MASSACHUSETTS INSTITUTE OF TECHNOLOGY
A. I. LABORATORY

Artificial Intelligence
Memo No. 238 April 1972

ITS STATUS REPORT

Donald E. Eastlake

ABSTRACT

ITS is a time-shared operating system designed for the Artificial Intelligence Laboratory DEC PDP-10/PDP-6 installation and tailored to its special requirements. This status report described the design philosophy behind the ITS system, the hardware and software facilities of the system implemented with this philosophy, and some information on work currently in progress or desirable in the near future.

Work reported herein was conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under Contract Number N00014-70-A-0362-0003.

1. ITS System Status Report

The ITS system is an operating and time-sharing system tailored to the hardware of the Aritificial Intelligence Laboratory DEC PDP-10/PDP-6 computer system and the special hardware and software development requirements of this research installation. The following sections describe the current state of the system and include comments on its development to its current maturity, possibilities for further enrichment, and the considerations behind certain system design choices.

The following persons have been involved in recent ITS development: Jeffrey B. Rubin, Thomas F. Knight, Richard D. Greenblatt, and Donald E. Eastlake. The following persons were involved in early ITS development: Fredrick H. G. Wright, Stewart B. Nelson, Thomas F. Knight, John T. Holloway, Richard D. Greenblatt, Jerry S. Freiberg, and Donald E. Eastlake.

(At this time a new ITS Reference Manual is being prepared. Those interested in using a feature mentioned herein that is not explained in the current ITS Reference Manual should contact one of the persons named above.)

1. ITS System Status Report 1

2. Design Philosopy 2

2(a). The Level of Service 2
2(b). The Personnel Techniques 3
2(c). Is Protection Necessary ? 4
2(d). Implementation in Assembly Language 6
2(e). The Organization of User Procedures 7
2(f). Procedures as a Representation of Procedure Status 10
2(g). The Scheduling of Procedures 14
2(h). The Debugging of Programs 16
2(i). The Transaction of Input-Output 18
2(j). Input-Output Buffering 20
2(k). Is Compatibility Necessary ? 22

3. Hardware 25

3(a). Memory, Paging, and Swapping 25
3(b). The Dual Processors 30
3(c). Secondary Storage 31
3(d). Display Facilities 34
3(e). Vision Facilities 38
3(f). Analog Input and Output 40
3(g). User Terminals 43
3(h). The Plotter and IPL Facilities 48
3(i). Clocks 49
3(j). Miscellaneous Hardware Devices 50

4. Additional Software Details 52

4(a). Daemon Procedures 52
4(b). Inter-Procedure Communication 55
4(c). Disowned Procedure Trees 56
4(d). Direct Input-Output Instructions 58
4(e). Software Interrupts 60
4(f). Miscellaneous Software Devices 61

5. Work in Progress 63

5(a). The ARPA Network 63
5(b). The Mathlab System 64

6. Recommendations 65

6(a). Hardware Development 65
6(b). Software Development 65

Bibliography 67

2. Design Philosophy

2(a). The Level of Service

The prime purpose of the ITS system is to improve the efficiency of utilization of both the human and hardware resources of the Artificial Intelligence Laboratory. The nature of the advanced research use to which the system is put dictates the desirability of a high level of service to a relatively limited number of users. Sometimes, in fact, an extremely high level of service to one user experimenting with an advanced real-time program is necessary and a simultaneous low level of service to other users is tolerable (which will still allow them to perform editing or other lower level functions).

The ITS system resides in the DEC (Digital Equipment Corporation) PDP-10, where normal user programs are executed. It controls almost all input-output and it allocates hardware resources among users. One of these resources is the PDP-6, an older nearly compatible computer. The availability of this additional computer with significant input-output capability, free from interference by other users, can solve the most demanding real-time control requirements (see section 3(b) below). Nevertheless, numerous features are incorporated in

ITS to enable unequal resource allocation to a certain procedure or procedures running on the time-shared computer.

Rapid response, especially to the more important users, is one goal of ITS. User program time quanta are sufficiently short to frequently allow response within a teletype character time. Although swapping has been added, the large core memory available (see section 3(a) below) enables most active user procedures to be kept in main memory as well as allowing researchers to efficiently work on large programs. Most swapped out procedures are dormant or disowned (background).

The recent growth in the number of users accomodated has resulted in some reduction in services especially when one researcher is heavily loading the system resources. In early operation, a peak load on ITS was six or seven users but now there are 39 ports (including 7 with at least character display capability) and 16 users is not uncommon.

2(b). The Personnel Techniques

The ITS system is not the result of a human wave or crash effort. The system has been incrementally developed almost continuously since its inception. It is indeed true that large systems are never "finished." As the system has matured there have always been new features to add to those under consideration as others were implemented or discarded.

Normally there are two or three persons most actively working on the time-sharing system. In several respects this number seems optimal. The organizational problems of a large group are avoided while the problems of a single person "getting stuck" or basing changes on a limited single point of view are lessened. Major changes are normally widely but informally discussed before implementation to form a wider base of opinion.

In general, the ITS system can be said to have been designer implemented and user designed. The problem of unrealistic software design is greatly diminished when the designer is the implementor. The implementor's ease in programming and pride in the result is increased when he, in an essential sense, is the designer. Features are less likely to turn out to be of low utility if users are their designers and they are less likely to be difficult to use if their designers are their users.

2(c). Is Protection Necessary ?

Most time-sharing systems have extensive secondary storage allocation and protection mechanisms. Most time-sharing systems go to great lengths to assure that a user is "authorized" and that no ordinary user procedure can damage the system or disrupt input-output. In ITS almost any user procedure can trivially read, write, or delete any user's disk files and can store in any absolute memory location (possibly clobbering ITS) or perform direct hardware input-output instructions (possibly disrupting ITS input-output). Over many years, with hundreds of different persons using ITS at the Artificial Intelligence Laboratory, no significant problems have occurred due to this lack of protection.

The situation might be different if sensitive files were frequently stored on the Artificial Intelligence Laboratory system or its absolutely continuous operation were vital. But often the sensitivity of files is overrated. It would appear that in a cohesive open research environment most "protection" mechanisms are a counter-productive encumberance. They divert the efforts of systems implementors, inconvenience most users, and for some users act as a challenge which diverts their efforts to warring with the protection implementors.

Just the right number of safeguards have been installed in ITS to reduce to an acceptable level the probability of accidentally damaging the system or another user's files. The ability to execute direct input-output instructions and to modify the system while it is running are powerful tools that have been used with due respect for their dangers by the advanced researchers that ITS is written to serve.

No "password" mechanism currently exists in ITS to stop anyone from using the system. Initially there was no restriction to any user storage of files on secondary storage. Watchfulness and moral suasion have controlled unauthorized machine usage while producing no unneccessary barriers to short demonstration uses, an authorized user logging-in under more than one name, or other advantageous flexibilities. A recent modification to ITS to provide control over the authorization of new users to store files on the disk coupled with the long standing moral suasion and fact that anyone can delete any file have controlled disk usage with no significant complaint.

2(d). Implementation in Assembly Language

The ITS system is entirely written in MIDAS, an assembly language with macro facilities. Although some parts could have been written in a compiler with little adverse effect, most of ITS is input-output coding and much of the remainder (such as the scheduler) is time critical for the level of service it is desired to provide. The flexibility, efficiency, and ease in interactive debugging of machine language are not matched by any available compiler.

Writing an extremely complex system full of multilevel interrupts and page fault considerations to run on a bare machine is not the same as writing a program of moderate complexity to run within a system known to be functioning correctly. When a system crashes, what do you do? If the basic console lights and manual controls don't work then it must be a hardware problem. But if they work and the system intermittently fails there must be something in between to aid in debugging the software or peripheral hardware. In ITS this function is filled by a permanently resident interactive symbolic machine language debugger which includes a symbol table for the system.

In almost all cases where an internal error is detected, as opposed to an external device error, ITS attempts to halt immediately and minimize the amount of information destroyed by later consequences of the error.

2(e). The Organization of User Procedures

The actual use of ITS is entirely procedure oriented. Almost no "command" is executed directly as a result of user console input but rather there is always a procedure which interprets console input into a sequence of system calls to accomplish a desired task. When a user first makes the system aware of his presence at a console, ITS loads a fixed procedure for him with which he then converses.

Procedures in ITS are organized into hierarchical trees with each non-apex procedure having one immediate superior. Each user at a console commands one tree and usually executes an arbitrary program by instructing his system loaded apex procedure to create an inferior procedure and load into it and start the desired program. Facilities are provided for the transfer of control of the user's console between procedures in his tree. If faced with a recalcitrant procedure a user may directly command ITS to transfer control of his console to the procedure's immediate superior and thus ultimately to the system loaded apex procedure.

This organization of procedures means that no rigid "command interpreter" exists in the system itself. Indeed, it is easy to change the system loaded apex procedure, which is the highest level console input processor, by writing a file on the disk, with no interruption to system continuity. It is also possible for a user to cause a different program to be loaded as his apex procedure so that the highest level console input processor may vary from user to user. This last feature is not used that frequently as the procedure tree mechanism allows users great generality in creating lower command levels of their own.

A powerful highest level console input processor and experimentation with its enrichment are encouraged by the fact that new versions can be run and debugged, as with any other program, lower in a user's procedure tree. The consequences of an infrequently encountered bug in the system loaded apex procedure are minor in ITS, normally affecting only one user. In most other systems with a rigid "command interpreter" built in, a single false address calculation by such a program can crash the system.

This principle of separating a large cohesive body of code into a separate program has also been followed in the implementation of the "IPL" device. This set of interpretive translation routines allows a graphic display list to be output on an incremental plotter (see sections 3(d) and 3(h) below). However, attempting to stop fault propagation by splitting off sections of the basic ITS system would lead to increased complexity and lowered efficiency. The fundamental scheduling, system call, and input-output handling routines of ITS are richly interwoven, carefully optimized, and, by this point in the system's maturation, relatively debugged and static. The only other large, cohesive, program-like piece of code within ITS that could be split off is the interpreted display device ("IDS", see section 3(d) below) but, since it is now thoroughly debugged and static, little would be gained.

Certain simplifications are possible in ITS due to the hierarchical restrictions placed on control actions on procedures. Many system variables associated with a procedure can be modified only by the procedure itself or its immediate superior. Procedures may be created or destroyed only by their immediate superior, except for apex procedures, which have no superior. Apex procedures may be created or destroyed only through the system recognizing a request for service from a free console (see section 3(g) below) or recognizing a log-out system call from an apex procedure. (However, procedures may change from being non-apex to apex and vice versa as described in section 4(c) below.) These restrictions reduce the need for interlocking and the problems of authority between procedures but have not been onerous from a user's point of view.

2(f). Procedures as a Representation of Procedure Status

The fundamental mechanism in ITS for determining and changing the status of a procedure that is blocked during the execution of a system call deserves special comment. In many other systems the state of a procedure that is blocked (can not proceed until some external event or condition occurs) is represented by status bits in such a way that for a system to determine if it is still blocked may be difficult and it may be difficult or actually impossible to abort the procedure from its blocked state until it is unblocked. It may even be necessary for every event that might unblock a procedure to perform extensive manipulations on these status bits and determine if it has unblocked the procedure.

In ITS, when the system code for the execution of a particular system call gets to a point where it may be blocked, the blocking condition is normally summarized in a single instruction that skips if the procedure may proceed. This instruction may, with few constraints, call an arbitrary routine or may simply test a single variable or, indeed, a single bit. In the location after the blocking test instruction, where one might in a non-time-shared environment expect to find a transfer of control back to the test to form a wait loop, there is a call to a routine which causes the blocking test instruction to become the criterion for allowing that procedure to be scheduled. The scheduler is then called to determine what other procedure to run.

One result of this is that no special pains need be taken at any event which may unblock one or more procedures. Any change in the state of any variable may cause any number of blocking test instructions to skip when executed. It is also easy to determine at any time if a procedure is in fact blocked. The only barrier to arbitrarily subtle blocking conditions is the time taken by the scheduler in executing blocking test instructions that call complex routines.

The addition of a few system conventions and another simple mechanism make it easy in ITS to abort any procedure from within execution of a system call in a reasonably short time. Procedures are not abortable when "running in executive mode" (executing a system call and not blocked). This is obviously necessary to protect certain semi-critical areas of code where interruption without return would leave variables inconsistent. (This should not be confused with truly critical areas of code where clock interrupts to ITS are momentarily disabled to avoid the possiblility of the suspension of the current process and the resumption by the scheduler of another.)

System calls are normally written so that whenever a procedure is blocked in executing the system call it can be aborted out and the user procedure reset so that, if resumed, it will re-execute the system call. If the system call has performed actions that should not be repeated, this must be indicated by modification of a permanent system variable or of the users core image, usually by changing one or more of the arguments to the system call. Note that the procedure being aborted may be indicated as unblocked simply be signaling that it no longer has a blocking test instruction associated with it. No change need be made in the actions at any possibly unblocking event. In the case that some "clean up" operations need be performed to successfully abort, a list is associated with each procedure of interlocks it has seized, which must be unlocked, and of specifications of arbitrary "clean up" routines. These interlocks prevent simultaneous access to certain data bases or simultaneous execution of lengthy (where turning off clock interrupts is impractical) critical areas of code by more than one procedure.

This abort facility has manifold uses. It has made it easy to implement a software interrupt facility by which user procedures may receive interrupts in much the same manner as the PDP-10 provides hardware interrupts to the monitor (see section 4(e) below). This interrupt feature incurs less overhead than the techniques used in some other systems where a separate procedure must be created for each condition or event it is desired to be signaled by.

If one procedure desires to temporarily or permanently stop a second procedure, say to modify system variables that may be in flux if the second procedure is executing a system call, it need not wait until the call is complete (a condition that might never occur, for example on output to a stuck device). It can simply abort the second procedure from any system call it is in. If the second procedure is already stopped or is running in user mode, there is no problem, of course. If the second procedure is actually running in executive mode, a feature is provided to block the first procedure and abort and stop the second procedure, unblocking the first, as soon as the second procedure is next blocked or returns to user mode.

2(g). The Scheduling of Procedures

When the quantum time runs out for a procedure, or it encounters a blocking condition, or it causes an interrupt by some program action (memory violation, stack overflow, etc.)

the scheduler is run. It examines the variables in the system associated with each procedure and decides which to run next. It simultaneously performs those modifications to the variables necessary to provide the software interrupt feature of ITS (see section 4(e) below). In the case of a "fatal" error in a procedure, an interrupt is given to its superior procedure (see section 2(e) above).

The best runnable procedure is chosen on the basis of two "usage" variables associated with each procedure. One of these is the amount of processor time used "recently" by the process. That is a quantized exponentially weighted average fraction of available processor time used with "current" usage weighted one and earlier usage weight decaying to 1/e for usage about seven seconds old. The second variable associated with a procedure is in fact the same quantity but computed for the usage of the entire procedure tree in which the particular procedure resides. Then, of any two runnable procedures, the one chosen to run is normally that with the lowest tree usage or, if tree usages are equal, the one with the lowest procedure usage.

This scheduling algorithm has several good effects. Machine time is basically divided equally between consoles and then equally between all procedures running for a particular console. A compute bound job is guaranteed its "fair" share in the long run yet an interactive program is guaranteed high priority if it has not run for a while. Note that, in contrast with many other systems, a procedure that is frequently blocked but only blocked for a very short period of time is quite likely to get its fair share of time and, in any case, is guaranteed to run again in at most one quantum time if it has been getting a low fraction of machine time.

Of course, there are numerous embellishments on the scheduling algorithm that make things a little more complicated in practice. Special consideration is given various "daemon" and disowned procedure trees (see sections 4(a) and 4(c) below). One procedure may be placed in a special state called master mode where it is given twice the normal priority (by pretending its usage variables are half as large). Master mode also gives a procedure priority to the DEC 340 display (see section 3(d) below). A feature called real time mode is available. This allows one procedure to, within limits, specify a high priority time interval and a larger frame time such that for the high priority time at the beginning of each frame it is given absolute priority. If not blocked it is run in preference to all other procedures. A real time airplane simulation has been successfully developed in ITS using this feature.

The present ITS scheduler has great flexibility and generality. However it examines every procedure every schedule time, though some are only glanced at. The current scheduling algorithm is not incompatible with some classification of procedures into queues and a resulting increase in efficiency.

2(h). The Debugging of Programs

Each user's standard system loaded apex program in ITS is a modified and extended version of the well known DDT debugging system. This automatically makes the usual symbolic memory examination, modification, and search facilities available. Additional powerful debugging facilities are available through DDT in conjunction with hardware and software features of the ITS system.

Modifications to the DEC PDP-10 processor and features of the memory paging hardware (see section 3(a) below) that have been installed on it allow a procedure to be single stepped by its superior procedure and provide means to guarantee suspension of a procedure and the informing of its superior on specified types of reference to a specified memory location. In most computer systems these facilities, if available, require manual intervention from a computer operator's console and are thus inconsistent with remote console time-sharing or at the very least inconsistent with providing these facilities to more than one user at a time. Software features in ITS enable DDT (or any other procedure) to install many breakpoints in its inferiors even at locations which are program modified within certain limits.

In some other systems, the debugging program actually resides in the same core image as the program to be debugged. Not only does this unduly impinge on the design of programs, but the program being debugged can clobber the debugger! In some other systems, much of the system's information concerning a program is stored in the program's core image. No only does this unduly impinge on the design of both the program and the system, but by clobbering these locations a procedure may destroy information of great utility in debugging the procedure or, indeed, debugging the time-sharing system. Furthermore, barriers are placed in the way of a simple algorithm for procedure swapping to secondary storage by the necessity to retain much information in main memory that normally resides in the procedure's core image.

A research environment, perhaps more than any other, requires that a high priority be given to ease in debugging. Despite considerable success in efforts toward universal debugging aides, it has been found that beyond a certain level of complexity and uniqueness there is no practical alternative to designing tailor made debugging and auditing features directly into the program being developed.

A good example of this is the system call abort test feature built into ITS. As should be clear from section 2(f) above, procedures can usually be interrupted out of a system call at several places. For a complex system call, the probability of being aborted at any particular point may be very low, expecially if the point is a possible blocking point where the blocking condition has a low probability. The system call abort test feature allows a single procedure to be selected so that all system calls it executes will be successively aborted at each point where a conditional call to the test feature has been inserted. After each test abort the system call is restarted and allowed to pass the previous abort test point and go on to the next. This provides a reasonably thorough test of interrupt sensitivity and of any clean up routines that are used.

2(i). The Transaction of Input-Output

The majority of the code in the ITS monitor is devoted to input-output. A general and uniform schema of system calls, with symbolic device and file specifications, has been extended to as many devices as was reasonably possible. Many additional system calls are available to provide the user with the full capability of certain devices that do not fit the standard system calls at all or whose potentiality is not fully subsumed by them. Those device-specific calls that are of sufficient importance are described with the device in section 3 below on hardware.

The general complex of input-output system calls provides for the transfer of a character, word, or arbitrary size block of words or characters at one time. Provision is made for interrupting out of a block transfer in a resumable manner with clear indication to the user of the extent of the transfer's completion. A user procedure is never required to know about the nature of the physical blocking of a device it is using.

The input-output transfers are effected by system calls that refer to the area of the procedure to be written from or read into and a logical input-output channel number. Initially a particular device, file, and direction of transfer are associated with a logical channel by a system call in which they are symbolically specified. A very similar symbolic specification is used in the calls to delete or rename files. A feature is provided in ITS whereby these symbolic file and device specifications are subject to a mapping which may be specified for a procedure by itself or its superior procedures. Any procedure may make or delete these input-output mapping entries.

There are many system calls to affect an established input-output transfer, referring to it by the logical channel with which it is associated. Some of these allow a procedure to treat a file on certain devices as an area of random access storage. Others allow a procedure to cause the ITS system to suspend a transaction, storing its status and freeing the associated logical channel, and later resume the transaction on the same or a different logical channel. This is frequently used for "insert" commands in files which cause a program processing the file to logically insert another file by suspending processing of the first, processing the inserted file, and then resuming the processing of the initial file. Another system call is available which resets system buffers for the transaction in question. This is frequently used in "quit" or "silence" commands which reset console input and/or output buffers.

2(j). Input-Output Buffering

The necessity for input-output buffers in the system when character or word at a time input-output is provided should be obvious. The many advantages from system resident buffers, especially if dynamically allocated, may not be as clear. As mentioned above, and as opposed to some other systems, the user is never required to be aware of the physical file blocking on a device he is using. The efficiency of memory utilization is greatly increased and input-output delays are greatly reduced by the dynamic allocation of buffers for the major high speed multi-user file-structured devices. Even for devices which are implemented with a fixed system buffer, such as the paper tape reader, one large efficient system buffer is all that need exist regardless of how many programs there are with the potentiality of using the device.

For single user slow speed non-file-structured devices, a fixed size system buffer has been used for simplicity. These buffers are set to a size proportional to the speed of the device, so the buffer normally represents a certain amount of device operation time. Since, for many of these devices, user intervention at the device is advantageous, or even necessary, a fixed "time" buffer is useful in maintaining a humanly reasonable lag or lead between the device and the procedure using it.

Input-output to user terminals presents special control, bi-directional synchronization, and buffering problems (see section 3(g) below). Using the above design philosophy, fixed buffers representing a reasonable amount of time were chosen. Furthermore, these buffers are associated with the terminals and not with individual procedures. Almost any advantages from different buffers for each procedure, wastefully provided in some other systems, can be simulated using system calls that allow procedures to "output" characters into input buffers. This allows a superior procedure to fully simulate a separate input buffer for its inferior.

Using buffers in the system allows real input-output to proceed almost entirely without consideration for the state of the procedure requesting it as real transfers are into or out of system memory. (In particular the procedure may have been swapped out to secondary storage.) A user procedure may easily be interrupted, swapped out, moved in core or otherwise affected by the system with no difficulty, even while executing or blocked in a system call requesting an input-output transfer. Such a system call normally just transfers data between system memory and the procedure's core image.

2(k). Is Compatibility Necessary ?

ITS is an acronym for Incompatible Time-Sharing. This naming was a reaction against certain attempts at "compatible" time-sharing systems. These attempts seem to result in a duplication of effort that produces an imperfect non-time-sharing compatibility mode embedded in a time-sharing system whose structure is such as to require considerable change in the external characteristics of programs if such programs are to efficiently mesh with and make use of their system.

To convert even a simple program to run under ITS requires change, but the structure and richness of ITS are designed to provide equivalents for all frequently encountered non-time-sharing features. As a result the difficulty of a system program conversion effort will normally be found linearly related to the amount of input-output programming to be converted and not much affected by its level of sophistication. Also, the external characteristics of many programs need be changed only trivially. This principle of design philosophy has not inhibited the inclusion in the ITS system of many pure time-sharing features not called on in any straightforward conversion of non-time-sharing systems programs.

A very simple feature has been included in ITS to allow the simulation, within the procedural hierarchy structure, of other time-sharing systems. This is done by allowing a superior procedure to set the state of an inferior to one in which all system calls and interrupts are communicated to the superior for it to interpret as it sees fit. This feature has been used to successfully simulate the manufacturer supplied time-sharing system. (This "DEC system" wisely does not attempt to provide non-time-sharing compatibility.)

In a research environment, the provision of a non-time-sharing compatibility feature may be an unnecessary effort sink doomed to lack of success in the sense of being able to run unmodified programs with sophisticated non-time-sharing input-output.

3. Hardware

Some information on the hardware facilities of the ITS system appears in section 2 above.

3(a). Memory, Paging, and Swapping

One of the important influences on the ITS system and research with it at the Artificial Intelligence Laboratory has been the availability of over a quarter of a million words of central core storage. This has encouraged large and sophisticated research programs with much rapidly accessible imbedded knowledge. It enabled early versions of ITS to keep all user procedures in core memory for minimum response delays on conversational input. Without this amount of memory, it is unlikely that the powerful procedure hierarchy organization of ITS would have been adopted (see section 2(e) above).

Recently, the crunch that would have resulted from the steady growth in the size of the programs being developed and the number of persons simultaneously using the system has been avoided by the use of paging and swapping. Before paging hardware, designed by the Artificial Intelligence Laboratory, was installed on our DEC PDP-10 computer, only a relocation and protection register was available. This restricted procedures to contiguity both in real memory and their own address space. Some additional workload was placed on ITS due to the necessity to "shuffle" the contents of memory on occasion when contiguous space needed could not be found. The workload would have increased further if swapping of procedures to secondary storage had been used before paging. It would have been continually necessary to find large contiguous spaces to swap into.

Nevertheless, it is doubtful that the elimination of this system overhead is adequate reason to install paging at current hardware and software costs. It is the numerous additional benefits of paging that tip the scales. With the Artificial Intelligence Laboratory paging system, as usual, a procedure's logical address space is divided into pages that are mapped into pages of real memory and possibly marked as being restricted to certain types of access. Since user pages need not be contiguous in real memory, the need to occasionally "shuffle" memory is gone. In addition, the user's address space need not be contiguous so that the user procedure can, for example, have several dynamically allocated tables without the need to relocate them or reserve space for the sum of their maximum sizes. More important, if the invariant parts of a system program are compacted in its address space so as to completely fill one or more pages, those pages can be made read-only (or "pure") and shared by all simultaneous users of the program. A feature for automatically providing this sharing on prepared programs has been incorporated into ITS so that, for example, the constant part of the standard apex procedure (see section 2(e) above) is shared by almost all users of the system.

It is also possible with paging to have a procedure partially swapped out. Those pages not in memory can be marked as not accessible and the system can swap in the page when interrupted by the error resulting from trying to reference it. In the current ITS, procedures are swapped out as a whole but pages are swapped back in only on demand. (Certain pages can be locked in core by the "direct to memory" input-output features of ITS (see sections 3(d), 3(e), and 3(f) below).) Thus rarely referenced pages tend to stay out of memory. The same blocking test instruction mechanism mentioned in section 2(f) above is used to suspend a procedure waiting for a page to be read in. As a result, a procedure can be easily interrupted out of this "page wait" status, something not usually possible in other systems.

In sum, due to these flexibilities, paging hardware is almost necessary for a modern general purpose time-sharing system. However, the case for segmentation hardware in addition to paging is not at all clear. The access between procedure core images and between a procedure and either the PDP-10's or PDP-6's absolute memory, which could be handled by segmentation hardware, is handled in ITS by paging with a one time set-up system call overhead or, for programs that want to treat memory as an input-output device, with system call overhead per word or arbitrary size block transfered. Segmentation hardware would not significantly, if at all, reduce overhead in the case of procedures that already access other segments with paging yet it would add hardware rigidity and expense. There also seems to be a tendency for persons using a system with segmentation hardware to operate on the assumption that almost no non-access forms of overhead are involved in using many segments instead of one. This mistaken impression leads to an unfortunate explosion in the number of segments.

Flexible system calls in ITS enables procedures to transform their paging maps in various ways. They can move their own pages around in their logical memory space or make them read-only. They can allocate additonal pages of memory in any logical page slot. They can insert pages into their map from any other procedure or the absolute memory spaces of the PDP-10 or PDP-6. If requested, write permit will be given for pages inserted from an immediate inferior if the inferior had write permission. Procedures can declare any page for which they have write permission to be a "public page" which any procedure may then insert in its own map with write permit. Finally, procedures may request the insertion of a page associated with a particular system wide identifier such that if no page exists associated with the specified identifier, a public page will be created. If such a page is in existence for the identifier it is made available to the user in the virtaul address slot requested.

The bandwidth of the current memory system and secondary storage swapping device seems adequate for the current level of use only. A significantly higher level of swapping would require a swapping device superior to the disk currently used, which must also support user file storage (see section 3(c) below). A faster swapping device would impose higher demand on main memory which is now mostly a single uninterleaved 2.75 microsecond Fabritek core memory. A significantly faster successor to the PDP-10 processor, which could execute instructions faster than its average of 5 microseconds, would be wasted instead of slightly slowed with the current memory. On the other hand, the installation of sufficient faster memory in <u>addition</u> to the current memory would not only allow for a much faster central processor but, by increasing the capacity of main memory, might obviate the need for a higher bandwidth swapping device.

3(b). The Dual Processors

The Artificial Intelligence Laboratory ITS system includes both a PDP-10 computer and a PDP-6, its slower predecesor. These are configured to give the PDP-6 a small private core memory which is accessible by the PDP-10, where the time-sharing monitor runs. Normally the PDP-6 can not access the memory being primarily used by the PDP-10. Each processor has exclusive control over some basic input-output devices attached to their input-output signal buss but beyond these devices the busses are time multiplexed into a single buss with an extra signal which indicates which processor has the shared buss for any particular cycle. Devices on the shared buss normally have simple assignment hardware which allows one processor at a time to seize each of them. Until released by their controlling processor they ignore commands from the other processor except that an attempt to read their status will reveal their processor assignment state.

All of the important robotics devices on the Artificial Intelligence Laboratory system are on the shared input-output buss and may be assigned to the PDP-6 where programs of unusual time criticality or input-output organization can be developed. Through ITS, the PDP-6 can be made to appear as a procedure in a user's procedure tree. Thus programs can be easily loaded into or dumped from the PDP-6 and to some extent controlled and debugged by a procedure running on the PDP-10. To further facilitate interprocessor communication, a device has been made accessible to each computer by which it can interrupt the other. If the PDP-10 is thus interrupted by the PDP-6, ITS will communicate this to the procedure in the time-sharing system that has attached the PDP-6. Finally, a system call is available in ITS to similarly interrupt the PDP-6.

3(c). Secondary Storage

Disk storage, standard magnetic tape, and DECtape (DEC microtape) are available on the ITS system for file storage. The disks are also used for swapping. The standard magnetic tape and DECtape provide off-line back-ups, archival storage, and easy interchange of information with other systems.

Disk storage is provided by three Memorex 2314-compatible drives which are attached to a Systems Concepts Incorporated DC-10 controller with direct memory access. The interchangeable disk pack feature of these drives is used primarily for back-up when new ITS disk routines are being debugged. Most users keep frequently used or currently active files on the disk system. A system of directories is maintained by ITS describing the location of all files on the disk and the location of free tracks.

Two tracks on each disk pack are dedicated to that pack's track usage table and master directory. The track usage table has use counts for each track on the disk indicating how many files include that track. Since facilities for including a track in more than one file have not yet been included in ITS, this count is presently equivalent to a one bit use flag. In some other systems, information on free tracks is kept in list form with each free track containing a pointer to the next. This requires that a read be done before each write, halving the efficiency of disk writing. In ITS, free tracks are easily available from the track usage tables which reside in core, although updated versions are frequently written on the disk packs. Similarly, the tracks in a deleted file may be freed in ITS by simply changing a track usage table without the necessity to write each track to link it into a free list as in some other systems. The track usage tables contain certain additional information such as the pack name and the size of the block of tracks, if any, on that pack dedicated to use in swapping (see section 3(a) above). Finally, the master directories on the disk packs (only one common copy is kept in core) contain the names of the users for which there are user directories on the disk.

Each user directory lists all files under that user's name and the location of each track in each file. Although user directories are currently limited to one track, the file location information stored in them is very compact and no problem has been encountered due to this limitation. Various pseudo-user directories are available for system programs, large joint projects, and other uses. Also stored with each file name in the user directory is information concerning the time of the file's creation. Provision is made for symbolic links where a particular "file" in a user directory is a symbolic specification of user and file name which is actually used if a read is being done. These links may be chained up to 100 levels. A write or delete will write over or delete the link and not the file linked to.

It turns out that the most frequent operation performed on a user's file directory in ITS is probably to list it. This is because most system programs, when run from a terminal with character or graphic display capability, display the most recently referenced directory after file commands. Also, users frequently examine several directories explicitly before actually deciding to read, write, or delete a file. File diretory design for an interactive system, especially one with high speed (e. g. display) output at many user terminals should take this into account.

One Digital Equipment Corporation (DEC) TU-20B magnetic tape drive is available interfaced by a DEC TM-10A controller. This is a seven track IBM-compatible unit used primarily for disk back-up and exchanges of large amounts of data with other computer systems. The ITS magnetic tape routines provide system calls for the usual special operations such as backspace and rewind but currently records are limited to a maximum of 1024 words and there is no provision for a system-recognized directory on a tape.

Several DECtape drives are available. These utilize small magnetic tapes with a pre-recorded fixed format. They are the principle type of off-line storage for almost all users of ITS. Due to the importance of this level of user controlled back-up and archival storage, system calls are available to temporarily assign and deassign particular DECtape drives to particular users. Normally DECtapes are used in a file structured manner very similar to the disks under ITS. System calls are available to initialize a DECtape's directory and to set its tape name. A mode is also available in which all the data on a tape can be read in its physical order, without regard for its usual directory structuring.

3(d). Display Facilities

Real time graphic displays provide the highest bandwidth communication from computer to user now available. They provide for natural input of graphic data or control information. In developing advanced programs to grapple with the real world, especially in conjunction with the Artificial Intelligence Laboratory vision facilities, described in section

3(e) below, graphic representations are indispensable and graphic displays are of inestimable value. As described below in this section, there are serious deficiencies in the the current ITS display hardware.

The prime display facility on the ITS system is a DEC 340 display with extended character generator character set. This display is placed near a hardwired system teletype terminal and also drives two slave monitor displays, a DEC 343 and a Hewlett-Packard 1300A. The DEC 340 has character, vector, and incremental modes as well as X-Y point plotting but it is not fast enough to maintain more than one complex display image.

There are three different ways of using the DEC 340 under ITS. With all three methods system calls are available to make use of its light pen and to control the refreshing of the display so that it may be synchronized with a camera shutter. Since actual display monitors are near particular system terminals, a pecking order has been established whereby users at certain consoles can take the display away from a user at a lower precedence terminal.

The most common method for graphic displays involves system calls that cause various areas of the user procedure's core image to be used as display lists to refresh the display. This gives the user the maximum power to dynamically change what is being displayed. He can simply modify his own core image. With this method, the user also has the advantage that he can get an incremental plotter version of his display output by using the interpreted plotter device as described in section 3(h) below.

The remaining two methods of using the DEC 340 buffer display lists in system memory. In the first method, the symbolic device "DIS" is used and characters can be output in character mode to appear on the display. ITS also simulates a blink feature whereby characters output between a blink-begin and blink-end character are intensified or de-intensified every half second. In word mode, output to DIS is buffered by ITS as absolute DEC 340 commands to be used to continually refresh the display.

The last method of using the display is the symbolic device "IDS" or interpreted display. This activates an extensive piece of sophisticated code in ITS that simulates a fictitious display processor by interpreting instructions for this display processor in the user's core image. This simulation code makes use of the DEC 340 hardware to perform many calculations and stores in a system buffer a 340 display list to produce the same display as the fictitious display processor would. The user simply sets up his display in the form of instructions for the IDS, sets up a display push-down list pointer for the use of the IDS, and outputs an initial display processor program counter. Other than the slowness of the initial interpretations and the denial of dynamic display modification due to system buffering, the user appears to have a sophisticated display processor at his disposal.

Certain hardware drawbacks of the DEC 340, which have been ameliorated by ITS software, make it inherently unsuitable for full utilization in a paged time-sharing system. Before ITS was paged, there were only two problems. First, that certain display lists will cause the 340 to hang-up with no indication to the computer. This requires various timing routines in ITS to detect this condition and reset the display. Second, certain display lists (for example the "display list" of all zeros, which does not cause anything to be displayed) are processed so rapidly by the 340 that it takes up all available memory cycles. This problem is aggravated by the fact that the 340 is not designed for use via a data channel but uses the PDP-10 computer's "BLKO" facility which uses three memory cycles for each word of data output. Non-trivial accounting routines are required in ITS to detect this condition and take appropriate actions.

With the addition of paging, the worst problem is added. Since data must be "BLKO"ed to the 340 from executive address space, ITS must laboriously simulate paging for each display list pointer and for each contiguous display list. This problem in addition to those above has meant that an attempt to make sophisticated use of the display with dynamic multiple display lists produces an enormous load (sometimes over 50%) on the system due to the exorbitant bookkeeping mandated by the DEC 340 hardware.

The replacement of the DEC 340 by a modern high speed display processor could provide the sophistication simulated by the IDS device (see above, this section) without its drawbacks and provide this to many users simultaneouly. Yet, there would be less load on ITS than that caused by the current limited facilities provided one user by the DEC 340.

A color display with X-Y point-plotting capability only is also attached to the Artificial Intelligence Laboratory system. The slowness of this display and various problem related to focusing and convergence presumably led to its manufacturer's generosity in giving it to the Artificial Intelligence Laboratory. There are currently no routines in ITS for the use of this device but it has been used to a limited extent from the PDP-6 (see section 3(b) above).

3(e). Vision Facilities

Available in the ITS system is a television-camera-like device enabling the light intensity at points in its field of view to be read in by a user. Points may be accessed in random order and the device, known as a vidissector, operates by, within limits, integrating the light at a point until it reaches a specified value and returning the time taken. The vidissector focus and iris are computer controlled (see section 3(f) below). Fetures for using it in the ITS system are described below.

There are three different methods of using the vidissector in ITS which provide increasing levels of overlap to the user. The simplest mode is as symbolic device "NVD". This device is opened on a logical channel (see section 2(i) above) and then input-output system calls are done specifying this channel and pointing to words containing the coordinates of points to be examined. The contents of these words are replaced at the time of the system call with the light intensity at the specified points.

The second method of using the vidissector is as symbolic device "TVC". In this method, TVC is opened on two logical channels, one for output and one for input. Coordinate pair words may then be output on one channel where they will be buffered by ITS, replaced asynchronously by the light intensity at the points they specify, and be made available for input in the same order on the other channel. Thus a user procedure can output some points, proceed with other computations, and later retrieve the vidissected values.

The final method of using the vidissector provides the maximum amount of overlap. A special system call allows the user to specify a rectangular array of points to scan, an arbitrary homogeneous transformation on their coordinates into vidissector field-of-view space, and an array of words in the users core image for the resulting light intensity measurements to be stored in. The vidissection caused by this system call is completely overlapped with user computation, if requested. Another system call is available to test the progress of a scan, hang up until its completion, or abort it.

Thus ITS provides vidissector routines of various levels of sophistication to match that desired by the user. At the expense of slightly increased complexity and organization on the part of a using procedure, greater speed, overlap, and patterning are available from the system.

3(f). Analog Input and Output

The Artificial Intelligence Laboratory system has various analog sensors and effectors attached to it. These include mechanical arms and hands frequently used to manipulate objects in the field of view of the system's vision facilities (see section 3(e) above). Analog channels to and from these devices are interfaced for digital control as described below.

The digital to analog multiplexor, or output multiplexor, can be used from within ITS in two ways. The first is as the OMX device which may be used by more than one procedure at once. The user simply outputs a word specifying in different fields which of the sixty four output lines he is setting and which of 4096 values he is setting it to. Since the output multiplexor requires refreshment every half second to maintain its output values (it has only analog memory) ITS maintains a table of values for each channel which it outputs periodicly as long as a procedure has the OMX device open or is using the special system call described below. When the user associates the OMX device with one of his logical channels he can specify that output is to be effective immediately or, for slightly less overhead, is to be only stored in ITS's table to take effect within a half second.

The output multiplexor may also be used via a special system call that is available to only one procedure at a time. With it, for a limited number of output channels, the desired destination value and a velocity (rate of change) limit can be specified and the current position and velocity tested. These operations are performed for the user by interrupt routines in ITS. Since this system call can specify changes for a list of output channels which may represent joints of an arm, the call is uninterruptable while information is being transferred to ITS. Otherwise uncoordinated arm motion with disasterous consequences might occur. This uninterruptable period is limited by limiting the maximum length argument block that may be given to this system call.

The analog to digital multiplexor, or input multiplexor, can also be used from ITS in two ways. The first is as the IMX device which may be accessed by more than one proedure at once. This device is used by opening it on a logical channel and then doing system calls on that channel pointing to a word (or words) containing input channel numbers. Each number is replaced by the digitalization of the analog signal present on the corresponding multiplexor line.

The second way of using the input multiplexor involves a special system call by which the user can cause program parameters to appear to be directly controlled by analog inputs. For a limited number of channels a procedure can cause a floating point word or a fixed point byte (possibly a word) to be pseudo-continuously set by a particular input multiplexor channel. The user also gives limits which tell what digital amount the maximum and minimum analog values are to represent. Linear interpolation is done inbetween. In addition a procedure specifies whether a particular parameter is to be set absolutely to the value represented by the input signal or is to be incrementally adjusted by it. In the incremental case no change will occur at the initial "connection" but change in the input signal will cause changes proportioned by the "limits" set. Since the normal use of incremental mode is to control a program by manipulating a potentiometer attached to the multiplexor it is desirable to keep the potentiometer centered and avoid saturating its range. This is accomplished by exaggerating the incremental effects of upward changes in the signal in the upper third of its range and of downward changes in the lower third of its range.

The input and output multiplexors are a good example of ITS input-output philosophy . As much as possible is made available through standard input-output system calls. If there are desirable capabilities or extremely useful effects that are not available through standard calls, appropriate special facilities are added.

3(g). User Terminals

One of the most important aspects of a time-sharing system from the user's point of view is the console interface. ITS is internally oriented to the use of 7-bit ASCII character codes. Only two characters have been reserved by the system on input so that they are slightly harder to type at a user program. In this section, a "^" before a character means the code produced by striking that character with the control key held down on a teletype. Since, for generality, all possible "character" inputs must be representable in any internal character streams, the "break" signal, from consoles that can supply it, is deliberately unrecognized by ITS.

Of the two characters recognized by the teletype routines on input, one, ^Z, provides the only direct user control over his procedure tree. When ITS sees a ^Z from an idle console, it loads the standard apex procedure (see section 2(e) above). If ITS sees a ^Z from a console controlling a procedure tree it does nothing if the console is attached to the apex procedure. When a console is attached to a lower level procedure, ^Z provides a way of alerting higher level procedures so that they can take the console away from an inferior and then accept commands from the user.

The other recognized character, ^_, primarily provides a means of direct communications with the teletype routines themselves. A user can type in a ^Z or ^_ or an arbitrary character specified by its numeric code without control effect by preceeding them with a ^_. Using a ^_ followed by various other character string arguments, the user can inform the system of various properties of his terminal (including that it is one of several types of character display terminals) and can enter "communicate" mode with another teletype if not prohibited by the other teletype's user. More than two teletypes can be in communicate mode which causes characters typed on any teletype to be printed on all.

Thus we see that on type-in the user has been given great flexibility within the procedural organization of ITS. Only two characters have been taken from him and in return he gets a minimal but sufficient monitor signaling facility and a convenient way to "talk" to other teletypes and tell the system about special properties of his console. (Actually, in the case of consoles with obviously inadequate character sets like the IBM 2741, additional characters have special effects on input to simulate control-shift, alt-mode, etc.) Messages can still be sent between system loaded apex procedures using the core-link device (see section 4(b) below). This feature of the standard apex procedure, which sends entire messages at once and has no effect on the recipient console other than typing out the message, is sufficiently different from the teletype communicate feature to have been retained.

Directly related to teletype input are the questions of echoing, procedure activation, and teletype interrupts. In ITS, the ASCII characters are divided into twelve natural groups. System calls are available to read and set for a particular procedure, the effects of characters in each group. They may be echoed immediately on type-in, or when read by the procedure, or not at all (by ITS). They may be declared to be activiation characters or not (a procedure hung on type-in is not started until an activation character is received or the input buffer contains a large number of characters). They may be declared to be interrupt characters or not. Finally, various special submodes are available such as whether lower case letters should be converted to upper case on input or whether alt-mode, escape, and prefix should be standardized into prefix.

The ITS console input-output routines are designed to allow full utilization of full-duplex communications. Half-duplex is a micro-scale remnant of batch processing where one inputs a clump and waits for a clump of output with pushing a panic button as about the only action possible in between. Even if a program uses line at a time input (carriage return the only activation character) and has no interrupt characters, it is often convenient to start typing in before previous conversational output is through.

Program type-out is given higher priority than echo of type-in so that intermixing of streams on a printing terminal, even if immediate echo is selected, in unlikely for typical high speed output. On character display terminals, different screen areas are normally used for program output and echo as explained below.

When certain characters have been declared to be interrupt characters, their type-in causes an interrupt specifying the logical channel the teletype is open on (see sections 4(e) and 2(i)). A procedure thus interrupted can read the interrupting character in its interrupt routine with no effect on its main program's ability to later reread the character for normal input.

Turning from teletype input to teletype output, the complicating factor becomes the availability of various character display terminals. To enable procedures to simply and efficiently use the full capabilities of such terminals, system calls are available to read the screen size and current cursor position. "Normal" mode output tries to simulate a teletype (except that some non-printing characters are optionally rendered into two graphics) but a display mode of output is available where ITS interprets ^P followed by various special character sequences as a command to set the cursor postion, clear the screen, etc. These commands are uniform despite the variety of character terminals on ITS. It is also possible for a procedure to specify whether it wants input and output interspersed or a separate command echo region at the bottom of the screen.

The hardware consoles on ITS are primarily interfaced through two 16-line controllers. One, built at the Artificial Intelligence Laboratory, interrupts the processor on every character in or out. The other, a Systems Concepts, Inc. DK-10 controller is much more suited to time-sharing use and handles direct from memory output of character strings without additional effort by the PDP-10 processor.

Certain non-hardware teletypes, or pseudo-teletypes, also are implemented in ITS. These appear to be normal teletype devices on one "side" but, in fact read from and output to whatever procedure has opened the other "side", rather than a physical terminal. These provide added flexibility in the simulation of certain situations for debugging purposes and will be used in the initial implementation of the ARPA network interface (see section 5(a) below).

In summary, the ITS teletype routines are one of the prime reasons that users can provide their own subsystems with such generality. The system provides great flexibility with a minimum of protrudence into the resulting teletype input—output behavior.

3(h). The Plotter and IPL Facilities

At the Artificial Intelligence Laboratory, ITS provides hard copy graphic output on a CalComp 565 plotter. This device can be used in two ways. As the PLT device it appears to be a character output device where various bits in each character have effects such as step right, pen down, etc. Of course blocks of "characters" can be output with one system call.

The IPL, or interpreted plotter, device allows a particular procedure to be automaticly loaded and interposed between the using procedure and the plotter. The user's plotter output is in fact interpreted by this procedure which also has the capability of examining the using procedures core image. The IPL device allows the user to output drawings in a plotter oriented command system that provides vectors, characters, scaling, and similar features. The user may also output information to the IPL device specifying the position and length of a DEC 340 (see section 3(d) above) display list in the user's core image. This will then be interpreted and output on the plotter.

3(i). Clocks

Time is, of course, important in a time-sharing system. ITS has several clocks that are used for different purposes. The most important is a sixty cycle clock that provides interrupts to the processor. This is used to drive the scheduler, to update internal times and dates, and to drive a general clock queue facility. ITS uses this clock queue internally to remember things to do at the clock level in order of immediacy. The clock queue is used to run various periodic bookkeeping routines and to provide simple timing to various ITS functions. There is also a potential clock queue node associated with each procedure whereby the procedure can get periodic software interupts (see section 4(e) below). This block is also used and these interrupts provided when a procedure uses the real time facility (see section 2(g) above).

A date clock is also attached to ITS. It is powered by a special power supply that is not normally turned off and is used by ITS to initialize its internal times and dates. This time and date information can be read by procedures through various system calls and is also used to set the creation time of files written on the disk and for similar purposes.

There is also a quantum timer included in the Artificial Intelligence Laboratory paging box which is not used to initiate the scheduler but is used to measure the processor time used by a procedure. Finally, there is a sophisticated high frequency real time clock not used by the system and available for user robotics or other uses.

3(j). Miscellaneous Hardware Devices

There are numerious input-output devices on the Artificial Intelligence Laboratory system that have not yet been mentioned. Character at a time devices include a paper tape reader, paper tape punch, and Data Products Corporation line printer. These three devices are available through the standard symbolicly specified input-output system calls. Several special devices are also available, mostly for robotics work, that provide simple binary input-output. These can be used for remote control of lights or input from touch sensors or switches, etc.

Finally, graphic input is available via a Sylvania DT-1 tablet. This device can accurately measure the X-Y coordinates of a special pen on its surface and can also produce a few bits of pen height information. A procedure can read samples of these coordinates, buffered by ITS, at a rate it selects. To compress this information, ITS can supply one coordinate sample with a count if successive identical samples are read from the tablet.

4. Additional Software Details

4(a). Daemon Procedures

In ITS various actions are performed by daemon procedures, which are activated in various ways, rather than directly by a procedure requesting the action through a system call and then running in executive mode. In some cases, the prime advantage of these daemon procedures is that they can treat a particular system aspect in a central manner independent of the priorities of other procedures requesting action, if any. In other cases, daemon procedures are used which appear to be part of the system, to their user, but which in fact are general user written programs.

The most important two daemons are the permanently existant "system job" and "core job". These procedures are part of ITS and run in the executive environment. The core job handles memory requests. It can base its actions on the global memory situation and more easily handle the problems involved in updating the structures linking shared pages. It also reclaims certain types of input-output buffers and is, on occassion, required to do a small amount of memory shuffling if the system job area for procedure variables expands, since this area is currently contiguous.

As important as the core job, and with many more different tasks, is the system job. The most obvious thing done by the system job is to type out various messages on a dedicated teletype. These messages include information that the following actions have been performed: logins and logouts, writes and deletes of system files, deposits in absolute locations, etc. Also messages are printed when various errors occur such as core parity errors or a checksum failure in a constant block of ITS. The latter is detected by the system job periodically computing checksums for each constant area and comparing it with a precomputed checksum. If they do not match, additional precomputed checksum information is consulted that is adequate to uniquely identify the address and old value for any single word being clobbered. (Locations in ITS modified by the set absolute location system call do not cause alarms as the checksums are updated.) The system job types out its conclusions on its teletype. The system job also performs the spooling function of ITS by line printing and then deleting files in a particular disk directory when the line printer is not in direct use. The system job also performs certain periodic tasks that are not sensitive to jitter in the time they are done. Finally, the system job does most of the things related to the system going down feature whereby users are informed a miminum of five minutes before the system goes down in a planned manner and all users are finally automatically logged out.

A third standard daemon normally present in ITS is an accounting and monitoring procedure called the dragon. It writes usage information, and such things as number of page swap in requests, for each user on the disk in its own file directory. A separate program is available that prints out this information in tabular form.

The only other daemon procedures in ITS are used to implement certain pseudo-devices. The IPL, or interpreted plotter device is explained in section 3(h) above. More recently, the JOB device has been added where the file name given specifies a procedure to be loaded by the system. This procedure has various information available to it concerning the system call, associating it with a logical channel of its user, that loaded it. It can then open a symbolic device and run as a co-routine with its input or output connected to the output or input of the other procedure.

The core job, system job, and dragon, though not inferiors or superiors of each other, all point to the same procedure tree usage variable (see section 2(g) above) and are given twice the priority of a console procedure tree. Procedures created by the IPL and JOB devices run for the same procedure trees as their creator.

4(b). Inter-Procedure Communication

It is frequently desirable for various procedures in a system to cooperate with each other. They may wish to communicate directly with each other through input-output streams or to share a data base. Besides the obvious method of communicating by files and the method of a procedure using another as a direct unbuffered co-routine mentioned in section 4(a) above, there are two other means of interprocedure communciation in ITS.

The first is the core-link input-output device. Using this device any two procedures can symbolically specify and form a buffered link over which characters, words, or blocks of information can be transmitted. In addition, by using a special device name, a procedure can specify by file name another procedure that is to be given a "core-link" interrupt. This also opens the input side of a core-link channel (output from the orginating procedure) and inserts the name of the calling procedure as the initial data. The interrupted procedure may use a different special device name in an open which will automatically connect to and allow input from the core-link associated with the interrupt.

The other method of interprocedure communication is by means of shared core. A procedure can attach pages of other procedures or attach an "intercommunication" page specified by a system wide identifier in a very flexible manner as described in section 3(a) above. Programs have been developed that run under ITS and on the PDP-6 by attaching PDP-6 core (see section 3(b) above). This results in two processes cohabiting a possibly identical memory, one running directly on the PDP-6 processor with full access to its peripherals and the other running as a regualr time-shared procedure with normal access to ITS facilities.

To make interlocks and semaphores between time-shared procedures in shared core easier to implement, there is a system call which can be placed after a limited class of test and skip instructions. This system call essentially replaces a transfer to the previous location, which would form an inefficient wait loop. It causes the skip instruction to become the procedure's blocking condition (see section 2(f) above).

Once again we see the great utility of the ITS method of procedure blocking and, for the core-link interupt feature, the general software interrupt scheme it allows!

4(c). Disowned Procedure Trees

Not all procedure trees in ITS are run from a user console or are part of the system (see sections 2(e) and 4(a) above). It is sometimes desirable to run programs in a lower priority "background" mode when their initiator is no longer logged on the system. Also the user may wish to escape from unalterable adherance to the hierarchical organization of procedures and be able to pass around inferiors in his procedure tree or pass an inferior procedure to another user.

To this effect, any procedure with an inferior in a console controlled tree may "disown" the inferior. This results in the branch of the original procedure tree below and attached to the disowned inferior becoming a disowned procedure tree. No console is associated with these procedures and for scheduling purposes (see section 2(g) above) a single half priority tree usage variable is used for all disowned trees.

Any console controlled procedure may attach any disowned procedure tree by attaching the apex procedure as an inferior. This associates all of the procedures in the previously disowned tree with the attaching procedures console and modifies the user-name of the attached procedures to that of the attaching user. (Procedures in ITS are identified by a normally unique user-name job-name pair which is, in some respects, like a file name. The user-name of all procedures in a console controlled tree is the name the apex procedure was commanded to log-in with.) For scheduling purposes the tree usage pointer of the attached procedures are switched to the attacher's usage variable.

Procedures in a disowned tree suffer some very mild restrictions on the system resources available to them but no resource held by a procedure is ever removed by disowning it. As explained above, disowned procedures have lower priority for processor time. The apex procedure of a disowned tree has the power to "log-out" and excise the entire tree. A fatal error in a disowned apex procedure results in its being halted and the next procedure that attaches it and becomes its superior is given an interrupt.

Few other systems allow this level of flexibility in the creation of free standing procedure trees or allow the freedom to pass around entire structures of running jobs.

4(d). Direct Input-Output Instructions

ITS provides two ways for users to execute hardware input-output instructions. First procedures may request that they be run in "IOT-user" mode. This is a hardware mode that makes all instructions legal but provides the same memory mapping and protection as user mode. In keeping with ITS's protection philosophy, this mode will be granted any procedure not in a disowned tree, although a message is typed out by the system job giving the user's and procedure's name (see sections 4(a) and 4(c) above).

If a procedure not in "IOT-user" mode executes hardware input-output instructions, these trap to routines which interpret the instruction and either treat it as an illegal instruction or execute it for the procedure depending on certain permit bits in a system table with entries for each device. These interpretive routines allow, for example, any procedure to read the state of the PDP-10's console switches but prohibit procedures from normally affecting the disk controller.

ITS also has routines for handling spurious interrupts. These routines attempt to find suspicious devices ITS does not know about and devices it does know about that appear to be set to interrupt on the wrong hardware level. The spurious interrupt routines protect the system from unknown devices causing interrupts and are integrated with the input-output instruction interpreting routines so as to prohibit interpretive access to devices suspected of causing spurious interrupts.

With either direct or interpreted hardware input-output instructions a procedure can make a device status test conditional skip instruction its blocking condition by following it with a special system call (see sections 4(b) and 2(f) above). Thus a user may code efficient non-interrupt routines for devices ITS does not know about.

4(e). Software Interrupts

One of the more powerful features of ITS is the system of general interrupts it provides to user procedures. This interrupt system is implemented through the use of several variables, a set of which is associated with each procedure. These variables include an interrupt mask with bits on for interrupts a procedure wishes to enable and an interrupt request variable with bits on for pending interrupts. To allow certain timing errors to be avoided, means are provided for a procedure and its superior to not only read and write these variables but also to set and clear selected bits without affecting other bits.

There is also an interrupt enable flag associated with each procedure that inhibits all interrupts if off. This flag is cleared when an interrupt is simulated to a procedure. The interrupt request bits at the time the interrupt was simulated and the user location interrupted from are stored into the procedure and control transferred to the user's interrupt routine. There is a system call available that may be used to return to the main program and re-enable interrupts. The interrupt enable flag may also be explicitly set or cleared, however.

Interrupts are in fact divided into three categories of severity. The most severe or fatal errors cannot be masked on to interrupt to a procedure. Rather, they have the effect of stopping it and interrupting its superior (if this happens to be the apex procedure of a console controlled tree it is reloaded). Interrupts of intermediate severity may be masked on so as to interrupt to a procedure. But, if they occur when either not masked on or the procedure's interrupt enable flag is off, they are treated as fatal. The least severe interrupt conditions are simply ignored if masked off or buffered in the interrupt request variable if a procedure's interrupt enable flag is off.

4(f). Miscellaneous Software Devices

Several devices in ITS do not correspond to a physical peripheral device. Among those not mentioned in other parts of this paper is the "NUL" device. This device is a high speed source of zero words or characters on input and high speed infinite sink on output.

There are also certain software devices in ITS that are available for character input of various messages by procedures that frequently output them to the user. These character string producing devices include the "ERR" device which translates various system error codes, as specified in the file name used to open the ERR device, into readable messages. The reading of file directories is implemented in a similar way.

These devices are written as co-routines whose "type out" interfaces to the input transfer of their using procedure.

There are several special file directories on the disk (see section 3(c) above) that it has been found convenient to reference as though they were separate devices. Among these are the directory of system programs and a common directory in which is stored such things as interuser mail. There is also a special device that not only accesses a special file directory but also modifies the file names used to encode various information on files written in this directory that are to be line printed by ITS later (see section 4(a) above).

5. Work in Progress

5(a). The ARPA Network

The ITS system is being fully adapted for use on the Advanced Research Projects Agency computer network by Jeffrey B. Rubin. The desire to provide Telnet service to remote users on the network was the prime impetus for the inclusion of pseudo-teletypes in ITS (see section 3(g) above).

The network code includes the basic IMP (Interface Message Prcessor) device routines and the NCP (Network Control Program) imbedded in ITS and separate programs that provide the Telnet and other protocols. This system network code and the necessary IMP interface hardware have been developed and debugged with almost no interference with normal ITS operation. A skeletal pseudo-ITS was written to run on the PDP-6. It has all the necessary hooks to attach the network code and an even greater propensity than regular ITS to halt at the first sign of trouble. As a result of this means of development, the ARPA network will be usable, in a limited manner, from the Artificial Intelligence Laboratory even when ITS or the PDP-10 are unavailable.

It remains to be seen what the full impact of ARPA network connection will be on ITS. It is possible that a need to control usage from the network or problems due to users pre-frustrated by other systems will require changes in ITS's protection philosophy (see section 2(c) above).

5(b). The Mathlab System

The ITS system is to be used by the Project MAC Mathlab group on their own PDP-10 computer. This should increase the incentive for real modularity which has been lacking with a one installation system. (Actually the Project MAC Dynamic Modelling group uses a non-paged early offshoot of ITS on their PDP-10.)

Much of the work in setting up the initial Mathlab system is being done by Richard D. Greenblatt.

6. Recommendations

This section does not concentrate on the Artficial Intelligence viewpoint. Rather recommendations are given for the elimination of bottlenecks and general improvement of ITS as a general purpose system.

6(a). Hardware Development

ITS's strongest hardware need is for a reasonable graphic display controller (see section 3(d) above). If this controller has a character generator with upper and lower case capability, as it should, it would also meet the current need for better upper lower case editing facilities.

Less immediate but forseeable is the need for additional high speed memory (see section 3(a) above). This need becomes critical if the acquisition of a higher speed processor is contemplated.

6(b). Software Development

Software development is more of a continuous allocation decision rather than a purchase or not hardware decision. ITS is being continuously improved ("maintained") at its lower levels but major improvements are not as frequent as they once were when the system was less mature. In software development, there is always a trade off between changes that provide immediate improvement and changes that provide the groundwork for later improvement.

Among major changes being contemplated are the following: 1) the continued development of the nascent "new call" feature which will provide a new uniform system of more symbolic calls to ITS; 2) improvements in the scheduling algorithm to increase its efficiency and decrease system thrashing; and 3) decontiguizing the user variable area of the system.

Bibliography

```
    This memo was typed in edited with TECO:
AI memo 81    PDP-6 TECO, Peter Samson

    The ITS system is written in MIDAS:
AI memo 90    MIDAS, Peter Samson

    The latest memo on the system loaded apex procedure is
AI memo 147A  DDT Reference Manual, Eric Osman

    The latest reference manual on ITS is
AI memo 161A  ITS 1.5 Reference Manual, D. Eastalke, et al

    This memo was output with TJ6:
AI memo 164A  The Text-Justifier TJ6, R. Greenblatt, et al

    For more memos try
AI memo 191   A. I. Bibliography
```