all input/output operations. Any attempt by non-Supervisor programs to initiate input/output also results in a Program Fault.

Hardware Faults caused by hardware malfunctions. Under this heading are memory parity errors, channel and device failures and similar difficulties which may be directly detected by the hardware.

Request for Supervisor Service caused by the execution in a program of an instruction known as the Supervisor Call. Since the Supervisor allows a program to have access to none of the facilities of the hardware except the arithmetic and control units, provision must be made for programs to communicate their desires for a wider range of functions. This mechanism is the Supervisor Call. Typical requests for service are those to read and write logical records on a data set.

While the Hardware Interface responds to physical occurrences, the Interrupt Activation Routine concerns itself only with the logical consequences of these occurrences. Two queues, the Interrupt Queue, and the Priority Queue relate physical traps to logical interrupts.

The Interrupt Queue consists of one element for each logical entity for which it is desired to define an interrupt. Certain entries correspond to physical devices, e.g., a card reader; others correspond to classes of devices, e.g., tapes multiplexed on a channel; and others to conditions, e.g., Timer overflow or Request for Supervisor Service.

Each element is composed of three parts:

Device Identification indicating the logical class causing the interrupt.

Interrupt Status which specifies the nature of the interrupt.

Interrupt Director which indicates the routine, called an Interrupt Routine, responsible for servicing the logical interrupt.

By changing the Interrupt Director, the response to a logical interrupt may be dynamically altered by components of the Supervisor.

Priority among interrupts is established with the Priority Queue. The Interrupt Activation Routine scans the Priority Queue in a fixed sequence. The Priority Queue indicates the order in which logical interrupts are to be processed. By altering the priority assignments, the Supervisor may be tuned to process certain classes of interrupts more rapidly than others.

A logical interrupt is serviced by executing a series of routines beginning with that one indicated by the Interrupt Director. The interrupt is in progress, or active, until control is returned to the Interrupt Activation Routine. While only one Interrupt can be active at any time, physical traps may easily be generated at a rate faster than the Supervisor can dispose of them. These traps are mapped into their corresponding logical interrupts and stacked in the Interrupt Queue where they await servicing when the current interrupt has been completed.

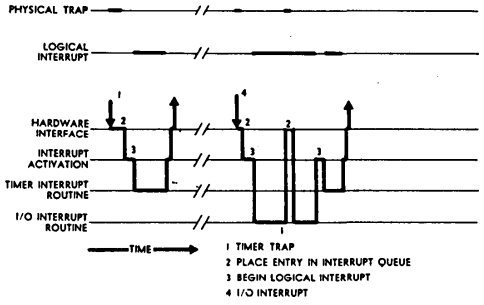

Figure 1 – Interrupt processor *typical interrupt sequence*

Input/Output processor

An Input/Output Processor is a set of routines which provides the operating system with the ability to communicate with classes of hardware devices.

The Processors schedule requests for input/output operations, initiate the physical data transfer, validate the correct transmission of data, and, when necessary, re-try operations found to be in error. The Processor is also responsible for notifying the ultimate requestor of the status of the operation upon its completion.

These activities may be grouped under the heading of:

Scheduling
Initiation
Physical Post Processing
Logical Post Processing

An Input/Output Processor generally consists of the following:

Request Queue
Schedular
Activation Routine
Input/Output Interrupt Routine
Associated Schedular Routine

Within a given time period, requests for input/output activities may be generated faster than the operations can be initiated. Requests for pending operations are placed in Request Queues by routines known as Schedulars, where they await servicing by an Activation Routine.

An entry in a Request Queue describes completely the nature of the operation to be performed. This description contains the specifications of the device

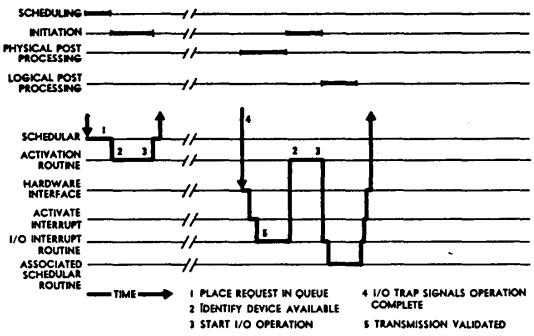

Figure 2 – Input/output processor *typical I/O sequence*

upon which the operation is to take place, the operation itself, and the identification of the routine to notify when the operation is logically complete.

The schedular is responsible for entering a request in the proper sequence into a Request Queue. Once the entry has been made in the queue, an attempt is made to start any pending operations for the class of device for which the requests are queued.

The Activation Routine scans the Request Queue, identifies devices available to service the request, and initiates all requests which may be started. When a request is started, an Interrupt Director for the logical interrupt expected is entered into the Interrupt Queue. This Interrupt Director defines the logical interrupt associated with the operation.

Upon completion of a request, an I/O trap is generated and transformed into a logical interrupt by the Hardware Interface. The operation must now be checked for correctness and the ultimate requestor, the routine which scheduled the operation, must be informed of its completion.

The Input/Output Interrupt routine specified by the Interrupt Director is responsible for validating the transmission and requesting retransmission as necessary. Once the physical operation has been accepted, the routine named in the Request Queue, known as the Associated Schedular Routine, is notified. This routine is associated with the request for operation; invoking it signals that the operation scheduled is logically, as well as physically, complete. As a part of the post processing path, pending requests are examined in the Request Queue and an attempt is made to start them. In this way the physical facilities associated with the class of request are driven at as high a rate as possible.

Timer administrator

As can be seen in Figure 3 by the interaction between Schedular, its Activation Routine, Interrupt Routine and Associated Schedular Routine, the Supervisor is basically asynchronous by nature. That is, there generally is little relationship between the order in which operations are started and that in which they terminate. Further, an operation does not have to terminate before other operations may be initiated.

Moreover, there are occasions when it is not possible to initiate an operation, but when it is desirable to be about other tasks rather than simply to wait. For this reason, the Supervisor provides itself with the ability to return to a given activity after a specified time period expires.

The Timer Administrator consists of a set of routines collectively responsible for the maintenance of physical and logical timers available for this purpose.

Logical Timers are created by the Timer Schedular and consist of entries in the Timer Queue. Each entry contains a time field and the identification of a routine, called a Timer Routine, to invoke when the time expires.

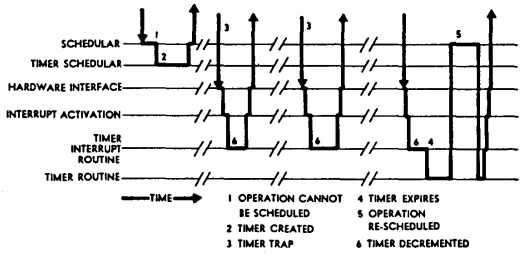

Figure 3 – Timer administrator *typical time delay sequence*

Periodically, as part of the processing of a logical timer interrupt, the timers in the queue are decremented, and upon detection of an expired Timer, control is given to a Timer Routine.

Programs

The Supervisor exists for one purpose only; to provide services to entities known as programs. A program performs some logically complete function and may not invoke or directly communicate with other programs. Communication with other programs is achieved through data sets which may be read or written by the programs. Additionally, a program is characterized by the existence of a Program Definition Area, available to the Supervisor only, which describes total environment for each program.

The primary direct service that the Supervisor provides for programs is that of Input/Output. To protect the integrity of itself, and of other programs which may be concurrently occupying Execution Storage, the Supervisor must exercise rigid control over the

total environment. Not only are programs isolated from one another by inviolable storage protect measures, but the nature and scope of Input/Output operations are limited to prevent one program from dominating the facilities of the operating system to the detriment of others.

Requests for operations forbidden to programs are made through the mechanism of the Supervisor Call, generally an instruction which generates a physical trap. Given the specification of the operation desired in the program, this specification is transformed into the corresponding action in the Supervisor as shown in Figure 4.

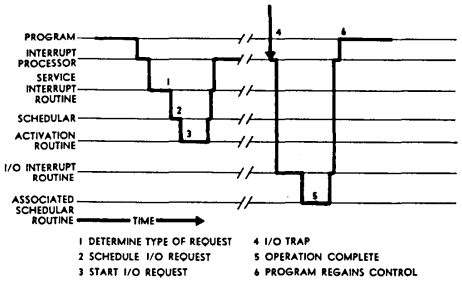

Figure 4 – Programs *typical supervisor call*

Program storage administrator

In the course of the execution of a program, the program occupies a portion of a larger space called the Program Storage. Program Storage contains the Program Definition Areas, Instruction Areas, Data Areas, and Buffers for all programs actively engaging the attention of the operating system.

That section of the Program Storage in which a program resides while making use of the arithmetic and control sections of the processor is known as Execution Storage. The remainder, that part which provides passive storage for the program, is known as Extended Storage.

At any given time, the total requirements for the Execution Storage may be expected far to exceed its availability. The Program Storage Administrator is responsible for the orderly transition of a program from Extended Storage to Execution Storage and back again.

The relationship between Extended and Execution Storage is summarized in the Execution Storage Availability Table, which relates a section, or page, of Execution Storage to that program requiring it. In the Program Definition Area of each program the Program Page Table shows for any given instant the Execution and Extended Storage assignments for the program. Programs not in a position to utilize Execution Storage are placed on Extended Storage, and the free Execution Storage pages are assigned to other programs.

In order to retain an acceptable responsiveness to programs attached to devices such as remote inquiry keyboards, the Timer Administrator periodically interrupts the execution of programs active in Execution Storage. These programs are forced onto Extended Storage while other programs are activated. This swapping process, as shown in Figure 5, insures that each program will have periodic access to the facilities of the central processor.

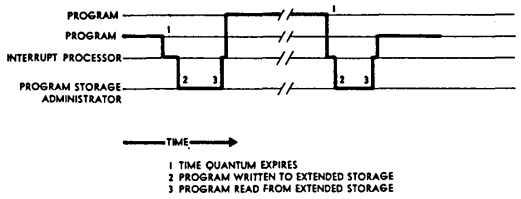

Figure 5 – Program storage administrator *typical swapping storage*

Facilities administrator

Before a program can be placed in execution, all of its requirements for facilities of the operating system must be satisfied. These facilities include catalogued data sets, such as would normally be found on bulk storage; temporary or permanent tape files; and printers or card readers. The Facilities Administrator attends to the task of satisfying requirements for given facilities. The sum of these requirements is contained in the Requirements List portion of the Program Definition Area. Once all requirements are assigned, the program may be placed in the program state.

Program administrator

The operating environment exists in one of three mutually exclusive states:

The Wait State
The Supervisor State
The Program State

The Wait State is the ground state of the operating system. It is entered whenever there is no load on the system. From this state the Supervisor State is entered whenever it is discovered that an activity has been completed or must be initiated.

It is in the Supervisor State that all interrupts are serviced, Input/Output is initiated, and response made to Supervisor Calls.

When all pending operations in the Supervisor State have been initiated, and no further processing can take place, either the Wait State is re-entered, if no program can be placed in execution, or the Program State is entered. The operating system is in the Program State as long as a program has effective control over the central processor. When this control is relinquished, either voluntarily as upon execution of a Supervisor Call, or involuntarily because of the expiration of a time quantum associated with the program or other physical trap, the system reverts to the Supervisor State. This is shown graphically in Figure 6.

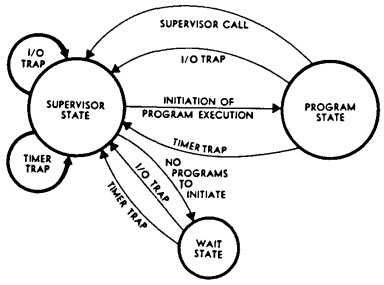

Figure 6 – Program administrator *system state diagram*

Separate from the states of the operating system are the states in which a program may find itself throughout the time allotted to it for execution. These states are governed by the attachment of the Program Definition Area to one of the following queues called, collectively, Program Queues:

Program Definition Queue
Program Queue
Execution Queue
Service Queue
Service Pending Queue
Service Delay Queue

If the Interrupt Processor is the heart of the Supervisor, the Program Administrator is the heart of the operating system, for it governs the flow between these states and controls the priorities among programs.

The Program Definition Area of a program contains descriptions of the environment of the program. The area is divided into:

The Program Descriptor which defines the number of files attached to the program, the location and extent of the program, the current location counter and accounting information.

The Machine Environment which contains the volitile machine registers and indicators. This environment is saved by the Program Administrator when the program relinquishes control of the central processor, and is restored when the program is in position to regain control.

The Requirements List which defines all data sets and physical facilities assigned to the program.

The File Control Blocks for all files attached to the program. Each file control block specifies the properties of the file and contains information relating to the current file positioning.

The Program Page Table which relates the location of pages of the program to both Execution and Extended Storage.

The Program Symbol Table which relates symbolic labels to locations within the program.

While the Program Definition Area is being constructed, and the appropriate facilities assigned to a program, the program is in the Program Definition State. (See Figure 7.) When all facilities are assigned, with the exception of Execution Storage, the program moves to the Program State. In this state the program is in contention for available Execution Storage. Once sufficient Execution Storage is made available for a program, it enters the Execution State. Programs in this state are either using the central processor or are immediately able to do so.

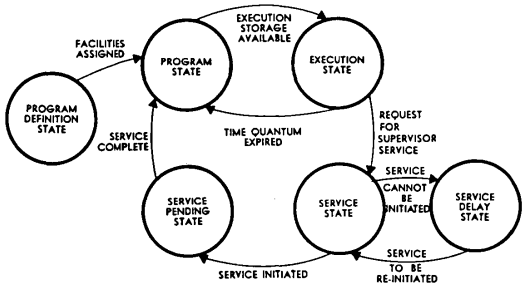

Figure 7 – Program administrator *program state diagram*

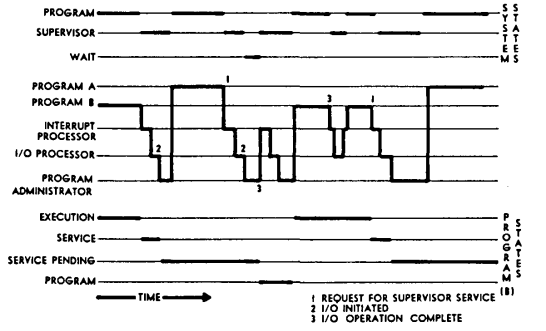

Figure 8 – Program administrator *typical program sequence*

When a program makes a request for a supervisor service, it enters the Service State where it remains until the service requested can be initiated. If the service can be initiated, the Service Pending State is entered until the service is completed. If the service cannot be started, the Service Delayed State is entered. Periodically, the program is placed in the Service State and the request re-initiated. Upon completion of the requested supervisor service, the program enters the Program State where it is again in a position to regain control of the central processor.

In order to minimize switching between the Supervisor and Program States of the operating system, all activities in the Supervisor are brought as close to completion as possible before attempting to enter the program state. If no further Supervisor activity can be attended to, a suitable program is chosen, the machine environment for this program is established, a time slice assigned to the program so that the Supervisor can ultimately regain control through a timer interrupt, and control is passed to the location counter indicated in the Program Descriptor.

SUMMARY

As systems become increasingly complex, it is increasingly important to be able to isolate the logical functions that comprise the system. The preceding paper is an attempt to specify the major functions required for the implementation of a time-shared operating system. By defining a reasonably formal structure for each function, independent of a specific implementation, a generalized machine-independent design emerges. Prototype versions utilizing this design have been implemented for two different manufacturers' machines.