Introduction to RSX-11M

Order No. AA-2555D-TC

RSX-11M Version 3.2

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

First Printing, May 1974
Revised: November 1976
December 1977
June 1979

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
VAX	VMS	SBI
DECnet	IAS	PDT
DATATRIEVE	TRAX	

CONTENTS

CONTENTS

FIGURES

PREFACE

MANUAL OBJECTIVES

This manual introduces the basic concepts and capabilities of the RSX-11M and RSX-11S operating systems.

INTENDED AUDIENCE

To understand the subjects discussed in this manual, readers should have a general knowledge of computing terms and principles. Previous knowledge of the RSX-11M or RSX-11S operating system, however, is not required.

STRUCTURE OF THIS DOCUMENT

The manual contains the following chapters:

Chapter 1 -- RSX-11M and RSX-11S

Chapter 2 -- RSX-11M Applications

Chapter 3 -- Real-Time and Multiprogramming Operations

Chapter 4 -- System Operation

Chapter 5 -- Program Development

Chapter 6 -- File System and I/O Operations

ASSOCIATED DOCUMENTS

The RSX-11M/RSX-11S Documentation Directory defines the intended readership of each manual in the RSX-11M/RSX-11S set and provides a brief synopsis of each manual's contents.

For detailed information about PDP-11 hardware, refer to the appropriate PDP-11 processor or peripheral handbook.

CHAPTER 1

RSX-11M AND RSX-11S

1.1 RSX-11M

RSX-11M is a disk-based operating system for PDP-11 computers that can be used both as a multiprogramming system and as a real-time system. The primary function of RSX-11M is quick response to real-time events. However, the multiprogramming capability of the system allows you to combine this real-time activity, with less time-dependent activities such as program development or text editing.

An RSX-11M system can be anything from a stand-alone system controlling one process (where quick response is the most important factor) to a multiuser system, supporting program development activity at several terminals.

Initially, you will receive a minimal RSX-11M host system on whatever distribution medium you select. You decide what you want your RSX-11M system to do (choosing from a wide range of hardware and software options.) You can then create an RSX-11M target system to suit your exact requirements, using a process called System Generation. The combination of hardware and software options you choose is called your target system configuration.

You can change the configuration at any time by performing another System Generation.

The user interface to RSX-11M is called the Monitor Console Routine (MCR). It serves as a channel for communication between the user and the operating system.

1.2 RSX-11S

RSX-11S is a memory-only version of the RSX-11M operating system. Its main purpose is to run tasks on a processor that has limited peripheral devices (tapes and disks). Since RSX-11S does not have a file system, checkpointing, nonresident tasks, editors, compilers, or assemblers, tasks must be written, assembled or compiled, and task built on an RSX-11M or RSX-11M-PLUS host system before they can run on an RSX-11S system.

RSX-11S can operate with a minimum of 8K words of memory and a maximum of 124K words. (This includes space for a 4K Executive; the remainder is available for user-written tasks.)

RSX-11S is fully compatible with RSX-11M, both internally and at the user level. It supports the same I/O driver interface and any drivers written to run on one system can be incorporated without change to run on the other.

Any nonprivileged user task that runs on RSX-11S can run on RSX-11M if you Task Build the file containing the object modules so that it fits in the available RSX-11S memory. Tasks that run on RSX-11M can also run on RSX-11S if they are correctly task built, as long as the tasks do not include memory management directives (also called Programmed Logical Address Space or PLAS directives). RSX-11S does not support dynamic allocation of system-controlled partitions or memory management.

The user interface to RSX-11S is Basic MCR (a subset of the RSX-11M interface, MCR). It is compatible with MCR.

RSX-11S can also include other system tasks:

1. The Online Task Loader (OTL), which installs, loads, and fixes tasks in the RSX-11S system.

2. The System Image Preservation program (SIP), which saves an image of the operating system for subsequent use.

3. The System Activity Display programs which provide video terminal displays of information about RSX-11S system activity.

4. A subset of File Control Services (FCS) for record devices. This feature does not include directory support.

CHAPTER 2

RSX-11M APPLICATIONS

This chapter discusses several real-time jobs that RSX-11M can handle. It also introduces RSX-11M program development facilities and network communications (DECnet).

Real-time events demand immediate and full access to available computer resources at regular and irregular intervals. When RSX-11M is not responding to these events, it permits the computer resources to be used to perform less time-dependent jobs, such as program development or payroll processing. RSX-11M can immediately preempt these programs whenever it needs to respond to a real-time event. Therefore, nonreal-time applications can make up the major work load for an RSX-11M system, without interfering with the system's real-time capabilities.

2.1 REAL-TIME APPLICATIONS

The following sections explain RSX-11M operations, for data acquisition, process monitoring and control, manufacturing monitoring and control, and laboratory and medical data processing.

2.1.1 Data Acquisition

Data acquisition refers to the process of collecting physically generated data in a form that can later be evaluated. A typical data acquisition application might require a number of programs to respond to a single burst of data. RSX-11M multiprogramming allows this apparently concurrent execution of several programs. For example, three separate RSX-11M tasks could process data generated by one real-time event and recorded on three different devices: one task could record data from a spectrometer; another could record data from a flowmeter; and a third, data from a thermocouple.

RSX-11M provides a rapid response to external stimuli because the programs that control peripheral devices are linked directly to the PDP-11 hardware interrupt vectors. This permits the system to change rapidly from one device to another and from one task to another.

2.1.2 Process Monitoring and Control

A process control application usually involves first acquiring and analyzing data, and then returning information based on the analysis. The information obtained is used to control some actual process. Oil refineries and steel rolling mills could perform such process control activities.

The signals that the PDP-11 hardware receives from devices that monitor and control a process are called process inputs. RSX-11M uses these inputs to monitor many facets of the process, such as temperature, flow rate, or the amount of raw material used. Instructions stored in RSX-11M user programs can determine how to keep the process running properly. In addition, control optimization programs can make adjustments based on the relationship of various parts of the process. After analyzing all of the data, RSX-11M, directed by user programs, generates signals called process outputs to control the valves, switches, and relays that, in turn, control the process.

2.1.3 Manufacturing Monitoring and Control

RSX-11M tasks can monitor manufacturing operations, test the quality of products, furnish data to other production systems and inventory control systems, and manage the flow of work between departments. Manufacturing data can be collected directly from sensors or entered manually through terminals dedicated to that use.

2.1.4 Laboratory and Medical Data Processing

Medical and laboratory settings require small computer systems to perform precisely defined real-time operations. The monitoring of patients for example, demands instant response to all changes in a patient's condition. A PDP-11 connected to monitoring devices can keep a continual log of the patient's condition, while running other programs to diagnose symptoms. Other possible laboratory and medical applications include:

- X-ray diffraction measurement
- Gas chromatography
- Psychological testing
- Particle acceleration

2.2 SUPPORTED LANGUAGES AND PROGRAM DEVELOPMENT

The real-time activities described above require specialized software systems (compilers or assemblers). RSX-11M supports several programming languages so you can choose the ones most suited to your particular applications. These languages include:

- MACRO-11
- BASIC-11
- BASIC-PLUS-2
- COBOL
- CORAL-66
- FORTRAN IV or FORTRAN IV-PLUS

The range of languages that RSX-11M supports allows you to improve existing applications and to develop new ones without changing operating systems. For example, a company that uses RSX-11M for a real-time process control application might decide to use the same system to handle its payroll. All the process control programs may have been written in FORTRAN IV, not the best language for the payroll operation. The company could buy the RSX-11 COBOL compiler to use for developing payroll programs without significantly affecting the existing system.

2.3 COMPUTER NETWORKS

You can use an optional software package, DECNET-11, to establish a computer network.

A network connects several computer systems, communication devices, and I/O devices. The computer systems, called "nodes," are connected by hardware, called "physical links."

In addition, DECNET-11 allows:

- **Device sharing.** Device sharing allows a DECNET-11 user to connect to and use the peripheral devices of a remotely located node.
- **File sharing.** File sharing permits you to read from, write to, and update files on a remotely located node.
- **Program sharing.** A loadable program can be transferred to a remotely located node where it can be loaded and executed.

DECNET-11 allows tasks to communicate with each other across nodes, even if the nodes are geographically remote from each other. A task executing in one node can send data to a task executing in another node. DECNET-11 monitors the data transfer, performing such functions as error detection.

CHAPTER 3

REAL-TIME AND MULTIPROGRAMMING OPERATIONS

RSX-11M real-time and multiprogramming operations require the interaction of the following system elements:

- **Memory.** Memory is the hardware storage medium in which the executive and user and system programs reside and run.
- **The Executive.** The RSX-11M Executive is the operating system software that directs all program execution.
- **User and System Programs.** RSX-11M executable programs are called "tasks." Tasks can either be user-written or supplied by DIGITAL (system tasks).

This chapter introduces the memory and executive facilities that combine to produce the RSX-11M real-time and multiprogramming facilities.

3.1 MEMORY ORGANIZATION

A task runs in a contiguous area of memory called a partition. The size and location of these partitions are set at system generation. Each partition has:

- A name
- A defined size
- A fixed base address
- A defined type

The relationship between a task and the partition in which it runs depends on whether the system is mapped or unmapped, and whether the partition is system- or user-controlled.

3.1.1 Mapped and Unmapped Systems

RSX-11M is designed to run on almost all PDP-11 computers. The PDP-11 allows a task to directly address only 32K words of memory. A special hardware device, a memory management unit, is available to allow you to use memory larger than 32K words. This unit is available with PDP-11/23/34/35/40/45/50/55/60/70 processors. The memory management unit associates addresses used in tasks (virtual addresses) with actual locations in memory (called physical addresses). Virtual addresses can range from 0 to 32K words, and physical addresses can

range from 0 to 124K words on all processors except the PDP-11/70. Physical addresses on a PDP-11/70 can range from 0 to 1920K words.

A PDP-11 system that includes a memory management unit is called a mapped system. Systems without the unit are unmapped.

RSX-11M users create tasks differently, depending on whether the system is mapped or unmapped. Before you can run an object module created by a language compiler or assembler, it must be processed by the Task Builder. If the task is to run on an unmapped system, you must specify to the Task Builder the base address of the partition in which it is to run. The task cannot run at a base address different from the address specified in the Task Builder command.

In a mapped system, however, every task has a virtual base address of 0. The memory management unit maps the virtual addresses of a task to the actual physical addresses in which the task resides. This mapping is invisible (transparent) to the person who is running the task. Therefore, a task in a mapped system can run in any partition large enough to contain it.

The only exception to this mapping method occurs when a privileged task is set to run in the memory space allocated to the executive. Such tasks will not be relocated.

3.1.2 **Partition Types**

RSX-11M tasks can execute in two types of partitions:

- System-controlled
- User-controlled

In a system-controlled partition, which can only be used on a mapped system, the Executive allocates available space to make room for as many tasks as possible at one time. This allocation can involve shuffling tasks already in memory (resident). The shuffling arranges available space into a contiguous block large enough to contain a requested task. Only mapped systems contain system-controlled partitions.

A user-controlled partition is allocated exclusively to one task at a time. This type of partition can be used with both mapped and unmapped systems.

3.1.3 **Subpartitions**

A user-controlled partition may be subdivided into as many as seven nonoverlapping subpartitions. Like the main partition, a subpartition can contain only one task at a time. Since the subpartitions occupy the same physical memory as the main partition, tasks cannot be simultaneously resident in both the main partition and any subpartitions. However, since the subpartitions can each contain a task, up to seven tasks can run together within a main partition.

Subpartitioning reclaims large storage areas in unmapped systems. For example, when a large task that requires a main partition is no longer active or can be checkpointed, subpartitioning allows the space to be used by a number of smaller real-time tasks.

3.2 EXECUTIVE CONTROL

The RSX-11M Executive allocates CPU time to various system and user tasks. The Central Processing Unit (CPU) is the part of the computer that actually carries out instructions provided by system or user tasks; only one task at a time can control the CPU. Multiprogramming is possible because task execution almost always involves more than CPU usage. A real-time task that initiates a process and then waits for the process to complete may not need CPU time while it is waiting. Therefore, while one task waits for an event to complete, the Executive gives control of the CPU to another task. This happens so rapidly that many individual users seem to have control of the CPU at the same time (apparent concurrency; see Figure 3-1.)

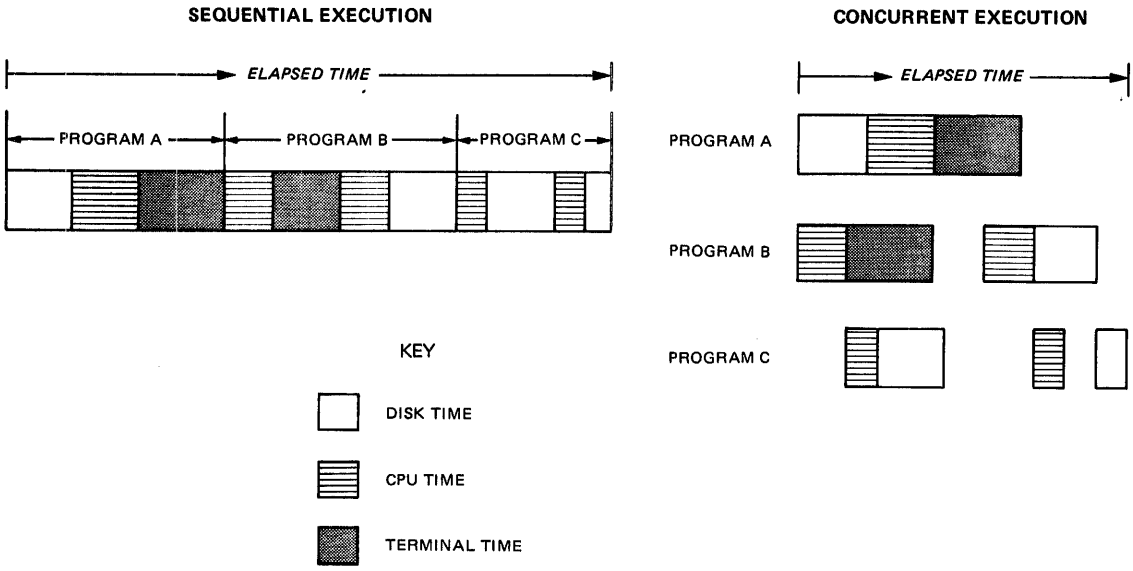

Figure 3-1 Comparison of Sequential and Concurrent Execution of Programs

Figure 3-1 illustrates the advantages of multiprogramming. When tasks A, B, and C run in a system without multiprogramming, they run one after the other. Task A reads some information from disk, operates on it, and generates a report. Task B performs some computation, generates a message, performs some more computation, and writes the result to disk. Task C performs some computation, reads some information from disk, performs additional computation, and writes the result to disk. While one part of the system, such as the disk drive, is busy, other parts, such as the CPU, are idle.

A different sequence of operations is possible if the three tasks run concurrently in a multiprogramming system. The CPU, disk drive, and terminal are active simultaneously; and the concurrent execution of the three tasks takes less time than the sequential execution of the same tasks.

In RSX-11M, the Executive coordinates the execution of all tasks in memory to achieve both efficient use of system resources and rapid response to real-time demands. The following factors affect the way this coordination works:

- Task state
- Priority
- Checkpointing
- Round-robin scheduling
- Swapping
- Significant events

3.2.1 Task State

When you install a task (by issuing an MCR command from a terminal) the system records information about the task in the System Task Directory (STD). The STD is a table that the Executive creates and updates to keep track of all the tasks it knows about. The parameters recorded in the STD include the name and size of the task, the disk address at which the task starts, and the name of the partition in which the task is to run.

An installed task is defined as a task that has an entry in the STD. It is neither resident in memory nor competing for system resources. The Executive considers the task to be inactive until a running task or a command issued from a terminal requests the Executive to activate it. Therefore the Executive recognizes two task states:

- **Dormant.** A dormant task is one that has been installed but is not running.
- **Active.** An active task is an installed task that is running. It remains active until it exits, terminates, or is aborted, when it returns to a dormant state.

  An active task can be either ready-to-run or blocked.

  - Ready-to-run. A ready-to-run task competes with other tasks for CPU time on the basis of the task's priority. The ready-to-run task with the highest priority obtains CPU time and becomes the current task.
  - Blocked. A blocked task is unable to compete for CPU time because it needs access to some resource (memory, a file, a device, etc.) that is not available.

The distinction between dormant tasks and active tasks is important in RSX-11M. A dormant task requires very little memory. However, when the task is needed to service a real-time event, the Executive can quickly introduce it into active competition for the system resources.

An installed task's STD entry enables this quick response because it contains all the parameters the system needs to retrieve the task. Note that the number of installed dormant tasks can far exceed the number of active tasks.

When the Executive receives a request from a terminal or another task to activate a dormant task, it does the following:

- Allocates required memory resources
- Brings the task into memory
- Places the task in active competition with other resident tasks for system resources

If the partition in which a task is to be installed is fully occupied and no other task in that partition can be checkpointed, the newly installed task is placed in a queue by priority with other activated tasks, each waiting for space to become available in its partition. At this point, the task is blocked.

3.2.2 Priority

Active tasks compete for system resources on the basis of priority and resource availability. The priority of a task is determined by a number assigned to the task when it is built, when it is installed, or when it is running. Task priorities can be established by both privileged and nonprivileged users. However, only privileged users can run tasks created at a high priority. The priority is in the range 1 to 250(10), where a higher number indicates a higher priority. The ready-to-run task with the highest priority gains control of the CPU. When that task becomes blocked (waiting for an I/O transfer to complete, for example), the Executive looks for the ready-to-run task with the next highest priority.

In an RSX-11M system that mixes real-time tasks with other jobs, the real-time tasks should be assigned higher priority numbers. This arrangement ensures that the real-time tasks receive CPU time ahead of the less time-dependent tasks.

For example, the text editors commonly used in program development spend much of their time waiting for terminal I/O to complete. During these waiting periods, the editors are in a blocked state. However, when the I/O finishes, the user wants a rapid response to his or her next request. To gain this response, the system manager can assign a priority to the text editors that is higher than more CPU-dependent tasks like the Task Builder or the Assembler.

3.2.3 Checkpointing

Checkpointing allows tasks not currently resident in memory to gain control of the CPU. In some instances, an installed task cannot compete for the processor because the partition in which it was installed is fully occupied. If the partition contains a task that has a lower priority and is checkpointable, the Executive can move that task out of memory to make room for the higher priority task. This process is called checkpointing. When the latter task is finished, the checkpointed task is reactivated and continues processing from the point at which it was interrupted.

To be checkpointable, tasks require a space on disk equal to the size of the task or its partition. While a checkpointed task is stored in this space, a higher priority task occupies memory. Checkpoint space is allocated either statically at Task Build time or dynamically at runtime.

When submitting a compiled or assembled program to the Task Builder, you can specify that checkpoint space be allocated in the task image. Therefore, checkpoint space is always available on disk while the task is running, whether or not the Executive actually needs to checkpoint the task. This is called static allocation of checkpoint space: the location and size of the area do not change unless the object module is task built with different parameters.

Dynamic allocation of checkpoint space allows a more efficient use of disk storage. Instead of reserving disk space for each checkpointable task -- space which may not be needed -- checkpoint files can be created on disk to contain all checkpointed tasks. The size of the files depends on an estimation of the checkpoint space required on the system at any given time. When the system allocates checkpoint space dynamically, tasks do not need to be built as checkpointable. The user determines whether a task is checkpointable when it is installed. Then, when the Executive needs to checkpoint a task, it determines that the task is checkpointable and rolls it out to a checkpoint file. The main drawback to dynamic allocation of checkpoint space is that space in a checkpoint file may not always be available.

3.2.4 Round-Robin Scheduling

When several tasks have equal priorities, the Executive tends to give CPU time more often to those tasks that appear first in the STD queue. (Entries with equal priorities appear in the STD in the order in which the tasks were installed.) However, RSX-11M provides a system generation option called round-robin scheduling. This option periodically rotates tasks of equal priority within the STD. The overall effect of the round-robin scheduler is to distribute use of the CPU more evenly because each equal priority task has its turn to be at the head of the queue.

3.2.5 Swapping

Another problem arises when several active tasks with equal priorities compete for partition space in memory. A task cannot normally cause the Executive to checkpoint another task of the same or higher priority. Therefore a task of equal or lower priority could be prevented from accessing memory.

Swapping enables the Executive to checkpoint tasks with similar priorities in and out of memory. (The tasks must be checkpointable.) When a task begins to run, the Executive adds a "swapping priority" to the task's normal running priority. The Executive then decrements the swapping priority (which eventually has a negative value) as the task runs. When the sum of the decremented swapping priority and the task's running priority creates an actual priority less than that of a competing task, the Executive checkpoints the running task to make room for the competing task. The Executive then places the checkpointed task at the end of the queue of active tasks competing for memory. (The swapping priority only affects allocation of space in partitions. It does not affect CPU scheduling or I/O dispatching, which are governed solely by the task's running priority.)

3.2.6 Significant Events

Certain occurrences called significant events cause the Executive to reevaluate the eligibility of all active tasks to run. When a significant event occurs, the Executive scans the queue of active tasks and runs the highest priority ready-to-run task. Significant events include the following:

- An I/O completion
- A task exit
- The removal of an entry from a clock queue
- The execution of some system directives issued by a task
- The execution of the round-robin scheduler

3.2.7 Example of a 16K Unmapped System

Figure 3-2 shows a possible layout of memory for a 16K unmapped system. The 8K Executive region consists of a system-controlled partition, the RSX-11M Executive, device drivers, and a user-controlled partition called SYSPAR.

In this example, SYSPAR contains the file system (F11ACP). The Monitor Console Routine (MCR) and the Task Termination Notification routine (TKTN). Since the F11ACP is checkpointable and has a lower priority than MCR or TKTN, MCR or TKTN can cause the Executive to checkpoint the F11ACP. The higher priority task can then take over the system resources.

The exact location and size of F11ACP, MCR, and TKTN are determined at system generation.

The 8K User region consists of a user-controlled main partition called PAR8K and three subpartitions, SUBA, SUBB, and SUBC. PAR8K contains program development facilities such as language processors and the Task Builder. These tasks usually have a low priority and are checkpointable.

SUBA, SUBB, and SUBC are available for real-time tasks. If one of the real-time tasks in these partitions needs the CPU and has a higher priority than the task currently occupying the main partition, the task in the main partition is checkpointed.

If multiple tasks in SYSPAR are ready to run, control of the CPU is determined by task priority.

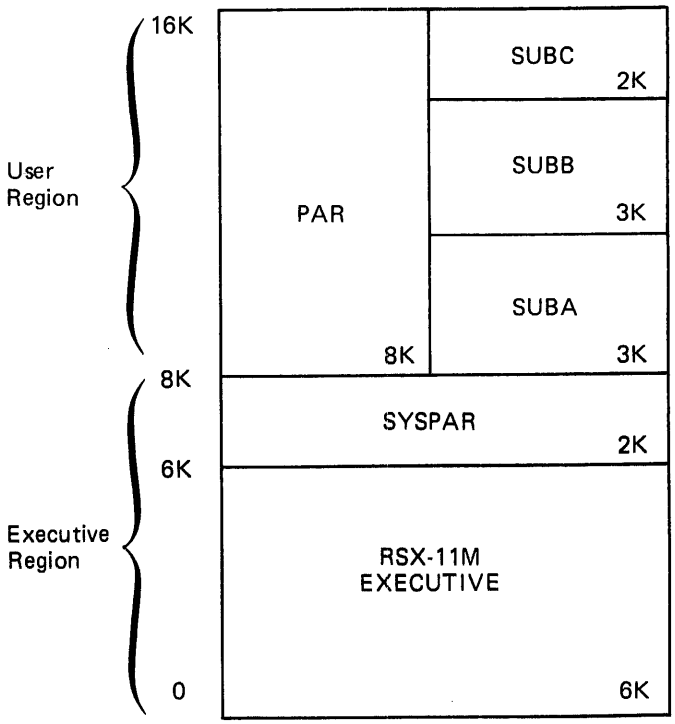

Figure 3-2 Sample Unmapped System Memory Layout

3.3 SYSTEM DIRECTIVE FUNCTIONS

A system directive is a request from a task to the Executive to perform an operation. System and user tasks use some directives to control the execution and interaction of tasks. The execution of other system directives causes significant events to occur. Therefore, directly or indirectly, system directives affect the way the Executive shares system resources among concurrently active tasks.

System directives enable tasks to:

- Obtain task and system information
- Measure time intervals
- Perform I/O functions
- Manipulate logical and virtual address space
- Suspend and resume execution
- Request the execution of another task
- Exit

System directives allow tasks to use some major RSX-11M features, including:

- Event flags
- System traps
- Extended logical address space

3.3.1 Event Flags

Significant events affect Executive management of task execution and allow tasks to coordinate internal task activity and communicate with other tasks. For example, a task can issue a system directive to associate an event flag with a specific significant event. When that event occurs, the Executive sets the flag. Therefore, a task can determine whether the event has occurred by testing the state of the flag.

Ninety-six event flags are available to enable tasks to distinguish one event from another. Each event flag has a corresponding event flag number. The first 32 flags are local to each individual task and are set or cleared as a result of the task's operation. The second 32 flags are called System Global Event flags and can be used by any task. The third flags are called Group Global event flags and are available to all of the users in a single user group (such as [303,...]).

Tasks must use global flags to communicate with other tasks, since one task cannot refer to another task's local flags.

3.3.2 System Traps

System traps transfer control within the system and provide tasks with the means to monitor and react to events. The Executive initiates system traps when certain events occur. The trap then transfers control from the running task to routines associated with the event. This gives the task the opportunity to service the event by entering a user-written routine.

There are two kinds of system traps:

- **Synchronous System Traps (SSTs).** SSTs detect events directly associated with the execution of task instructions. They are "synchronous" because they always occur at the same point in the program. The running task controls when the trap occurs. For example, an illegal instruction causes an SST to occur.
- **Asynchronous System Traps** (ASTs). ASTs detect significant events that occur "asynchronously" during the task's execution; that is, the task has no control over the precise time that the event occurs. For example, the completion of an I/O transfer causes an AST to occur.

To use system traps, a task issues system directives that establish entry points for user-written routines to respond to the event. Entry points for SSTs are specified in a single table. AST entry points are set by individual directives for each kind of AST. When a trap occurs, the task automatically enters the appropriate routine (if its entry point is specified).

3.3.3 Extended Logical Address Space

An RSX-11M task specifies an address in a 16-bit word. The largest address that can be expressed in a 16-bit word is 65,536 bytes or 32,768 words (commonly referred to as 32K words). Therefore, a task can directly address only 32K words. To avoid limiting the size of a task to its addressing capability, a task can use overlays that are defined at Task Build time, or it can use memory management directives.

An overlaid task is arranged into segments: a root segment, which is always in memory when the task is active, and overlay segments, which can be read into memory as required. The segments concurrently in memory cannot exceed 32K words.

Memory management (PLAS) directives allow task segments resident in memory to exceed 32K words. The directives use the KT11 memory management hardware to map task virtual addresses to different logical addresses. Therefore, the task can reside entirely in memory and map its virtual addresses to different physical addresses.

RSX-11M uses three kinds of address space:

- **Physical address space.** Physical address space consists of the actual physical memory in which tasks reside and execute.

- **Logical address space.** Logical address space is the total amount of physical address space to which the task has access rights.

- **Virtual address space.** Virtual address space corresponds to the 32K of addresses that the task can explicitly specify in a 16-bit word. If a task does not use memory management directives, its logical and virtual address space are exactly the same. If a task does use these directives, it can map its virtual addresses to different parts of its logical address space. The net effect is to allow a task's logical address space to exceed 32K.

The memory management directives also allow a task to expand its logical address space dynamically while it runs. In other words, a task can access logical addresses that are not part of its static task image. This expansion is accomplished when a task issues directives that create a new region of logical space and then map a range of virtual addresses to the newly created region. A task can also map virtual addresses to logical areas that are part of another task. The other task's logical addresses then become part of the original task's logical address space.

The ability to map to virtual addresses also allows tasks to interact by using shared regions of memory. For example, a running task can create a new region of logical address space and writes a large amount of data to it. Other tasks can then access that data by mapping some of their virtual addresses to the same region. Interacting tasks also allow a greater number of common routines. Moreover tasks can map to required routines when they run rather than link to them at Task Build time.

CHAPTER 4

SYSTEM OPERATION

RSX-11M operating procedures vary from one system to another, depending on the function of the computer and the hardware options it supports. A laboratory may have a PDP-11/10 used for one real-time application with only one disk and one terminal. A lab technician might oversee this computer system as one of many laboratory duties. In contrast, a manufacturer may have a PDP-11/70 that performs a process control job, all the company's accounting and payroll applications, as well as substantial program development. Such a system could have a dozen or more terminals and numerous disk and magnetic tape drives. A system of this size requires a full-time operator to maintain the equipment and manage the use of the system.

Although an RSX-11M system may not require a full-time computer operator, someone must know how to operate RSX-11M. The term "operator" refers to anyone who interfaces with or oversees RSX-11M. This chapter discusses some basic concepts required to operate an RSX-11M system. Most of the concepts apply to all systems; however, some features (multiuser protection, for example) are optional and may not be present on your system.

4.1 THE MCR INTERFACE

You can communicate with RSX-11M by entering commands at a terminal. The terminal sends the commands to the Monitor Console Routine (MCR) processor, which either executes the commands itself or activates a system- or user-written task to execute the commands.

MCR commands allow an operator to:

- Start up the system
- Manage peripheral devices
- Control task execution
- Obtain system and task information
- Activate system- or user-written tasks that request input from the terminal

An operator uses MCR commands to establish the base of installed tasks, which the Executive, active tasks, and further MCR commands manipulate in later operations.

To restrict the use of commands that directly affect system performance, RSX-11M considers some MCR commands and command options to be privileged. You can issue a privileged command only from a privileged terminal.

4.1.1 External Scheduling of Task Execution

An important MCR function is the external scheduling of task execution. This type of scheduling works in conjunction with the Executive's priority-driven internal scheduling of active tasks. The operator can include time parameters with the command that activates an installed task. The time parameters request the Executive to run a task:

- At a specified time from the current moment
- At a specified time from clock unit synchronization
- At an absolute time of day
- Immediately

All of these time options are available with or without periodic rescheduling. RSX-11M also supports an unlimited number of programmed timers for each task in the system. The user task can create its own timer, which the Executive then decrements at regular intervals. When the timer reaches zero, the Executive sets an event flag or generates an Asynchronous System Trap (AST) that passes control back to a task.

4.1.2 Indirect Command Files

An indirect command file contains a list of commands exclusive to, and interpretable by, a single task. The interpreting task is usually an RSX-11M system program, such as MCR, PIP, the MACRO-11 Assembler, or the Task Builder.

To execute the commands in indirect files, the user enters a file specifier preceded by an at symbol (@) in response to a prompt from the appropriate task.

For example, to execute a file containing only MACRO-11 commands, enter:

```
MAC  @INPT.CMD
```

The MACRO-11 Assembler then executes the commands in INPT.CMD.

Indirect files can invoke other indirect files; the maximum nesting depth varies with each task.

4.1.3 The MCR Indirect File Processor

Most tasks read and respond to commands contained in an indirect file as if the commands were entered directly from a terminal. MCR, however, has an indirect file processor that interprets indirect commands. An MCR indirect command file can contain both MCR commands and commands (or directives) to the indirect file processor itself. MCR optionally displays on the entering terminal all of the commands executed from the indirect command file.

The indirect file processor first reads the command file and interprets each command line either as a command to be passed to MCR or as a request for action by the indirect file processor itself. Directives to the indirect file processor are distinguished by a period (.) as the first character in the line.

The MCR indirect file processor also enables the user to define symbols. These symbols can subsequently be tested to control the operation of the indirect command file. The symbols consist of 1 to 6 ASCII characters that can be a true or false value, an octal or decimal number, or a character string. The first directive to specify a symbol defines that symbol; subsequent references test, compare, or redefine it.

4.1.3.1 **Symbol Value Substitution** - The symbols described above are used in directives to the indirect file processor. Indirect MCR commands can use the values assigned to string symbols by replacing a normal parameter (for example, a device-unit) with the symbol name enclosed in single quotation marks (for example, 'DEV'). The indirect file processor replaces the symbol name enclosed in single quotation marks with the string value assigned to the symbol. When the processor encounters a single quote, it treats the subsequent text, up to the second single quote, as a symbol. The processor searches the table of symbols for the corresponding string value and substitutes it in the indirect command line before the line is interpreted.

4.2 TERMINAL OPERATION

In RSX-11M, a variable number of terminals can operate concurrently. In addition, each terminal operates independently of others in the system so that each can run a different task. In a system that supports multiuser protection a user must log onto a terminal before issuing further commands. In nonmultiuser protection systems, a user can issue commands whenever the terminal displays an appropriate prompt.

In multiuser protection systems, individual users are either privileged or nonprivileged; when a user logs on, the terminal assumes the privilege status of that user. In nonmultiuser protection systems, however, a terminal's privilege status is determined at system generation. After system generation, a user can issue an MCR command at a privileged terminal to modify the privilege status of any other terminal connected to the system. The privilege status of a terminal determines what commands you will be able to issue from that terminal.

4.2.1 Attached Terminals

RSX-11M permits tasks to solicit input from a specific terminal or terminals. While the task is receiving input, the terminal is attached to it. Thus, all of the terminal's output goes directly to the attached task, with one exception. The exception is a <CTRL/C> character (the C key typed while pressing the CTRL key). This command gains the attention of MCR and allows you to issue one MCR command. Note that attaching to the terminal is a function of the task rather than the user.

Some applications require that a user be denied access to MCR. In that case, a task can attach to the terminal with a special subfunction that causes the system to generate an AST whenever someone enters unsolicited input, including <CTRL/C>, from the attached terminal.

4.2.2 Slave Terminals

RSX-11M also permits you to dedicate a terminal exclusively to one or more specific tasks, using an MCR command or special I/O functions in the task.

A terminal restricted in this way is called a slave terminal. The difference between a slave and attached terminal is that the system ignores all unsolicited input from a slave terminal, including <CTRL/C>. Until the user issues an MCR command from a privileged terminal to delete the slave status, the terminal can only be used to communicate with the task soliciting input. A task can also issue an I/O function to delete the slave status of a terminal. Slave terminals are often dedicated to critical real-time applications.

4.3 MULTIUSER PROTECTION

Multiuser protection, a system generation option, allows you to monitor and control individual users of an RSX-11M system. A system manager assigns a User Identification Code (UIC) to each user. The UIC determines whether the user is privileged or nonprivileged. UICs in groups less than or equal to 10(8) are privileged. Those in groups numbered above 10(8) are nonprivileged. When logging on a terminal, the user supplies a last name or UIC and a password. The system then checks that the password matches the last name or UIC, and sets the terminal's privilege status, according to the UIC.

4.3.1 Public and Private Devices

Multiuser protection systems allow both privileged and nonprivileged users to issue MCR commands to allocate a device (a disk drive, for example) as the user's private device. Allocating the device prevents other nonprivileged users from accessing it, although privileged users can still access it.

A nonprivileged user can use a device allocated to him or her to perform MCR functions that are privileged in nonmultiuser systems. These functions include preparing a disk or magnetic tape for use by the RSX-11M file system and putting the disk or tape online and offline.

Multiuser protection systems also allow a privileged user to make certain devices public. Public devices cannot be allocated to individual users. When a line printer is public, for example, all users have equal access to it.

4.4 SYSTEM MAINTENANCE FEATURES

4.4.1 Error Logging

RSX-11M provides an error logging subsystem as a system generation option for systems that are 24K words or larger. The error logging subsystem monitors the reliability of system hardware. Routines in the Executive and specific Error Logging tasks share control of Error Logging.

The error logging task (ERRLOG) continually records information detected by the Executive or device drivers about hardware errors, regardless of whether the error was recoverable (soft).

At user-determined intervals, another task, (PSE) can be run to format the error information so that reports can be generated on some or all of these errors.

Finally, the report generating task (SYE) produces reports on the errors you select. You can generate a wide variety of reports from the data that Error Logging collects. For example, you can specify a report to cover a certain time period, a certain device or group of devices, or a certain type of error. You can also request a report that contains only information on individual errors, only summary information, or both.

The Executive automatically retries operations involving recoverable errors. However, you could be unaware that the error occurred, if it was recovered, unless your system included Error Logging.

In summary, the Error Logging subsystem:

- Gathers information from the Executive and device drivers about hardware errors
- Stores the information in a file
- Formats the information to produce an error report
- Generates a report according to the user's specification

4.4.2 **Power Failure Restart**

RSX-11M includes a power failure restart routine to handle intermittent short-term power fluctuations with minimal loss of service or data. The routine operates in four phases:

- When power begins to fail, the CPU traps to the Executive, which stores register contents and halts system operations with very little damage.
- When power is restored, the Executive restores the preserved state of the system, including the stored registers.
- The Executive then reactivates the device drivers that were active at the time of the power failure at their power-fail entry points. Drivers can be reactivated either:

  1. Whenever power fails, or
  2. Only when power fails while the driver is processing an I/O request.

  The drivers can then restore their own state (repeat an I/O transfer, for example).

- The Executive then determines if any user tasks requests notification of power failure. (User tasks do this by issuing a system directive that requests an AST on power recovery.) The Executive initiates ASTs for those tasks that requested them.

CHAPTER 5

PROGRAM DEVELOPMENT

Program development on an RSX-11M system usually requires four steps:

1. Creating a source file, using an RSX-11M editing program
2. Compiling or assembling the source file into an object module, using an RSX-11M-supported compiler or assembler
3. Task Building (linking) the object module or modules to create an executable unit called a task
4. Running the task

If errors occur in the program, it is necessary to add a fifth step: debugging the program.

This chapter describes some of the system resources available to create, run, and debug RSX-11M tasks. Figure 5-1 summarizes the steps involved in creating a FORTRAN program.

5.1 EDITING UTILITIES

RSX-11M provides two interactive utility programs to create and edit source files. They are EDT (the DIGITAL Standard Editor) and EDI (the line text editor). Some features of these editors are described below. The RSX-11 Utilities Manual contains complete information on EDT and EDI.

5.1.1 EDI

EDI uses buffers in operations to create and edit text files. These buffers control the amount of data EDI accesses at a time. It reads a line or group of lines from an input file into an EDI buffer and permits the user to edit the material in the buffer from the terminal. When the material is edited, another terminal command can write the data to a new file and read in another group of lines.

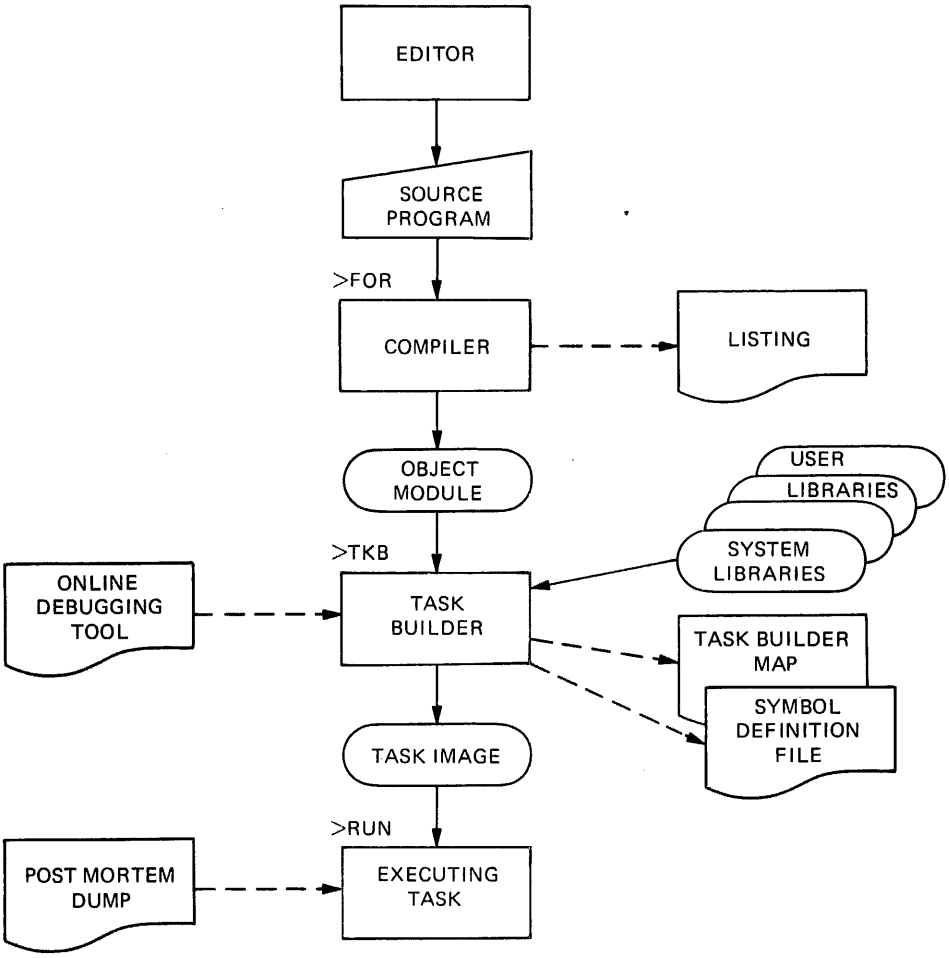

Figure 5-1 Steps to Creating a FORTRAN Program

5.1.2 EDT

EDT permits you to create and modify text files or source programs, on either a line or character basis. EDT also permits you to edit large files without breaking them up into smaller sections.

Features of EDT include:

- English language commands
- Video or hard copy display
- Online documentation of error conditions
- File and buffer I/O
- Maneuverable cursor (in character mode) and line pointer (in line mode)

5.2 PROGRAMMING LANGUAGES

RSX-11M supports several programming languages, including:

- MACRO-11
- BASIC-11
- BASIC-PLUS-2
- COBOL
- CORAL-66
- FORTRAN IV AND FORTRAN IV-PLUS

Source programs must be created according to the requirements of the language you are using. MACRO-11 is the only language that is distributed as part of the RSX-11M system. Compilers or assemblers for other languages are optional and must be purchased separately.

5.2.1 MACRO-11

The MACRO-11 assembly language lets the programmer work directly with the PDP-11 hardware. MACRO-11 also allows the programmer to define sections of code called macros with dummy values and to invoke them to generate repetitive coding sequences.

The MACRO-11 assembler includes the following features:

- Program and command line control of assembly and listing functions
- Specification of input and output device and file names
- Error listing on output device
- Formatted, alphabetized symbol table
- Optional cross-reference symbol listing
- Relocatable object modules
- Global symbols to link separate object modules
- Conditional assembly directives
- Program section (.PSECT) directives
- User-defined macros and macro libraries
- System macro library
- Indirect command files to control the assembly process

5.2.2 BASIC-11

BASIC-11 is an easy-to-learn language that uses English words, abbreviations, and familiar mathematical symbols to perform operations.

The BASIC-11 interpreter performs functions handled by the editor, compiler, and Task Builder for other languages.

Special features of BASIC-11 include:

- Immediate executions of programs after input
- Immediate mode operation for debugging programs and use as a desk calculator
- ASCII files compatible with FORTRAN
- PRINT USING statement for formatting output
- String manipulation functions and dynamic allocation of string storage
- User-defined functions
- CALL statements to pass data to assembly language subroutines
- Graphics and laboratory peripheral support

5.2.3 **BASIC-PLUS-2**

BASIC-PLUS-2 is a compiled language that combines the execution speed of a compiler with the ease of use and familiarity of BASIC-11. The language is an extension of BASIC used in many commercial applications.

Special features of BASIC-PLUS-2 include:

- Virtual arrays
- Enhanced string manipulation capability
- Long variable names
- Complete matrix functionality
- IF...THEN...ELSE statements
- ON ERROR GOTO statements
- Statement modifiers
- User-defined functions
- Multistatement lines and multiline statements

5.2.4 **COBOL**

COBOL provides rapid data processing, primarily for commercial applications. Source programs conform to the ANSI standard for COBOL-74.

PDP-11 COBOL features beyond the ANSI standard include:

- CALL sequences for external subroutines
- String and substring manipulation capabilities
- MERGE utility to combine ODS files
- Reformatting utility to reformat source files
- Report generator

5.2.5 CORAL-66

CORAL 66 is a high-level block-structured language, which performs assembly language operations in industrial and commercial applications.

The CORAL-66 compiler provides:

- BYTE, LONG (32-bit integer) and DOUBLE 64-bit floating point) numeric types
- Generation of reentrant code at the procedure level
- CORAL 66 programs can run on any RSX-11S system that includes the Extended Instruction Set (EIS)
- Programs can optionally select target PDP-11 instruction set
- Optional code optimization
- Option to check the bounds of array variables
- Conditional compilation of defined parts of source code
- INCLUDE keyword to incorporate CORAL 66 source code from user-defined files

5.2.6 FORTRAN-IV and FORTRAN IV-PLUS

RSX-11M supports two versions of PDP-11 FORTRAN: FORTRAN-IV and FORTRAN IV-PLUS. Both versions of FORTRAN meet the specifications of ANSI Standard FORTRAN and include substantial extensions to those specifications. The major differences betweeen the two FORTRANs are:

1. FORTRAN IV-PLUS produces highly optimized code, which executes in less time than FORTRAN-IV code.
2. FORTRAN IV-PLUS can use the floating-point processor option.
3. FORTRAN IV-PLUS can produce shareable code.

Both FORTRAN language processors consist of a compiler and an Object Time System (OTS). The OTS is a set of object modules that can be Task Built with user programs to perform mathematical functions and to detect errors when the program runs.

RSX-11M also provides a set of optional FORTRAN callable process control subroutines that meet Instrument Standard of America requirements.

5.3 BUILDING THE TASK

The Task Builder creates an image called a task by linking one or more object modules. This image can then be installed and run, using MCR commands. The modules usually include the object module created by a compiler or assembler, and object modules containing system or user library routines.

The Task Builder:

- Links object modules
- Resolves references to system or user libraries of object modules
- Allocates virtual address space to the task
- Produces an optional Task Builder map that describes memory allocation, object modules, and global symbol references
- Produces an optional Symbol Definition File
- Builds an overlaid task
- Maps the task to shared regions of memory

5.4 RUNNING THE TASK

To run a task on an RSX-11M system, issue an MCR RUN command, which instructs the system to:

- Locate the task image on the disk where it is stored
- Load a copy of the image into memory
- Execute the task

The RUN command includes options to run tasks at an exact time of day, at a specific time increment, or at a time increment from another clock parameter (at the next hour, minute, second, or clock tick). The command also permits you to run the task immediately or to install, run, and remove the task with one command.

5.5 DEBUGGING THE TASK

RSX-11M provides two system features that can help programmers to diagnose errors in programs. The Online Debugging Tool (ODT) lets you debug MACRO programs interactively, and the Post Mortem and Snapshot Dump (PMD) analyzes data when a user task terminates abnormally.

5.5.1 ODT

ODT is an online debugging program which is incorporated into a task when it is Task Built.

An ODT breakpoint allows the user to run and debug the task at the same time. The task runs to a predetermined point, the breakpoint. There, the task stops and the user can examine and modify the contents of task registers before continuing to run the task.

ODT permits the user to:

- Print the contents of any memory location, in octal, and examine them at the terminal. (These contents can later be altered with the RSX-11M Patch utility.)
- Search for specific bit patterns in the object file.
- Search for words that reference a specific word in the object file.
- Fill a block with a designated value.

5.5.2 **Post Mortem and Snapshot Dumps**

The RSX-11M PMD task provides a dump of a task's registers when the task terminates execution abnormally, or when a user program requests a snapshot of the registers during task execution. Programmers can use these dumps to debug a program offline, when the program does not use ODT.

PMD dumps provide the following information:

- The task name and reason for the dump
- The contents of the task's registers, stack, and program counter at the time of the dump
- The devices and files being used at the time of the dump
- The status bits, event flags, and I/O count for the task
- The terminal that ran the task
- The device and physical address of the task
- The task's virtual memory allocation, in octal words, octal bytes, RADIX-50, or ASCII

CHAPTER 6

FILES AND I/O OPERATIONS

This chapter introduces the RSX-11M file system and describes some basic aspects of I/O operations.

6.1 RSX-11M FILE SYSTEM

In RSX-11M terminology:

- A file is an owner-named area on a volume.
- A volume is a magnetic medium that RSX-11M can recognize, such as a disk, a DECtape, or a magnetic tape.
- An operator is anyone who uses or maintains the RSX-11M system.

6.2 FILES-11

FILES-11 is an RSX-11M system task that oversees the storage and handling of files on volumes. Files-11 volumes are magnetic media that have been specially formatted with an MCR command called Initialize Volume (INITVOL). Volumes that are not in FILES-11 format are described as foreign volumes. Although FILES-11 cannot access these foreign volumes directly, the Files Exchange Utility (FLX) translates files from the foreign DOS or RT-11 format to FILES-11 format. User tasks can also perform I/O operations on foreign volumes, without using FILES-11. This is often necessary to handle real-time applications.

6.2.1 File Ownership and Directories

The information that follows applies primarily to systems having multiuser protection. On nonmultiuser protection systems, users do not have a password or UIC to identify their files. However, they can create UFDs with an MCR UFD command.

When a user receives authorization to use a multiuser protection system and is given a UIC and a password, he or she also acquires a default UFD. The default UFD, which has the same number as the UIC, is in the format [g,m]. The user's group number is g, and the member number within that group is m.

The UFD provides protection for user and system files on the system. A user can use a file name that has already been used in another UFD and the system will recognize it as the file of that name associated with that UFD.

For example:

[303,33]CAT.MSE is not the same file as [303,13]CAT.MSE

All of the UFDs on a volume are listed in the volume's Master File Directory (MFD). The name of the UFD in the MFD is a combination of the group and member numbers, along with a file type of .DIR. For example, the UFD [303,12] becomes 303012.DIR in the MFD.

Both the MFD and the UFD contain the names of files as well as pointers to each file's header. The file header contains information about the physical location of the file segments.

6.2.2 File Protection

To access a file, you must know the UFD in which it is listed and satisfy the conditions in a protection mask associated with the file. The mask specifies four types of access allowed to four user groups. The four types of access are:

- Read access - user group indicated can read the file
- Write access - user group indicated can write to the file
- Extend access - user group indicated can extend the length of the file
- Delete access - user group indicated can delete the file

The four user groups are:

- System -- Tasks and users with a privileged UIC
- Owner -- Tasks and users running under the owner's UIC
- Group -- Tasks and users whose UIC is in the same group as the file's owner
- World -- All other tasks and users

6.2.3 File Specifications

To refer to a file in an RSX-11M command line, use a file specifier in the following format:

volume:[g,m]filename.filetype;versionnumber

volume:

The volume holding the volume that contains the file.

[g,m]

The UFD in which the file is listed (the file owner).

filename

The name of the file, as it appears in the UFD.

filetype

An abbreviation describing the file's contents.

versionnumber

A number that differentiates between several copies of the same file.

The file system uses the information in the file specification to retrieve the file as directed by the command line in which the specification occurs. The same file specification is used with all RSX-11M tasks, including compilers, editors, and the Task Builder.

6.2.4 File Manipulation

The Files-11 system disguises differences between files on different types of volumes and enables you to transfer files from one type of volume to another, without knowing how the file was originally formatted.

Before Files-11 can access a file, the volume containing the file must be known to the system. Issue an MCR Mount command to perform this operation. Once the required volumes are ready, the user can manipulate files with system utilities or user-written tasks. The following sections describe some of the system utilities.

6.2.4.1 **PIP** - The most commonly used file manipulation utility is the Peripheral Interchange Program (PIP). The major functions of PIP are:

- Copying files from one Files-11 device to another
- Deleting and Purging files
- Renaming files
- Listing User File Directories

6.2.4.2 **Queue Manager** - The Queue Manager and the associated Print command handle the orderly operation of the line printer or other despooling device. The Queue Manager maintains lists of files to be printed and sends the files to the printer, usually in chronological order. The Print command places the names of files in queues.

The major functions of the Queue Manager are to:

- Control the operation of the system line printer or despooling device
- Control the format of line printer output (for example, output on special forms or standard line printer paper)
- Prevent files from monopolizing printer resources or interrupting other line printer operations

6.3 TASK I/O OPERATIONS

User tasks running on RSX-11M can access data with three sets of subroutines:

- File Control Services (FCS)
- Record Management Services (RMS)
- RMS-11K (a separate option that allows you to use indexed files)

FCS and RMS software are to individual files what FILES-11 software is to entire volumes. As discussed previously, FILES-11 routines oversee the storage and handling of files on volumes. FCS and RMS, on the other hand, oversee the storage and handling of data within each file. They provide access to individual files, impose logical structures upon them, and maintain the integrity of the data they contain.

Through FCS or RMS, tasks access files on FILES-11 volumes and process their contents, using either block-oriented or record-oriented I/O operations.

Block-oriented operations treat files as arrays of equal-sized structures called virtual blocks. The size of virtual blocks in a file is determined by the type of volume containing the file. On disk volumes, virtual blocks in all files always contain 512 bytes. On ANSI-compatible magnetic tapes, in contrast, the user can specify block sizes other than 512 bytes.

Record-oriented I/O operations treat files as collections of logical records. The size of individual records is specified by the user.

(See the RSX I/O Operations Reference Manual for more information on FCS and RMS.)

6.3.1 File Control Services (FCS)

FCS routines let you read and write files on FILES-11 structured devices and process files as logical records. You can write a collection of data to a file in a way that allows it to be retrieved at will. You do not have to know anything about the format in which the data was written. Therefore, FCS is transparent (invisible) when you are using it.

FCS views the contents of a file as a continuous sequence of user records (see Figure 6-1). The size of individual records is determined by the user rather than by the physical medium.

FCS provides two access modes for storing and retrieving individual records. Sequential mode stores successive records physically adjacent to previous records and retrieves records, based on this adjacency. Random mode is permitted only for disk files containing equal sized records. In this mode, tasks can read and write records by specifying their relative position in the file.

6.3.2 Record Management Services (RMS)

RMS routines let you read and write files in one of three ways:

- Relative
- Sequential
- Indexed

In sequential file organization (see Figure 6-1) records appear in the same order in which they were created.

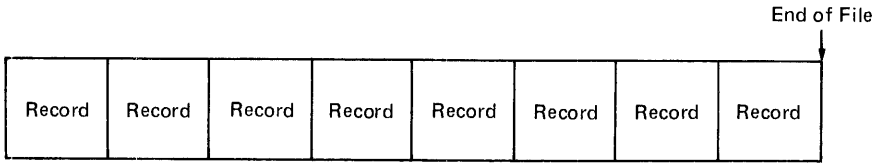

Figure 6-1 Sequential File Organization

In relative file organization, RMS creates a series of fixed-size record cells. The cells are numbered according to their location relative to the beginning of the file. Each cell can contain a single record. However, empty cells can be interspersed among cells containing records. Figure 6-2 illustrates the structure of a relative file.

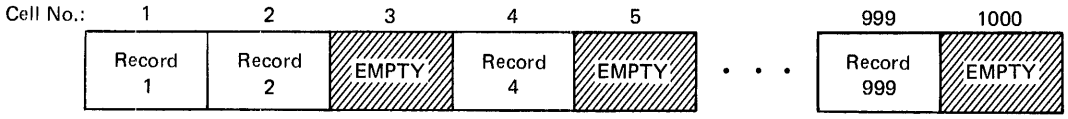

Figure 6-2 Relative File Organization

The optional RMS-11K software permits a third type of file structure: indexed file organization. Unlike the physical ordering of records in a sequential file, or the relative ordering in relative files, indexed file organization is transparent to the program using it. Keys in the records of the file, established by RMS-11K, determine the placement of records in an indexed file. These keys, unique character strings that appear in every record of the file, tell the operating system where to look for the next record. Figure 6-3 illustrates the structure of an indexed file.

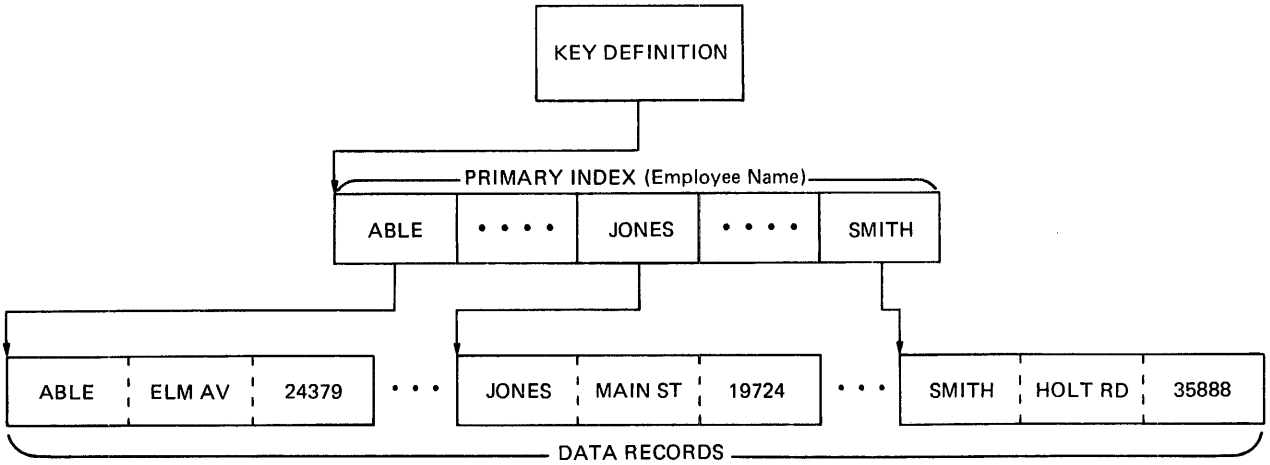

Figure 6-3 Indexed File Organization

6.3.3 Device Independence

RMS and FCS allow RSX-11M programmers to write code independent of the physical characteristics of devices. Therefore, the programmer does not have to know what devices the program will eventually use for its I/O operations. Note, however, the requirement that random access mode can be used only on disk files. A program performs I/O on Logical Unit Numbers (LUNs) that the programmer or operator assigns to specific devices ahead of time.

An operator can subsequently issue an MCR command to change the physical device used for I/O if the device fails, for example. This command will redirect all I/O intended for one device to another of the same type. This redirection has no effect on LUN assignments; FCS or RMS changes the LUNs transparent to the programmer.

You can also use logical devices to associate LUNs with physical devices. A task can assign a LUN to a logical device and then issue an MCR command to associate the logical device name with a physical device unit. A logical device name does not correspond to a physical device until a device is explicitly assigned to it.

6.4 PHYSICAL I/O OPERATIONS

Physical I/O operations involve the transmission of data between main memory and connected peripheral devices. In RSX-11M a connected device cannot be operated unless there is software in memory for that device type. This software can be either an I/O driver or a user task that connects directly to a hardware vector associated with the device.

An I/O driver performs functions that enable physical I/O operations to occur. Drivers for most device types are built into the Executive at system generation. RSX-11M includes drivers for all the supported devices. In addition, RSX-11M permits user-written I/O drivers for nonstandard devices. This capability is essential for real-time applications that communicate directly with devices such as production or laboratory equipment (refer to the RSX-11M Guide to Writing an I/O Driver for more information).

Some systems have devices that are used infrequently. To avoid wasting memory with permanent drivers that are seldom used, you can select a system generation option to allow drivers to be loaded and unloaded.

An alternative approach to drivers is the connect-to-interrupt vector facility provided by the RSX-11M Executive. Using this facility, the user task itself can receive the hardware interrupts generated through hardware vectors dedicated to the device.

A

B

C

D

E

F

H

I

L

M

N

O

P

Q

R

S

T

U

V

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code or Country_____

Please cut along this line.

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

RT/C SOFTWARE PUBLICATIONS TW/A14
DIGITAL EQUIPMENT CORPORATION
1925 ANDOVER STREET
TEWKSBURY, MASSACHUSETTS 01876

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line