Operating Systems

B. RANDELL, Editor

The LACONIQ Monitor: Time Sharing for Online Dialogues

DANIEL L. DREW
Lockheed Research Laboratory,
Palo Alto, California

The LACONIQ (Laboratory Computer Online Inquiry) Monitor was developed primarily to support non-numerical applications such as retrieval from very large files by means of a "dialogue" between a system user and a retrieval application.

The monitor was designed so that it could work with a small computer (an IBM System 360/30). Therefore techniques for resource allocation were important. For this reason the use of core storage, computational facilities, and input-output were all scheduled.

An unusual feature of the system is that it is event-driven rather than clock-driven. The program segments called into execution by the remote CRT consoles are invariably run to completion rather than "rolled-out" to be brought back at a later time.

1. Overall Configuration

INTRODUCTION. The basic requirement in the design of the Laboratory Computer Online Inquiry (LACONIQ) Monitor was that it should facilitate the programming and operation of online "dialogues." These dialogues typically consist of inquiries directed to a large data file, to which pre-prepared programs respond in an attempt to help the system user find and display specific information. The prototype application is that of document reference retrieval.

This approach led to an event-driven monitor, as opposed to the usual "clock-driven" time sharing monitor in which application program execution is carried on during a fixed interval, then interrupted until the program has another "turn." In the LACONIQ monitor, each application program is processed to completion. (This is only practical, of course, because the programs are limited in their duration.) Each such program corresponds to at most one step in the dialogue. Long operations might require that a sequence of such short programs (called program segments in the following descriptions) be executed, and this is foreseen in the monitor design. The basic event that governs the timing of happenings in the monitor is then the completion of processing of a program segment.

This decision to set an upper limit on the amount of processor time each segment is allowed rather than to interrupt the segment ("roll it out" to a peripheral storage unit, and bring it back for later continuation) was based on a tradeoff between constraining the application programmer and increasing system response time. Given the conversational nature of the foreseen applications, it was felt that limiting segment execution time would not be a serious constraint because very few segments would approach this limit. (This has turned out to be the case.) The improvement in system response due to this doctrine derives not only from the smaller number of disk references but from the simpler mechanisms needed to handle the segments.

A second major design decision was to make the monitor poll the remote consoles, i.e., to accept input at the system's convenience rather than in response to interrupts generated by user's consoles ("contention" mode). Polling is made practical by the existence of adequate local buffers at the remote consoles. The principal advantage of this mode is that remote input can be scheduled, so no dedicated input buffer areas are required.

All the resources of the system—communication lines, core storage, use of the CPU, and use of the I/O channels—are scheduled in one way or another. The most fundamental scheduler, the work-in-progress scheduler, calls in both program segments and systems routines as appropriate and when needed.

The monitor was not primarily designed to facilitate programming at the user console, although incremental assemblers or compilers can be added in the form of applications. The basic criterion was rather to make it easy for a skillful programmer to prepare an application with which the (usually untrained) user, at his console, could interact in a language natural for that application. This monitor might be described as a three-level system, with the programmer mediating between the layman-user and the computer, and is a step toward making the power of the computer more generally accessible.

It is probably obvious, but perhaps worth mentioning that a further requirement on the monitor was that it should be capable not only of supporting several terminals processing a given application, but also several different simultaneous applications.

Many of the detailed features of the monitor are directly related to the IBM System/360, and some even to the

configuration of the particular computer system at hand, which consists of a 32K 360/30 main-frame with two 2311 disk drives, a 2321 data cell drive, a tape unit, and multiple remote CRT consoles (IBM 2260 and Sanders 720). The disk drives each store 7.2 million bytes with average access time of 85msec, the data cell stores 4.8 million bytes with a maximum access time of 600msec. The CRT consoles display, respectively, 960 and 1024 alphanumeric characters at 120 characters per second line rate). Some hardware factors which have been important in their influence on monitor design are the small core storage, the large peripheral storage, and the local buffers for the console displays.

The first version of LACONIQ was operational in December 1966, supporting three IBM 2260 consoles and using two disk drives and a data cell. It has been continually upgraded and at this writing it is being set up to have a high degree of compatibility with the IBM Disk Operating System and OS/360. The changes include exploiting a larger core storage (64K bytes) and introducing a background capability. Depending on the size of the background partition and the application mix, it is foreseen that the monitor will support between six and twelve CRT consoles with an (approximate) 2-sec response time. System overhead is estimated at 25 per cent, and it is believed that more careful measurement will show this to be pessimistic.

Application Program Segments. A given application consists of data and directory files, and a library of program segments. Each segment consists of two sections, one of which performs the processes the programmer has planned and the other describes the segment and all of its possible successors. When a segment is selected and input, the executable portion is read from disk into general working storage and the descriptive portion is read into a protected area dedicated to the logical console which called in the segment.

Segments are not permitted to perform I/O, to preempt storage that has not been allocated by the monitor, or to directly alter any part of the protected storage where the monitor and the "console areas" are resident. These operations are performed by the monitor at the request of the segment. Communication between the segment and monitor is carried on by "service calls" which generate interrupts and change the state of the CPU from "problem", to "supervisory." There are 15 such calls implemented at present. They permit the segment to request input from disk or data-cell; to output to disk, data-cell, tape, or remote console; and to alter the console-dedicated area, request more core storage or release storage blocks, and make error and normal exits.

Monitor Facilities. A capsule description of the monitor might be as follows: it is a set of programs which input an application segment, give it CPU time, supply it with data and storage areas on request, handle its output to remote terminals and to interim storage, handle input from terminals, using this input and the self-descriptive section of the segment select the next segment in sequence, and, to complete the cycle, input the successor so selected. To relate these functions to the programs that perform them (see Figure 1), application segments are input by the peripheral I/O programs in the order indicated by the peripheral I/O scheduler. The principal executive routine, the work-in-progress scheduler, gives control of the CPU to the segment and interprets its requests for main storage, I/O from peripheral storage, and output to the associated console. These requests are passed on, respectively, to the storage allocation program in the executive, to the peripheral I/O scheduler, and to the remote I/O scheduler. The latter schedulers queue these requests to be performed by the peripheral I/O programs and the console I/O program (called "line control"). When segment execution is complete, the executive releases the main storage it has occupied and scans the self-descriptive part of the segment to determine whether this segment has indicated a unique successor and if so whether it is to be input at once or only after a response from the console. If there is more than one candidate in the table, selection of the successor will be performed by the input analysis routine, which compares the eventual user input with the criteria in the successor table (matching strings or matching input categories).

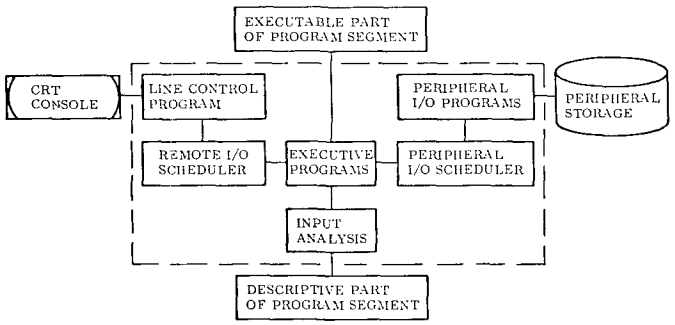

Fig. 1. Programs that make up the monitor

To summarize, the executive acts as interface between the application and the rest of the monitor, and calls in the other system routines as required to furnish the application with the facilities it needs. The Remote I/O Handler and Line-Control programs deal with transactions with remote terminals. The Input Analyzer selects the next segment in sequence. The Peripheral I/O Scheduler and Peripheral I/O Program handle the local data-bases, the application-program segment libraries, and interim file storage.

It should be noted that, for clarity, Figure 1 shows only one program segment and its descriptive part whereas normally several such pairs will be in core, either in execution or awaiting processing.

All the boxes inside the dashed lines represent core-

resident system programs. System programs which are infrequently used, such as diagnostics, are disk-resident.

Some of the monitor functions are implemented by program segments indistinguishable in form from the applications segments. This technique gives an almost unlimited range of potential facilities.

2. System Programs

DEFINITIONS:

Program Segment. The program segment is a self-contained logical unit of programming with two sections, one executable and the other used by the monitor primarily to determine what segment should succeed the one being processed. The executable portion of the segment can be at most 8 storage units in length (a storage unit is currently defined as 256 bytes) and actual execution time cannot exceed an arbitrary limit currently set at 600msec. There are arbitrary limits on other demands the segment can make of the monitor, such as on the total amount of data it can input.

Console-Dedicated Storage ("Console Area"). A 640-byte block of storage is dedicated to each (logical) console. The first 256 bytes of this area are used exclusively by the monitor. The descriptive part of the program segment occupies the first section of the remaining 384 bytes. The rest of this "segment description area" contains a segment-to-successor-segment communication buffer and a buffer for input from the remote console.

Scatter-Loading. Scatter-loading is a technique which permits properly prepared program segments and data to be input in fixed-length sections to noncontiguous blocks of storage. The scatter-loading unit in use is two storage units. Scatter-loading and the concomitant operation, gather-writing, are optional.

EXECUTIVE ROUTINES. The principal executive routines are the work-in-progress scheduler (WIPS), the service-call routines, the I/O interrupt handler, and the input analysis routine.

The function of the WIPS, the basic scheduler in the monitor, is to find and eliminate obstacles to program segment execution. It does this by supplying appropriate facilities: CPU time, core storage, and I/O, using the associated systems programs to do so. This system resource allocation is governed by constantly updated WIP status words, one word for each active remote console.

The service-call routines process requests made by program segments in execution. These include requests for data, scratch storage, output, etc. This technique was developed so that resource allocation would be controlled by the monitor rather than the application, to make it equitable and free of conflicts.

The I/O interrupt handler has the function of analyzing the source of such interrupts, calling the appropriate interrupt routine, and updating WIP status on completion of I/O activity.

The input analysis routine is called when a segment just executed has indicated that it expects a response from the corresponding remote console, and such a response has been received. (A program segment can indicate that it does not expect a response or that any response received is irrelevant to the selection of its successor. In either of these cases a "unique successor" is indicated.) "Analysis" consists of comparing the input with a "successor table" read into the console area by the segment just terminated in an attempt to match an input string with a key string.

Communication between the WIPS and other monitor routines is simple and standardized. The WIPS makes entries in the queues of the other routines (where applicable) and transfers control to them. Before they return, they set status bits in a communications area (the WIP status word).

The Work-in-Progress Scheduler operates by examining, in turn, the WIP status words stored one in each console area. When a set of simultaneous conditions is found to hold true for a given console, indicating that the current program segment is ready to start or to resume execution, control of the CPU is turned over to this segment. If the segment is not in this ready status, one of the pending operations required to put it in this status is initiated, and the WIPS goes on to test the status of the segment corresponding to the next console in sequence. When all consoles have been checked, the scheduler recycles to the start. This is a round robin scheduling as far as application segments are concerned, but an on-demand scheduling of execution of system programs.

The sequence of status tests, outcomes, and resulting actions can be summarized as follows:

Is the segment successor selection waiting for input from or output to the remote console?
- Yes: Go to remote I/O handler.
- No: Is there an unanalyzed console input in core?
  - Yes: Go to the input analysis routine.
  - No: Has the next segment been selected, but not input?
    - Yes: Go to the peripheral I/O scheduler.
    - No: (Note that if the tests reach this point the segment to be processed must be in working storage.) Is the segment waiting for a scratch area?
      - Yes: Go to the core storage allocator.
        (In storage allocator)
        Is storage immediately available?
        - No: Return to the WIP scheduler.
        - Yes: Allocate and continue.

      (Continuation)
      Is there any tape output pending?
      - Yes: Go to remote I/O handler.
      - No: Does the console have any pending requests for peripheral I/O?
        - Yes: Return to the WIP scheduler.
        - No: Set up the segment and put it in execution.

In the case of the "yes" responses (resulting from the determination that there is a requirement for some system resource or some further information before the segment can be input or can execute), the corresponding monitor activity is initiated and control is returned to the WIP

scheduler which then examines the status of the next active console. Two exceptions are pending requests for scratch storage and peripheral I/O which cause a direct return to the WIPS.

The Service Call Routines provide the means by which the program segment can request I/O, storage, and information from the monitor. They are called into action when the segment makes a properly coded SVC (supervisory call). This instruction generates a program interrupt and transfers control to a conventional location in the monitor area. Most service calls can request a number of "items" such as data records to be input, etc. Fifteen calls have been implemented, and as experience with the requirements of diverse applications grows further services may be added. The current list follows.

(1) *Segment exit.* This call returns control of the CPU to the monitor at the end of segment execution. It also initiates a monitor routine which checks for a unique successor, releases core storage, and resets the appropriate status words.

(2) *Input from disk storage.* One or more disk addresses are included in this request, either as hardware locations or keys to a directory file.

(3) *Input from data cell to disk.* This call was devised to permit a segment to request up to ten items (logical records) from data cell storage. These items are read into core storage and output to disk for the use of later segments.

(4) *Input from data cell to core.* This call will bring one item (logical record) from data cell to core storage. It can be used only when a unique successor is specified, as the input of the data will trigger the execution of the segment called in to process it.

(5) *Output to disk and exit.* This call causes the output of a logical recordtodisk storage after which the segment is terminated.

(6) *Output to data cell.* This call incorporates both core and data cell addresses and causes the SVC routine to make an entry in the data cell peripheral I/O queue. It implies segment exit.

(7) *Output to tape.* This call changes the status of a specified buffer to "output" status and makes an entry in the remote I/O queue. The call implies exit.

(8) *Output to remote console.* This call causes a change of status of a buffer. It contains a parameter specifying whether the output is to blank the CRT screen, to output "normally," to output with screen line-addressing, or to output to the typewriter associated with the console rather than to the CRT display.

(9) *Store halfword in console area.* This call permits the segment to store/alter a halfword in the (protected) console-dedicated area.

(10) *Store block in console area.* A block of characters specified by length and location is transmitted to the corresponding console area at a location indicated in the call.

(11) *Place console area address is base register.* The register is specified in the call. Control is returned to the segment.

(12) *Provide scratch storage.* The call specifies the quantity desired. The appropriate SVC routine calls in the storage allocation program to try to find space. If the request cannot be satisfied, control is returned to the WIP scheduler.

(13) *Release scratch storage.* This call is always satisfied at once, and control is returned to the segment.

(14) *Output to disk and wait.* This is the same routine as (5) (output to disk and exit) except that the segment is not terminated.

(15) *Error exit.* This is effectively a request for diagnostics. A core dump (to disk) is provided, and a map of the section of core the segment was using when it detected the error will be supplied.

Note that several service calls can be "stacked." Even though one or more of them imply segment exit, all will be processed before the SVC routines return control to the WIP scheduler.

The I/O Interrupt Handler will respond when the computer is in "problem state," that is, when application segments are being processed. When the monitor is in control, interrupts are disabled.

The routine, IOINT, responds to the interrupt first by identifying the console which had a segment in execution at the time the interrupt occurred and storing the program-segment status (the general registers, Program Status Word, timer clock value) in the corresponding console-dedicated area. Next it analyzes the interrupt to determine to which of three categories it belongs: remote I/O (including tape), peripheral I/O (disk or data cell), or other (console, printer, or card reader). In the first case, IOINT transfers control to the interrupt handler in the line control program, in the second case to the interrupt handler in the peripheral I/O program, and in the third case to the IBM 8K Basic Operating System interrupt handler.

The first two interrupt handlers are described in the sections on line control and peripheral I/O. Use of the BOS handler is restricted to I/O devices not currently used by the monitor. Later, when a facility for handling background jobs (programs executed while the CPU is momentarily idle) is added, interrupts from the card reader and the printer will be handled by the monitor.

The Input Analysis Routine is used to compare input from the user console to key strings left in console-dedicated storage by the segment which has just been executed. Each key corresponds to a unique successor segment, which will be input as soon as possible after it is selected. Many segments will have only one successor indicated; if this is the case, a "unique successor" flag is set.

The sequence of events in successor-selection is as follows.

At the time of segment exit, the exit routine checks the associated console area to determine whether the segment

has a unique successor. If not, it sets a flag to indicate that remote input will be required to determine the successor and thus that the input analysis routine will be needed.

The input analysis routine is called when the WIP scheduler checks the console in question and finds that an input has occurred, and finds that the "analysis needed" flag is on. Before it is called, the amount of input and the address of the first byte will have been stored in a conventional location.

The analysis routine starts by examining a fixed location in the console area to determine how many bytes of the input string are to be used in matching the input against the keys left by the previous segment.

Two comparison tables will have been left in the console area by the previous segment. The first contains blocks, each of literal character strings of a given length. The analysis routine selects the block of strings of length corresponding to the input (as modified by the length indicator) and compares this string character by character to the keys in the block. If an exact match is found, the segment linked to the key will be input.

If no match is found in the first, or specific, table of comparands, the input string is categorized by type (alphabetic, numeric, special character, etc.). The second table of comparands identifies keys by type and a match in the second table has the same consequences as a match in the first.

If this second attempt fails, a segment, whose address is found in a fixed console area location, will be input to deal with the "no match" case. This might be an input analysis routine written by the application programmer to supplement that provided by the system.

The input analysis routine, after identifying the successor segment, returns the successor segment identification and control of the CPU to the WIP scheduler.

Communication Areas. For each logical console, a two-section communication area (the "console area") is provided. The sections are contiguous and storage-protected. The first is used for the transfer of the information between program segment and monitor and between successive segments. The other is for communication between monitor routines. It contains the work-in-progress status word, provides storage for the program status word and general registers 2 through 13 when segment execution is interrupted, contains storage maps of segment and data section locations, etc. All console and system status information (aside from that found in I/O queues) is displayed in this area.

For each of the peripheral I/O devices (currently three), a queue of requests is loaded by an executive routine, and processed by the peripheral I/O scheduler described in the next section. An entry in one of these queues describes the I/O in terms of length, location on peripheral device, location in core (for output), request status, etc. There is provision for a priority ranking. Most entries in these queues are made in response to service calls issued by program segments. The queue is scanned to find requests which may be satisfied each time an entry is made and each time a peripheral I/O interrupt is detected.

PERIPHERAL INPUT/OUTPUT. Three routines are used to schedule and perform information transfer between core and peripheral (disk and data-cell) storage. They are the Peripheral I/O Scheduler, the Storage Allocator, and the I/O Program which (using IBM Basic Operating System Physical IOCS) actually makes the transfer. The Scheduler finds requests for service in its queue, calls the Allocator (on a request for input) to determine whether the request can be satisfied and, if so, gives the needed parameters and control to the I/O program.

The Peripheral I/O Scheduler (PIOS) is called each time an entry is made in its queues of requests for service and each time an I/O interrupt occurs in the selector channel. It then scans its queues in the following order: first data-cell, then disk 1, and finally disk 2, searching for a request which can be initiated at this time.

The first test determines whether a channel is available, then, if so, whether the device corresponding to the queue being checked is available (operational and not busy). If it is not, the next device is tested and if no device is available the PIOS returns control to the WIPS.

If a device is found to be available, its queue is scanned to determine if there is a pending request for I/O from/to the cylinder on which the read/write head is currently positioned.

If such an item is found, the next question is whether or not the request implies a new demand for core storage. If not, the I/O operation is initiated. If so, the allocation routine is called. If the latter finds sufficient storage, it is then allocated to this item and I/O initiated. If not, a request status bit is set to indicate "not enough core available at this time" and the next request is examined.

If no requests satisfy these criteria, the queue is scanned a second time to find the "best" item not on the current cylinder (currently "best" means "closest," but other factors will be considered later). A "seek" command is then issued, to move the read/write head to the cylinder containing this item.

If the chosen request demands no core storage, or if the allocator finds that adequate storage can be reserved at this time, a label is attached to the request: "no further storage needed." Whether or not allocation is successful, control then returns to the PIOS.

The scheduler will continue to scan its queues until either all have been scanned (and up to three seeks initiated) or until a read/write operation is initiated. Control is then returned to the WIPS.

As previously mentioned, the PIOS and executive communicate by placing requests and setting status bits in the peripheral I/O queue. One of the status flags set by the scheduler informs the executive that the request for a given item of I/O has been satisfied and hence that

the corresponding slot in the queue is available for a new entry.

The Storage Allocator. This discussion of the algorithm used for core-storage allocation requires three definitions:

(1) Free storage is storage not currently allocated (abbreviated FS).

(2) Storage-in-waiting is storage currently assigned to those program segments which have already been given all the space they need to complete their execution. Note that storage-in-waiting (SIW) will eventually become available no matter what other storage assignments are made.

(3) Usable storage is the sum of FS and SIW.

The storage requirements of a given program segment are classified as immediate (that corresponding to the current request) and future (the total needs of the segment less the amount already allocated). The immediate and future needs of a segment requesting storage are compared to the free and SIW storage, respectively. If either need exceeds the corresponding resource, the allocator rejects the request. If both can be satisfied, the allocator locates the free storage for the immediate need and tentatively updates the storage map.

At this point it would be natural, if it were not so costly in efficiency, to subtract the future requirements from the SIW pool and hence reserve all the space the segment will need. This is inefficient because the demands on space may be serial rather than parallel, and may occur only toward the end of segment processing. This technique would be wasteful in that it might reserve space never used, or not used more than a small fraction of the time during which it was reserved.

What is done is to consider the maximum momentary demand any one of the segments in core (or if the request is for the input of a segment, the requesting segment) can make on working storage. If this maximum will fit into usable storage (after the immediate need of the requesting segment has been satisfied), the request is granted. If not, a further criterion is invoked: is the requesting segment in core already? If not, the request is denied (the segment will not be input); if so, the request will be granted. In the latter case, as a precaution, space for the complete processing of the requesting segment is then reserved. In either case it will be seen that enough storage has been reserved so that at least one of the segments in core can process to completion. (For further detail, see [1].)

The Peripheral Input/Output Program has two aspects, that of handling interrupts in the selector channel and that of executing seeks and I/O operations at times and locations specified by the PIOS.

Each time a seek or an I/O is complete on the selector channel, the executive transfers control to the peripheral interrupt handler which analyzes the input to determine its source, its nature, and whether an error condition is signaled. If the interrupt was generated by a completed seek, control is transferred to the PIOS. If I/O is possible at this time the scheduler will return control to the I/O program (at a different entry point) to initiate the data transfer.

In the case of a completed I/O operation flagged as erroneous, several attempts are made to achieve correct transmission by repeating the operation. If these fail on a write operation and if it seems that the peripheral storage device is at fault, an alternate track assignment may be made. Other techniques are used to attempt to deal with other problems, the extreme being to inform the user that a file is inaccessible and that the application cannot continue.

If the interrupt was generated by a successful input or output, the nature of the operation is determined and the WIP status of the corresponding console updated. If a program segment was input, or a data input from disk or an output-and-wait performed, control then returns to the PIOS. If the interrupt was due to an input in response to a request to transfer information from data cell to disk, the pending record count is decremented and the record put in the peripheral I/O queue for output to disk. The other possibilities, both of which imply that the segment has exited, are input from data cell to core and output to disk and exit. When an operation of this type is complete as signaled by the I/O interrupt, the console WIP status is examined to determine if there are any more pending requests for I/O for this console. If not, the core storage allocation routine is called to release storage blocks held as output buffers.

In all cases the PIOS attempts to initiate further I/O and then returns control to the WIPS.

File Structures for Peripheral Storage. The basic organization of the disk and data-cell storage has been kept simple: only two file structures are currently used. One, the "directory file" on disk, can be addressed by an alphanumeric key of up to 256 characters in length. The other, the "data file" on disk or data cell, is addressed by its hardware location.

Physical records and directory logical records have a maximum length of 512 bytes. A data logical record or program segment may include up to four physical records. Input and output are in terms of logical records.

An area on disk is dedicated to each logical console to provide interim storage for the application segments. It is organized as a data file.

To permit the monitor to deal with the application files, a table describing them is maintained in core.

REMOTE INPUT/OUTPUT. The second major class of I/O the system must support is that from/to the CRT consoles. These will be called "remotes" even in the case in which the console controllers are directly connected to the multiplexor channel of the computer. This designation reflects the decision to deal with "local" and "remote" consoles in a uniform manner, as though at the end of a telephone line.

The two routines involved are the remote I/O handler

(RIOH) and the Line Control program. The latter is called by the former as well as by the WIPS.

The Remote I/O Handler is called by the WIPS when the dialogue at a given console cannot continue without an input from the console, or when the program segment being processed has requested that an output be made to the associated console. Different entry points are used in the two cases.

In the case of input, two resources must be available to satisfy the request—a core buffer and the transmission line. The first act of the RIOH is then to determine whether a 1-K (1024) byte block of storage is available. (This block size corresponds to the console buffer storage, the maximum amount of data that can be transmitted.) If not, a status flag is set and control returned to the WIPS.

If a buffer is found, a test is made to determine whether the transmission line to the given console is available. If not, a line-busy status flag is set and control returns to the WIPS.

If both buffer and line are available, the RIOH calls the line control program, giving as parameters the console identification and the location of the core buffer, and setting the appropriate console status for "console being polled." After the line control program starts the input operation, it returns control to the WIPS.

The next pertinent event is the end-of-transmission interrupt generated in the multiplexor channel. When it occurs, control is given to the interrupt-processor entry point in the line control program for analysis. If the latter finds that the interrupt flags a successful transmission, control is given to the RIOH. The RIOH tests to see if there was any input, and if not restores the console remote-input flag, releases the buffer storage to the allocator program, and returns to the WIPS. If there was an input, the RIOH sets the appropriate flag in the area describing remote input status ("awaiting input analysis") and tests to see if the input data is less than 33 bytes in length. If so, the data is transferred to the microbuffer in console-dedicated storage and the input buffer is released. Whatever its size, its length and the address of the first byte in the input string are stored in a conventional location in the console area. The RIOH then returns control to the WIPS.

In the case of *output* to remotes, only one resource, the transmission line, is needed. When called by the WIPS, the RIOH tests for line availability, and, depending on the result, either returns to the WIPS or transfers to the line control program after setting appropriate status flags. The line control program initiates the output and returns to the scheduler.

When the interrupt signaling successful output has been so interpreted by the line control program, control is given to the RIOH. The latter sets the console status for "output complete," releases the output buffer, and returns control to the WIPS.

The Line Control Program has two major functional parts, the interrupt handler and the programs that actually initiate the I/O operations.

The line-control interrupt handler is called by the interrupt routine in the executive when an interrupt occurs in the multiplexor channel due to remote I/O or output to tape.

It first analyzes the interrupt to determine its source and whether transmission was successful. If not, and if the error arose in transmission from console to CPU, the console will be disabled and a call put in for engineering aid. If the error appears in output to the console an application program segment may be called (if the application has prepared for this possibility) as this error might be unimportant or recoverable. If the error was in output to tape it will be ignored. In all these cases, it is assumed that normal recovery routines (re-reads, re-writes) have been attempted by the line control program and, for the message in question, have failed before further action was taken.

Conclusions

This approach works. It has, as intended, made it quick and easy to set up an online application (ten are in existence or in development) to permit a layman to create, retrieve, and manipulate files with the help of a computer.

The event-driven aspect, which simplified the design and cut down system overhead considerably, has not created any appreciable problems in applications programming.

Acknowledgments. The design and development of this monitor was definitely a group effort. The work-in-progress scheduler was conceived by D. B. Jack Bridges, who programmed all of the executive routines except for input analysis, the work of Stephan Burr. Dr. Allen Reiter developed the interesting algorithm and program for storage allocation, and programmed both the peripheral I/O scheduler and remote I/O handler. The peripheral I/O program and file design are due to E. R. Estes, and the remote I/O program to Sidney Shayer.

Equally important, all these gentlemen made valuable contributions in areas aside from those in which they had direct responsibilities.

RECEIVED MARCH, 1967; REVISED JULY, 1967

REFERENCE

1. REITER, A. A resource allocation scheme for multi-user on-line operation of a small computer. Proc. AFIPS 1967 Spring Joint Comput. Conf., vol. 30, pp. 1–8.

BIBLIOGRAPHIC NOTES

Programming Real Time Systems, by James Martin, Prentice Hall, 1965, is an excellent general discussion of the problems encountered and dealt with in the SAGE, Project Mercury, SABRE, PANAM, New York Stock Exchange, and other systems.

Information on the well-known Project MAC is available in Memoranda, Technical Report, and Progress Report form from Project MAC, MIT, Cambridge, Mass.