DOCUMENT RESUME

ED 088 476 IR 000 309

AUTHOR Fitzhugh, Robert J.
TITLE Laboratory Control With A Medium-Scale Time-Sharing System.
INSTITUTION Pittsburgh Univ., Pa. Learning Research and Development Center.
SPONS AGENCY National Science Foundation, Washington, D.C.
REPORT NO PU-LRDC-1973-25
PUB DATE Oct 73
NOTE 23p.; Paper presented at the Annual Meeting of the National Conference on the Use of On-Line Computers in Psychology (3rd, St. Louis, Missouri, October, 1973)

EDRS PRICE MF-$0.75 HC-$1.50
DESCRIPTORS Behavioral Science Research; *Computer Based Laboratories; Computer Programs; *Computers; *Computer Science; Costs; Input Output; *On Line Systems; Program Descriptions; *Time Sharing
IDENTIFIERS ETSS; *Experimental Time Sharing System; Interfacing; Memory Management; Multi Language Computer Systems; Task Scheduling

ABSTRACT

A description is provided of the Experimental Time-Sharing System (ETSS), a multi-language, general-purpose time-sharing system designed to support a wide range of computing applications. Included among these are the control of an on-line behavioral research laboratory. Major topics discussed in the report are: 1) the system's hardward configuration; 2) the operating system software; 3) interfacing and the control of laboratory devices; 4) the ETSS input/output command structure; 5) the initiation and timing of multiple input/output processes; 6) task scheduling and memory management; and 7) overall system performance and cost. (PB)

R&D 60

# THE R AND D CENTER

## RESEARCH AND DEVELOPMENT

### LOCALIZATION CONTROL WITH A TIME-SHARING SYSTEM

ROBERT J. FITZHUGH

1971

# LABORATORY CONTROL WITH A MEDIUM-SCALE TIME-SHARING SYSTEM

Robert J. Fitzhugh



Learning Research and Development Center

University of Pittsburgh

October 1973

The research reported herein was supported by a grant from the National Science Foundation (NSF-GJ-540X) and by the Learning Research and Development Center, supported in part by funds from the National Institute of Education, United States Department of Health, Education, and Welfare. The opinions expressed do not necessarily reflect the position or policy of the sponsoring agencies and no official endorsement should be inferred. Paper presented at the Third Annual Meeting of the National Conference on the Use of On-Line Computers in Psychology, St. Louis, Missouri.

## Abstract

A multi-language, general-purpose time-sharing system designed to support a wide range of computing applications including the control of an on-line behavioral research laboratory is described with particular emphasis on the system's laboratory control capabilities.

# LABORATORY CONTROL WITH A MEDIUM-SCALE TIME-SHARING SYSTEM

Robert J. Fitzhugh

Learning Research and Development Center

University of Pittsburgh

During the past decade, there has been a continuing and growing debate in the computing field between the proponents of extremely large, utility-like computer systems each serving many users with diverse computing requirements, and the proponents of small, single-application systems each serving a relatively homogeneous population. The issues involved are complex, and the debate has yet to be resolved. Those who favor the large utility argue that there are economies of scale in computing, just as there are in electric power generation, and that a single, very large system is far more cost effective than multiple, small systems each with its own support facility and staff. In addition, it is argued that a large system can offer a powerful and varied range of computing services some of which cannot be provided by a smaller system. Large-scale number crunching is an example. The small-computer enthusiasts counter by pointing to the dramatic decline in the cost of small computers that is making them increasingly cost effective Microcomputers the size of a pack of cigarettes are already available for under $200. They argue that the very large systems that seek to be all things to all users have grown cumbersome, overly complex, and difficult to manage. A substantial amount of the memory and the processing power of these machines is often comsumed by massive, general-purpose operating systems.

It is clear that the small and large computer issue will not be settled for some time. In fact, a probable outcome is that neither side will prevail and that there will be a mix of large and small depending upon application. For the control of an on-line psychological research laboratory, most workers in the field would agree that the small, single-application minicomputer is more appropriate than a large, shared processor. Historically, this has certainly been the case. University computing centers have been unwilling and, more often, unable to provide the services required. The dedicated minicomputer is usually the only solution for the psychological researcher interested in on-line experimentation. However, as many will attest, this is a path fraught with difficulties. Although a number of special-purpose operating systems and languages have been developed in recent years, most of the small systems in use are difficult to program and operate, particularly when compared to large systems and the high-level services they provide. A disproportionate amount of time and effort often is spent preparing for rather than running an experiment.

As a possible solution to these problems, a multi-language, general-purpose time-sharing system designed to support a wide range of computing applications including the control of an on-line, psychological research laboratory has been developed and is operational. Based on a medium-scale DEC PDP-15, the system, called ETSS for Experimental Time-Sharing System, has been successfully operating twelve hours each day, five days a week for nearly two years. The development of this system grew out of the belief that there is a viable third alternative to either a very small, special-purpose laboratory computer system or to a large, general-utility system that is shared by a great many users with diverse and often conflicting requirements.

Hardware Configuration

ETSS is currently operating on two independent computer systems. The original and primary implementation described here is based on a DEC PDP-15 computer housed in a specially modified van and located at a suburban elementary school. ETSS was subsequently installed on an older PDP-7 computer used largely for batch processing and located at the University of Pittsburgh. The PDP-15-based system is shown in Figure 1 and includes 32K words of memory, memory protection/relocation hardware, a one million word Vermont Research Drum, two IBM 2314-equivalent disk drives, two Dectape drives, papertape equipment, several internal clocks, a telecommunications controller, a hardware system bootstrap, and a terminal controller capable of controlling 64 terminals of which 32 are currently installed.

The high speed drum is used as a program swapping device and is modified to transfer four tracks in parallel to increase the data transfer rate. This modification is relatively minor and of nominal cost. The terminal controller can control terminals of varying data rates each transmitting and receiving 8-bits bytes bit-serially. The four clocks are used by the operating system for a variety of internal purposes including the accurate timing of input/output events. The remaining components of the hardware configuration including the central processor are standard and are unmodified with the exception of the hardware system bootstrap. This inexpensive unit operates through the papertape hardware read-in mode logic and permits the operator to load and start the operating system with the push of a button. Using an inexpensive read-only memory, the hardware bootstrap loads into memory and starts a 32-word program which, in turn, loads and starts a program from the drum. This speeds and simplifies the loading of the operating system and is particularly useful with unskilled operators.

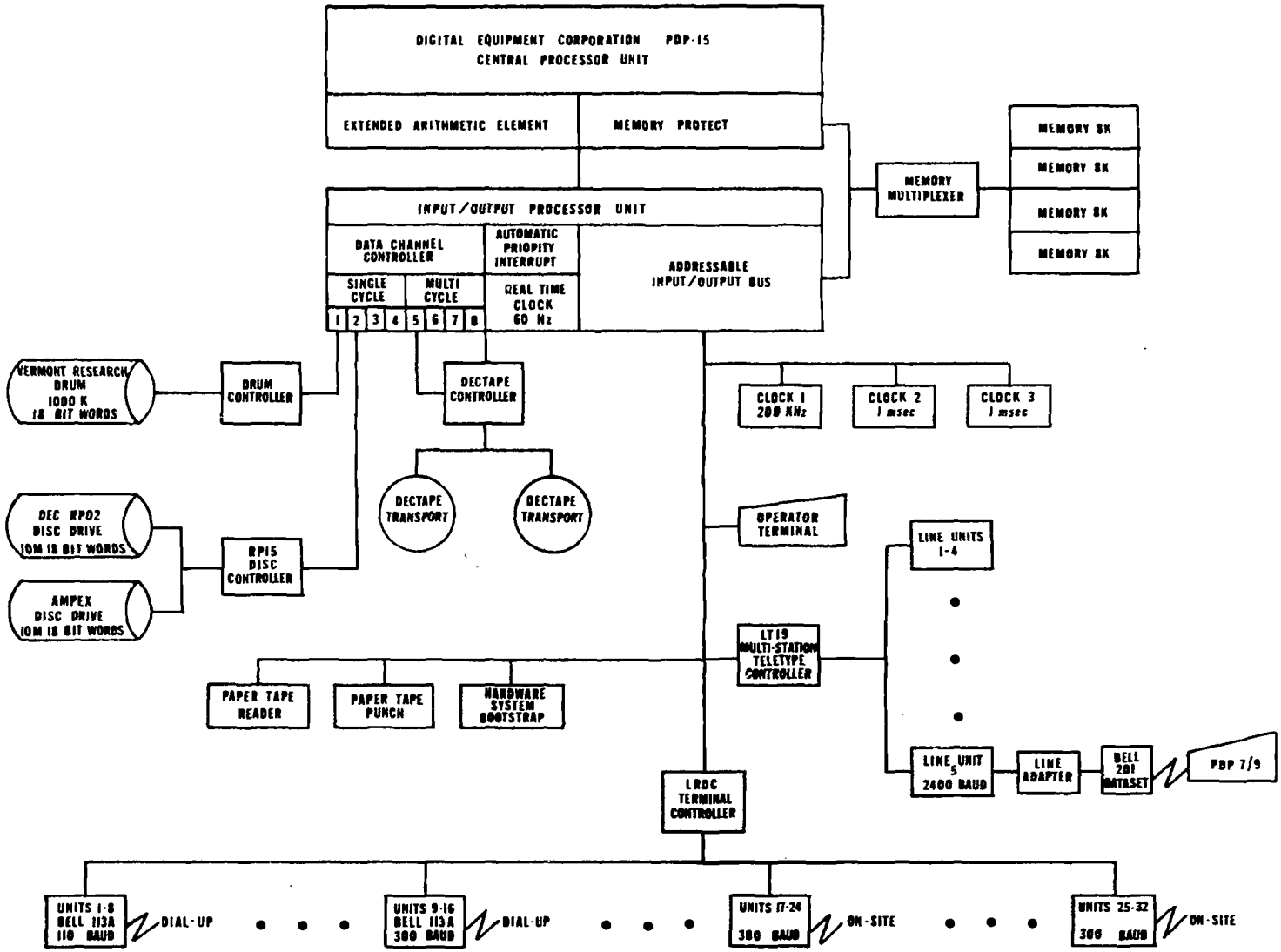


**Figure 1. LRDC Experimental Time-Sharing System Hardware Configuration**

In general, the total hardware configuration is relatively standard. The nonstandard, specially constructed or specially modified components were required simply because the appropriate hardware was not available when the system was constructed several years ago.

## Operating System Software

The ETSS operating system consists of five major procedures, a collection of peripheral device control subprocedures, a body of common subroutines and a variety of tables and context blocks, some permanent and some dynamically created and destroyed through time. These operating system components reside in a portion of main memory called 'syspace' for system space. User programs execute in the remaining portion of memory called 'uspace' for user space. When required, portions of uspace may be transferred to and from an auxiliary swapping device although, as will be shown, the system managers have the ability to selectively limit swapping or to eliminate it entirely if this is required by a fast-response laboratory application.

Although the five procedures share a body of common subroutines, each procedure is independent of the others and is responsible for a major system function. Figure 2 shows the five procedures and the lines of communication between them. The EXECUTIVE procedure is the single most important procedure and is responsible for the allocation and control of all memory and computational resources. Any procedure can request the EXECUTIVE to create a 'task' or job with a specified set of characteristics and schedule it for execution. During the task's lifetime, it is under the exclusive control of the EXECUTIVE although the procedure that requested its creation can also request that it be suspended or destroyed at any time.

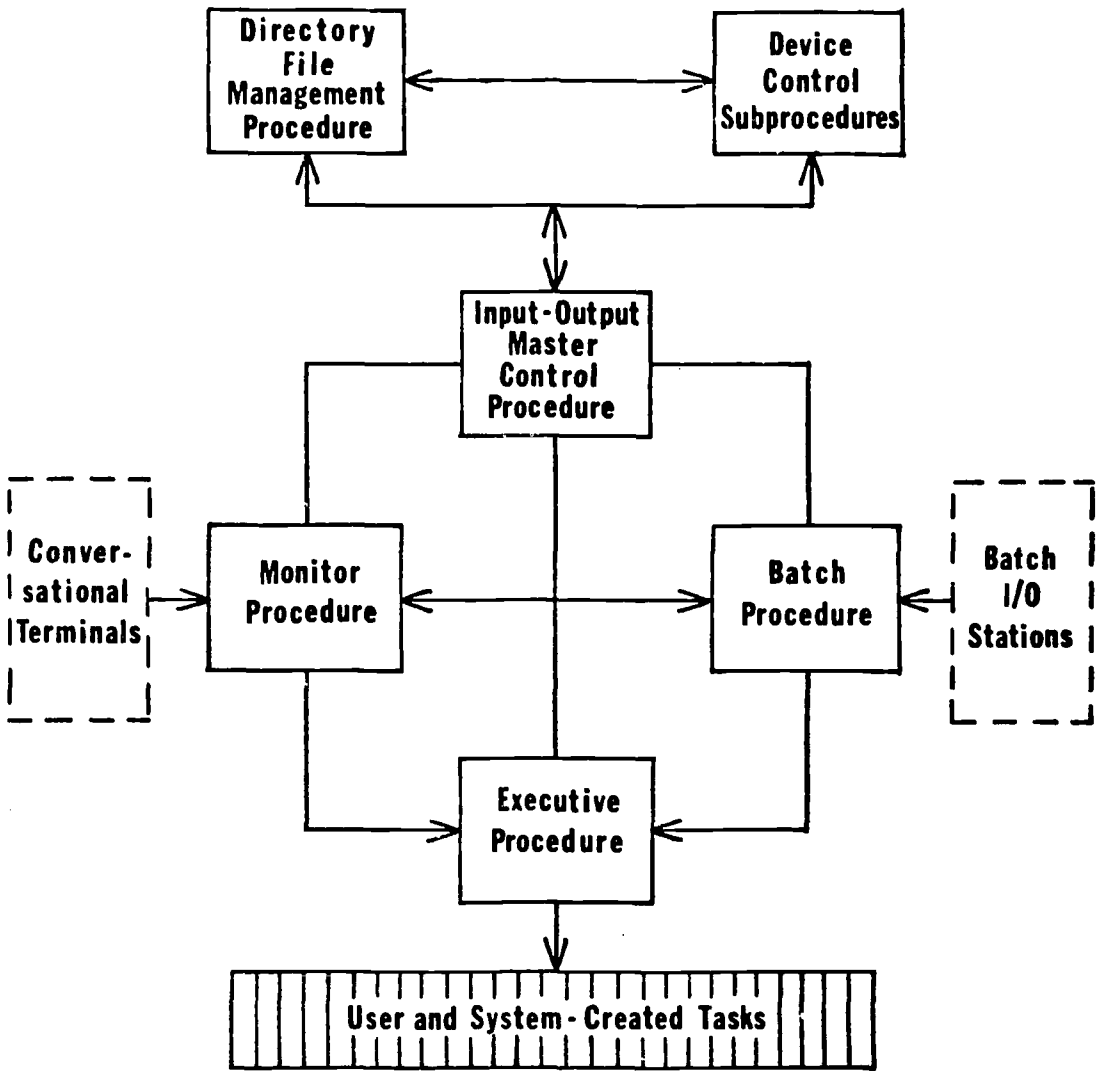


Figure 2. Major System Procedures and Their Lines of Communication

Requests to create and destroy tasks most commonly come from the BATCH and MONITOR procedures although the INPUT/OUTPUT MASTER CONTROL (IOMC) procedure does request the creation of I/O spooling tasks. The BATCH and MONITOR procedures each function as a software interface between the EXECUTIVE procedure and the human users of the system. The MONITOR procedure supports conversational time-sharing and provides the user at a terminal with a MONITOR Command Language (MCL) through which the user is able to enter and exit the system, initiate tasks, acquire and return system resources, display and modify dataset directories, request information on system status and performance and invoke a wide variety of language processors and utilities. The BATCH procedure maintains one or more batch processing streams and interacts with batch input/output stations. A BATCH Command Language (BCL) is used to specify tasks to be run. Although the MONITOR and BATCH procedures may create tasks with different characteristics, the tasks are indistinguishable to the EXECUTIVE and are able to share all system resources including a common file structure.

The INPUT/OUTPUT MASTER CONTROL (IOMC) procedure is responsible for the allocation and control of all input/output resources including I/O channels, space on auxiliary storage devices and access to system peripherals. By masking hardware idiosyncrasies, the IOMC procedure enables user programs and other system procedures to interface with a virtual input/output structure in which programs reference 'files' assigned to any appropriate device or dataset. The IOMC procedure serves as a translator converting file-oriented I/O requests into requests to actual physical devices or datasets.

During the processing of a file request, the IOMC procedure determines if the request refers to a 'directory' or to a 'nondirectory'

device. Directory devices contain data as well as directories with the names and the locations of the permanent entries of 'datasets' stored on the device. A disk unit might be used as a directory device, and a line printer would be a typical nondirectory device. If the file request refers to a nondirectory device, the IOMC procedure interacts directly with the appropriate device control subprocedure. All requests referencing directory devices are sent by the IOMC procedure to the DIRECTORY FILE MANAGEMENT (DFM) procedure.

The DFM procedure permits the programmer to treat a directory device as a file-oriented medium without any concern for the device's physical properties or addressing structure. Data written to a directory device can be left in a temporary state or it can be cataloged as a dataset. Datasets may be cataloged in a user's private directory, in a directory available to all called the 'LIBRARY,' or in the directory of another user if permitted. Protection keys are provided to restrict dataset access when required. During the processing of a request, DFM interacts with the one or more device control subprocedures responsible for the control of the actual physical devices involved.

## Interfacing and Control of Laboratory Devices

The ETSS strategy for the hardware and software interfacing of 'non-standard' laboratory devices is to develop character-oriented interfaces that mask, as much as possible, any idiosyncratic control characteristics. Whenever possible, the interface is simply an encoder/decoder unit, called an Intervening Black Box (IBB), that stands between the device and an ordinary time-sharing terminal port on the computer. No special operating system software is required, and the device appears to accept and generate ASCII records. If an IBB is not possible and the

device must be interfaced directly to the processor, the hardware interface or the operating system software interface performs these encoding/decoding functions.

The result is an orderly and systematic, virtual input/output structure that is not cluttered with one-of-a-kind commands. Since all laboratory device input/output is in the form of conventional ASCII records, programs controlling unusual devices can be written in a commonly available higher-level language such as FORTRAN IV or in a special-purpose experimental control language if one is available. In neither case do the languages need to be modified as new devices are added or existing ones altered since the operating system input/output structure remains uniform. If the interface is an IBB on a terminal port, the operating system software need not be modified as well. This is, perhaps, one of the most attractive outcomes of the Intervening Black Box approach as frequent system modifications often lead to lower operating reliability. In addition, the operating system remains smaller in size than it might be if special I/O software was required for each laboratory device.

It is the dramatic decline in the price of electronic components and in computer memories that has made this character-oriented interface approach possible. With relatively inexpensive computer memories, larger, more sophisticated operating systems with the big machine features of ETSS are possible on smaller computers. Cheap electronic components with low power requirements permit low cost, compact IBB's to be constructed for the control of unusual devices. For example, Texas Instruments, Inc. now markets a single 'transceiver' chip for under $7 that performs all the functions of the two $150 Teletype transmitter/receiver modules DEC uses on the PDP-15. As an illustration of the

sophisticated functions that are available in single, solid-state components, a bidirectional Hollerith-to-ASCII and ASCII-to-Hollerith code converter on a single chip is available for under $50.

Two interfacing examples will illustrate this character-oriented interface approach. The first describes a random access audio device in which the significant portions of the interface are in operating system software; the second is a touch-sensitive surface that is interfaced to a conventional terminal port using an IBB. The random access audio device consists of a closed-loop, mylar belt six inches wide that moves past fixed read/write heads. The heads can be moved across the belt to selected recording 'tracks.' There are 128 tracks, each eight 'segments' in length, where a segment is approximately one second of recorded sound. The hardware interface for this device was constructed a number of years ago and is rudimentary and low level making the device cumbersome to control. The operating system software interface, however, masks the complex control characteristics and permits the device to be controlled with ASCII records.

When programming on the assembly language level, the first step is to ASSIGN a file name to the device. The file is then OPENed causing the audio device to reset in preparation for a subsequent command. Thereafter, the program WRITEs records to the file, and it is the contents of those records that specify some action to be performed by the audio device. The first character of each record specifies the function (move, play, record), the next three specify a track number, the next character identifies a starting segment number and the next four specify the number of segments to be played or recorded. These last eight characters may be repeated up to 30 times in a record so that a single audio message can be created from a number of discrete pieces recorded in noncontiguous locations on the tape.

The touch-sensitive surface is an 18 by 18 inch translucent surface upon which slide images can be displayed from the rear using a conventional Kodak slide projector under computer control. The surface is designed so that the computer can determine the X-Y coordinates of any point touched. An IBB allows the unit to be controlled over a terminal port and accepts control information in the form of ASCII records. A teletype also can be connected to the IBB, and the ASCII records will be directed to it if the IBB is switched to the 'teletype mode' by a special control character. The teletype is used most often at the beginning of an experiment to enter subject-identifying information although it can be used alternately with the touch-sensitive surface throughout the course of an experiment. All control information to the unit, including slide numbers to be displayed, is in the form of ASCII records as is data generated by the unit. Because the IBB operates off an ordinary time-sharing terminal port, the touch-sensitive terminal can be located outside the laboratory and operated over a dial-up data-phone link.

ETSS Input/Output Command Structure

User programs request I/O services through the use of programmable EXECUTIVE CALLS or EXCALLS. Through the EXCALL, the programmer is able to draw upon a powerful set of device independent input/output and file manipulation services and is relieved of the responsibility of direct device control. Since all input/output requests specify logical files rather than unique physical devices, the actual device assignments can be altered prior to the execution of a program either through direct MONITOR commands at the user terminal or through programmable EXECUTIVE calls. This full device independence is extremely useful in a laboratory environment, particularly during

program debugging. Since all device control information is in the form of ASCII records, these records temporarily can be directed to the programmer's terminal permitting the programmer to debug and test a program from any time-sharing terminal. Laboratory equipment is not tied up through lengthy debugging sessions, and the actual equipment need only be used during final program check-out.

In order that the average programmer faced with a conventional programming problem need not learn and understand a large number of complex, technical commands, the overall ETSS design philosophy was to include in the EXCALL command set only a limited number of general and logically consistent functions. The experienced programmer faced with a complex or unusual laboratory control problem is provided with a set of 'modifiers' which can be appended to the general commands if additional options and control features are required. Although space limitations prohibit a full description of each command, the input/output EXCALLS recognized by ETSS are listed to provide a feel for the command structure and its level of generality:

1. ASSIGN . . .(Establishes an association between a file name and a device or dataset.)
2. DEASSIGN . .(Destroys that association.)
3. OPEN . . . .(Readies a file for subsequent activity.)
4. CLOSE . . .(Places a file in an inactive state.)
5. READ . . . .(Receives a record from a file.)
6. WRITE . . .(Transmits a record to a file.)
7. CONTROL . .(Performs record skipping and other operations.)
8. WAIT . . . .(Wait, conditionally or not, for the completion of some file activity.)

9. TEST . . . .(Test on the active/inactive state of a file.)
10. DELETE . .(Delete a DATASET.)
11. RENAME . .(Rename a DATASET.)
12. CATALOG. .(Catalog a DATASET.)

In addition to modifying these basic EXCALLS, the programmer also is able to test for the occurrence of a wide variety of input/output errors and conditions through a feature called the 'selective notification of exceptional conditions.' Through bit settings in the EXCALL, the program can specify whether or not control should return to the program or to the terminal MONITOR in the event of specified errors or exceptional conditions. If the program fails to request notification and an exceptional condition occurs, the operating system will terminate the program, display a diagnostic message on the user's terminal, and place the user in communication with the terminal MONITOR. If notification is requested, control will be passed to a notification address specified in the EXCALL and a bit pattern will be stored specifying which exceptional conditions have occurred.

This exceptional condition feature was designed to enable a program to retain full control over its environment even in the face of unusual conditions or device malfunctions. In addition to the usual tests for parity errors and the like, a program running under ETSS can test for conditions such as an illegally formatted dataset name within an EXCALL or an attempt to perform a file operation without previously opening the file. This ability to selectively request notification for over 40 exceptional conditions provides a high degree of programming flexibility and control that needs be exercised only when required. A programmer faced with a conventional and straightforward problem can simply refuse all notification and need not be burdened with the task of writing unnecessary error handling routines.

Initiation and Timing of Multiple I/O Processes

In order to permit a program to initiate multiple and simultaneous input/output processes and to compute while an input/output process is underway, the input/output EXCALLS allow the program to specify if control should be returned immediately or only after the requested service is complete. If the program does not elect to wait, control is returned immediately, and the program is free to continue computing or to make another EXCALL request. The program then must have some means of determining when the file process is complete and if it was successful. Two separate EXCALLS are provided for this purpose.

The WAIT EXCALL suspends the program until some previously requested file process is complete. The TEST EXCALL is a variant of the WAIT EXCALL and permits the program to test whether or not a file process is complete without the risk that the program will be placed in a wait state. TEST is useful in certain laboratory or process control applications where a high degree of device control is required. In order to avoid unnecessary delays, or waits, on slower devices, a program wishing to maintain or monitor several simultaneous input/output processes would use the TEST instead of the WAIT EXCALL. However, WAIT must be called prior to making another input/output request of a file even though the TEST EXCALL indicates that the file process is complete.

An additional degree of control over an input/output process is provided by the 'conditional time-delay' option of the WAIT EXCALL. If a time-delay is specified, control is returned when the file process is complete or upon the expiration of the time-delay, whichever occurs first. The expiration of a time-delay is an exceptional condition, and if notification is requested, control will be passed to the notification

address if the time-delay expires prior to the completion of the input/output process. As with the occurrence of any exceptional condition, the file process is interrupted and is not allowed to proceed. This conditional time-delay feature is a requirement of many laboratory applications in which a subject at an experimental device is given a limited and measured time to respond to a program-generated stimulus.

As a second option of the basic WAIT EXCALL, the program also can request that the time of day when a file process begins and ends be stored as an ASCII record in a program-specified buffer. This timing information is accurate to the millisecond level regardless of the time-sharing load on the operating system and is most often requested to measure subject response latency.

ETSS Task Scheduling and Memory Management

A major feature of ETSS is its ability to support a mix of jobs or 'tasks' including fast response laboratory control tasks, terminal-oriented conversational time-sharing tasks and batch processing tasks in one or more batch streams. System resources are dynamically allocated to the tasks based on criteria established by the system manager. The EXECUTIVE procedure within the operating system is primarily responsible for this task management and resource allocation function.

Any procedure within the system can request that the EXECUTIVE create a new task with a unique set of characteristics called a 'task profile.' The task profile fully defines the task and its relationship to other tasks in the system and specifies the extent to which it can gain access to system resources. Included in the task profile are parameters defining the task's priority range, initial quantum class, preemption class, memory size class and memory residency class. By controlling

the type and mix of the tasks that are created, procedures other than the EXECUTIVE can perform high-level or 'course' scheduling. However, once created, the low-level or 'fine' scheduling of a task is under the exclusive control of the EXECUTIVE procedure.

The EXECUTIVE maintains multiple queues of tasks that are ready to run, one queue for each priority level, and a task's priority range defines the highest and the lowest priority levels it can assume during its lifetime. The initial quantum class specifies the size of the time-slice the task is given when it begins execution for the first time, and the memory size class specifies the maximum amount of memory the task can acquire. During the lifetime of a task, it will tend to drift upward and downward in priority, within the constraints of the priority range, depending upon CPU and memory usage. As a task drifts downward in priority because of heavy CPU usage, it will be assigned successively longer time-slices although it will be allowed to run less frequently. Memory and CPU usage are additive in their effect upon priority so that, if permitted, a large, compute-bound task will drift to a lower level than will a small, compute-bound task or a large, I/O-bound task. To ensure that extremely active higher priority queues do not lock out tasks on lower priority queues, each queue is assigned a 'queue ratio' that specifies the frequency with which a queue must be serviced even though higher priority tasks are ready to execute.

The task also is assigned to one of three preemption classes. A ready to run task in the 'immediate preemption' class will immediately preempt a currently executing task of lower priority. A task in the 'delayed preemption' class will immediately preempt lower priority memory resident tasks and swapping tasks that have been in memory for more than one time quantum. However, if the task to be preempted

was swapped in immediately prior to its execution, preemption will be delayed until the executing task has been allowed to run for a specified period of time. The intent is to avoid unnecessary swapping while still ensuring that the higher priority task receives service within a short period of time. A task in the 'no preemption' class is not permitted to preempt a lower priority task until that task has completed its time-slice, made an input/output request or is removed from the processor for some other reason.

Memory space for tasks is allocated and deallocated dynamically. Fixed-size partitions are not used, and through EXCALLS, tasks can acquire and release memory as required. The task profile assigns each task to one of four 'memory residency' classes. Tasks in classes one through three are eligible for swapping with each successive class having a higher 'sticking priority.' The sticking priority determines the order in which tasks are to be swapped with tasks with a lower sticking priority eligible for swapping before tasks with a high sticking priority. Class four tasks are memory resident and are never swapped.

This scheduler and memory management design permits the system to be tailored to a wide range of applications. By altering the queue ratios and the task profiles of the tasks created by the MONITOR and BATCH procedures, the operating system can be dynamically reconfigured while the system is running. For example, if the application is foreground, laboratory control and background program development in a time-sharing mode, the laboratory control tasks would be created with a high sticking priority and a high priority range and would be placed in the immediate or delayed preemption classes. The background program development tasks would be assigned to a lower priority range and would be given a lower sticking priority. If necessary, the foreground

laboratory control tasks could be made memory resident if extremely fast response time is required.

Within each type of task (i.e., laboratory control or program development), the scheduler will dynamically allocate computational and memory resources based on task performance and will favor small, I/O-bound tasks over large, compute-bound tasks. However, the priority range concept ensures that the laboratory control and program development tasks do not overlap in priority unless this is felt to be desirable and is specified. Depending upon the queue ratios that are assigned, response time for the background program development tasks can be maintained at some minimum level, or response time can be permitted to degrade to the lock-out level during periods in which the laboratory control tasks are extremely active.

Performance and Cost

ETSS has been successfully operating more than 300 hours each month for nearly two years. Actual system up-time measured as a percentage of scheduled up-time exceeds 99 percent for the entire period and mean time before failure is in excess of one operating month. Usage has steadily increased throughout the period, and total connect time for time-sharing users now averages approximately 200 hours per day. Of the 50 million characters of available on-line disk storage, approximately 25 million have been allocated in more than 5000 datasets. For ordinary time-sharing users who are not favored by the scheduler, response time to a conversational command is in the .1 to 1.0 second range 99 percent of the time with worst case response times in the 1.5 to 2.0 second range. The average program size for a time-sharing user during the busiest six hours of the day is 14.5K words.

ETSS currently offers as on-line subsystems a text editor, a macro assembler, a FORTRAN IV compiler, a linking loader, an expanded version of DEC's FOCAL, a string-processing language called T64 similar to Mooer's TRAC, an on-line debugging aid called DEBUG, a statistical package, a file and data manipulation package and a variety of utility programs.

The total cost of the original PDP-15-based prototype constructed four years ago is more than $200,000. This figure includes equipment that was acquired for strictly research and development purposes that would not be required by an operational system in the field. A more powerful PDP-11-based version of the system could be constructed today for $50,000 to $90,000 depending upon disk storage requirements. The basic system, excluding disks, would include a PDP-11/40, a one-half million word swapping drum, 65K of DEC and independently acquired memory, some miscellaneous hardware, and a 32 port terminal controller capable of controlling terminals running from 110 to 2400 baud.

## Conclusion

ETSS is an operating demonstration that a powerful, general-purpose operating system with the capability to support on-line laboratory applications, conversational time-sharing and one or more batch streams is possible on a medium-scale computer. As equipment costs continue to decline, a system such as ETSS providing large computer features and programming flexibility should become an increasingly attractive alternative to the small, specialized minicomputer in on-line psychological research.

## References

Fitzhugh, R. J. LRDC experimental time-sharing system reference manual. Pittsburgh: Learning Research and Development Center, 1970. 3 Vols.

Fitzhugh, R. J. LRDC experimental time-sharing system internal reference manual. Pittsburgh: Learning Research and Development Center, 1971. 2 Vols.

Fitzhugh, R. J. Oakleaf Computer Project. Pittsburgh: Learning Research and Development Center, February, 1972.

Fitzhugh R. J. & Katsuki, D. The touch-sensitive screen as a flexible response device in CAI and behavioral research. Behavioral Research Methods and Instrumentation, 1971, 3(3), 159-164.