J. R. Calva

# PCOS: A Process Control Extension to Operating System/360

**Abstract:** This paper discusses the design and implementation of an extension to IBM Operating System/360 that is called PCOS and is intended specifically for use in "real-time" control of IBM manufacturing processes and testing. The most important part of PCOS is a Real-Time Control Program (RTCP) which is initiated at system start-up time as a "never-ending" task. The RTCP controls the execution of application programs and permits system response to requests for their execution within a time on the order of 100 msec. Other contributions are an "express path" for handling input/output operations, an "interpartition communication" program that provides common core storage for use during the execution of application programs, and an appendage to the OS/360 supervisor program to serve a specially designed high-speed multiplexor called the "transmission control unit."

## Introduction

The Process Control Operating System (PCOS) is an extension of IBM OS/360 that was developed specifically for use in the company's manufacturing plants. PCOS resides in the central computer of the satellite computer network discussed by Stuehler.[1] In this network a number of satellite computers are connected to a central computer through a high-speed multiplexor called a Transmission Control Unit (TCU); this unit is described in a companion paper by Thoburn.[2]

The planning and development of PCOS took place in 1968. This development was the result of decisions to upgrade the IBM 1460/1440 technology of a satellite computer system (COMATS)[3] then existing at the IBM plant in San Jose, and to provide a single software package that would satisfy the diverse needs of computer-aided testing and process control applications at most of the IBM manufacturing plants.

As an extension of OS/360 PCOS not only provides the normal services and facilities of that operating system, but also contains new features intended to 1) support the TCU, 2) achieve near-optimum response to service requests initiated at the satellite computers and 3) schedule the execution of service programs requested by the satellites.

The innovations included in PCOS are based on a philosophy of real-time operating systems that is also exemplified in other extensions of OS/360. RTOS (Real-Time Operating System)[4] was developed for manned spaceflight ground-support systems at NASA, Houston. BTAM, QTAM and TCAM[5] are operating system extensions provided to support teleprocessing applications of computers. Despite great differences in the manufacturing, spaceflight and teleprocessing environments for computer operation, they all impose the same "real-time" constraint that the system must respond to requests for service within a time on the order of 100 msec. It should be noted that IBM has recently announced for potential customer use a generalized real-time extension of OS/360 called the Real-Time Monitor (RTM).

To be specific in discussing what is meant by real-time systems, it is useful to quote from the recent paper by Weiler et al.[4] (substituted terms are in brackets):

"A basic difference between real-time systems and other data processing systems is that real-time systems are data driven. . . . A real-time system cannot explicitly request the next unit of data to be processed. Instead, it must be prepared to accept data whenever it arrives and, if processing cannot be immediately accomplished, store the data in queues for later processing. As soon as it is recognized that data may be placed in queues awaiting processing, another characteristic of real-time systems is identified, i.e., responsiveness. . . . The requirements of being data driven and highly responsive are satisfied by the design of [the RTCP and the transmission control unit]. . . ."

Before beginning the discussion of PCOS it is important to note its relationship to the TCU (Fig. 2 of Stuehler's

The author is with the IBM General Systems Division at the Laboratory in Rochester, Minnesota 55901.

paper). The TCU is an essential hardware component of the process control network for which PCOS provides software services. That is, PCOS cannot provide service to satellite computers or terminals unless they are connected to the central computer via the TCU.

The present paper has been written to provide a comprehensive review of PCOS development and operation from a programmer's viewpoint. The heart of PCOS is the Real-Time Control Program (RTCP) of which the "supervisory state" extensions to OS/360 are the most important. To make the description of the RTCP understandable to readers lacking a detailed knowledge of OS/360, the paper has been organized as follows.

A section on "History, requirements and design criteria" summarizes the justification for and considerations leading to the development of PCOS. Next, a section on "PCOS architecture" discusses the structure of PCOS and reviews the multiprogramming characteristics of the MFT version of OS/360. Then, a section called "Key functions of OS/360" presents background information critical to an understanding of PCOS design. This section is included especially for those readers who are unfamiliar with OS/360. The following section, "Supervisory state extensions," describes in considerable detail the design and implementation of the major PCOS components. The section on "Problem state functions of the RTCP" is included to provide completeness to the description of PCOS. Here, the level of detail is reduced for the sake of brevity. The final major section, "Present status of PCOS," points out some areas in which development is continuing.

## History, requirements and design of PCOS

The decision to develop PCOS was made in February, 1968. To reach that point, several months had been spent gathering and studying requirements of several plants within the Systems Manufacturing Division. A modification to Disk Operating System/360, known internally in IBM as CIMPAC, had been evaluated in October, 1967. Serious consideration was given to converting San Jose's 1460-based COMATS into a System/360 process control system. Moreover, the desirability of creating a completely new programming system tailored to suit the precise needs of process control was an issue that is still being considered today.

A committee of engineering personnel from five domestic IBM manufacturing facilities was formed to evaluate alternatives and make recommendations for a process control system design.

The use of small computers at "stand alone" test stations clearly emphasized the need for a central system. The updating of test programs at each station was becoming a major problem. Many stations also needed the same data and occasionally data from another station. The cost of I/O equipment for inputting, outputting, and storing programs and data was incremented with each new test station. The requirements of a central system were well established.

The next consideration was a programming system at the central system. The characteristics of Operating System/360 were well known. Although it is not conventionally a real-time system ("jobs" are initiated slowly, the operator may have to respond to messages before, processing can continue, etc), OS/360 does allow multiprogramming with full support of FORTRAN, PL/1, and many "data management services." Additionally, the maintenance and improvement of a programming system plus field engineering support were important factors. The potential value of all these facilities, especially at a central system, could not be denied.

The consideration of OS/360 was not complete without giving some attention to the Disk Operating System/360 (DOS).[7] The basic criteria for using either OS or DOS had been established. From a previous study, it was known that DOS could support a maximum of three (problem program) tasks; the MFT (multiprogramming with a fixed number of tasks) version of OS/360 would support fifteen while the MVT (multiprogramming with a variable number of tasks) version would support any number of sub-tasks related to any one of fifteen tasks. The use of FORTRAN or PL/1 was considerably restricted under DOS. The judgment was that OS/360 was preferable to DOS.

Operating in two System/360 computers, CIMPAC could satisfy all the basic needs. But because CIMPAC uses a "single task" supervisor, multiprogramming was precluded. Furthermore, FORTRAN and PL/1 were not available with CIMPAC. It was decided that all the functions of CIMPAC could be incorporated as a task under OS/360.

Converting San Jose's 1460 COMATS into a System/360 process control system was discarded, not because of a lack of excellence, but because the architecture of COMATS would not fit the present generation of computers. Making a "one-to-one" conversion of 1460 COMATS to System/360 which has five types of interrupts was, the committee decided, impractical.

The last item considered was the creation of a new programming system. The "precise needs" of process control were enumerated by tabulating the requirements submitted from several plants. Most plants requested that the programming system operate in a System/360 Model 30F (64K bytes of main storage). All plants required a central system with data management services for large but undefined program and data banks. All plants desired the availability of FORTRAN and PL/1. Considering manpower and dollar resources, especially for support of FORTRAN and PL/1, the committee agreed that developing a new programming system was out of the question.

The only candidate that, in the committees' judgment, would meet immediate and future needs was OS/360. The MFT version was selected for one principle reason: the MFT supervisor program provides multiprogramming capability in a System 360 Model 40G (128K bytes of main storage)—one size larger than the desired minimum.

The design of PCOS had to satisfy the general requirements that the central system must
1) respond in a real-time environment,
2) provide storage facilities for satellite programs and data.
3) provide a communication facility through the Transmission Control Unit—from a satellite to the central system and back again to the same, or to a different, satellite.

To equal the performance of San Jose's COMATS, the response of the central system had to be within 500 msec 95% of the time. But this design criterion was applicable specifically to the system load (the satellite environment) in San Jose. Attempts to state a response time criterion that would satisfy all potential user locations were unsuccessful. Hence, the response time design goal was not rigidly specified. It was felt that the performance of PCOS would equal or better that of COMATS. This was purely a matter of judgment based on using the new generation of computers. On the basis of experience with COMATS it was judged that the new system could expect an average of one interrupt per second from satellites requesting service; the average transaction would be 2000 bytes of data. It was also specified that with all input queues empty at the central computer the interval from interrupt time to the start of execution of a service program had to be 20 msec or less if the service was in core storage and 200 msec or less if the service had to be fetched from disk or drum storage. These specifications assumed the use of a System 360 Model 40 as the central computer.

To provide storage facilities for satellite programs and data was, on the surface, a design criterion. But a data management structure applicable to satellite systems and OS/360 data management services had *not* been defined. Satellite systems had been developed independently of any other system; there was absolutely no programming experience with OS/360 among those initially involved in the project. As a result the concept of Service Modules was derived from the CIMPAC study and a basic design goal was established:
PCOS was to be an "open-ended" system. Process control users could use the central system by providing their own Service Modules. The open-endedness of PCOS would give each plant the means to request, load and execute its own service program. PCOS would provide a real-time environment and support the execution of a multiplicity of service programs.

To provide communication facilities via the TCU was, to some extent, a straightforward programming problem. But to do it in the context of a real-time system was not. The design criteria in this instance were implicit: support of the TCU, the processing of request codes, and the scheduling of Service Modules had to be complementary if a real-time response was to be a meaningful goal. Moreover, Service Modules should function independently from the processing of request codes and a diagnostic capability should be available through the TCU. A result of these considerations was the design goal of functionally separating automatic polling, data transfer and diagnostic testing. The specification of three unit addresses for one TCU device, as described in the section on "Supervisory state extensions," was a direct result of this goal.

It should be noted that the design of PCOS is a reasonable compromise. It satisfies the real-time requirements without significant internal modifications to the MFT supervisor. Additional interfacing routines have been developed which are modular and for the most part independent of the continuing developments of OS/360 itself. By taking this approach the designers of PCOS have minimized the problems of interfacing with the future releases of OS/360.

## PCOS architecture

The architectural relationship between the main elements of PCOS is shown in Fig. 1. The OS supervisor is the system control program that executes the functions needed to support a multiprogramming environment. In PCOS no changes have been made to the standard OS supervisor. To the OS supervisor PCOS consists of two separate "tasks"*: 1) the Real-Time Control Program, including its attention routine and appendages and the Service Modules (the application programs which serve the satellite computers, and 2) Core Common.

The Real-Time Control Program (RTCP) is the part of PCOS that permits the central computer to operate in a real-time environment. Execution of the RTCP is initiated as if it were a normal "problem" program operating under control of the standard OS supervisor. The RTCP in turn controls the execution of all Service Modules, which, in the case of PCOS, are the programs presented to the central computer for solution. Since the RTCP is executed as a "never-ending" task, it is necessary to perform the time-consuming task-initiation procedure only at system start-up time.

The RTCP, its Attention Routine and Appendages perform several important functions: 1) they provide software support to the Transmission Control Unit (TCU)

* In some of the literature[6] on OS/360 the overall work of data processing may be defined in terms of jobs, job steps, system tasks, and subtasks. In a given OS context, any one of these terms may have a specialized definition. For the purposes of the present paper, a *task* is defined as program executed in a single partition of storage under the direct control of the supervisor. A *subtask* is a program executed under the control of a "task" program.

2) they provide the means to schedule and load Service Modules and 3) they provide internal services necessary for the execution of the Service Modules.

The RTCP prepares itself to accept data called a "request code" after it commands the TCU to start automatic polling of satellite computers. When a satellite system needs service from the central system, it responds "positive" to polling and transmits request-code data to the TCU. The TCU buffers each request code and generates an I/O interrupt to the central system with "attention status." This causes the RTCP to read the buffered request code (which it is prepared to accept) and the TCU then continues automatic polling. If a valid request code cannot be serviced immediately, the RTCP queues and, subsequently, retrieves the request code when the service can be performed.

The block labelled Core Common in Fig. 1 refers to a program that is loaded by the OS supervisor to provide a temporary storage facility in the central computer for use during the execution of Service Modules. Service Modules employ an RTCP routine to enter or retrieve data into or from the Core Common partition of storage. As demonstrated in a later section of the paper, the fact that the Core Common program performs no work other than initiating itself (i.e., it becomes "dormant") is used to advantage in executing programs.

As previously stated, PCOS is based on the MFT version of OS/360. In this version problem programs execute in partitions of core storage, numbered P0, P1, P2, ··· , P51 (a maximum of 52 partitions). The size of each partition must be defined before a problem program is loaded for execution. Figure 2 illustrates a typical core storage layout. The term "fixed number of tasks" refers to the number of core storage partitions that must be defined before any programs can be loaded. The tasks and subtasks performed in a given parition are treated as a single task by the MFT supervisor. Thus, with the partitions defined as in Fig. 2 the multiprogramming supervisor controls and monitors four tasks:

1) Core Common in P0,
2) the Real-Time Control Program and Service Modules in P1,
3) the Output Writer in P2 (an OS service program) and
4) any other "job" or "series of jobs" in P3.

The reader familiar with OS/360 will recognize that the Real-Time Control Program (RTCP), its Appendages, Attention Routine, and all Service Modules are treated as a single task by the OS supervisor, be it MVT or MFT. However, because the RTCP contains special routines for the loading of disk-resident Service Modules, for passing control to all Service Modules and for terminating the programs, it is convenient to consider Service Modules as sub-tasks of the RTCP.

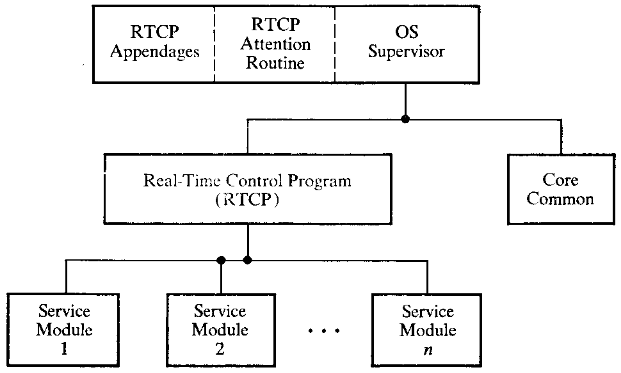
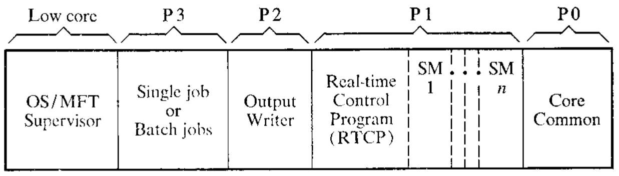


**Figure 1** PCOS architecture: major software components.

**Figure 2** Core storage layout of central computer in a system using Operating System/360 multiprogramming with a fixed number of tasks (MFT). The supervisor resides in lower addresses of core storage; the balance of core storage is divided into four partitions—P0, P1, P2 and P3. A problem program located in P0 will have the highest execution priority; one in P3 the lowest. Following initialization, Core Common is dormant; thus the RTCP and Service Modules in P1 will then have the highest execution priority.



## Key functions of OS/360

The design of PCOS is based on two key functions of OS/360: the control of input/output activity performed by the OS supervisor and the routing of data through the OS supervisor. It is important to be familiar with these functions if one is to understand the differences between PCOS and standard OS/360 operation.

• *Control of input/output activity*

One function of the supervisor program is to control the execution of all input-output activity. This function is especially important in a multiprogramming environment where devices and data files may be shared by several users. The supervisor must resolve all conflicts among users that desire access to a particular resource. Typical resources are I/O devices (including the Transmission Control Unit of the satellite system) and data files.

After the OS supervisor program has loaded a user program (a problem program) it passes control of the computer to the user program. The user program may continue processing until it requires an I/O operation. It must then pass control back to the supervisor with the request that a particular I/O operation be initiated. If

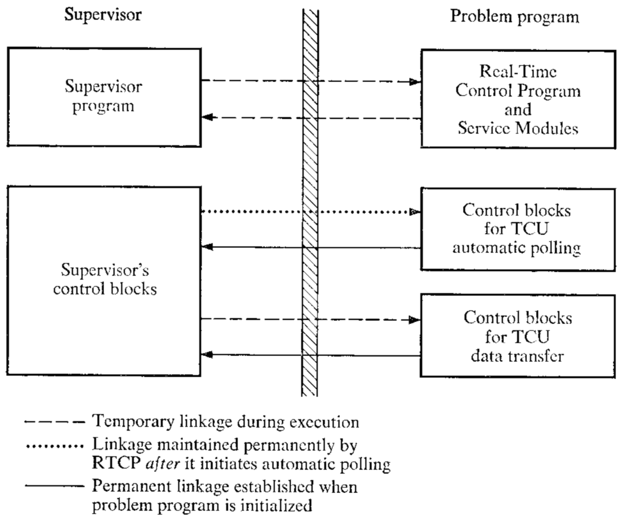

Supervisor
Problem program
Supervisor program
Real-Time Control Program and Service Modules
Control blocks for TCU automatic polling
Supervisor's control blocks
Control blocks for TCU data transfer
Temporary linkage during execution
Linkage maintained permanently by RTCP *after* it initiates automatic polling
Permanent linkage established when problem program is initialized

**Figure 3** Linkages between supervisor and problem program for I/O work.

the I/O resource is "busy" (in use due to any previous request) the supervisor must enqueue the current I/O request until the resource is no longer busy; otherwise the supervisor initiates the I/O operation. In either case, whether the request is queued or initiated, control of the computer is again passed to the user program so that it may continue processing. Eventually, the user program must pass control back to the supervisor, indicating that it wishes to wait for completion of one or more I/O requests. In a multiprogramming environment, the supervisor may then give control of the computer to another "task" (a problem program) of lower priority. Whenever an I/O request is completed, the supervisor automatically retrieves and tries to initiate any queued I/O requests before returning control to some problem program. The task having highest priority will always receive control if it is waiting on the completed I/O operation.*

The passing of control between the supervisor and a problem program is depicted in Fig. 3. Temporary linkages (those that exist for relatively brief periods) are indicated by dashed lines; permanent linkages (those that may be established for the total duration of problem program execution) are indicated by solid lines. Before a user program can request an I/O operation it must initialize the required "control blocks" by means of special OS/360 service routines invoked by the OPEN macro-instruction. The OPEN routines of OS establish the permanent linkage between the user's control blocks and the supervisor's control blocks. When a user program requests an I/O operation it provides the address to a particular set of user control blocks so that the supervisor can establish the temporary linkage from its own control blocks to the user's control blocks in order to initiate and complete the I/O operation.

The user's control blocks of Fig. 3 contain special control words (e.g., device commands and addresses of data buffers and I/O workspace) and software indicators ("flags" and codes) required by the supervisor in order to execute an I/O request. The supervisor's control blocks contain the "unit address" of each device attached to the system, some control information peculiar to each device ("device characteristics"), workspace required by procedures common to all I/O requests (including device status conditions), and the control words ("queue elements") needed to link the supervisor's control blocks with the user's control blocks when an I/O request is made.

A given set of user control blocks may be used repeatedly after each I/O request is completed. It is also possible to make several I/O requests to the same resource (device or data file) before previous ones are completed if the user's program contains as many sets of control blocks as the number of I/O requests that will be made before the first one is completed. The latter case is typical of a PCOS environment; any number of Service Modules may independently make I/O requests for the same resources, in particular for the Transmission Control Unit.

It should be stressed that the linkage from the supervisor's to the user's control blocks is established only by an explicit request for the next unit of data; when the I/O operation is completed the OS supervisor deletes the temporary linkage between control blocks. This characteristic is in direct conflict with a real-time system requirement. In particular, as far as PCOS is concerned, the response to automatic polling by the Transmission Control Unit would be adversely affected. In a later section it is described how an RTCP Appendage converts the (normal) temporary linkage to a permanent linkage when automatic polling is first initiated. From then on, the RTCP is ready to accept request code data in a continuous fashion without re-initiation of an I/O request.

## • *Data routing through the supervisor*

An area of particular interest for PCOS concerns the data routing through the OS supervisor that is triggered by an I/O interrupt. Figure 4 illustrates all the possible OS data paths from component to component from an I/O interrupt to a problem program task. Readers familiar with OS/360 may observe that the diagram does not illustrate data routing directly from a problem program

* In OS/360 terminology, the execution priority of tasks and subtasks is determined by a "dispatching priority"; in MFT, this priority is determined by the partition in which a problem program may reside. As illustrated in Fig. 2, P0 has the highest priority and P3 the lowest. For optimum operation, high priority tasks must wait on I/O activity or other external events so that lower priority tasks may have execution time.

to the Execute Channel Program Supervisor or to the Dispatcher. The former occurs when a problem program requests an I/O operation, and the latter when a problem program issues the WAIT macro-instruction so that the dispatcher may give control to a lower priority program. For simplicity, consider the data routing that begins with the completion of an I/O request, i.e., an I/O interrupt.

The shortest possible data path occurs if the problem program does not include supervisory state extensions to the OS supervisor. In this case, the data route will be through the interruption supervisor and dispatcher only. In Fig. 4 the data path numbers (for the shortest path) are 1 to 6 and 11.

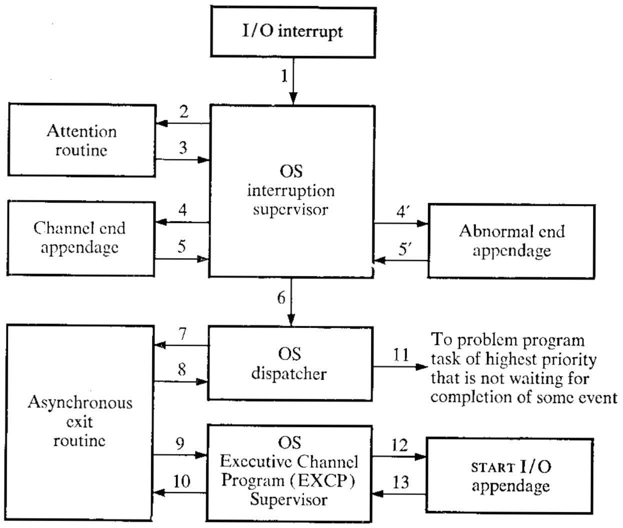
The inclusion of one or more supervisory state extensions will, of course, increase the length of the data path and the time required to traverse it since each extension will receive control to execute its own instructions. The extension routines (or appendages) have coding restrictions and conventions that should be followed. Additionally, OS/360 provides special macro-instructions that must be coded for execution *before* the extensions are needed (e.g., when a program begins operating). In this way, the OS supervisor establishes linkages (addresses) to each extension identified through coding parameters. A given set of extension routines, appendages, and macro-instructions is associated only with a specific I/O device which, in turn, is fully identified by an I/O interrupt. For a specific device, such as the TCU, the OS supervisor can link to applicable extensions by using the device each time an I/O operation is completed.*

Assuming that all the extension routines or appendages shown in Fig. 4 exist, let us consider the conditions that must prevail so that each may receive control. On receiving an I/O interrupt the OS interruption supervisor will first link to the attention routine defined for the I/O device if the status from the device indicates "attention." As indicated in Fig. 4, the attention routine returns control to the interruption supervisor. If the status examined by the interruption supervisor contains only the "attention flag" then only the attention routine will receive control before the interruption supervisor gives control to the dispatcher.

The channel end appendage *or* the abnormal end appendage is entered if the status words of the I/O channel and I/O device indicate the completion of an operation. Upon a given I/O interrupt, the interruption supervisor may give control to either appendage but not to both.

Normally, the dispatcher immediately gives control to some problem program. But, if the attention routine or either appendage has requested the scheduling of an asynchronous exit routine, the dispatcher will first give control to the appropriate asynchronous exit routine.

* System/360 hardware stores the address of a device in the "old I/O program status word" when an I/O interrupt takes place.



I/O interrupt
1
Attention routine
2
3
OS interruption supervisor
Channel end appendage
4
5
4′
5′
Abnormal end appendage
6
Asynchronous exit routine
7
8
OS dispatcher
11
To problem program task of highest priority that is not waiting for completion of some event
9
10
OS Executive Channel Program (EXCP) Supervisor
12
13
START I/O appendage

**Figure 4** Data routing through the OS supervisor. Diagram illustrates the data routings that may be triggered by an I/O interrupt. The attention routine channel end appendage, abnormal end appendage, the asynchronous exit routine and the START I/O appendage are conventional problem program extensions to the OS supervisor; these extensions operate in supervisor state.

It may be helpful to consider a typical application to define what is meant by an "asynchronous exit routine." The figures and characters displayed in an IBM 2250 display station are driven through explicit I/O requests that provide data to a given unit. Each of these operations is accompanied by an I/O interrupt when the (output) data transfer is completed. These interrupts are "synchronized" with problem execution because they occur as a consequence of an I/O operation directed to the device. But requests made by an operator at a display station may occur asynchronously to problem program execution. To handle this case, the display stations are designed to generate a status of "attention" when the operator indicates that he wishes to communicate with the computer. The result: an attention routine requests that the dispatcher take an "asynchronous exit" in place of its normal (synchronous) exit to a problem program.

An asynchronous exit routine can perform any number of I/O requests (the same or different) through the Execute Channel Program (EXCP) Supervisor (e.g., to read information being keyed in by an operator). Conventionally, the attention routine cannot perform any I/O requests. The channel end and abnormal end appendages can request (through modified exits) that the same I/O operation be repeated, but the appendages cannot initiate any new I/O request. We shall see in the following section that in the case of PCOS, if a conventional implementation had been made, an asynchronous exit routine would be required to read request codes buffered by the TCU.

Finally, with reference to Fig. 4, a few comments about START I/O appendages are in order. A START I/O appendage receives control from the EXCP supervisor whenever an I/O request is made for its associated device; control is given to the appendage before the I/O request is executed. As indicated in the figure, a START I/O appendage returns control to the EXCP supervisor. A "normal return" allows the execution of the I/O request and a "modified return" disallows the I/O request. PCOS uses a START I/O appendage to prevent the incorrect use of commands that may be issued to the TCU (e.g., service modules may not start or stop automatic polling).

## PCOS supervisory state extensions of OS/360

PCOS was designed to enhance the effectiveness of OS/360 performance in the real-time environment represented by IBM's plants. Specifically, 1) it improves the ability of the OS supervisor to respond quickly to data-driven I/O interrupts that request other I/O operations and 2) it eliminates the need for all other activity in an entire partition to wait until the loading of a program, such as a Service Module, is completed. In PCOS these enhancements are accomplished by three supervisory-state extensions to OS: the RTCP Attention Routine, the RTCP Channel End Appendage, and the RTCP Fetch Module. Moreover, through the first two extensions the TCU is supported in a data-driven mode. This section describes how these operations are accomplished. In addition, the PCOS support for "interpartition communication" (another supervisory state extension) is also described.

It should be stressed that the PCOS express path, the PCOS method of loading disk-resident Service Modules, and PCOS support of the TCU provide a means of handling TCU request codes in a very efficient way (when using OS). By combining the individual effects of these supervisory state extensions, the net effect is most significant. Request codes can be quickly processed so that the TCU may continue polling, Service Modules can be loaded while other Service Modules continue their execution (or while a subsequent request code is processed), and Service Modules can use the TCU without disrupting or delaying the handling of request codes.

• *PCOS express path for reading a request code*

Let us direct our attention to the express path and consider a request for a core-resident Service Module. Assume that the TCU function of "automatic polling" has been started.

The TCU buffers request codes, one at a time, as they are received from a satellite computer. As soon as a request code has been buffered, the TCU generates an I/O interrupt to the central system with attention status. As described previously, the OS interruption supervisor will then give control to the RTCP Attention Routine which, conventionally, would never "request" nor "execute" an I/O operation. But the RTCP Attention Routine issues a START I/O instruction to read the request code buffered in the TCU into an input queue and then completes the operation by issuing a TEST I/O instruction. START I/O and TEST I/O are "privileged instructions" to be used only by the supervisor, but the attention routine can issue them because it operates in supervisory state.*

A TEST I/O instruction retrieves ending status due to an I/O operation. Thus, when the attention routine issues a TEST I/O to the TCU, the status contained in the System/360 channel status word is changed from "attention" to "channel end/device end" (obtained from the TCU). When this is done, the attention routine returns control to the interruption supervisor which then detects a status of "channel end" (in addition to "device end") and passes control to the RTCP Channel End Appendage.

The RTCP Channel End Appendage examines the request code read by the attention routine to determine the identity of the Service Module (SM) requested. If the SM is in use but is serially re-usable, the current request code is chained to a previous one in the input queue for later retrieval. If the SM is disk resident, the RTCP Channel End Appendage will set parameters so that the OS dispatcher will take an asynchronous exit to the RTCP Fetch Module. If the SM is core resident—which is the case we are considering—the appendage marks the SM ready for execution. Finally, the RTCP Channel End Appendage keeps the automatic polling control blocks (a set of user control blocks) permanently open so that the attention routine will continue to process request code data. Control is then returned to the OS interruption supervisor.

The OS interruption supervisor passes control to the OS dispatcher which, in turn, gives control to the RTCP problem program. Any SM that has been marked "ready" for execution will receive control (in problem state) from the RTCP SM Initiator.

In summary, the sequence for initiating execution of a core-resident SM is: 1) an I/O interrupt with attention status is generated by the TCU, 2) the RTCP Attention Routine reads the buffered request code through a START I/O-TEST I/O sequence, 3) the resulting status due to TEST I/O is channel end/device end, 4) the RTCP Channel End Appendage marks a core-resident Service Module ready for execution and 5) the dispatcher gives control to the RTCP (in problem program state).

As indicated in Fig. 4, the linkage to the attention routine, channel end appendage and to the RTCP is accomplished via the OS interruption supervisor and the OS dispatcher. And, to emphasize the PCOS express path for reading a request code, note that only *one* I/O inter-

* The procedure followed is identical to the one in the "sense subroutine" of the OS Supervisor.

rupt takes place and the path numbers in Fig. 4 are 1-2-3-4-5-6 and 11.

**•** *Conventional approach to reading a request code*

To read a request code using a conventional approach, *two* I/O interrupts are required. The path numbers for the first I/O interrupt are 1-2-3-6-7-9-12-13-10-8; this interrupt is followed immediately by a second I/O interrupt with path numbers of 1-6-7-8 and, finally 11.

In a conventional implementation, the logical sequence for the first I/O interrupt is: 1) an I/O interrupt with attention status is generated by the TCU, 2) the RTCP Attention routine requests that the dispatcher link to an asychronous exit routine—the request code is not read, 3) the asynchronous exit routine makes an I/O request to the EXCP supervisor to read the request code buffered by the TCU, 5) the EXCP supervisor links to the RTCP START I/O appendage before executing the I/O request, 6) the I/O operation is initiated and the EXCP supervisor returns control to the asynchronous exit routine, 7) the asynchronous exit routine issues a WAIT macro-instruction to cause a wait for completion of request code input and 8) the dispatcher receives control as a result of the WAIT macro-instruction.

When the dispatcher enables I/O interrupts, the interrupt will occur immediately because the TCU will have completed transferring its buffered request code (this operation is very fast).

The logical sequence for this second I/O interrupt is: 1) an I/O interrupt with channel end/device end status is generated by the TCU (the channel end appendage is *not* an essential requirement in this case and may be assumed *not* to exist), 2) the dispatcher gives control to the asynchronous exit routine because the event it was "waiting on" has now completed, 3) the asynchronous exit routine marks a core resident Service Module ready for execution and 4) the dispatcher gives control to the RTCP (in problem program state).

Clearly, the express path used by PCOS is much shorter than the path that would be used in conventional OS operation. Under normal operating conditions, many milliseconds are saved by avoiding the queueing and retrieving of the I/O requests which can occur when the System/360 channel is busy servicing some other I/O request.

**•** *Fetch of disk-resident Service Modules*

As indicated above, whenever the RTCP Channel End Appendage finds that a Service Module is disk resident, it sets parameters so that the OS dispatcher will take an "asynchronous exit" to the RTCP Fetch Module. The path through Fig. 4 to the RTCP Fetch Module, an asynchronous exit routine, is 1-2-3-4-5-6-7.

There are several important steps involved in the loading of disk-resident Service Modules. Some are performed within the channel end appendage before the fetch module is entered and the others within the fetch module.

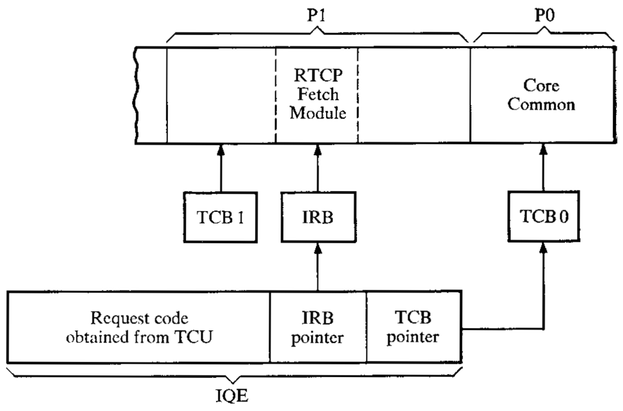


**Figure 5** Fetching disk-resident Service Modules in overlap mode. The task control block pointer within the interrupt queue element addresses the partition containing Core Common. Service Modules in P1 can then continue executing while a disk-resident Service Module is being loaded into P1.

With reference to Fig. 5, in order to link to an asyncronous exit routine the dispatcher requires what is called an "interrupt queue element" (IQE). The IQE must contain a "pointer" (an address) to an "interrupt request block" (IRB) which, in turn, contains a pointer to some routine or program (in PCOS it is the RTCP Fetch Module). The IQE must also contain a pointer to the "task control block" (TCB) that the dispatcher associates with a problem program. The RTCP Channel End Appendage initializes appropriate parameters but, in addition, the TCB pointer is altered so that its content addresses the TCB for Core Common instead of the TCB for the Real-Time Control Program. This situation is depicted in Fig. 5. Result: the OS dispatcher treats the fetch module in P1 as if it resided in P0.

To state it more simply, the asynchronous exit routine requested in the RTCP Channel End Appendage is made to belong to another task, i.e., the Core Common problem program.

When the RTCP Fetch Module is entered, this asynchronous exit routine does not perform the actual loading of disk-resident Service Modules. Instead, it initializes parameters required by the OS Fetch Routine and branches directly to it. OS fetch loads Service Modules into the Real-Time Control Program partition (P1) in the same way it loads any other disk-resident program. In so doing, when OS fetch issues a WAIT macro-instruction, the OS dispatcher will receive control and it will place the task in P0 (Core Common) into the wait state (because of the TCB pointer in the IQE) and pass control to a lower priority task not waiting for completion of some event.

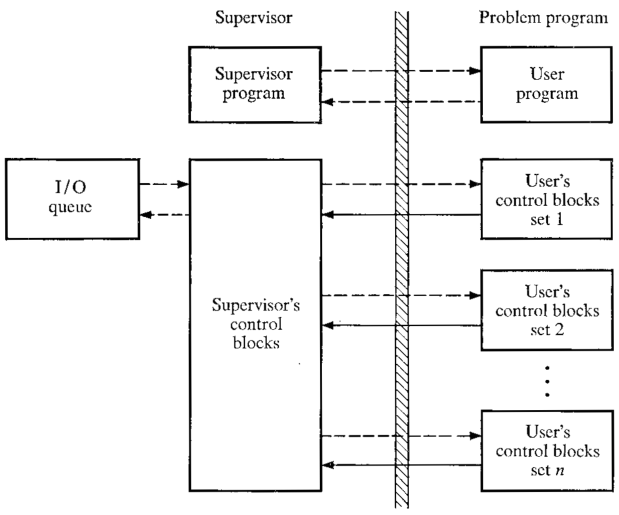

Supervisor
Problem program
Supervisor program
User program
I/O queue
User's control blocks set 1
Supervisor's control blocks
User's control blocks set 2
User's control blocks set *n*
---- Temporary linkage during execution
——— Permanent linkage established when problem program is initialized

**Figure 6** Real-Time Control Program support of Transmission Control Unit.

It was stated previously that dormancy of the Core Common task is used to advantage. It should be clear that on the occurrence of an I/O interrupt for a disk-resident Service Module, if the Real-Time Control Program or its Service Modules are in some stage of execution, processing in P1 will be resumed at the point of interruption as OS fetch loads another Service Module into the same partition. The problem program task waiting for completion of loading will be the one in P0, Core Common.

When OS fetch finishes loading a Service Module it returns to the RTCP Fetch Module (the asynchronous exit routine) which resumes processing in supervisory state. To finish its work, RTCP fetch sets the "ready flag" for the SM just loaded; the SM flags field is located in the request code input queue. Then, RTCP fetch checks the event control block (ECB)* used by (RTCP) SM-Initiator Servicer. If the SM initiator is waiting for the completion of a request code interrupt, RTCP fetch sets the completion flag in SM initiator's ECB so that the OS dispatcher will give control to SM initiator. If the SM initiator is *not* waiting for completion of a request code interrupt, then the Real-Time Control Program or a Service Module has been interrupted from its processing. The dispatcher will automatically return control to the RTCP or one of its Service Modules (so that processing will continue at the point of interruption), and the SM initiator will eventually find a newly loaded SM ready for execution.

Finally, as far as disk resident Service Modules are concerned, the data routing through Fig. 4 is 1-2-3-4-5-6-7-8 (when OS fetch issues a WAIT to the dispatcher) and 11. After OS fetch completes the loading of a Service Module, the RTCP Fetch Routine (working as if it were in P0) will terminate its operations. The OS dispatcher will again give control to the RTCP or to one of its Service Modules in P1.

* An ECB is one control block within a "user control block" set of Fig. 3.

## • *Program support of the TCU*

The PCOS express path and the method of loading disk-resident Service Modules would hardly be effective without the support provided for the Transmission Control Unit. The real-time response depends largely on the PCOS software-TCU hardware relationship. For programming purposes, the hardware interface between the TCU and the central system was designed for three distinct functions: 1) automatic polling, 2) data transfer and 3) diagnostic testing. The programming of these functions can be performed by using *three* unit addresses. That is, although the TCU is actually one device, it will accept commands from the central system on three different addresses.*

The Real-Time Control Program reserves the first unit address solely for automatic polling and the next higher unit address (of the TCU) for data transfer. As stated previously, a user program must have a set of user control blocks for each I/O request. This situation is depicted in Fig. 6, where the user control blocks of Fig. 3 have been assigned to "automatic polling" and "data transfer."

As indicated in Fig. 6, the control blocks for automatic polling remain permanently active once automatic polling has been started. To accomplish this, the RTCP Channel End Appendage executes one additional instruction and it uses a modified return to the OS supervisor. The modified return is a standard option for any OS appendage. Some of the details are discussed in the following four paragraphs.

When the TCU accepts the command to start polling it presents a status of "channel end/device end" (attention status is not generated until the TCU buffers a request code) and, as explained previously, the RTCP Channel End Appendage will receive control. The appendage recognizes the issuance of a start poll command and sets two flags in the supervisor's "unit control block" (UCB) containing the unit address for automatic polling. They are called UCBBUSY and UCBPOST. These flags indicate to the OS supervisor that an I/O request is pending, and any subsequent I/O interrupt (from the unit assigned to automatic polling) will be treated as an explicit I/O request.

The RTCP Channel End Appendage terminates its processing by using a modified return to the OS supervisor. Through this standard option, the OS supervisor will retain what is called a "request queue element" (RQE) in an active status. An RQE is a unit of software logic

* For example, at the plant in Rochester, Minn., the system uses TCU address 260 for automatic polling, 261 for data transfer and 262 for diagnostic testing.

used by the OS supervisor for each explicit I/O request. Among other things, an active RQE contains a pointer (an address) to a set of user control blocks (in this case, those associated with automatic polling).

Two important things have now been explained: 1) flags have been "set" in the supervisor's control blocks so that a subsequent I/O interrupt will be treated like an explicit I/O request and 2) the linkage to a user's control blocks is retained. The result is that when the TCU generates an I/O interrupt with attention status the OS interruption supervisor will handle it as if an explicit I/O request had been made.

The RTCP Channel End Appendage repeats the same procedure whenever it processes a request code. Thus, because the TCU does not require a command to continue polling, it should now be clear why the Real-Time Control Program is data driven. That is, request codes buffered by the TCU are processed without the use of explicit I/O requests to the OS supervisor.

With the first unit address of the TCU reserved for automatic polling, what of the second unit address? The software is conventional in design and use. It is reserved by the RTCP for data transfer operations by any Service Module.

As for the third unit address of the TCU, the RTCP does *not* use it, and diagnostic tests can be performed without any conflicts from a separate OS partition while the RTCP is in operation.

• *Software implications of multiple TCU addresses*

From a software standpoint, the advantages in having three unit addresses for the same TCU are: 1) responsiveness to automatic polling is optimized, 2) it is logically convenient to assign each unit address with the specialized functions of polling, data transfer and diagnostic testing and 3) it is possible to assign the unit address for diagnostic testing to a separate problem program. As to the latter, once a "unit address" has been assigned to a "task" (a problem program) of OS/360, the OS supervisor will not assign the same "unit address" to another (independent) task; however, because the TCU will accept three unit addresses, it could in fact be assigned to three different independent OS tasks.

• *Hardware implications of multiple TCU addresses*

The relationship between OS control blocks, unit address, and devices led to a specification for three unit addresses. But the TCU will accept all of its commands on any one of its addresses. Thus, the TCU is independent of any programming system and is designed to work with any System/360 computer.

It is possible to program the TCU by using only its first unit address. Nevertheless, there are characteristics of the TCU that offer important programming advantages. When the TCU buffers a request code, it does two important things: 1) it resets the "polling latch" so that the satellite system is not polled and 2) the I/O interrupt with attention status will occur only on the first unit address. From the latter, a programmer wishing to treat automatic polling as a separate function (such as in PCOS) must use the first unit address for this purpose.

When a central system program (such as a Service Module) selects a satellite computer, the TCU will automatically re-enable polling of the satellite system. However, automatic polling will be re-enabled only if the satellite system is selected on the first or second unit address. Thus, a programmer may use the first unit address for automatic polling and data transfer. He may, optionally, separate the two functions (such as PCOS). Data transfer on either the first or second unit address will produce the same result—the TCU will resume automatic polling to the selected satellite after it has been serviced.

If a central system program selects a satellite computer on the third unit address of the TCU, the selected satellite will not be re-enabled for automatic polling. In this way, the third unit address can be reserved for diagnostic tests and the satellite system will not be polled automatically. (It is possible to poll a satellite system through software. It is also possible to enable or disable a satellite system for automatic polling by explicitly using a MASK command. This may be done on any unit address, including the third.)

The TCU characteristic of not re-enabling automatic polling (unless explicitly requested) on the third unit address is significant. It is possible to test one or more satellite systems without affecting the Real-Time Control Program. If polling were re-enabled, the TCU would buffer what could be an undesired request code which, in turn, would be processed by the RTCP.

The design of the hardware interface of the TCU is complementary to the design of PCOS. Each unit address of the TCU provides features that assist the programming of three distinct functions—automatic polling, data transfer and diagnostic testing.

• *Elimination of stored polling list*

Another important characteristic of the TCU is its ability to automatically poll satellite systems *without* the continuous access of terminal addresses. The TCU accepts a "polling list" through what is called a MASK operation. Though this operation is performed by a problem state function of the RTCP, it is considered here because of its specification for (and relation to) real-time programming.

Through the MASK operation, the RTCP TCU Initialization Servicer (in problem state) provides a polling list to the TCU. Once completed, the TCU does not require a polling list from the central system. In contrast to teleprocessing devices, a real-time response is not compromised by I/O channel interference. Moreover, after the command to start automatic polling has been issued, no software instruction or polling list data is needed (except for error conditions). The only normal requirement is the ability to process I/O interrupts for request code handling. In PCOS, a polling list is provided only when the RTCP begins its operation. The RTCP Attention Routine and the RTCP Channel End Appendage take care of request code handling. The need for use of a permanently core-resident polling list is eliminated.

• *Inter-partition communication*
As illustrated in Fig. 2, the Core Common partition is a separate area of core storage. For this reason, to the OS supervisor Core Common is an independent task with its own "storage protect key." A problem program that attempts to store data into another partition will be abnormally terminated by the OS supervisor. However, the PCOS Core Common Servicer provides the means to store or retrieve data into or from the Core Common partition.

The Core Common Servicer was designed for RTCP Service Modules, but any other problem program may use this servicer which stores or retrieves data only into or from the Core Common partition. This function is implemented through the use of PCOS macro-instructions and a PCOS SVC Routine.* A user must provide a unique eight-byte "label" to store data and, subsequently, to retrieve data; another user may retrieve data by using the same label. Provisions are available to prevent storing data with duplicate labels and for the deletion of labeled data.

The core buffer used by the Core Common Servicer has a fixed size that can be determined at each user installation. To prevent core storage from becoming fragmented when data records are deleted from a string of records, the Core Common Servicer links pieces of core together when necessary. Core Common is thus limited by the size of the core buffer but not by core fragmentation.

To use the PCOS Core Common Servicer, the Core Common partition must be initialized so that a PCOS SVC routine can place the address of the Core Common data buffer into the communications vector table (CVT) of the OS supervisor. (The CVT contains addresses to routines and control blocks used by the OS supervisor or by SVC routines. A pointer to the CVT is stored in core location 16 in all versions of OS/360 when the OS supervisor is initialized.)

## Problem state functions of the RTCP

The problem state routines of the Real-Time Control Program are loaded by the OS supervisor as one problem program. As the RTCP begins its execution, linkages to the supervisory state extensions are established through normal conventions of OS/360 and the RTCP initializes all the functions it needs to begin its work. Additionally, the RTCP provides many service routines for its own use and for Service Modules. The following paragraphs describe the highlights of the RTCP services.

• *Data Control Block Servicer*
A requirement common to all Service Modules is the need to access I/O devices and data files. The use of these I/O resources must be planned before any Service Module can be coded. When the RTCP begins its execution, the Data Control Block (DCB) Servicer initializes all the sets of user's control blocks (Fig. 3) required by Service Modules and by the RTCP. Service Modules use the DCB Servicer to obtain any defined set of control blocks. An I/O request can then be initiated without delay.

The DCB Servicer uses the OPEN macro-instruction to initialize all sets of user control blocks. The significance of "opening data control blocks" when the RTCP begins its execution may be understood by considering a typical case. Service Modules of the PCOS installation at IBM Rochester use 13 data files. The processing performed by the open routines of OS takes 15.6 sec—an average of 1.2 sec per set of control blocks. Once initialized, control blocks remain "open" for immediate use. The time saved for a real-time system environment is self-evident.

• *TCU Initiation Servicer*
The TCU Initiation Servicer performs the mask operation and issues a command to the TCU to start automatic polling (via I/O requests). It should be emphasized that the automatic polling control blocks provide control information to the OS supervisor and *also* to RTCP functions. In particular, the automatic polling control blocks contain the address of the input queue for request code data required by the RTCP Attention Routine and the RTCP Channel End Appendage. This all-important address is made available when the TCU Initiation Servicer starts automatic polling.

• *SM Initiation Servicer*
The SM Initiation Servicer gives control to Service Modules that are ready to begin or to resume their execution. SM execution is resumed when any of its I/O operations have completed.

When two or more SM's are waiting for completion of an I/O operation, the SM Initiation Servicer "looks ahead" to see if other SM's are ready to begin or resume execution. If no Service Module is ready for execution, this servicer branches to the PCOSWAIT Servicer to wait for the completion of an I/O interrupt (when another SM is ready for execution).

* An SVC "supervisor call" routine operates in supervisor state as yet another extension to the OS supervisor. An SVC routine for Core Common operates with all System/360 interrupts disabled (excluding machine check) to lock out any interference.

• *PCOSWAIT Servicer*
Whenever a problem program performs an I/O operation, if it cannot continue processing it should issue a WAIT macro-instruction to wait for the completion of its I/O request. The OS supervisor will return control immediately if the I/O operation has completed. Otherwise, control is given to a problem program of lower priority. When the I/O operation is completed, the OS supervisor interrupts the lower priority program (triggered by hardware interrupt) and passes control to the higher priority program.

In the case of RTCP Service Modules, the use of the WAIT macro-instruction is not allowed because a lower priority partition may gain control while other Service Modules are in some stage of execution. The PCOSWAIT macro-instruction is used by Service Modules instead of the normal WAIT macro-instruction to invoke the PCOSWAIT Servicer routine.

In performing its functions, the PCOSWAIT Servicer uses what is called a "multiple wait," a special form of the WAIT macro-instruction. At the very least, the SM Initiation Servicer will be waiting on the completion of a request code interrupt. Additionally, one or more Service Modules may be waiting for completion of an I/O request.

When the PCOSWAIT Servicer issues a multiple wait, the OS supervisor gives control to a lower priority program if *none* of the multiple I/O events have completed execution. But if one or more of the multiple I/O events has been completed, or whenever one or more does become completed, the OS supervisor returns control to the PCOSWAIT Servicer which, in turn, passes control to the SM Initiator.

• *Program Check Servicer*
A "program check" is a name given to error or fault conditions associated with the execution of a program. In System/360, a program check will cause a special interrupt which is similar to, but separate from, an I/O interrupt. The OS supervisor processes a program check and, in many cases, it will allow a problem program to continue operation according to its own error or fault handling routines. In OS/360 these can be provided through the use of SPIE and STAE macro-instructions. The RTCP Program Check Servicer provides necessary routines and issues SPIE (System Program Interrupt Exit) and STAE (System Task Abend Exit). Its own routines *and* Service Modules can recover from arithmetic errors. In addition, the servicer protects all Service Modules from those operations that may generate nonarithmetic errors (e.g., an attempt to store data in another partition, overflow of a data file, and many others).

• *SM Termination Servicer*
When a Service Module completes its execution it uses the SM Termination Servicer. This servicer is also used by RTCP routines that (abnormally) terminate a Service Module. The SM termination servicer frees all I/O resources that may have been requested by a Service Module from the DCB Servicer. The SM Termination Servicer also clears flags in the input queue to make the space available for another request code.

## Present status of PCOS

PCOS was first installed at IBM's Boulder manufacturing plant in September, 1968. San Jose, Kingston, Raleigh and Rochester each installed a PCOS facility within a year. By September 1969 the user plants had gained experience and acquired programming skills at the central system as well as at satellite systems, and it was time to improve the original design.

Again, a development mission was assigned to the manufacturing plant in Kingston. PCOS was modified to support "re-entrant" Service Modules. Additionally, with the availability of Release 18 of OS/360, PCOS could use the STAE feature described earlier in the paper.

Re-entrant Service Modules are those that can accept many request code inputs regardless of their execution stage. In contrast, a "serially re-usable" SM is one that can handle only one request code input at a time. Re-entrant SM's are most useful when they can provide a commonly used function. The most obvious common functions are program storage or retrieval and data storage or retrieval. A core-resident Service Module that is used often enough can provide faster service if it is re-entrant since a satellite user does not have to wait for the completion of a previous service request.

Efforts have been made, notably by the San Jose plant, to provide a set of re-entrant Service Modules for common use. These efforts have been partially successful. There has been some difficulty, however, in defining a data management scheme common to all plants. Service Modules offered from one plant may include features peculiar to its own installation, e.g., the way error conditions are handled, use of different error control codes, need for additional services, and so on. Thus, satellite systems at the various plants are not using identical Service Modules. Those SM's that are shared are usually modified.

In interfacing the RTCP with OS/360 no attempt was made to develop new core management or data management techniques. Thus, in addition to the problems involved in developing a set of compatible re-entrant Service Modules, some problems exist that are not specifically restricted to the real-time process control environment in which PCOS operates.

One problem associated with OS/360 is called "core fragmentation." Its existence has been recognized since the first development of PCOS but, up to now, there have been ways of overcoming it by providing "enough" core storage. All the details concerning core fragmentation are beyond the scope of this article. The effect, however, is that at some point a Service Module cannot obtain core for its work or the RTCP cannot load another SM. If this point should ever be reached the central computer would become inoperative.

There is a problem, too, if Service Modules require many different data files. For maximum speed, the core required for "user control blocks" becomes prohibitively large. At 200 bytes per set of control blocks, 50 data files will require 10,000 bytes of core storage. IBM Rochester has overcome this problem by using two data files for most Service Modules. The first one is an "index file" which contains the location of each record or data file stored in the second file. This procedure conserves core at the expense of time.

Proposals have been made to take care of known deficiencies. Kingston has suggested that selected control blocks be stored on disk after a data file has been "opened." Rochester has suggested improvement of its system by keeping a core-resident directory of commonly used data records and data files and also a new core management technique.

The acceptance and development of PCOS have continued to progress satisfactorily in spite of certain problems. The design goals and performance specifications of 1968 have been met.

## Acknowledgments

The implementation of PCOS was carried out at the plant in Kingston, New York. M. L. Mauldin led the development effort and incorporated the Service Module concept. He and T. G. Reilley implemented the RTCP Fetch Module and almost all of the problem-state portions of the RTCP. D. J. McKenna developed the Core Common functions and, with A. C. Christiano, implemented most recent improvements to PCOS. W. D. Houle managed the project at Kingston.

The author is especially grateful to R. W. Henry of San Jose who provided him with much of the programming information needed to functionally specify the TCU-to-central computer interface and who, with the author, developed the RTCP support for the TCU. The engineering skill of A. H. Rall was crucial to the design of a TCU that could operate compatibly with PCOS.

The important contributions of W. Moore in coordinating the efforts of managers from the various IBM plants during development of the process control network are acknowledged and the author's special thanks go to G. L. Gorsuch of Rochester, Minnesota for his foresight and commitment to the project.

## References

1. J. E. Stuehler, "An Integrated Manufacturing Process Control System: Implementation in IBM Manufacturing," *IBM J. Res. Develop.* **14,** 605 (1970), this issue.
2. F. W. Thoburn, Jr., "A Transmission Control Unit for High-speed Computer-to-computer Communication," *IBM J. Res. Develop.* **14,** 614 (1970), this issue.
3. J. E. Stuehler and R. V. Watkins, "Computer Operated Manufacturing and Test System," *IBM J. Res. Develop.* **11,** 452 (1967).
4. P. W. Weiler, R. S. Kopp and R. G. Doman, "A Real-Time Operating System for Manned Spaceflight," *IEEE Trans. Computers* **C-19,** 388 (1970).
5. R. M. Winick, "Line Control and Terminal Management in OS/360-A Proposed Control System," *Software Age* **4,** 23 (January, 1970).
6. G. H. Mealy, "The Functional Structure of OS/360, Part One, Introductory Survey," *IBM Syst. J.* **5,** 3 (1966).
7. G. Bender, D. N. Freeman and J. D. Smith, "Function and Design of DOS/360 and TOS/360," *IBM Syst. J.* **6,** 2 (1967).

*Received August 14, 1970*