pdp11

INTRODUCTION TO RSX-11M

Order No. DEC-11-OMIEA-A-D

digital

INTRODUCTION TO RSX-11M

Order No. DEC-11-OMIEA-A-D

digital equipment corporation · maynard, massachusetts

First Printing, May, 1974

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.

CONTENTS

FIGURES

INTRODUCTION TO THE RSX-11M REALTIME EXECUTIVE

This manual is an introduction to Digital Equipment Corporation's RSX-11M Realtime Operating System. It assumes familiarity with a few process-control and data processing terms and should prove useful to managers, programmers, and operators contemplating use or acquisition of an RSX-11M based system. Topics covered include:

1. Applications of RSX-11M;
2. RSX-11M's Capabilities and Features, and
3. The Organization of RSX-11M.

CHAPTER 1

APPLICATIONS OF RSX-11M

1.1 INTRODUCTION

The RSX-11M Realtime Operating System can be used on any DEC PDP-11 computer* and is designed for applications requiring response to physical events as they occur. The system may function as an unattended, stand-alone process controller, as an operator-controlled** interface, or as part of a network of DEC computers.

1.2 REALTIME APPLICATIONS

1.2.1 Data Acquisition

Data acquisition is the source collection of physically generated data for subsequent use in evaluation and control. Data acquisition applications frequently must respond to bursts of data that can only be serviced by many user tasks operating concurrently. RSX-11M is designed to permit the concurrent execution of user tasks (multiprogramming) which makes the servicing of demanding data acquisition applications possible. RSX-11M, under user program direction, can process data acquired from instruments such as spectrometers, flowmeters, and thermocouples.

1.2.2 Process Control

A process control application usually involves data acquisition, analysis, and a feedback loop aimed at controlling the behavior of a continuous process. Oil refineries or steel rolling mills are typical examples of such processes.

The signals that the PDP-11 receives from devices attached to the process are called process input. By reading and operating on various process inputs, the PDP-11 can monitor the status of many parts of the process, such as temperatures, flow rates, and the amount of raw materials being used. Engineering and operational data stored in the system can be used to determine what action should be taken to keep the process running properly. These decisions can also be made by control optimization programs, which are programs that make adjustments based on interrelationships of various parts of the process.

* The minimum configuration is a 16K PDP-11/10 with a Teletype (Teletype is a registered trademark of the Teletype Corporation) or LA30 for the console terminal; one RK disk drive, which is the system distribution medium; a KW11-L/P for the system clock; a hardware bootstrap loader, and either one additional RK disk drive, a DECtape or cassette.

** RSX-11M may or may not require an operator in the conventional use of the term. In this document operator refers to anyone who is controlling the behavior of the sytem form a console in order to accomplish some task.

After operational decisions have been made, the PDP-11 generates signals that control the valves, switches, and relays that in turn control the process. These signals are called process output.

1.2.3 Manufacturing

In a manufacturing application, RSX-11M monitors manufacturing operations, tests the quality of products, furnishes production data to higher-level production and inventory control systems, and manages the flow of work between departments. Data in these applications can be collected directly from sensors or entered by operating personnel into special devices located in the manufacturing plant.

1.2.4 Laboratory Data Processing

Until approximately 1964, the concept of having a computer for use by researchers within the laboratory environment was almost unheard of. Equipment that could meet the laboratory environment requirements of hands-on, on-line, interactive computing was simply unavailable. DEC has been a leader in the LDP field since the introduction of the LINC processor in 1965 (LINC is an acronym for Laboratory Instrument Computer) which was the first computing system designed and priced to meet the needs of the laboratory setting. RSX-11M provides the basis for a natural cost-performance extension of DEC's pioneering concept.

1.3 BACKGROUND PROCESSING

Typically, a computer system designed to control a realtime process is engineered to provide an adequate service level at peak load. Any peak load design results in low resource utilization during the intervals when peak loads are not being experienced. To make use of these idle resources, background tasks are introduced into the system's workload. Background tasks usually have less stringent time constraints than those of the primary realtime application being serviced, Typical of these background tasks are program preparation, off-line data reduction, and payroll processing.

RSX-11M is designed as a priority system and, as such, services background tasks as just another unit of work requiring processor attention. Tasks are given priorities and all processor resources are doled out based on the task's priority and resource availability. Thus, to distinguish between levels of urgency, the operator simply assigns different priorities. And, if the urgency of tasks changes, the operator can alter the priorities of installed tasks without halting system operation.

To further enhance the processing of background tasks, RSX-11M has a checkpointing feature which will remove a lower priority task from memory, replace it with a task of higher priority, and, after the servicing of the higher priority task has completed, resume the original lower priority task by restoring it to active competition for processor resources.

1.4 COMMUNICATIONS

The ability of a computer system to communicate with both humans and other computer systems is becoming increasingly important. RSX-11M supports a variety of hardware and software interfaces which permit it to communicate with other DEC computers, and also with IBM 360/370 systems.

Asynchronous communications devices using ASCII code are supported at line speeds from 110 to 2400 baud. Devices in this category include DEC's LA30 DECwriter and the VT05B Alphanumeric display. These devices are full duplex and permit both data entry and display.

Synchronous transmission provides a more efficient communications mechanism when large amounts of binary or ASCII data must be moved from one digital system to another, as is the case in computer networks or hierarchical systems. In order to transfer programs and data between systems, RSX-11M can communicate with other systems over synchronous lines.

1.5 PROGRAMMING

The RSX-11M System performs the applications just discussed by executing programs. A program is a sequence of instructions that directs the computer in manipulating data to achieve some goal.

Application programs are highly specialized and unique, and are, in general, prepared by the user to meet the requirements of his installation.

System programs are common to all applications and are delivered as components of an RSX-11M system. These include programs to:

Compile and assemble programs written in FORTRAN IV and MACRO-11;

Service input/output devices;

Respond to operator requests, and

Control the allocation of system resources.

System programs make it possible for the user to concentrate on writing applications programs. The next chapter discusses in detail the functions supplied by the RSX-11M system.

CHAPTER 2

RSX-11M SERVICES

2.1 WRITING, STORING, AND EXECUTING PROGRAMS WRITTEN IN SYMBOLIC LANGUAGES

General purpose computers operate on and execute instructions that are internally represented in binary notation. The tedium of entering an instruction in binary was recognized very early in the evolution of computers, and this tedium was virtually eliminated by the development of computer languages more suited to human needs rather than the physical requirements of the equipment.

RSX-11M provides two source languages:

1. MACRO-11, and
2. FORTRAN IV.

2.2 RSX-11M LANGUAGE TRANSLATORS

For a language to be of any use, it must have an accompanying translator that converts statements in the source language to the object language of the computer itself. Thus the MACRO-11 Assembler translates MACRO-11 statements into PDP-11 binary and the FORTRAN IV compiler translates FORTRAN IV statements.

In Figures 2-1 through 2-4 interspersed in the text which follows, we have shown the same program, that of adding two numbers to produce a sum, in four language representations:

1. FORTRAN;
2. A MACRO-11 macro (with its macro definition);
3. A sequence of MACRO-11 statements, and
4. ~~Binary (the object language of the PDP-11 itself).~~

The FORTRAN IV source statement in Figure 2-1 contains a complete statement of the problem of producing the sum of two numbers and clearly conveys its intent. Thus, we say it has a high informational content.

```
A=B+C
```

Figure 2-1 FORTRAN Program

The macro SUM below contains the same information but does not convey its intent as naturally. Note that every macro requires a companion macro definition and, with the exception of those macros supplied with the RSX-11M software, the user must write this definition. With FORTRAN IV, the compiler is supplied; the statements written in the language are translated without user intervention. The macro, its definition, call, and expansion are shown in Figure 2-2.

```
MACRO DEFINITION

  .MACRO     SUM RESULT,FIRST,SECOND
  MOV        FIRST,RESULT
  ADD        SECOND,RESULT
  .ENDM

MACRO CALL

  SUM        A,B,C

MACRO EXPANSION

  MOV        B,A
  ADD        C,A
```

Figure 2-2 Macro Definition, Call, and Expansion

Next, shown in Figure 2-3, we have the sum calculated by a sequence of MACRO-11 statements. Note that the code statements are contained in the body of the SUM macro definition, which is not unexpected, since the function of the macro definition is to expand, from the macro call, the instructions necessary to perform the SUM calculation. The principal role of macros is to permit a compact representation of repetitive code sequences that appear in a program. Thus the single line:

```
SUM A,B,C
```

will be expanded into the two instructions needed to carry out the operation wherever it appears in a program.

```
MOV    X,Y
ADD    Z,Y
```

Figure 2-3 MACRO-11 Statements

Finally we have the binary machine representation actually used by the computer to carry out the sum calculation. Remember, it is the function of the FORTRAN IV compiler and the MACRO-11 assembler to convert statements in their respective languages into object form.

```
0001110111110111
1111111111111010
1111111111110110
0110110111110111
1111111111110110
1111111111110000
```

Figure 2-4 Machine Language

The choice of a language involves trading off simplicity of representation with the economics of execution speed and memory space. In general, a FORTRAN IV program will require more memory space and equipment time to execute than an equivalent assembly program. On the other hand, the programmer will be able to master the FORTRAN IV language in less time, and once mastered, produce operational programs more quickly than if he had selected MACRO-11 as his language.

2.3 RSX-11M TASKS

RSX-11M programs written in either MACRO-11 or FORTRAN IV require additional processing after they are translated into object language and before they can be entered into the system and perform their intended function. Programs which have been fashioned into executable units are called tasks, and the program which creates tasks is called the Task Builder.

The Task Builder is a vital element in the overall operation of a software system. The Task Builder exists:

1. To permit one or more output files of the MACRO-11 assembler and/or the FORTRAN IV compiler to be combined into a single task;
2. To create a task with an overlay structure;
3. To attach attributes* to a task, and
4. To store the task image on disk where it can be rapidly retrieved for execution.

Let's consider these functions individually.

2.3.1 Combining Language Translator Output

Often it is necessary or desirable to write a program using both MACRO-11 and FORTRAN IV. Building a task which is composed of output from two different languages is accomplished by writing the code of the different sections in the appropriate language, then submitting the output from the translators to the Task Builder, which constructs them into a single task. The same Task Builder services are used in combining previously written, commonly used subroutines into a task.

* Attributes are discussed in Section 2.3.3.

2.3.2 Creating Overlay Structures

The memory resource of a computer is fixed and finite. If a program will not fit in the available memory it must be split up or overlaid. Overlaid tasks share the fixed memory such that when one part of the program is complete, it is overlaid by another. It is the function of the Task Builder to create, from a set of user overlay specifications, the overlaid task structure.

2.3.3 Applying Attributes To A Task

Certain task characteristics are best applied to the task following language translation since they either have a dynamic quality that would otherwise require repeated translations to alter, or depend on the existence of a hardware feature. For example, a task using the Extended Arithmetic Element (EAE hardware option) requires resources not needed if the option is not going to be used. A task can be modified to run on a system not possessing the option by relinking rather than by a more costly retranslation.

2.3.4 Disk Storage Of Tasks

All tasks in RSX-11M are stored on disk and retrieved by name. This includes any overlays that are part of a given task. In a realtime system it is imperative that requested tasks and task overlays be retrieved with minimum delay. This requires a method for translating the name supplied by the caller to a physical disk location. One of the functions of the Task Builder is to place tasks onto disk, create a name directory entry for it, and link program requests for tasks and overlays into the algorithm for searching and loading the requested program section.

We can summarize the task creation process as a sequence of three steps. The user must:

1. Prepare a program section or sections in a supported language;

2. Submit each program section to the appropriate language translator, and

3. Submit the translated program sections to the Task Builder.

Such tasks are ready to be installed and initiated by the system operator.

2.4 RESPONDING TO REALTIME EVENTS

The success of a realtime system is gauged not only by the speed with which it can respond to realtime events, but also by the number of such events it can effectively satisfy at peak load. RSX-11M provides facilities for minimizing absolute response to a single event and maximizing the number of events it can service at peak load. These facilities are:

1. Multiprogramming;

2. Priority scheduling;

3. Multitasking;

4. Disk based operation;

5. Checkpointing;

6. Power failure restart, and

7. Contingency exits.

2.4.1 Multiprogramming

The natural synchronization requirements of programs, coupled with the disparity between the time required to transfer data in and out of the system and the time required to process it, produce idle time in all system resources. Multiprogramming is an attempt to improve equipment efficiency by building a queue of demands for resources. The demand is achieved by maintaining concurrently in main store more than one task waiting for resource usage. The concurrent tasks are then multiplexed among each other's dead time intervals.

Since in a single processor only one task can have control of the CPU at a time, the apparent concurrency is actually achieved because other system resources, in particular I/O units, can execute in parallel.

By interleaving task activity, the system can better accomplish the dual goals of minimizing absolute response time and maximizing the number of events at peak load.

The multiprogramming of tasks is accomplished by dividing available memory into a number of named, fixed size partitions. Tasks are built to execute out of a specific partition, and all partitions in the system can operate in parallel.

Further, every partition can be split into as many as seven subpartitions, and tasks executing out of the subpartitions may also execute in parallel. The execution of a task in the main partition, however, is mutually exclusive with that of tasks within a main partition's subpartitions. The existence of subpartitions makes it possible to reclaim the space of a large partition and distribute it among several smaller tasks. This is a very real requirement in a system which permits language translation concurrently with realtime processing, since language translators require large partitions and are used intermittently.

We can graphically depict the advantages of multiprogramming by a simple illustration. Programs A, B, and C are being executed in a system without multiprogramming. Program A reads some information from disk, operates on it, and displays a report. Program B performs some computation, displays a message, performs some more computation, and writes the result to disk. Program C performs some computation reads some information from disk, performs some more computation and writes the result to disk.

Figure 2-5 illustrates the sequence in which various operations would be performed when the three programs are executed one after the other. Notice that while any one part of the system, such as a disk drive, is being used, the other parts, such as the CPU, are idle.

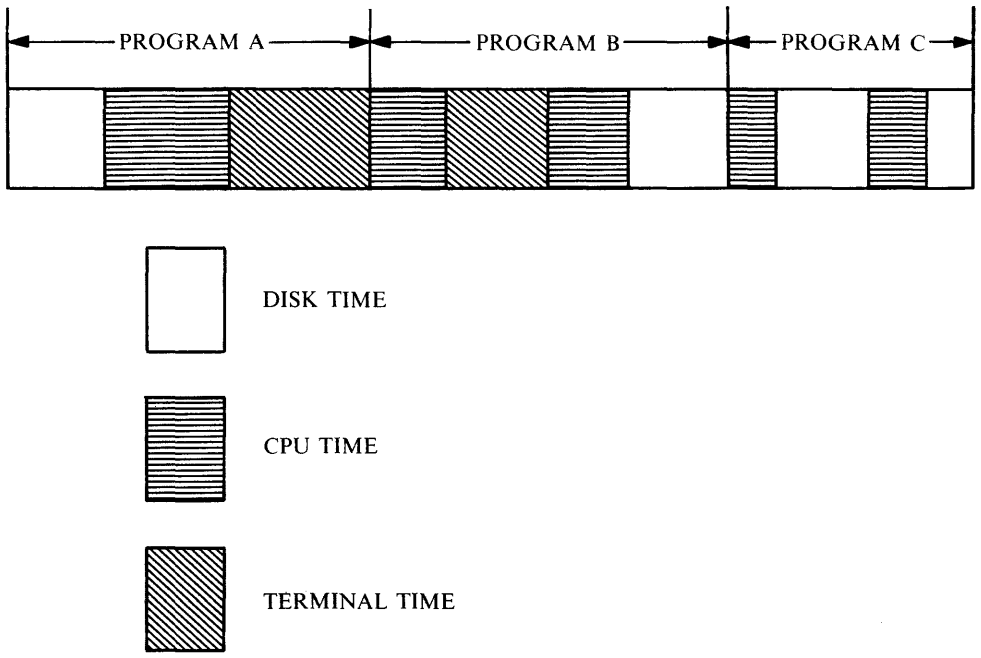

Figure 2-5 Sequential Execution of Programs.

Figure 2-6 shows the sequence in which the same functions might be carried out under RSX-11M. Note that the three resources involved (CPU, disk drive, terminal) are, at times, used simultaneously. Note also that concurrent execution of three programs requires less computer time than sequential execution of the same programs.

2.4.2 Interrupts

Multiprogramming functions only if a method exists for identifying when tasks in the system have become blocked, indicating that a switch to a task ready to use system resources should be initiated. The mechanism used in RSX-11M is the interrupt.

When an interrupt occurs, RSX-11M determines the source of the interrupt and then executes the program that has been specified to service the interrupt.

Interrupts can, as has been noted, be caused by realtime events. For example, if RSX-11M were controlling the testing of resistors as they were manufactured, an interrupt might be generated every time the testing mechanism finished testing a good resistor. A different interrupt might be generated when the resistor being tested was found to be defective. Different programs might be executed to service the two interrupts. In an actual process, the two programs might cause the resistors to be moved to different locations. In any realtime application events have priority relationships, and both the interrupts themselves and the subsequent processing of the interrupt reflect these relationships.

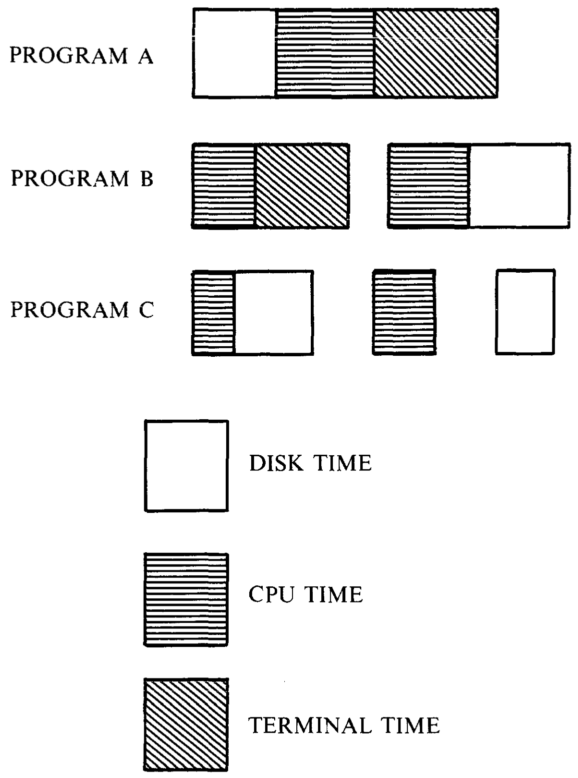

Figure 2-6 Concurrent Execution of Programs.

Interrupting sources have four levels of hardware priority numbered 7, 6, 5, 4, with level seven being the highest. An interrupt at level seven cannot be interrupted by any source until the interrupt is serviced. More than one source may exist at any of the four levels but sources tied to the same level cannot interrupt one another. Lower priority interrupt service routines interrupted by higher sources are automatically returned to when the higher priority request has been completed.

A second level of priority is maintained by the software, and this software priority determines which task in the multiprogramming mix will execute next. The software maintained priority contains 250 levels, and its service discipline is analogous to the four hardware levels.

~~When an interrupt occurs the Executive determines~~ if ~~the interrupt has~~ resulted in the blocking or unblocking of any of the tasks currently in the multiprogramming mix. If blocking or unblocking has occurred, a search is made to determine the highest priority task ready to run, and when this task is located it is given control of the CPU resources.

Thus the task priority is the sole determiner of which ready-to-run task has control of the CPU at any given time. The priority of a task is established by the programmer or system operator.

Using multiprogramming, multitasking (which is discussed next), interrupts, and priority scheduling, RSX-11M effects efficient use of the hardware complex, overlapping equipment resources and responding to events in the order of criticality established by the user.

2.4.3 Multitasking

Multitasking is the multiprogramming of two or more tasks which need to communicate among themselves and synchronize their activities.

In a uniprocessor environment*, some response time dependencies can only be achieved by inducing parallelism. For example, in an airline reservation system a single console request may require several independent file accesses, and to achieve the needed response time these accesses and the their subsequent processing must occur in parallel under the control of a single master task. To exercise this control, the master task must be able to start, stop, and synchronize activity with related subsidiary tasks. In effect, a multitasking capability gives every user task the ability to become an executive and thus extends the benefits of parallel execution to the user tasks. RSX-11M provides the services necessary to support multitasking.

2.4.4 Disk-Based Operation

Except for rare, dedicated applications, the total code in a system always exceeds the available main memory. A disk-based system uses random access peripherals both as an extension of executive main memory and as the principal data interchange medium. The use of disk as the system data storage medium provides the base for program development facilities, a common file system, checkpointing, and rapid initiation of tasks, The Task Builder makes it possible for the user to build overlaid tasks and call these overlays from disk. The total effect is to extend significantly the achievable peak load while still maintaining system response time requirements.

2.4.5 Checkpointing

A task currently active in a partition, but of a lower priority than another task requesting the partition, can be pre-empted, rolled-out to disk, and later, after the higher priority task has completed its execution, be rolled-in and restored to active execution at the point where it was previously pre-empted. Checkpointing, which is optional on a per task basis, is another method of making it possible to load the processor with as much work as it can possibly absorb, and still meet its realtime commitments.

2.4.6 Power Failure Restart

Power failure restart is the ability of a system to smooth out intermittent short-term power fluctuations with no apparent loss of service and without losing data, all the while maintaining logical consistency within the system itself and the application tasks. Power failure affects absolute response time and peak load capacity differently from the facilities previously discussed, since it applies to the aggregate system performance rather than increasing performance when the system is actually in operation. A system is not performing when it is shut down, and if the Executive can reduce the shutdown periods due to power failure restart, aggregate performance is increased.

* In multiprocessors, intra-task parallelism is needed to exploit the availability of more than one CPU.

In systems which do not support power failure restart, service disruptions due to power failure are often lengthy, resulting in reduced equipment effectiveness. Furthermore, redundancy (a second processor) does not increase system uptime in the power failure situation. Either the system itself provides for recovery, or the user, if his application requires it, must purchase auxiliary power, a rather expensive alternative.

Under RSX-11M power failure restart functions in four phases:

1. When power begins to fail, the processor traps to the Executive where volatile register contents are stored in non-volatile memory, gracefully bringing system operations to a halt.

2. When power is restored, the Executive again receives control and restores the previously preserved state of the system.

3. The Executive then determines if any user level tasks have requested notification of power failure. The contingency exit mechanism discussed in the next section is used for notification.

4. The Executive then schedules all device drivers that were active at the time the power failure occurred at their powerfail entry point. Drivers have the option of being scheduled:

    a. On power failure, or

    b. On power failure when the driver has an outstanding I/O order.

    These drivers can then, if required, make those restorations of state (like repeating of I/O) they deem necessary.

This approach is quite efficient because the repeating of I/O is placed nearest the source most likely to know how to make the restoration. Basically, RSX-11M takes a position which assumes it does not know more about an application's requirements than the code expressly written to service the application.

2.4.7 Contingency Exits

Subroutines automatically entered as the result of an unanticipated synchronous condition (for example, an attempt to execute an illegal instruction) or as the result of an asynchronous condition anticipated or unanticipated (for example, an I/O termination) are called contingency exit routines, and the conditions which triggered their entry are called contingency exit conditions.

Synchronous exit conditions are those which are the result of a specific instruction encountering an unanticipated event; if the code sequence up to and including the instruction is repeated under identical conditions, the same unanticipated event occurs. As cited above, an attempt to execute an illegal instruction has these characteristics.

By contrast, asynchronous conditions are not associated with a specific instruction's execution and the point of exit for this condition in the code is unpredictable; an I/O termination has these characteristics.

Contingency exits improve both the structural design of a program and the response efficiency of a task. A contingency exit capability contributes meaningfully to the peak load a given task can sustain and, as such, the peak load the system can sustain.

2.4.8 Responding To Realtime Events - A Summary

Each of the seven features:

1. Multiprogramming;
2. Priority Scheduling;
3. Multitasking;
4. Disk based operation;
5. Checkpointing;
6. Power failure restart, and
7. Contingency exits

exists to improve the cost performance of the system by increasing peak load capacity while maintaining the ability to meet absolute realtime deadlines. They also contribute to the system's capability for servicing sophisticated applications and to make use of the system's resources when realtime demands are relatively inactive.

2.5 MULTILEVEL ACCESS TO INPUT/OUTPUT DEVICES

Input/output servicing has been traditionally among the more difficult aspects of applications programming. From the user's point of view, I/O processing is a necessary evil since the application itself, though driven by data, is not particularly concerned with the computer interfacing subtleties required in data acquisition. And any service supplied by the system to ease the I/O programming burden can contribute substantially to reducing the time required to produce operational applications. RSX-11M eases the I/O programming requirements by supplying comprehensive device independence.

Device independence, in its most general form in RSX-11M, consists of six entities:

1. An I/O Language;
2. A Common File System;
3. A Record I/O Package;
4. Device Drivers;
5. Logical Unit Numbers (LUNs), and
6. The Physical Unit Directory Group.

The goal of device independence is to provide a package of services that will enable the construction of a task that can substitute devices at runtime without necessitating a single change in the task's code. Many systems claim device independence when they can substitute at runtime devices of similar characteristics, for example, a terminal for a line printer. Both these devices output a single record at a time. Device independence which permits the substitution of random access devices involves a quantum jump in complexity and user flexibility. Many systems claiming device independence do not permit substitution of random access devices for unit record devices and thus cannot match the I/O flexibility of RSX-11M which does permit such substitutions.

2.5.1 The RSX-11M I/O Language

The creation of user-level device independence in RSX-11M begins with the I/O language. The I/O language includes all the I/O service requests necessary to logically interface with devices. The requests make files available (OPEN), cause data to be transmitted to and from the task to devices (GET, PUT), and disconnect files from the task (CLOSE).

The device independent I/O language specified in RSX-11M communicates with the File System and the Record I/O Package.

2.5.2 The Common File System

A file system is the collection of system services which permits a user to view his I/O as a transaction between his program and a named, protected collection of records.

The RSX-11M File System manages public storage. It finds, opens, and closes files, and it maintains a directory of named files existing on the systems data volumes. It provides the mechanism by which a file destined for a printer can be re-directed to a named file in random access storage. Without such a file system to maintain the integrity of public store and thereby not permitting a named storage area on random access devices to be treated as a record oriented device, device independence reduces to a substitution of one record oriented device for another. This substitution of similar device types severely impacts the benefits accruing to both users and equipment when contrasted with complete device independence provided in RSX-11M.

2.5.3 The Record I/O Package

Record I/O manages buffering, blocking, and device control functions. Using the I/O language, the user task communicates its I/O requirements to the Record I/O package. Record I/O, based on the characteristics of the device to which the data is being sent, builds the I/O requests it will transmit to a device driver, The entire process just described is transparent to the user level task.

2.5.4 Device Drivers

Every physical device in the system has an associated device driver. A driver accepts requests from the File System, Record I/O, and user tasks*. Drivers perform the physical functions implied in the requests issued to them, and perform error recovery when needed.

The structure of the RSX-11M I/O system makes it easy for a user to build drivers unique to his application.

2.5.5 Logical Unit Numbers and Physical Unit Directories

Interchanging devices at runtime implies a method for communicating these changes to the system. In RSX-11M two tables provide this communication:

The Logical Unit Table, and

The Physical Unit Directory Group.

In his task, a programmer refers to devices by logical unit numbers (LUNs). At task installation, or dynamically at runtime, the task connects a LUN to a device driver and a physical unit number. Now the web of device independence is complete. We can illustrate it by an example.

Suppose a task which normally places its output on a printer desires to substitute a disk file.

At runtime, the task will connect its LUN for the printer to a disk driver and supply a filename. When the device is opened, the File System will locate the file and note it in an appropriate table. PUTs issued to the file flow through the Record I/O Package which will block the records, append appropriate control information, and request the disk driver to transmit it to the random access device. Later this file can be read and printed by a file utility.

Two facilities, higher level languages and comprehensive I/O services, are the principal means by which a user simplifies his programming task. Most 16K-based systems offer both FORTRAN IV and a macro assembler, but few offer the I/O service capabilities of RSX-11M; these services are usually found only in systems requiring a much larger minimum configuration.

2.6 SHARING OF COMMON ROUTINES

If a system is designed to support many levels of multiprogramming, the possibility that many tasks will need the same code sequences is very high. A system usually selects one of two alternatives for incorporating common, simultaneously required code sequences:

1. Include the common code in every task which needs it, or

2. Devise a mechanism to permit all tasks to share a single physical copy of the code.

* User tasks may bypass the file System and Record I/O, but in so doing forfeit the system's capability to provide device independence.

The first alternative is simple to implement and, if many tasks will need the same code simultaneously, then many duplicate copies will exist in physical memory, as well as on peripheral memory. Avoiding this duplication is the aim of alternative number two.

The mechanism for code sharing in RSX-11M is shareable libraries. A shareable library consists of subroutines that are coded such that they may be interrupted asynchronously, service another request for either the current or a different task, then resume later at the point of interruption.

Shareable libraries under RSX-11M are constructed at system generation and installed into an operational system as needed.

2.7 SUPPORT OF USER PREPARED RE-ENTRANT ROUTINES

As stated in the previous section on shareable libraries, routines in the library must be so coded that if the routine is being executed on a given interrupt level and a higher level interrupt occurs, execution of the routine can be suspended and resumed after the higher priority interrupt has been serviced.

Users who observe the requirements for coding re-entrant routines can include their own shareable libraries into an RSX-11M system. All RSX-11M Executive macros have a format which simplifies their use when the user is creating re-entrant routines.

2.8 USER CONTROL OF EXTERNAL TASK SCHEDULING

Priorities are indicators supplied by users to control the internal scheduling of tasks in active competition for processor resources. But user tasks often require another level of scheduling. For example, a task may want to run every two minutes. This type of scheduling request is called an external schedule request. A user may request the Executive to run his task:

A delta time from now;

A delta time from clock unit synchronization;

An absolute time of day, or

Immediately.

All of these time options are available with or without periodic rescheduling. In addition, RSX-11M supports an unlimited number of programmed timers for each task in the system. Using a request to the Executive, the user task can create a timer and place a value in it. The Executive will decrement the timer at regular intervals until it reaches zero at which time it will generate an asynchronous system trap passing control to the task at a prespecified task address.

2.9 AVAILABILITY OF SYSTEM ROUTINES FOR PERFORMING DATA CONVERSIONS, ARITHMETIC CONVERSIONS, AND FUNCTIONAL CALCULATIONS

Part of the shareable library contains subroutines that are often needed by user programs. Some of these subroutines change the representation of data from one form to another. Others compute functions of variables such as sines, logarithms, and square roots. Still others perform arithmetic operations on data in various formats.

These subroutines reside on disk and must be made part of a task or included in a shareable library before they can be executed. The Task Builder can make any of these subroutines part of any task.

2.10 OPERATOR CONTROL OF RSX-11M

Use of RSX-11M assumes a human operator directing the system from an operator's terminal. RSX-11M will support any number of simultaneously active operator terminals. Access to, and control of the system is via an operator's language which directs the Monitor Console Routine (MCR) to carry out the specified functions. The language initiates services designed to:

1. Give the operator full access rights to the system;
2. Provide for emergency on-line software-fault servicing;
3. Provide services for initializing and controlling both the system and user tasks, and
4. Be simple to use and simple to modify or enhance.

2.11 SYSTEM SUPPORT OF USER CREATED MACRO LIBRARIES

Most computer installations discover that there are some sequences of assembler-language instructions that are used over and over again. The Macro Assembler makes it possible to condense each such sequence into a single instruction called a macro instruction. When the Macro Assembler encounters a macro instruction, it processes a prespecified sequence of assembler-language statements. These prespecified statement sequences reside in a Macro Library.

In some cases, it may be possible to use macro instructions to create a programming language tailored to the needs of specific applications. By writing several macro instructions with descriptive names, such as ORDER, TEST, or FLOW, it is possible to greatly simplify the programming effort.

2.12 COMPREHENSIVE ERROR RECOVERY

RSX-11M provides error detection and recovery services during language translation, task building, and execution. Errors detected by a language translator or the Task Builder are displayed at the completion of a translation or link, and can be used for subsequent correction of the program.

During program execution, the RSX-11M Executive makes numerous checks on the validity of requests, retries I/O operations, and restarts the system on power failures. In addition to rip-stop error detection, RSX-11M provides core dump routines and an online interactive debugger to assist the programmer in quickly resolving the cause of a program fault. Of course, other tasks independent of the faulting task continue normal operation.

2.13 PROTECTION OF PROGRAMS AND DATA

The RSX-11M makes a number of checks to ensure that the integrity of each task is preserved. These involve the verification of any passed parameters that, if in error, would cause either the system or another user task to fault. Checks are made in the file system to prevent a task from reading, writing, or deleting files for which it does not have permission to use as requested. Checks are repeatedly made to guarantee the consistency of critical system tables. If the hardware memory management option is part of the users system, protection among tasks and between the system and tasks is absolute; thus, unintended access to private areas is blocked by the hardware.

2.14 TAILORING A SYSTEM TO USER-LOCAL REQUIREMENTS

Users have varying requirements for the total services supplied by RSX-11M. To make it possible for each user to meet his local needs rather than to fulfill the global needs of all users, RSX-11M provides a System Generation program (SYSGEN).

System Generation is the process by which a collection of system services are tailored to meet the local physical constraints and performance requirements of the end-user.

RSX-11M consists of a set of independent program segments that can be linked so as to eliminate services not required at a given installation, thus improving system cost-performance. The user, for example, may eliminate certain Executive services like the panic dump routine, or vary the size of the dynamic storage region used by the system.

RSX-11M is delivered on disk, and the system generation procedure is employed, under user direction, to create an operable, locally tailored system.

2.15 NON-DISRUPTIVE GROWTH

RSX-11M provides two paths for non-disruptive growth. First, even though the system in its minimum configuration is quite modest, it is packed with facilities rarely found in 16K systems. Indeed, RSX-11M is classed as a small system because of its minimum configuration. Viewed from the services it provides, it is in the midi-system class. Growing within RSX-11M is greatly simplified by the availability of the system generation procedure discussed in Section 2.14. Through SYSGEN the user application may grow by peripherals or memory or by moving up to a more powerful member of the PDP-11 family of processors.

Second, RSX-11M is a proper subset of DIGITAL's widely accepted RSX-11D Realtime Operating System. RSX-11M Applications can move to an RSX-11D based system and be assured of file portability, common operator commands, and growth without reprogramming. In fact, RSX-11M's compatibility with RSX-11D is at the binary level thus, translated RSX-

11M programs (recompilation or re-assembly is not required) will run on RSX-11D following a re-link of the RSX-11M modules by the RSX-11D Task Builder.

Finally, compatibility is further enhanced by the strict programming standards used in the creation of all RSX-11M software. By following these standards, which are detailed in an appendix of the MACRO-11 Reference Manual*, the user can interface simply with DEC-produced software and be confident that programs will be compatible with all RSX-11M and RSX-11D system programs.

* DIGITAL manual number DEC-11-OXDMA

CHAPTER 3

THE ORGANIZATION OF RSX-11M

This chapter presents no new concepts or facilities to those already discussed in Chapter 2. This chapter is intended as a structural overview of the system for those readers whose interest or decision-making needs require further insight into how the system accomplishes the user services described in Chapter 2. Readers primarily interested in only the facilities of RSX-11M may omit reading this chapter.

The system delivered on the distribution medium, and from which the user site tailors his system, is identical in structure to the system the user will SYSGEN. The only differences which will exist are determined by the SYSGEN selection of services.

3.1 ORGANIZATION OF THE 8K EXECUTIVE

Figure 3-1* shows the memory layout of the basic 8K Executive. The descriptions of the individual regions include possible expansion through SYSGEN parameters.

3.1.1 Executive Component Descriptors

Trap Vectors:

> This region contains the hardware trap and interrupt vectors and requires 128 words. This region is expandable at SYSGEN to a maximum of 256 words.

System Stack:

> Used for nesting interrupts and internal calls made by the Executive. Forty words are required.

System Common Data:

> Contains pointers filled in by SYSGEN.

System Tables:

> Contains the data used to control system operation. Included are:
>
> > Partition Descriptions;
> >
> > The System Task Directory, and
> >
> > Device Tables.
>
> The total size of the table region is established by SYSGEN configuration selections.

* The figure is schematic and not to scale.

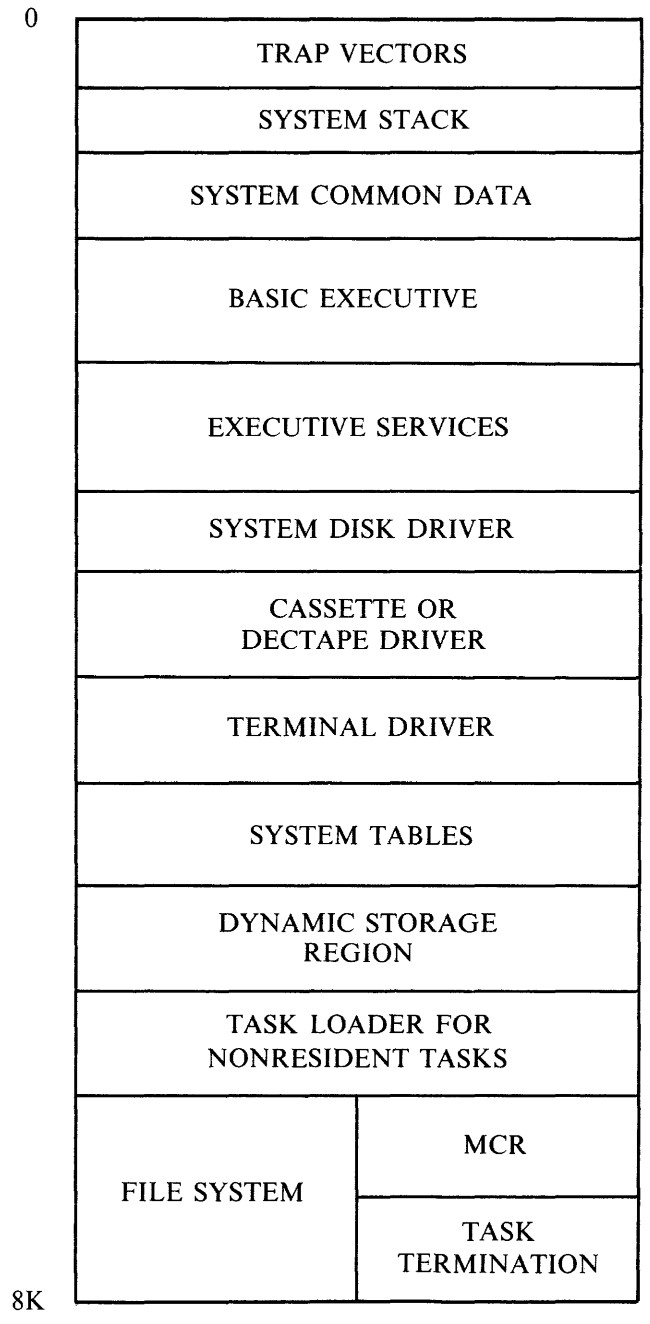

Figure 3-1 The Basic Executive

Dynamic Storage Region:

The Executive has continuing needs for temporary storage. Such storage is acquired, used, and returned to the available pool. If a given Executive service requests dynamic storage, and it is unavailable, the Executive will inform the user task, which usually waits for some storage to become available. The size of this region is important, for, if it is too small, wait periods will be induced; if it is too large, system effectiveness is lowered, since fewer tasks can fit in memory. The size of the region is a SYSGEN parameter.

The Basic Executive:

The Basic Executive contains the code that provides the facilities discussed in Chapter 2. All these facilities are included in the 8K Executive.

Executive Services:

This region contains the programs which respond to the directives issued by users to request Executive services. These programs make extensive use of the Basic Executive.

Device Drivers:

Three drivers are included in the basic 8K Executive:

Disk;

Cassette or Dectape, and

Terminal.

These are multi-unit drivers that can service any number of their respective devices.

In general, systems which grow beyond 8K will do so because of the presence of additional drivers. Drivers are included in the system at SYSGEN.

Task Loader For Nonresident Tasks:

This loader is a task and operates out of its own partition; thus, it can run in parallel with system and user task operation.

The loader, which is device independent:

1. Loads tasks on initial load requests;
2. Writes checkpointable tasks to disk when required, and
3. Returns previously checkpointed tasks to active competition for processor resources.

File System, Monitor Console Routine (MCR) and Task Termination (TKTN):

These three routines function as tasks. The file system occupies the main partition; MCR, and TKTN operate out of subpartitions. Whenever MCR or TKTN need to run, the file system is checkpointed. MCR and TKTN can run in parallel. Of course, the main partition itself operates in parallel with both the system and user tasks.

Panic Dump and Crash Modules:

These two routines respond to system software failures, providing core dumps and selective analysis. They are not included (or shown) in the basic system (8K) but are mentioned because of their fundamental importance in error analysis. Most program development systems (as opposed to dedicated online systems) will likely include these routines.

In a 16K system with an 8K Executive, the remaining 8K is available for user task partitions.

3.2 THE ORGANIZATION OF THE SYSTEM DISK

All data, whether data as such or programs to be loaded and executed, exist as named files on disk managed by the RSX-11M file system. The entire disk storage space is apportioned dynamically as files are created or deleted. Thus programs be they in source, object, or task images exist as named storage regions in the public storage space (disks) and are retrieved by name. It is one of the functions of the file system to locate files when presented with their names.

Task images on disk are structured to minimize the time required to load them. or to load overlays within a task as they are needed.

The tree structure of tasks is shown below in Figure 3-2.

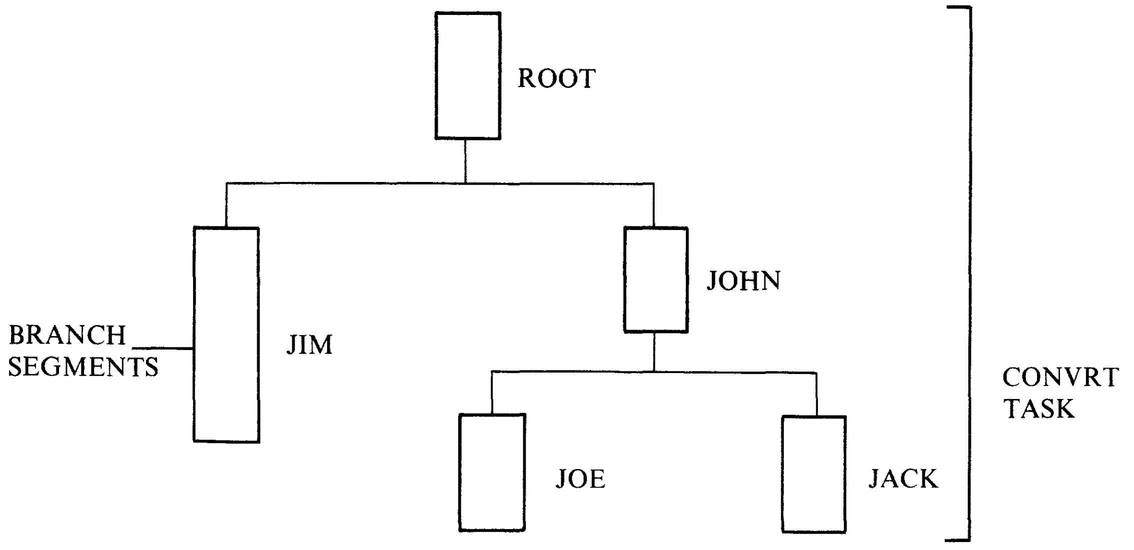

Figure 3-2 Overlay Task Structure

The root segment is always resident; the branch segments, JIM, JOHN, JOE, and JACK can overlay one another during task execution. The tree structure is created by user specifications submitted to the Task Builder.

Using the file system, the Task Builder creates a disk image of the task CONVERT; this image is shown in Figure 3-3.

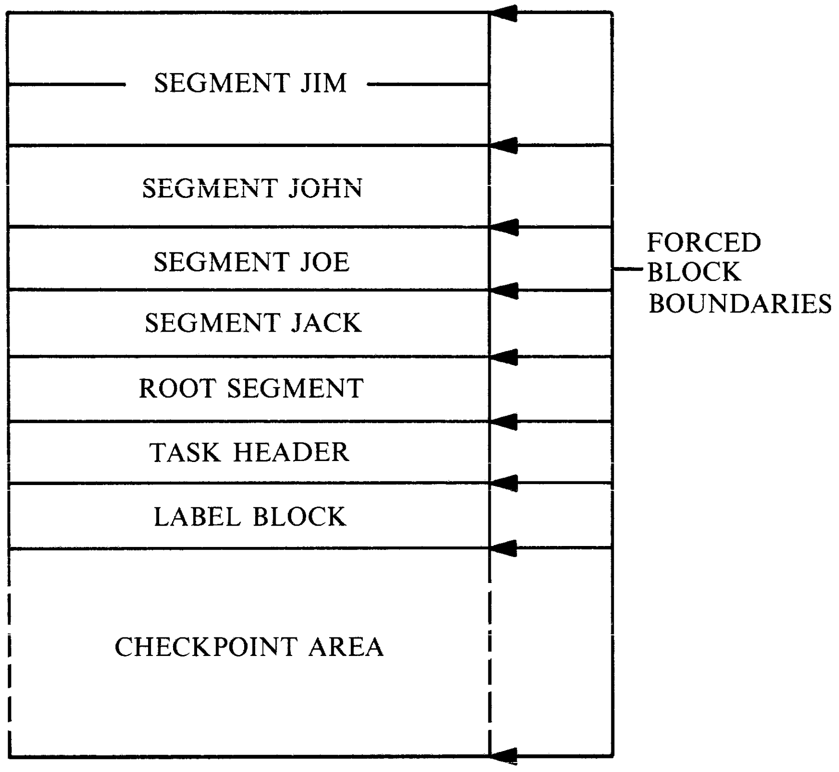

Figure 3-3 Task Disk Image

The checkpoint area exists only for tasks declared checkpointable and is equal in size to the partition or subpartition from which the task will execute.

When a task is installed in the system a resident task descriptor is generated into which the task name and its physical disk address are stored. The segments within a task are located on disk block boundaries, an arrangement which greatly speeds up the process of loading overlays when they are requested. Given that the physical disk address of a task is preserved in memory we can note two efficiencies of our structure:

1. The root segment, including the task header, is loaded with a single access. No lookups are required for task loading, yet the file can be retrieved by name through the file system.

2. Each segment of a tree structured overlaid task is also retrieved with a single access. Since the entire task is in a contiguous file, and the overlay loader can compute the block number of any segment, retrieval is as fast as the device itself

permits. And since the file is protected, no checks are required to validate header information, thus further reducing the loading and initiation overhead to the absolute minimum.

In realtime systems, response time is more than a psychological variable interpreted by the interacting human; it's a critical parameter that often determines success or failure quantitatively and absolutely. Every effort has been expended to ensure in the design of the task's organization on disk instantaneous response, constrained only by the application requirements of the user and the limitations of hardware.

INDEX

Introduction to the RSX-11M
Realtime Executive
DEC-11-OMIEA-A-D

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form

Did you find errors in this manual? If so, specify by page.

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

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

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

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name ______________________ Date ______________

Organization ______________________

Street ______________________

City ______________ State __________ Zip Code or Country __________

If you require a written reply, please check here. ☐

Fold Here

Do Not Tear - Fold Here and Staple

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

```
Software Communications
P. O. Box F
Maynard, Massachusetts  01754
```

digital

digital equipment corporation

Printed in U.S.A.