE OR DATABASE NAME:  CPU-DESCRIPTOR
ALIASES:                NONE
COMPOSITION:            CPU-DESCRIPTOR = CURRENT RUNNING
                                         PROCESS
                                       + POINTER TO HEAD OF
                                         READY-QUEUE
                                       + POINTER TO HEAD OF
                                         WAIT-QUEUE
ORGANIZATION:           TABULAR
COMMENTS:               DISPATCH PROCESS LEVEL


FILE OR DATABASE NAME:  INTERRUPT-VECTOR-TABLE
ALIASES:                NONE
COMPOSITION:            INTERRUPT-VECTOR-TABLE =
                              { INTERRUPT-NUMBER
                              + INTERRUPT-ROUTINE-ADDRESS}
ORGANIZATION:           SEQUENTIAL BY INTERRUPT-NUMBER
COMMENTS:               INTERRUPT HANDLER LEVEL


FILE OR DATABASE NAME:  PROCESSOR-QUEUE
ALIASES:                NONE
COMPOSITION:            PROCESSOR-QUEUE = READY-QUEUE
                                        + WAIT-QUEUE
ORGANIZATION:           LINKED LISTS
COMMENTS:               DISPATCH PROCESS LEVEL


FILE OR DATABASE NAME:  PROCESSOR-REGISTERS
ALIASES:                NONE
COMPOSITION:            PROCESSOR-REGISTERS = PSW
                              + GENERAL PURPOSE REGISTERS
ORGANIZATION:           TABULAR
COMMENTS:               SAVE AND RESTORE CPU LEVEL.


FILE OR DATABASE NAME:  PROCESSOR-STACK
ALIASES:                NONE
COMPOSITION:            PROCESSOR-STACK = CPU STORAGE
ORGANIZATION:           LAST-IN-LAST-OUT
COMMENTS:               SAVE AND RESTORE CPU LEVEL
```

FILE OR DATABASE NAME:	PROCESS-STRUCTURES
ALIASES:	NONE
COMPOSITION:	PROCESS-STRUCTURES = PROCESSOR-QUEUE + CPU-DESCRIPTOR + FILE-DESCRIPTORS + DEVICE-SERVICE-LIST
ORGANIZATION:	LINKED LISTS AND TABULAR
COMMENTS:	NUCLEUS OVERVIEW LEVEL

PROCESS DESCRIPTIONS
FOR THE NUCLEUS

PROCESS NAME: TEST FOR CURRENT PROCESS
PROCESS NUMBER: 1.1
PROCESS DESCRIPTION:
Compare front of READY-QUEUE to the CURRENT-PROCESS
If the same process
then execute CURRENT-PROCESS
else read the PROCESSOR-ID of the CURRENT-PROCESS
Select the front process in the READY-QUEUE to run.
Read the PROCESS-DESCRIPTOR and change the CURRENT-PROCESS.

PROCESS NAME: UPDATE PROCESSOR STATE
PROCESS NUMBER: 1.2
PROCESS DESCRIPTION:
Load the PROCESS-STATES into the PROCESSOR-REGISTERS

PROCESS NAME: RECORD PROCESSOR STATE
PROCESS NUMBER: 1.3
PROCESS DESCRIPTION:
Read the PROCESSOR-REGISTERS and record them in the PROCESS-DESCRIPTOR of PROCESS-ID

PROCESS NAME: LOCK
PROCESS NUMBER: 2.1
PROCESS DESCRIPTION:
Disable the CPU from any outside interruptions

PROCESS NAME: BLOCK
PROCESS NUMBER: 2.2
PROCESS DESCRIPTION:
Prevent a PROCESS from executing unless a specific condition is satisfied.

PROCESS NAME: AWAKEN
PROCESS NUMBER: 2.3
PROCESS DESCRIPTION:
Remove the effect of the BLOCK process and allow execution.

PROCESS NAME: UNLOCK
PROCESS NUMBER: 2.4
PROCESS DESCRIPTION:
Remove the LOCK process (2.1)

PROCESS NAME: SAVE CPU STATE
PROCESS NUMBER: 3.1
PROCESS DESCRIPTION:
Read the PROCESSOR-REGISTERS and write them to the
PROCESSOR-STACK

PROCESS NAME: IDENTIFY INTERRUPT SOURCE
PROCESS NUMBER: 3.2
PROCESS DESCRIPTION:
Compare the INTERRUPT-ID with interrupt sources to
determine the SOURCE-ID

PROCESS NAME: DETERMINE PRIORITY
PROCESS NUMBER: 3.3
PROCESS DESCRIPTION:
Read the INTERRUPT-PRIORITY of the SOURCE-ID
If the INTERRUPT-PRIORITY of the source is higher than the
current interrupt rountine
   then suspend the current interrupt
   CURRENT-PRIORITY = higher interrupt priority.

PROCESS NAME: DISABLE LOWER PRIORITES
PROCESS NUMBER: 3.4
PROCESS DESCRIPTION:
If interrupt priority is lower or equal to CURRENT-PRIORITY
then ignore interrupt.

PROCESS NAME: DETERMINE INTERRUPT ROUTINE LOCATION
PROCESS NUMBER: 3.5
PROCESS DESCRIPTION:
Match the HIGHEST-PRIORITY with the entry in the INTERRUPT-VECTOR-TABLE. Read the LOCATION of the interrupt routine.

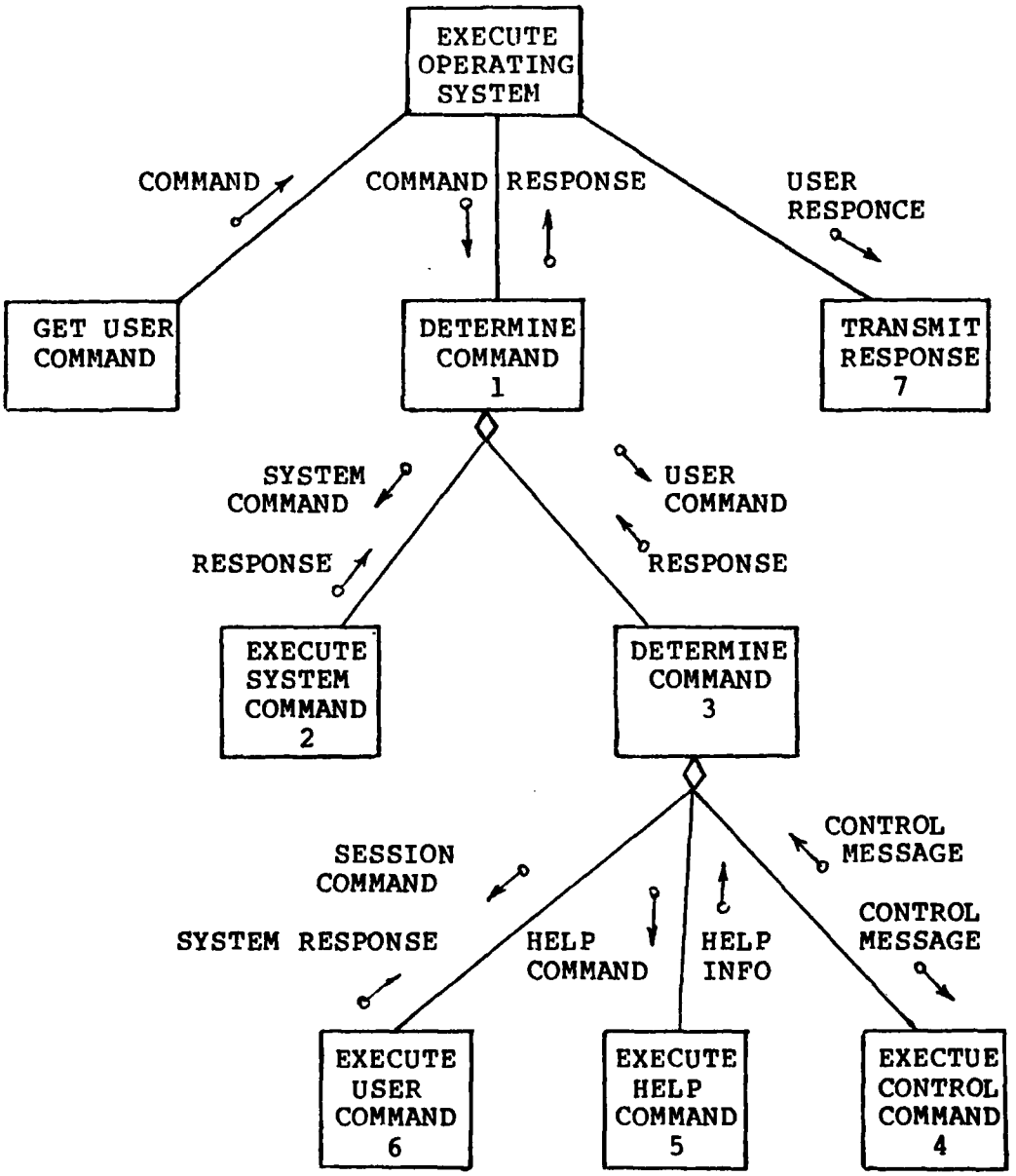
PROCESS NAME: SERVICE INTERRUPT
PROCESS NUMBER: 3.6
PROCESS DESCRIPTION:
Go to the LOCATION of the interrupt routine and execute.

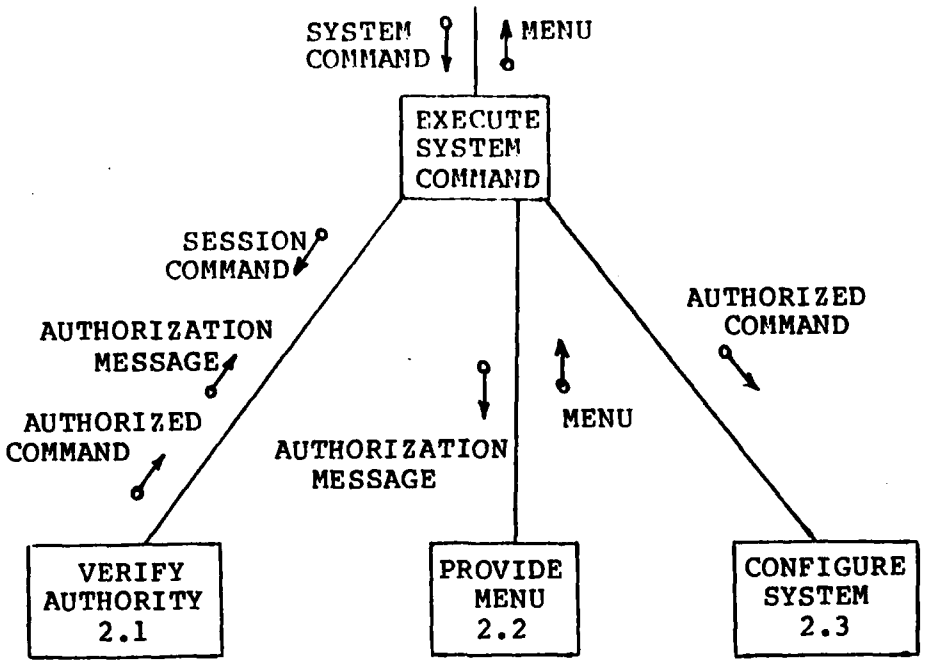
Appendix F

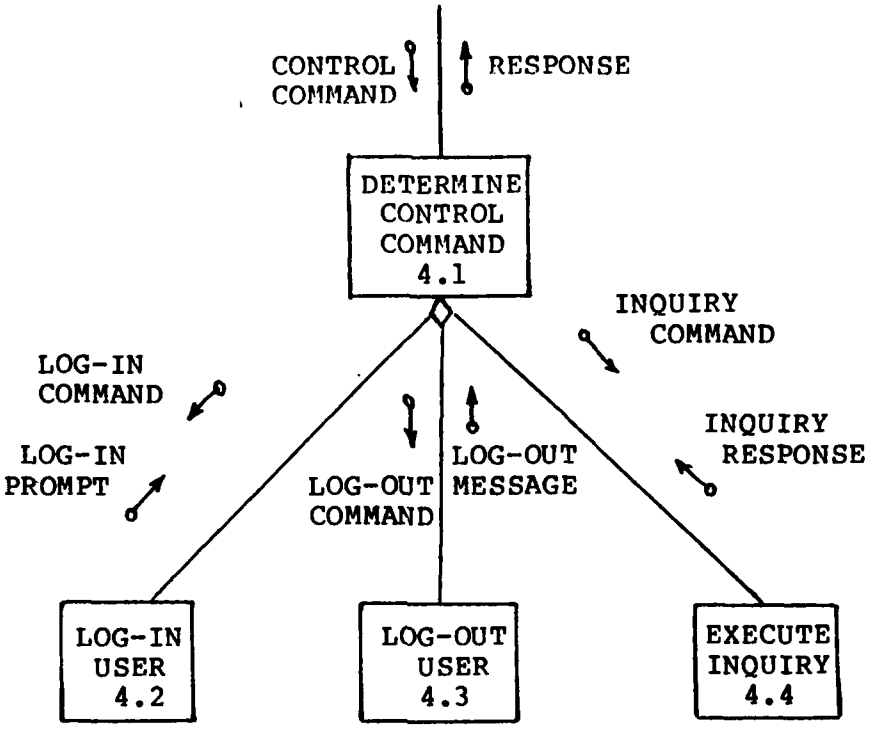
Module Structure Charts

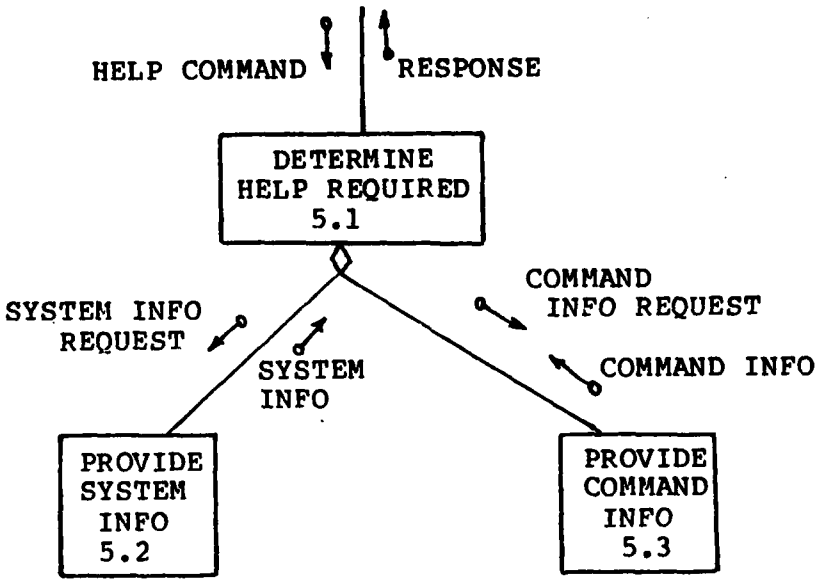
This appendex contains the module structure charts developed in Chapter Five. Techniques used to develop the charts are from Weinberg's Structured Analysis. A summary of notation and development can be found in Chapter Five.

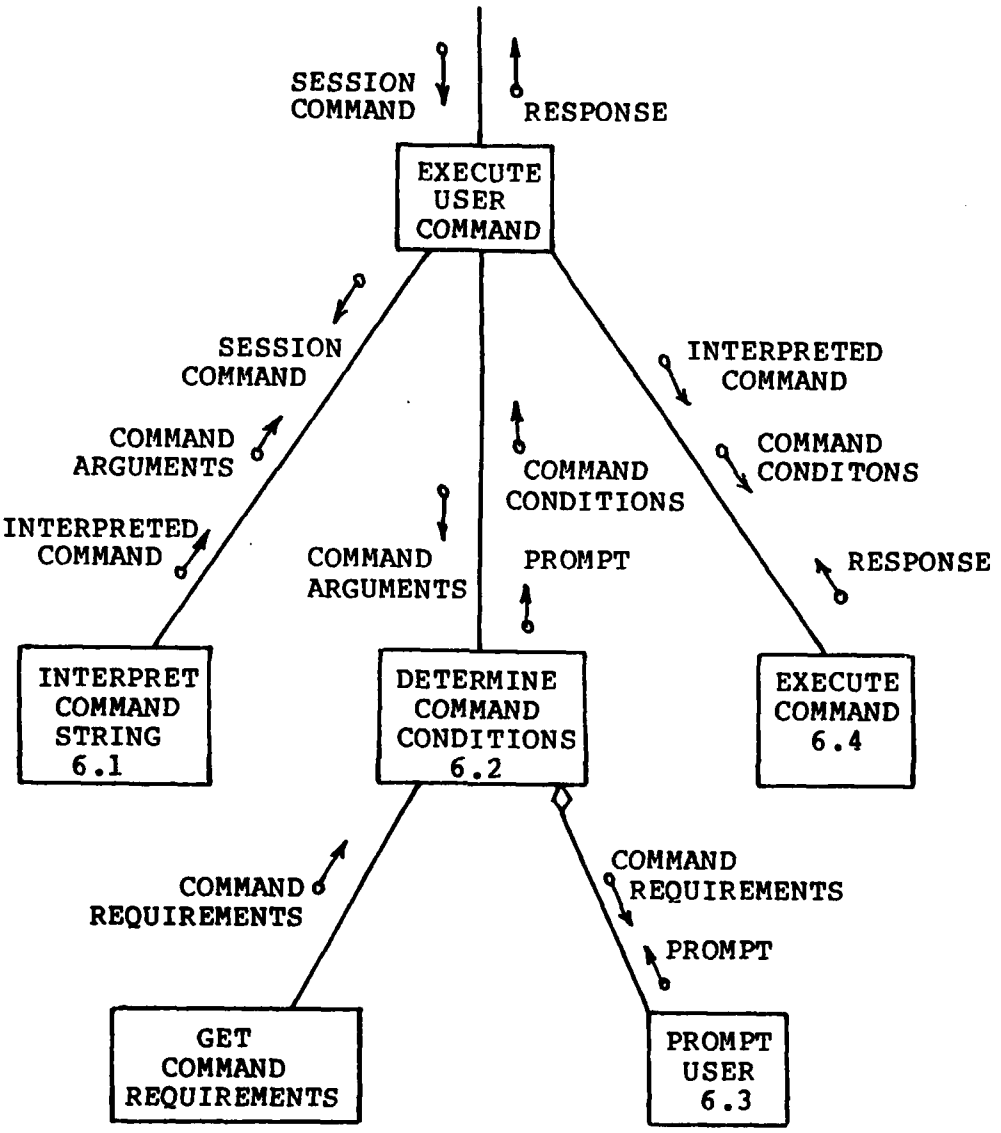
Index

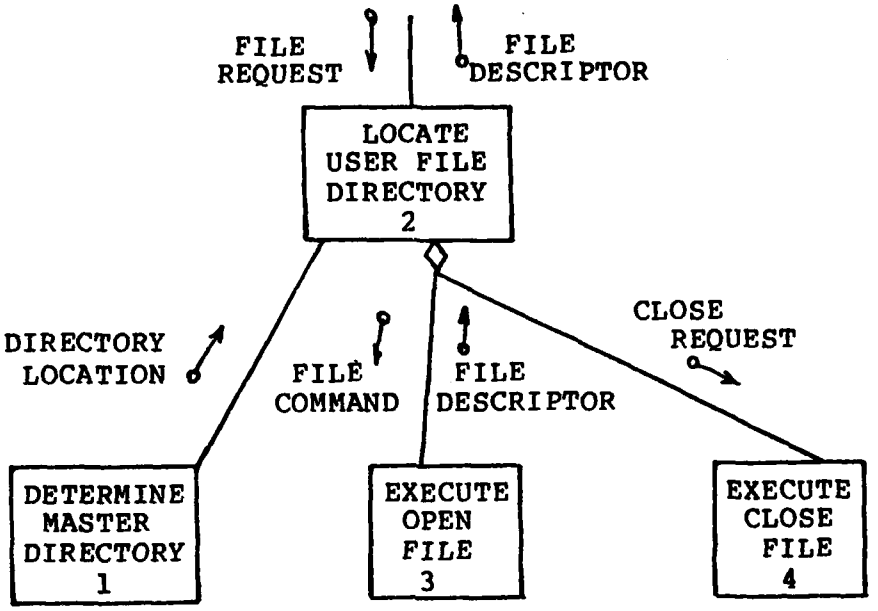

EXECUTE OPERATING SYSTEM
COMMAND
COMMAND
RESPONSE
USER RESPONCE
GET USER COMMAND
DETERMINE COMMAND 1
TRANSMIT RESPONSE 7
SYSTEM COMMAND
RESPONSE
USER COMMAND
RESPONSE
EXECUTE SYSTEM COMMAND 2
DETERMINE COMMAND 3
SESSION COMMAND
SYSTEM RESPONSE
HELP COMMAND
HELP INFO
CONTROL MESSAGE
CONTROL MESSAGE
EXECUTE USER COMMAND 6
EXECUTE HELP COMMAND 5
EXECTUE CONTROL COMMAND 4

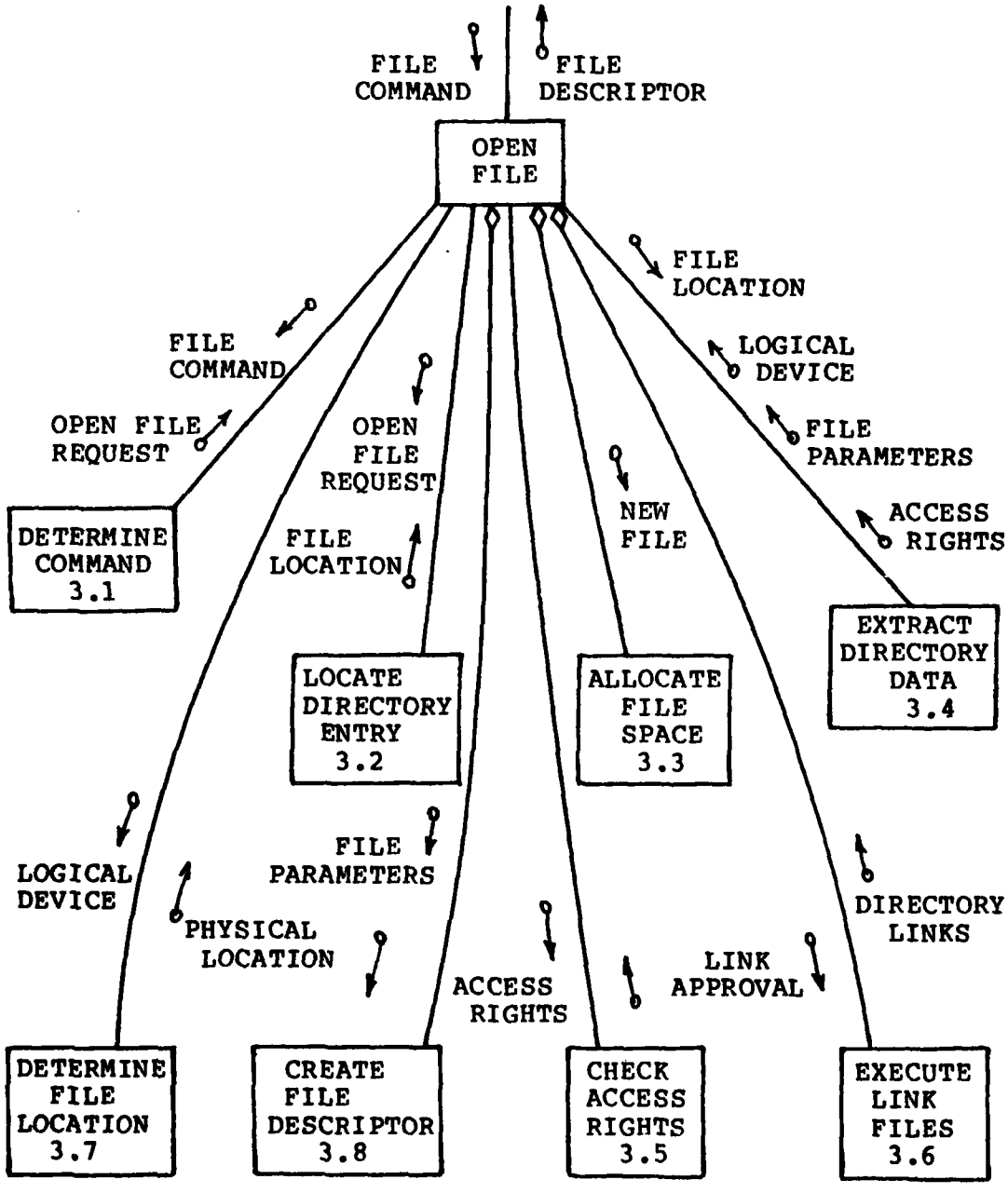

SYSTEM COMMAND
MENU
EXECUTE SYSTEM COMMAND
SESSION COMMAND
AUTHORIZATION MESSAGE
AUTHORIZED COMMAND
AUTHORIZATION MESSAGE
MENU
AUTHORIZED COMMAND
VERIFY AUTHORITY 2.1
PROVIDE MENU 2.2
CONFIGURE SYSTEM 2.3

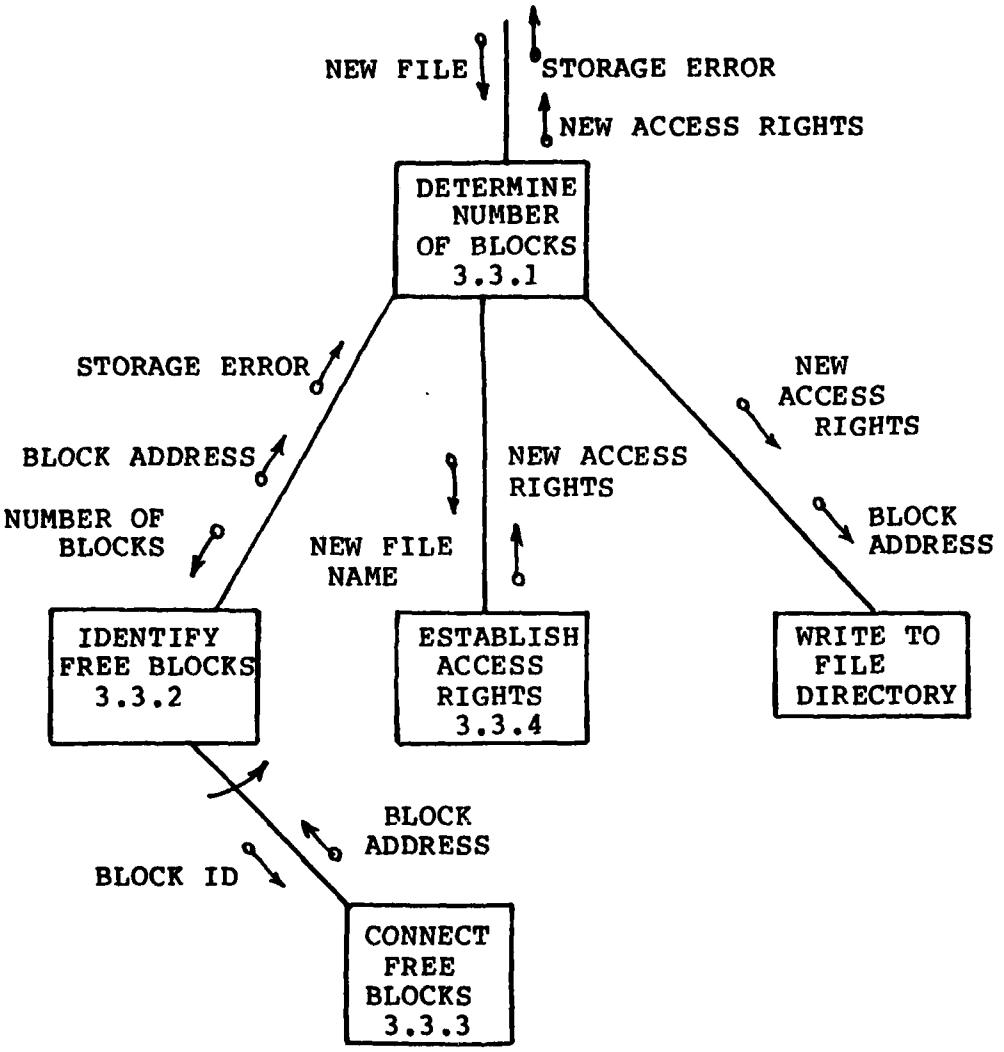

CONTROL COMMAND
RESPONSE
DETERMINE CONTROL COMMAND 4.1
INQUIRY COMMAND
LOG-IN COMMAND
LOG-IN PROMPT
LOG-OUT COMMAND
LOG-OUT MESSAGE
INQUIRY RESPONSE
LOG-IN USER 4.2
LOG-OUT USER 4.3
EXECUTE INQUIRY 4.4

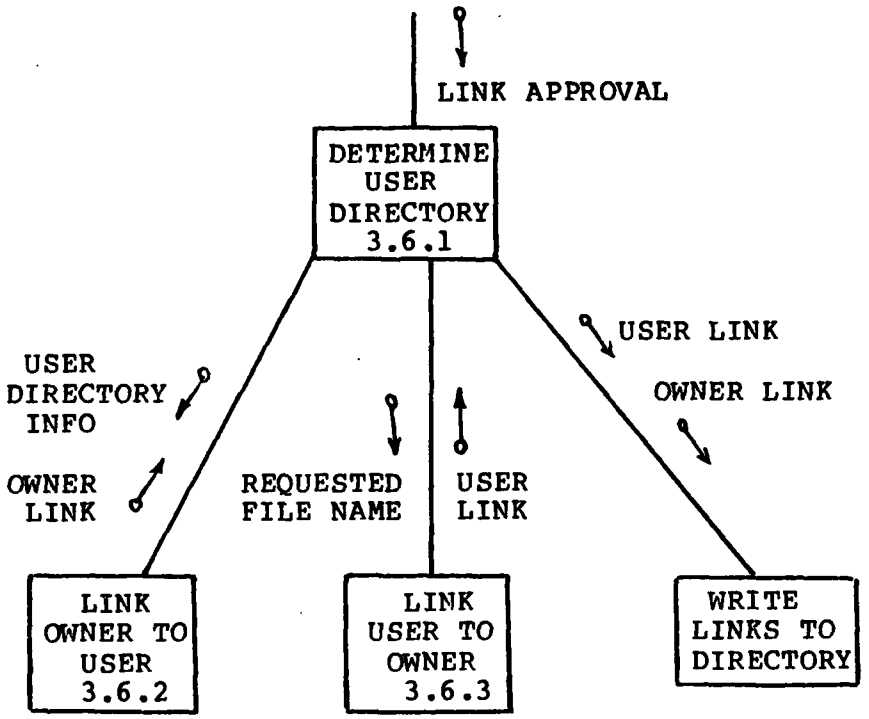

HELP COMMAND
RESPONSE
DETERMINE HELP REQUIRED 5.1
SYSTEM INFO REQUEST
SYSTEM INFO
COMMAND INFO REQUEST
COMMAND INFO
PROVIDE SYSTEM INFO 5.2
PROVIDE COMMAND INFO 5.3

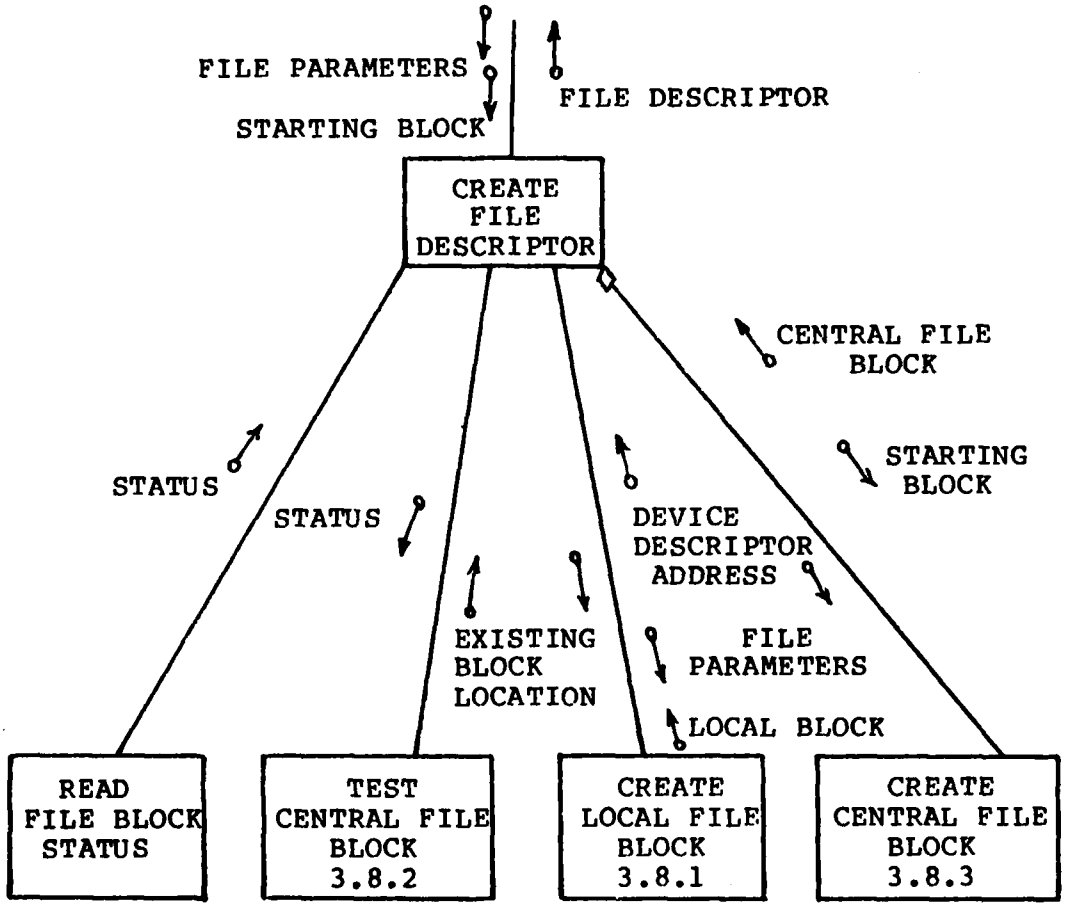

SESSION COMMAND
RESPONSE
EXECUTE USER COMMAND
SESSION COMMAND
COMMAND ARGUMENTS
INTERPRETED COMMAND
COMMAND ARGUMENTS
COMMAND CONDITIONS
PROMPT
INTERPRETED COMMAND
COMMAND CONDITONS
RESPONSE
INTERPRET COMMAND STRING 6.1
DETERMINE COMMAND CONDITIONS 6.2
EXECUTE COMMAND 6.4
COMMAND REQUIREMENTS
COMMAND REQUIREMENTS
PROMPT
GET COMMAND REQUIREMENTS
PROMPT USER 6.3

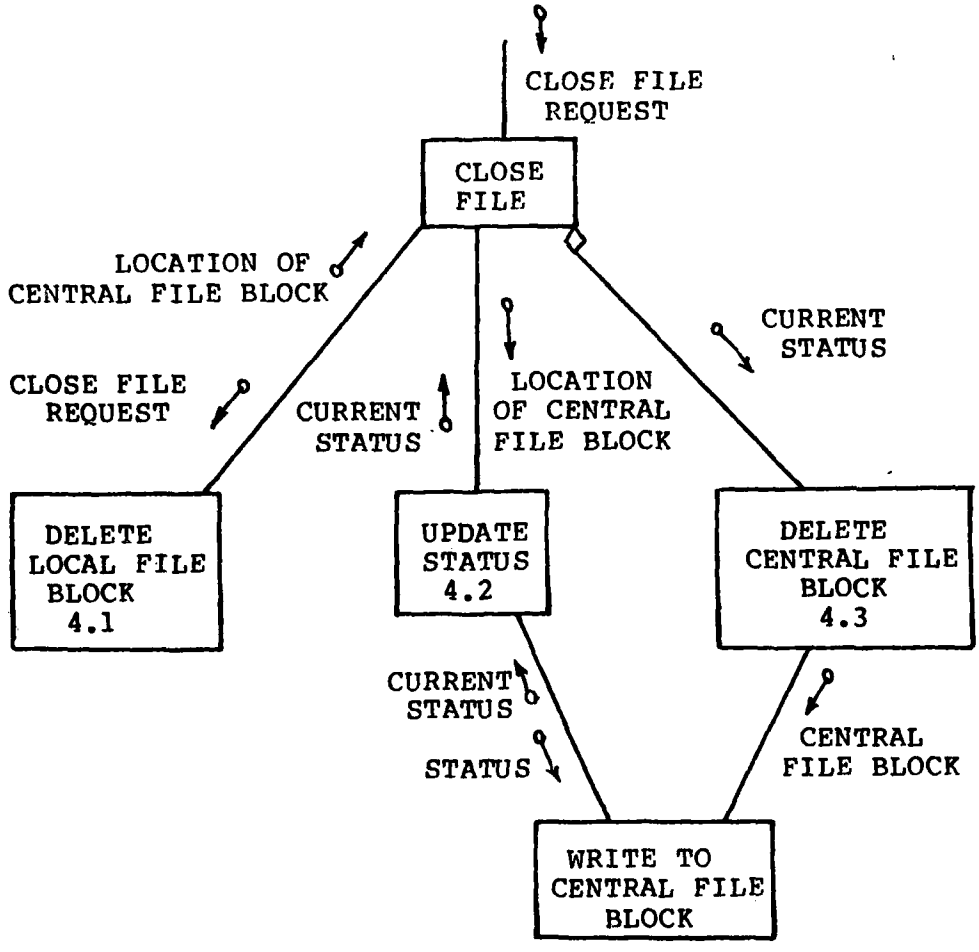

FILE REQUEST
FILE DESCRIPTOR
LOCATE USER FILE DIRECTORY 2
DIRECTORY LOCATION
FILE COMMAND
FILE DESCRIPTOR
CLOSE REQUEST
DETERMINE MASTER DIRECTORY 1
EXECUTE OPEN FILE 3
EXECUTE CLOSE FILE 4

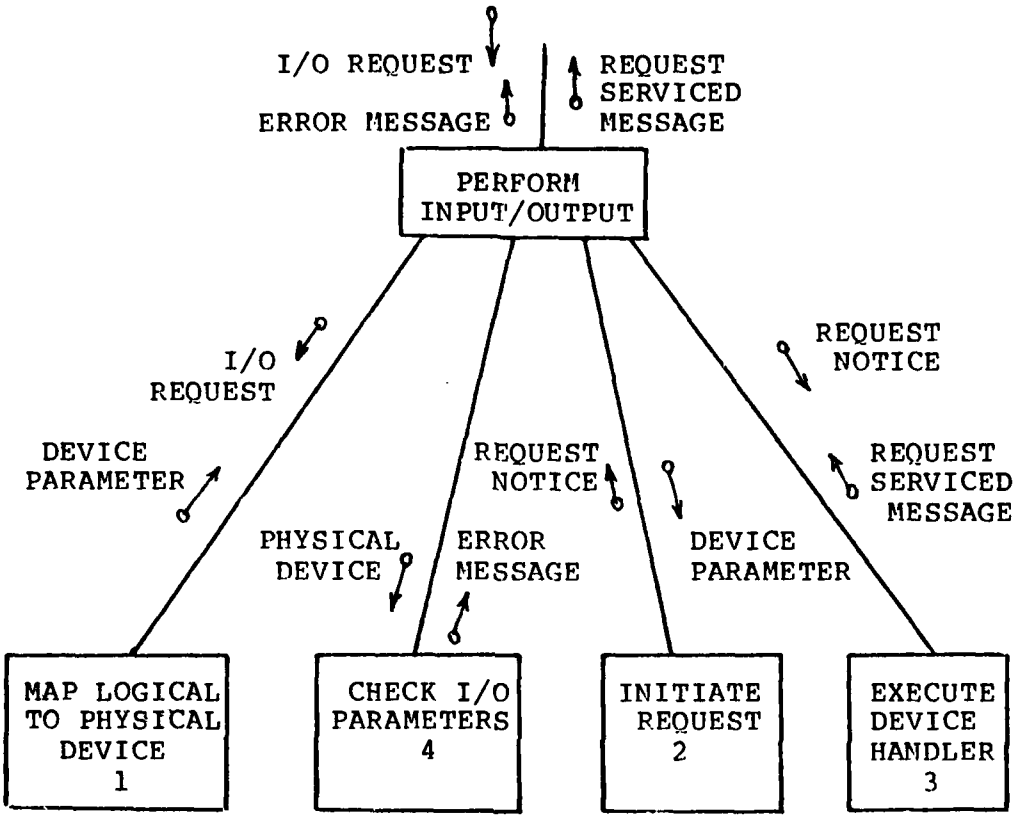

FILE COMMAND
FILE DESCRIPTOR
OPEN FILE
FILE LOCATION
LOGICAL DEVICE
FILE PARAMETERS
ACCESS RIGHTS
FILE COMMAND
OPEN FILE REQUEST
OPEN FILE REQUEST
NEW FILE
FILE LOCATION
DETERMINE COMMAND 3.1
LOCATE DIRECTORY ENTRY 3.2
ALLOCATE FILE SPACE 3.3
EXTRACT DIRECTORY DATA 3.4
LOGICAL DEVICE
FILE PARAMETERS
PHYSICAL LOCATION
ACCESS RIGHTS
LINK APPROVAL
DIRECTORY LINKS
DETERMINE FILE LOCATION 3.7
CREATE FILE DESCRIPTOR 3.8
CHECK ACCESS RIGHTS 3.5
EXECUTE LINK FILES 3.6

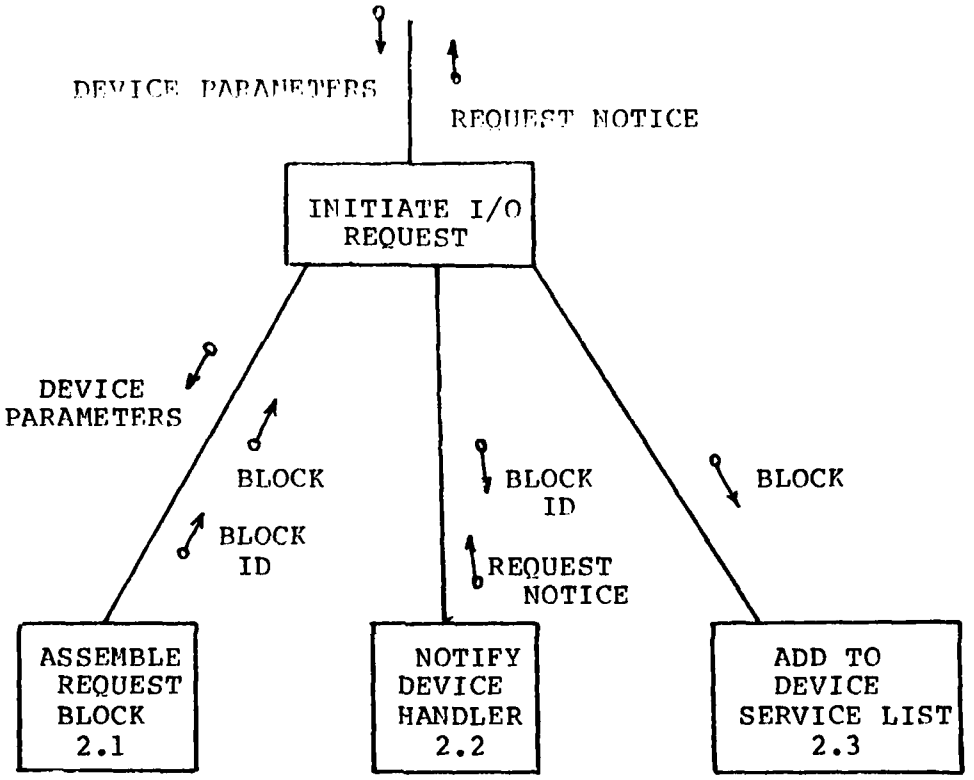

NEW FILE
STORAGE ERROR
NEW ACCESS RIGHTS
DETERMINE NUMBER OF BLOCKS 3.3.1
STORAGE ERROR
BLOCK ADDRESS
NUMBER OF BLOCKS
NEW FILE NAME
NEW ACCESS RIGHTS
NEW ACCESS RIGHTS
BLOCK ADDRESS
IDENTIFY FREE BLOCKS 3.3.2
ESTABLISH ACCESS RIGHTS 3.3.4
WRITE TO FILE DIRECTORY
BLOCK ADDRESS
BLOCK ID
CONNECT FREE BLOCKS 3.3.3

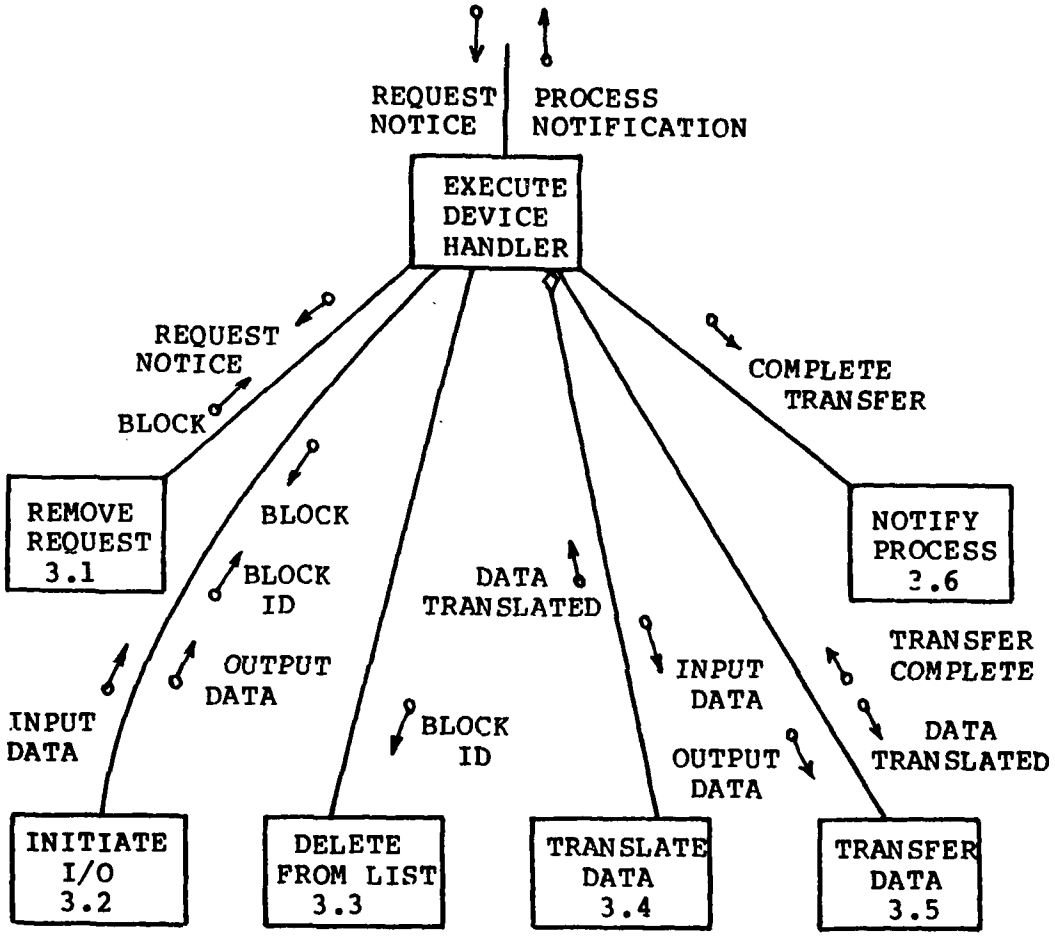

LINK APPROVAL
DETERMINE USER DIRECTORY 3.6.1
USER LINK
OWNER LINK
USER DIRECTORY INFO
OWNER LINK
REQUESTED FILE NAME
USER LINK
LINK OWNER TO USER 3.6.2
LINK USER TO OWNER 3.6.3
WRITE LINKS TO DIRECTORY

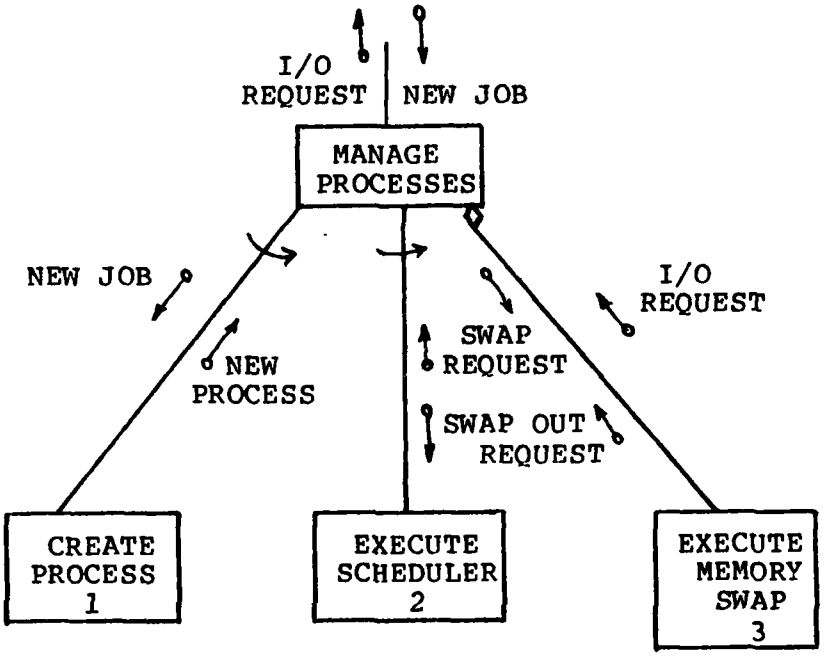

FILE PARAMETERS
STARTING BLOCK
FILE DESCRIPTOR
CREATE FILE DESCRIPTOR
CENTRAL FILE BLOCK
STARTING BLOCK
STATUS
STATUS
DEVICE DESCRIPTOR ADDRESS
EXISTING BLOCK LOCATION
FILE PARAMETERS
LOCAL BLOCK
READ FILE BLOCK STATUS
TEST CENTRAL FILE BLOCK 3.8.2
CREATE LOCAL FILE BLOCK 3.8.1
CREATE CENTRAL FILE BLOCK 3.8.3

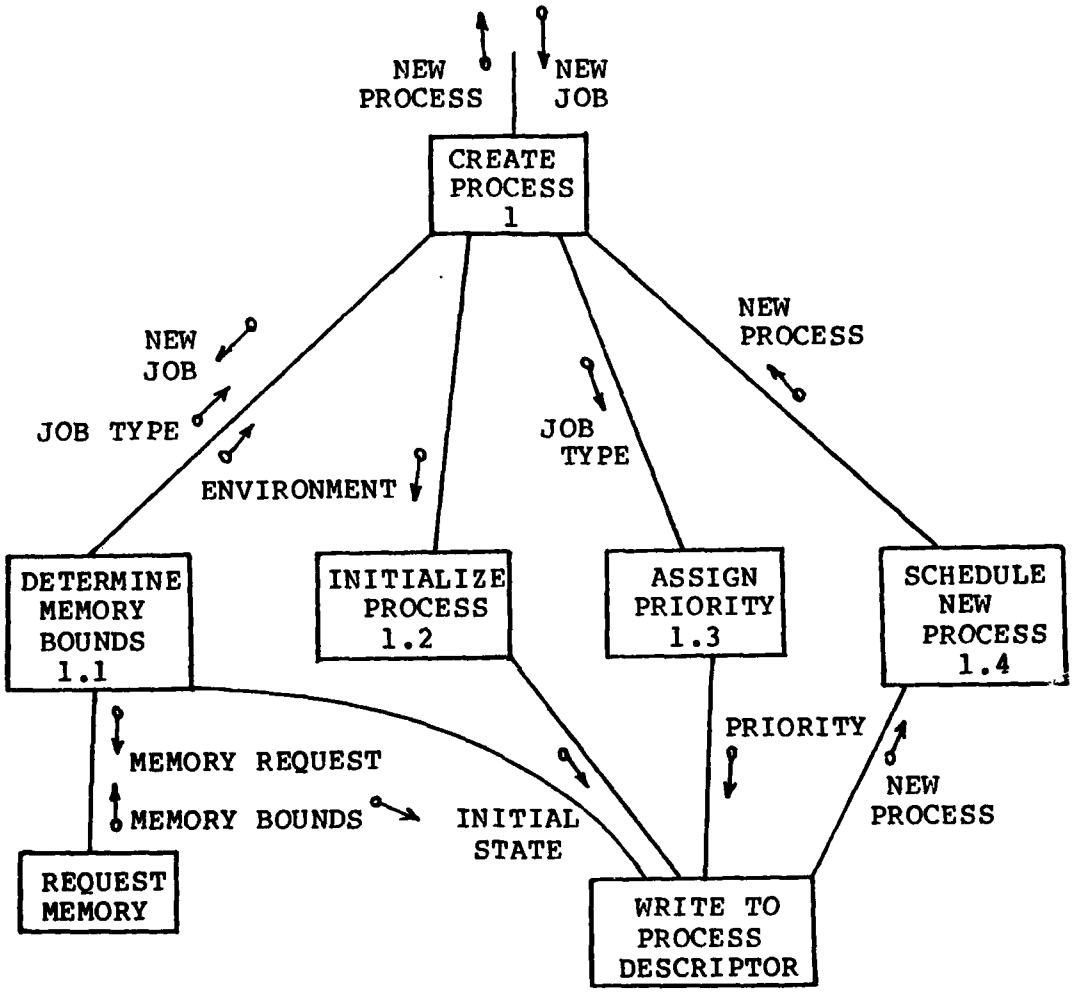

CLOSE FILE REQUEST
CLOSE FILE
LOCATION OF CENTRAL FILE BLOCK
CLOSE FILE REQUEST
CURRENT STATUS
LOCATION OF CENTRAL FILE BLOCK
CURRENT STATUS
DELETE LOCAL FILE BLOCK 4.1
UPDATE STATUS 4.2
DELETE CENTRAL FILE BLOCK 4.3
CURRENT STATUS
STATUS
CENTRAL FILE BLOCK
WRITE TO CENTRAL FILE BLOCK

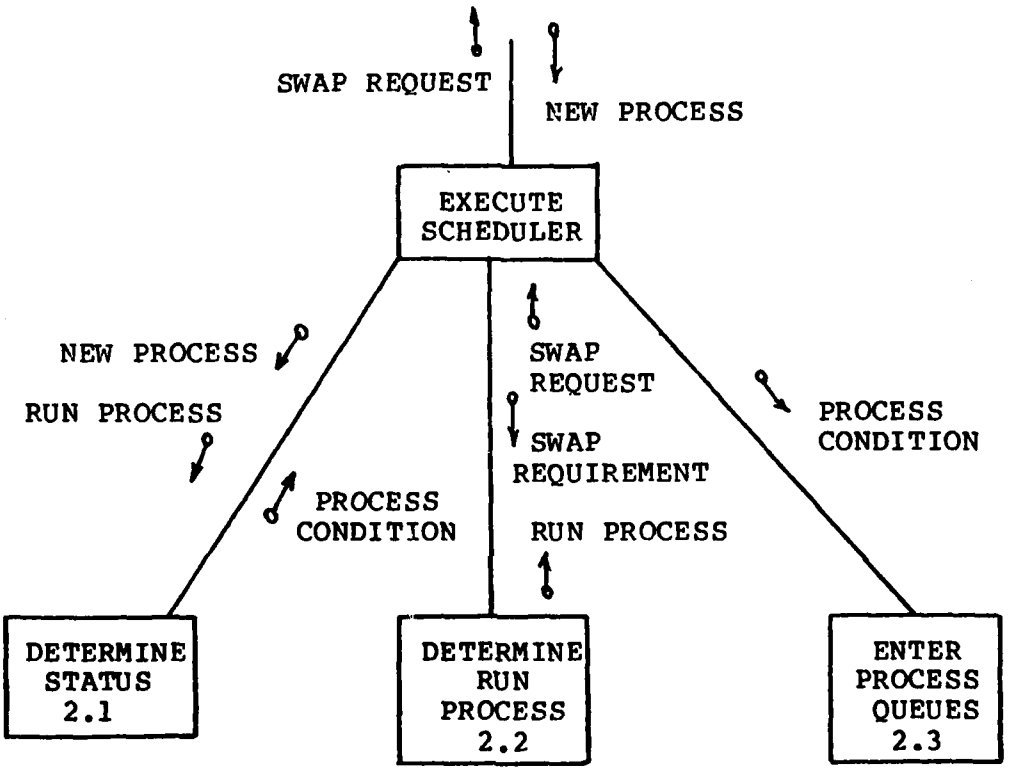

I/O REQUEST
ERROR MESSAGE
REQUEST SERVICED MESSAGE
PERFORM INPUT/OUTPUT
I/O REQUEST
DEVICE PARAMETER
REQUEST NOTICE
REQUEST NOTICE
REQUEST SERVICED MESSAGE
PHYSICAL DEVICE
ERROR MESSAGE
DEVICE PARAMETER
MAP LOGICAL TO PHYSICAL DEVICE 1
CHECK I/O PARAMETERS 4
INITIATE REQUEST 2
EXECUTE DEVICE HANDLER 3

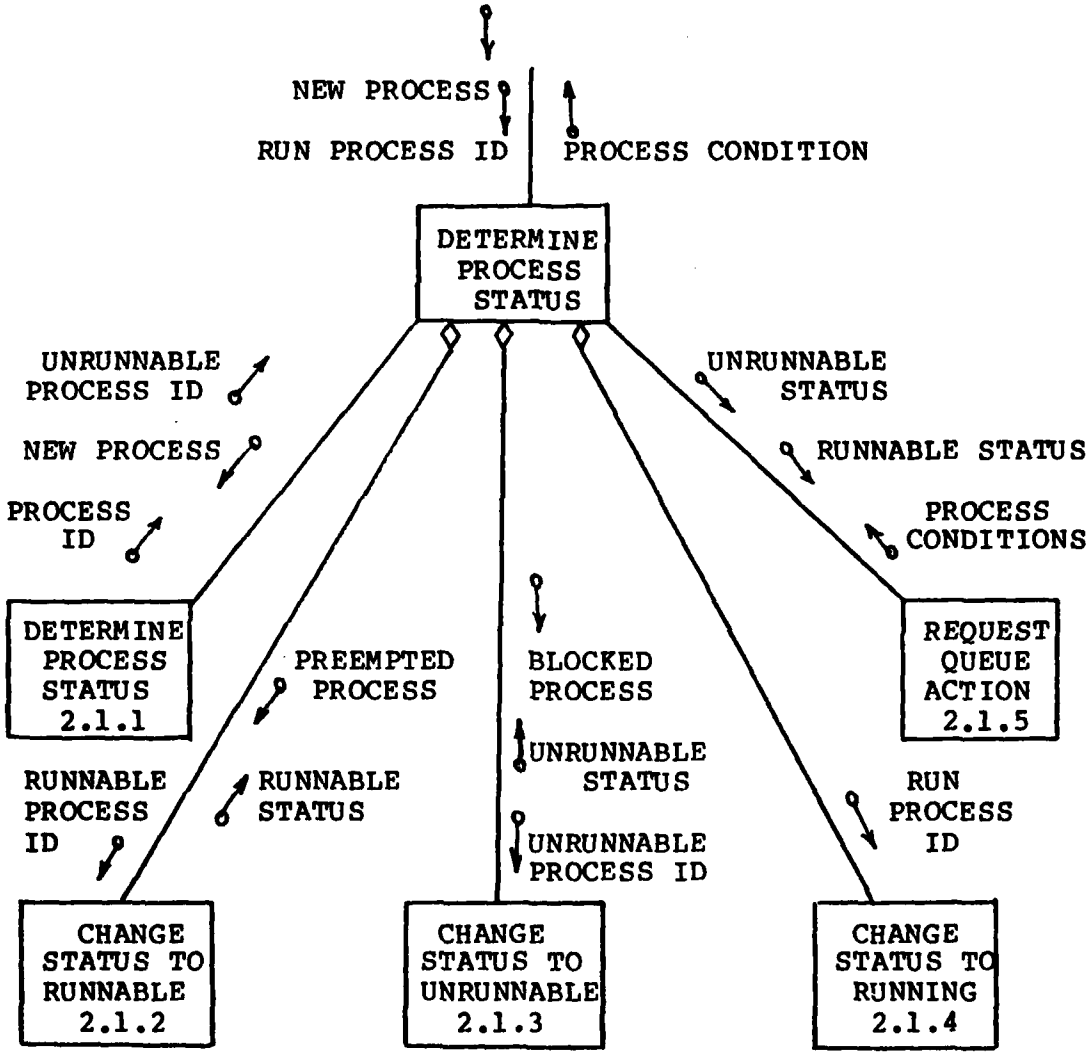

DEVICE PARAMETERS
REQUEST NOTICE
INITIATE I/O REQUEST
DEVICE PARAMETERS
BLOCK
BLOCK ID
BLOCK ID
REQUEST NOTICE
BLOCK
ASSEMBLE REQUEST BLOCK 2.1
NOTIFY DEVICE HANDLER 2.2
ADD TO DEVICE SERVICE LIST 2.3

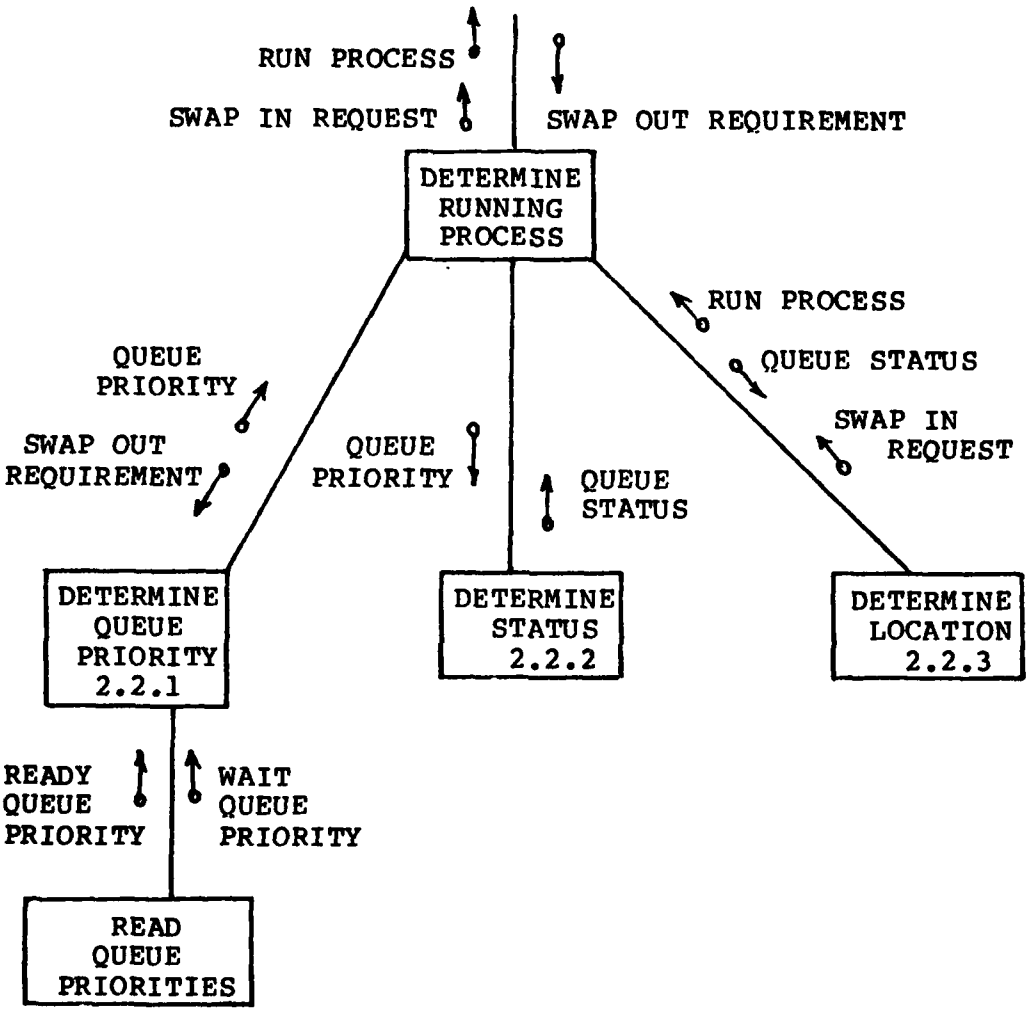

REQUEST NOTICE
PROCESS NOTIFICATION
EXECUTE DEVICE HANDLER
REQUEST NOTICE
BLOCK
COMPLETE TRANSFER
REMOVE REQUEST 3.1
BLOCK
BLOCK ID
DATA TRANSLATED
NOTIFY PROCESS 3.6
TRANSFER COMPLETE
OUTPUT DATA
INPUT DATA
INPUT DATA
BLOCK ID
OUTPUT DATA
DATA TRANSLATED
INITIATE I/O 3.2
DELETE FROM LIST 3.3
TRANSLATE DATA 3.4
TRANSFER DATA 3.5

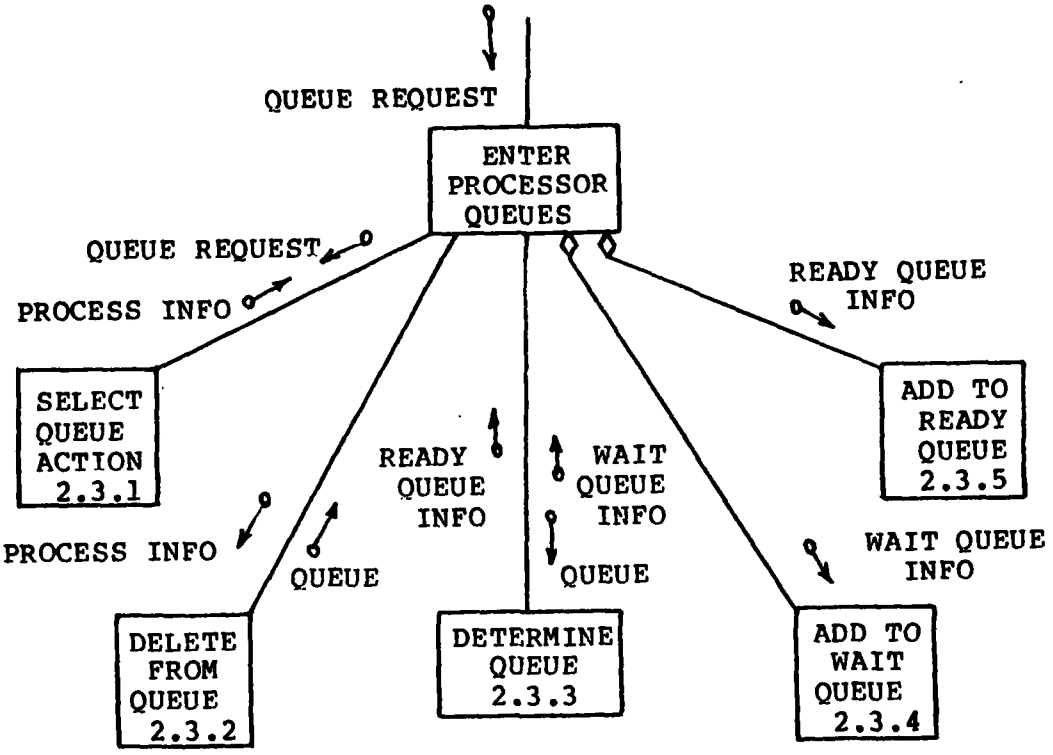

I/O
REQUEST
NEW JOB
MANAGE
PROCESSES
NEW JOB
NEW
PROCESS
SWAP
REQUEST
SWAP OUT
REQUEST
I/O
REQUEST
CREATE
PROCESS
1
EXECUTE
SCHEDULER
2
EXECUTE
MEMORY
SWAP
3

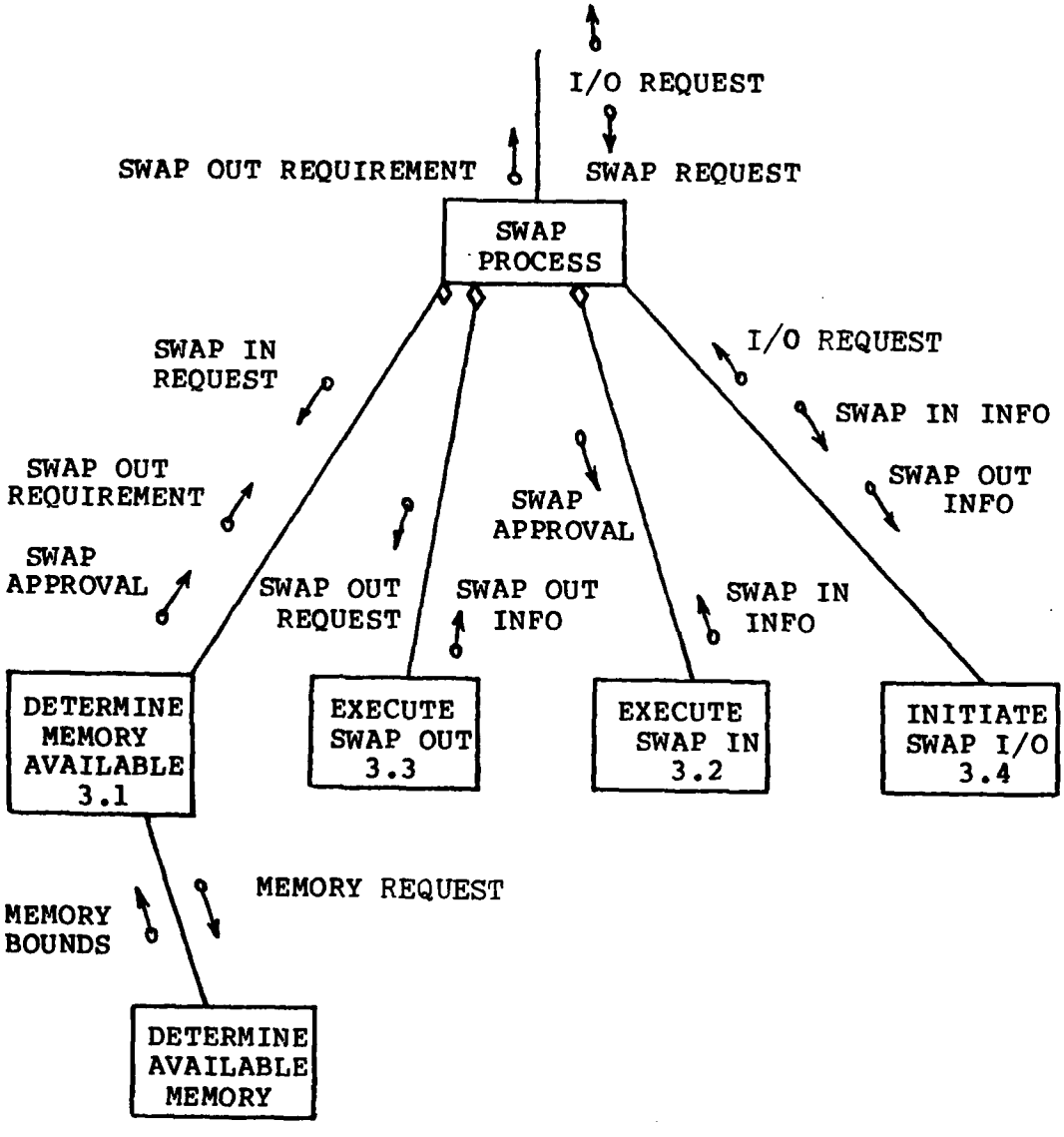

NEW PROCESS
NEW JOB
CREATE PROCESS 1
NEW JOB
JOB TYPE
ENVIRONMENT
JOB TYPE
NEW PROCESS
DETERMINE MEMORY BOUNDS 1.1
INITIALIZE PROCESS 1.2
ASSIGN PRIORITY 1.3
SCHEDULE NEW PROCESS 1.4
MEMORY REQUEST
MEMORY BOUNDS
INITIAL STATE
PRIORITY
NEW PROCESS
REQUEST MEMORY
WRITE TO PROCESS DESCRIPTOR

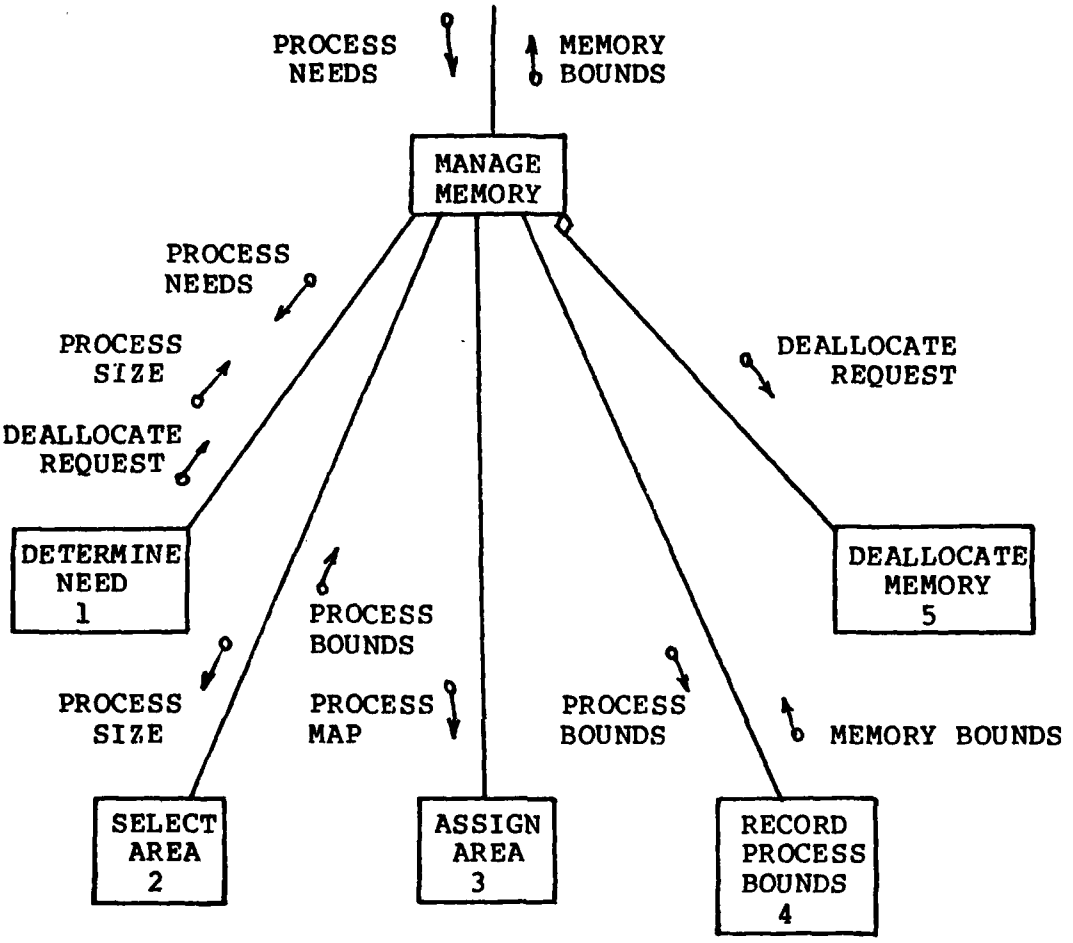

SWAP REQUEST
NEW PROCESS
EXECUTE SCHEDULER
NEW PROCESS
RUN PROCESS
PROCESS CONDITION
SWAP REQUEST
SWAP REQUIREMENT
RUN PROCESS
PROCESS CONDITION
DETERMINE STATUS 2.1
DETERMINE RUN PROCESS 2.2
ENTER PROCESS QUEUES 2.3

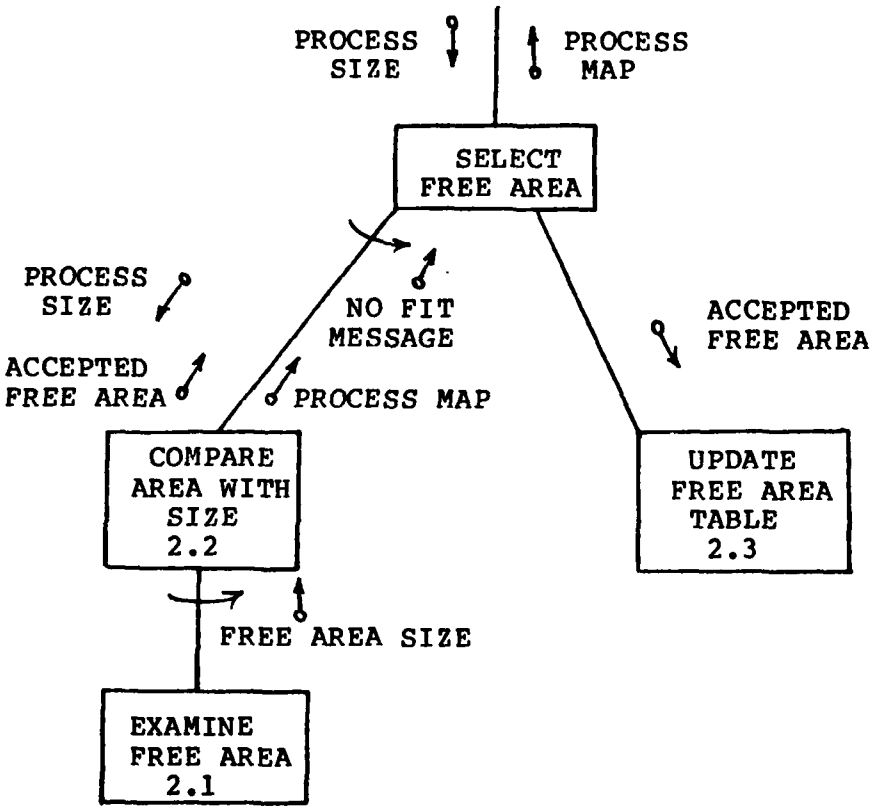

NEW PROCESS
RUN PROCESS ID
PROCESS CONDITION
DETERMINE PROCESS STATUS
UNRUNNABLE PROCESS ID
NEW PROCESS
PROCESS ID
UNRUNNABLE STATUS
RUNNABLE STATUS
PROCESS CONDITIONS
DETERMINE PROCESS STATUS 2.1.1
PREEMPTED PROCESS
BLOCKED PROCESS
REQUEST QUEUE ACTION 2.1.5
RUNNABLE PROCESS ID
RUNNABLE STATUS
UNRUNNABLE STATUS
UNRUNNABLE PROCESS ID
RUN PROCESS ID
CHANGE STATUS TO RUNNABLE 2.1.2
CHANGE STATUS TO UNRUNNABLE 2.1.3
CHANGE STATUS TO RUNNING 2.1.4

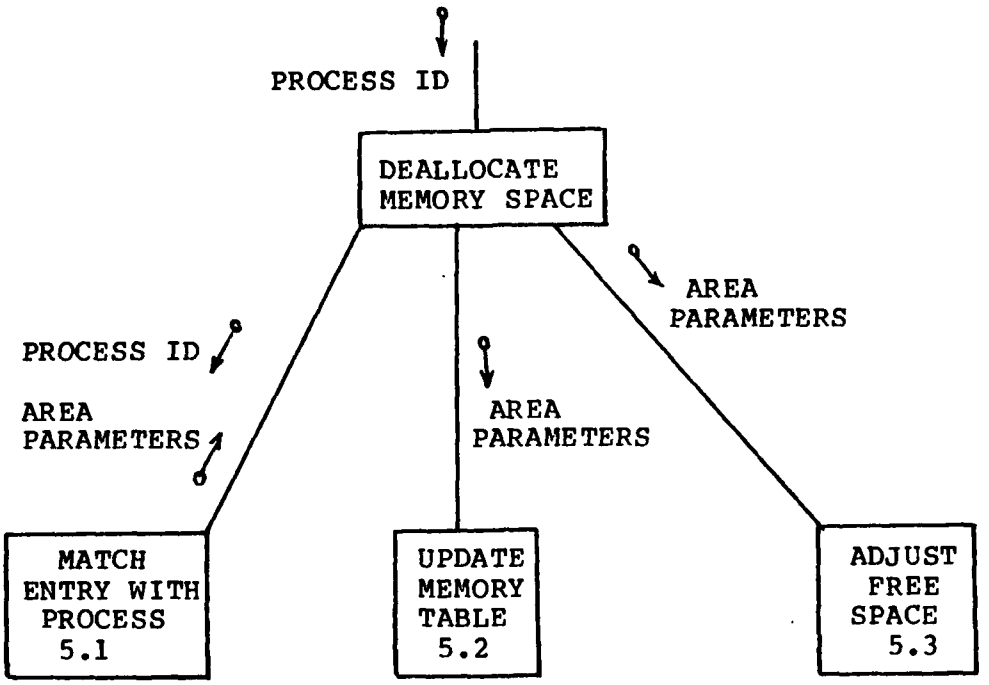

RUN PROCESS
SWAP IN REQUEST
SWAP OUT REQUIREMENT
DETERMINE RUNNING PROCESS
RUN PROCESS
QUEUE STATUS
SWAP IN REQUEST
QUEUE PRIORITY
SWAP OUT REQUIREMENT
QUEUE PRIORITY
QUEUE STATUS
DETERMINE QUEUE PRIORITY 2.2.1
DETERMINE STATUS 2.2.2
DETERMINE LOCATION 2.2.3
READY QUEUE PRIORITY
WAIT QUEUE PRIORITY
READ QUEUE PRIORITIES

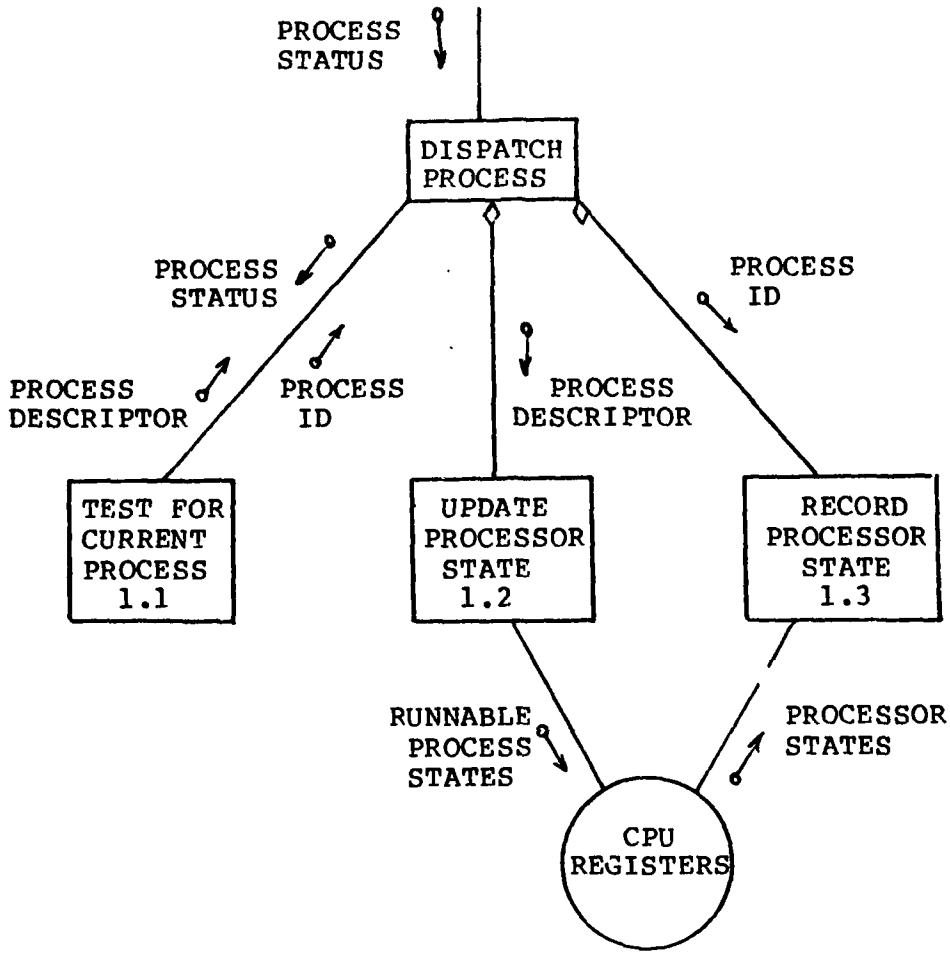

QUEUE REQUEST
ENTER PROCESSOR QUEUES
QUEUE REQUEST
PROCESS INFO
READY QUEUE INFO
SELECT QUEUE ACTION 2.3.1
ADD TO READY QUEUE 2.3.5
READY QUEUE INFO
WAIT QUEUE INFO
PROCESS INFO
QUEUE
QUEUE
WAIT QUEUE INFO
DELETE FROM QUEUE 2.3.2
DETERMINE QUEUE 2.3.3
ADD TO WAIT QUEUE 2.3.4


I/O REQUEST
SWAP OUT REQUIREMENT
SWAP REQUEST
SWAP PROCESS
SWAP IN REQUEST
I/O REQUEST
SWAP IN INFO
SWAP OUT REQUIREMENT
SWAP OUT INFO
SWAP APPROVAL
SWAP APPROVAL
SWAP OUT REQUEST
SWAP OUT INFO
SWAP IN INFO
DETERMINE MEMORY AVAILABLE 3.1
EXECUTE SWAP OUT 3.3
EXECUTE SWAP IN 3.2
INITIATE SWAP I/O 3.4
MEMORY REQUEST
MEMORY BOUNDS
DETERMINE AVAILABLE MEMORY


PROCESS NEEDS
MEMORY BOUNDS
MANAGE MEMORY
PROCESS NEEDS
PROCESS SIZE
DEALLOCATE REQUEST
DEALLOCATE REQUEST
DETERMINE NEED 1
DEALLOCATE MEMORY 5
PROCESS BOUNDS
PROCESS SIZE
PROCESS MAP
PROCESS BOUNDS
MEMORY BOUNDS
SELECT AREA 2
ASSIGN AREA 3
RECORD PROCESS BOUNDS 4


PROCESS SIZE
PROCESS MAP
SELECT FREE AREA
PROCESS SIZE
NO FIT MESSAGE
ACCEPTED FREE AREA
ACCEPTED FREE AREA
PROCESS MAP
COMPARE AREA WITH SIZE 2.2
UPDATE FREE AREA TABLE 2.3
FREE AREA SIZE
EXAMINE FREE AREA 2.1


PROCESS ID
DEALLOCATE MEMORY SPACE
AREA PARAMETERS
PROCESS ID
AREA PARAMETERS
AREA PARAMETERS
MATCH ENTRY WITH PROCESS 5.1
UPDATE MEMORY TABLE 5.2
ADJUST FREE SPACE 5.3

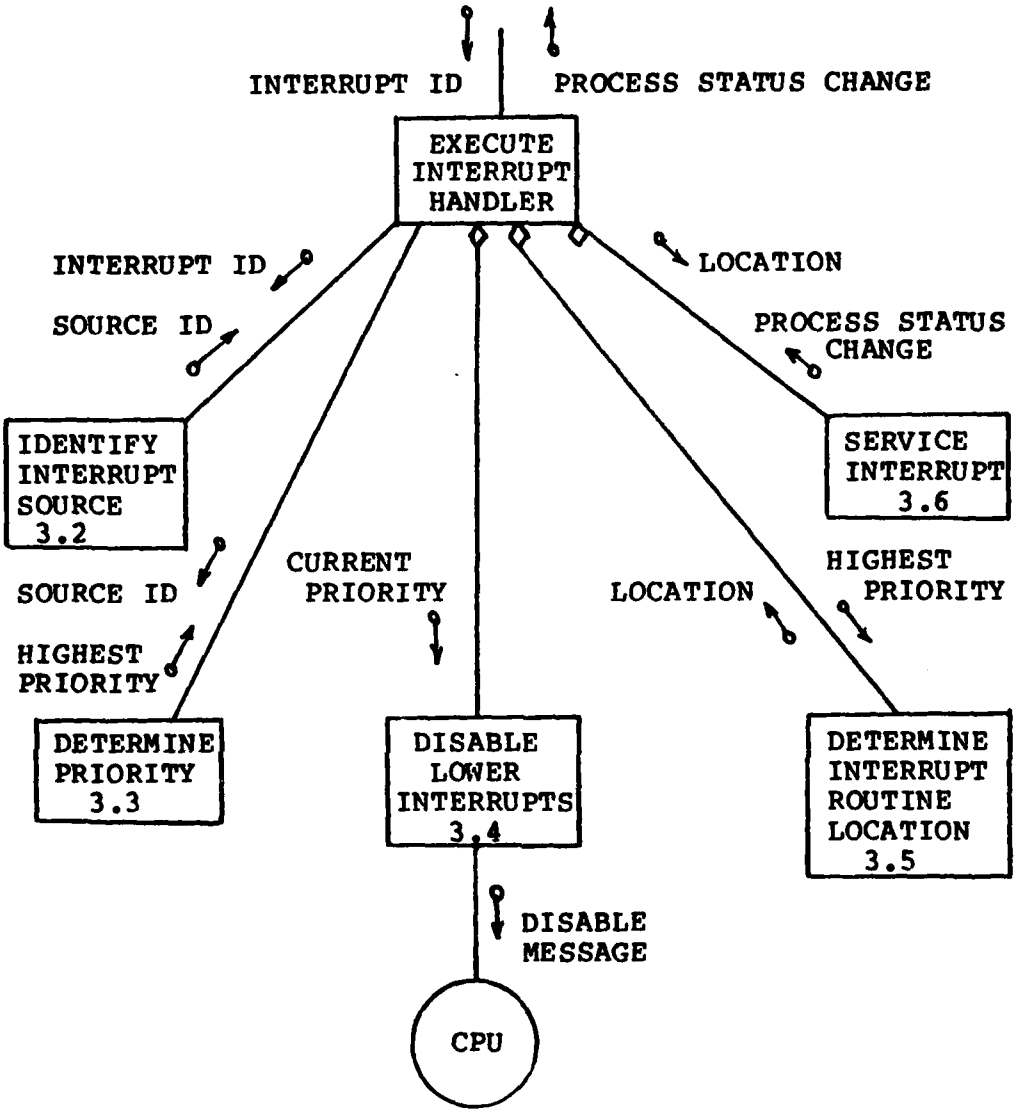

PROCESS STATUS
DISPATCH PROCESS
PROCESS STATUS
PROCESS ID
PROCESS DESCRIPTOR
PROCESS ID
PROCESS DESCRIPTOR
TEST FOR CURRENT PROCESS 1.1
UPDATE PROCESSOR STATE 1.2
RECORD PROCESSOR STATE 1.3
RUNNABLE PROCESS STATES
PROCESSOR STATES
CPU REGISTERS

AD-A115 614 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
DESIGN AND DEVELOPMENT OF A MULTIPROGRAMMING OPERATING SYSTEM F--ETC(U)
DEC 81 M S ROSS

UNCLASSIFIED AFIT/GCS/EE/81D-14 NL

4 - 4
AD
A115 614

END
DATE FILMED
8-8
DTIC


INTERRUPT ID
PROCESS STATUS CHANGE
EXECUTE INTERRUPT HANDLER
INTERRUPT ID
SOURCE ID
LOCATION
PROCESS STATUS CHANGE
IDENTIFY INTERRUPT SOURCE 3.2
SERVICE INTERRUPT 3.6
SOURCE ID
HIGHEST PRIORITY
CURRENT PRIORITY
LOCATION
HIGHEST PRIORITY
DETERMINE PRIORITY 3.3
DISABLE LOWER INTERRUPTS 3.4
DETERMINE INTERRUPT ROUTINE LOCATION 3.5
DISABLE MESSAGE
CPU

Vita

Captain Mitchell S. Ross was born on Febuary 28, 1952 in Stillwater, Oklahoma. In 1970, he graduated concurrently from Central High School and The Willard-Graff Vocational Technical School (Electronic Technology) in Springfield, Missouri. He earned a Bachelor of Science degree at Southwest Missouri University with a major in Industrial Technology and minor in Mathematics. Following graduation he attended the Signal Officer Course at Fort Gordon, Georgia and was assigned to the 11th Signal Brigade, 40th Signal Battalion, Fort Huachuca, Arizona. From December 1976 to December 1977 he was the Aide-de-camp to the Deputy Commanding General, Army Communications Command. He was subsequently assigned to the 86th Signal Battalion as a Microwave and Tropospheric Radio Officer. In December 1978, he became the Commanding Officer of the 19th Signal Company, Electronic Maintenance and Supply Facility (EMSF) for the 11th Signal Brigade. He entered the Air Force Institute in June 1980.

Permanent Address: 2115 Mt. Vernon
Springfield, Missouri

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: AFIT/GCS/EE/81D-14 — 2. GOVT ACCESSION NO.: AD-A115614	3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): DESIGN AND DEVELOPMENT OF A MULTIPROGRAMMING OPERATING SYSTEM FOR SIXTEEN BIT MICROPROCESSORS	5. TYPE OF REPORT & PERIOD COVERED: MS THESIS 6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): Mitchell S. Ross, Captain, USA	8. CONTRACT OR GRANT NUMBER(s):
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Air Force Institute of Technology (AFIT) Wright-Patterson AFB, OH 45433	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Institute of Technology (AFIT) Wright-Patterson AFB, OH 45433	12. REPORT DATE: December 1981 13. NUMBER OF PAGES: 280
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	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

Approved for public release; IAW AFR 190-17

FREDERIC C. LYNCH, Maj, USAF
Director of Public Affairs

Lynn Wolaver
LYNN E. WOLAVER
Dean for Research and Professional Development

Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB, OH 45433

4 JUN 19

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

Operating Systems
Timesharing
Man-Machine Interface
Multiprogramming
Interactive Computing

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

see reverse

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract

A timesharing operating system for the Air Force Institute of Technology Digital Engineering Laboratory was designed and developed with emphasis on the human interface. The functional requirements were developed by a thorough literature search on the user perceptions of computer operating systems and the justification for the success of popular systems such as UNIX, TENEX, and UCSD Pascal. Structured Analysis was used to produce a structured specification for the hierarchy of the operating system. The structured specification includes an operating system shell which allows a flexible user command structure, a hierarchical file structure, device independent input/output management, a scheduler which supports swapping, a general memory management scheme, and a system nucleus consisting of process dispatching, interrupt handling and interprocess communications. Weinberg's methodology, which is based on Yourdon and Constantine's Transform Analysis and Transaction Analysis Techniques, was used to develop the software design which consists of a set of module structure charts. The module structure charts are supported by data flow diagrams and a data dictionary.

Because of the depth needed to complete such a project, this first effort is intended to provide a basis for further expansion and development. Hence, the design is a broad overall approach aimed at 16-bit microprocessors and not detailed sufficiently for full implementation.