To the Graduate Council:

I am submitting herewith a thesis written by Mukesh Sundaram entitled "UNIX to DEC-10 -- A Simple Teleprocessing Link." I recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Charles E. Hughes, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:

Vice Chancellor
Graduate Studies and Research

UNIX TO DEC-10 -- A SIMPLE TELEPROCESSING LINK

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Mukesh Sundaram

August 1979

ACKNOWLEDGEMENT

The author wishes to express his indebtedness to Dr. Charles Hughes for his invaluable guidance and support, which were instrumental in the development of this thesis. Appreciation is due to Dr. Charles Pfleeger and Dr. Michael Moshell for their contributions and service on the committee.

Thanks are also due to Hari Nayak and Philip McCharen for their help in preparing this manuscript.

ABSTRACT

The objective of this thesis project was to create a software link between a PDP-11/34 minicomputer running under the UNIX operating system and a DECsystem-10 computer with the TOPS-10 operating system, both used at The University of Tennessee at Knoxville.

UNIX is a multi-user timesharing operating system developed at Bell Laboratories for the PDP-11 family of computers.

The software link provides the PDP-11 users the capability of batch job submission to the DEC-10. The hardware involved is minimal--either machine views the other as a simple terminal. The actual software that handles the communication is a simple process, not part of either operating system, per se.

A special device driver routine (added to the operating system) is necessary to handle the I/O under UNIX. The existing device driver routine has a limited buffer size of 256 characters. The communication software which reads this buffer competes for machine resources at the same priority as other user processes. Therefore, a buffer overrun and consequent information loss can occur unless the output of the DEC-10 is automatically inhibited by the device driver until buffer space is available.

At any time, a user of the PDP-11 can submit a file of commands to be interpreted line by line by the communication software, which performs the requisite processing. This submission merely results in the request being queued as a batch process, to be later initiated by an operator. When initiated by the operator, the queue of requests is serviced individually by first logging into the user's DEC-10 account (this information is system maintained). Any output produced by the DECsystem-10 is stored in a log file in the user's directory. The entire implementation is in the C language.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

Networking is a concept that has been in existence for many decades, but has become practical only with the advent of mini and micro computers. Distributed computing, as networking is often called, has the advantage of providing access to more comprehensive facilities than are available to any one computer.

The Computer Science Department at The University of Tennessee has a minicomputer laboratory that includes a PDP-11/34, with 64 K words of memory, three RK-05 disk drives and one floppy disk drive. It runs under the multi-user UNIX operating system, and at this time, supports four time-sharing terminals, three on-site and one remote.

The University of Tennessee Computing Center maintains, among other computers, a DECsystem-10. It is a 1080 configuration with a KL-10 processor, and supports up to 80 concurrent jobs.

This report deals with the interfacing of the PDP-11/34 with the DECsystem-10 and the creation of a batch mode teleprocessing system under UNIX to submit jobs to the DEC-10.

Why a batch mode system? This question is answered by two facts: Communication with the DECsystem-10 is possible only over a voice-grade telephone line, or a hardwired leased line. Owing to the limited number of physical lines to the PDP-11 and the high demand on it, it is infeasible to devote one line entirely to the DECsystem-10 interface. With the addition of a line multiplexor, this may become feasible; however, such an addition will neccessitate an expensive leased line to the DEC-10. All aspects considered, batch mode operation is the best solution.

The teleprocessing system is intended primarily for character file transmission between the two computers. It consists of two parts: one is the actual communication handler, and the other is the user interface which processes the user DECsystem-10 login information, the creation of the command file to comform to the communication handler's input format and the submission of the command file to the communicator's queue of requests. This division is necessary because of the batch mode operation.

CHAPTER II

BACKGROUND OF UNIX

UNIX is an operating system developed at Bell Laboratories. Designed primarily for the PDP-11 family computers, to date it has been implemented on only one other machine, the Interdata 8/32.

A concise summary of the UNIX system described by Ritchie and Thompson[1] follows:

> It offers a number of features seldom found even in bigger operating systems, including
>
> (i) A hierarchical file system incorporating demountable volumes,
>
> (ii) Compatible file, device, and inter-process I/O,
>
> (iii) The ability to initiate asynchronous processes,
>
> (iv) System command language selectable on a per-user basis,
>
> (v) Over 100 subsystems including a dozen languages,
>
> (vi) High degree of portability.

1. System Environment

The PDP-11/34 is a 16-bit word (8-bit byte) computer. As with other models of the same family, it features the

UNIBUS, which controls all inter-module communication. The I/O on the system is memory mapped and each physical device that is attached to the UNIBUS is assigned an address. A number of peripheral devices may be attached by this scheme. All processor-generated addresses are 16-bit quantities giving the machine a 32K (word) addressing capability. However, the UNIBUS has an 18-bit or 128K (word) addressing capability.

The upper 4 K (word) of the UNIBUS addresses (776000 - 777777 Octal) are dedicated to I/O devices. A memory-management hardware module translates all processor-generated addresses (16-bit) to 18-bit quantities, thus addressing up to 124 K (words) of memory, 4 K (words) being dedicated to I/O devices.

The memory-management hardware thus facilitates multi-processing, with each process addressing up to 32 K (words). When servicing multiple users, some form of memory protection is imperative. The memory-management module provides this protection, in addition to providing the CPU with two hardware states--Kernel and User. When the CPU is in the User state, it is prevented from executing instructions that would be disastrous in a multi-programming environment, such as halting the computer, or modifying the kernel program or other user programs. A program fault is generated if any of these situations arises.

The 32 K (word) virtual address generated by a user program is translated, by the memory-management module, into a real address. This virtual address space is divided into eight 4K (word) pages. Each virtual page is relocated separately to a real page. Each real page in memory has a 2-bit access control key. When the key is set to zero, the page is defined as non-resident. Any attempt by a user program to access a non-resident page is prevented, generating a program fault. Using this feature, only those pages associated with the current program have legal access keys. Control keys of all other pages are set to zero, preventing illegal memory references. The control keys can also be set to read-only, or to read-and-write access.

Each of the peripheral devices on the system is assigned a priority for the interrupt structure. A device may interrupt the processor only if the processor priority is lower than the priority associated with the device. The processor priority may be set. Interrupts that are not acknowledged by the processor (because of priority) are kept pending until acknowledged or, until the UNIBUS is reinitialized.

Machines of the PDP-11 family have 8 registers (numbered 0-7). Registers 0-5 are general purpose registers, 6 is the hardware stack pointer, the stack expanding in the direction of low memory, and 7 is the program counter.

The operating system, for the most part is written in a language called C, which was also developed at Bell Laboratories. The kernel portion of UNIX is written in assembler and is a small fraction of the overall code. C is a general purpose programming language which features economy of expression, modern control flow and data structures, and a wide variety of operators. It exploits the capabilities of the PDP-11 instruction set to the fullest.

2. File System under UNIX

The most important role of UNIX is to provide a file system. The UNIX file system is tree-structured and comprises files of three major categories:

(i) Ordinary files--those which contain data of any type and are byte (or character) oriented,

(ii) Directory files--those which contain pointers to files (of any type),

(iii) Special files--those which provide access to peripheral devices on the system.

Figure 1 is a typical file system structure. In this figure, an example of an ordinary file would be that named "ar". A directory is "bin" (as are "usr", "tmp", "dev" and "rk2"). A special file is "tty1".

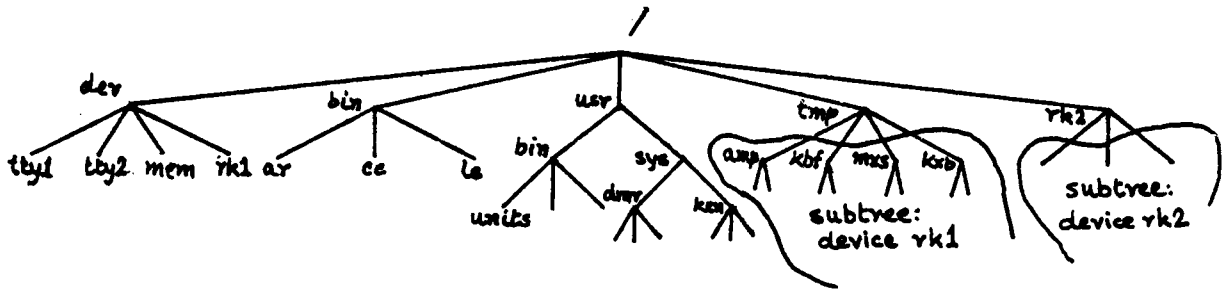

Figure 1. Typical File System Structure

There is no restriction on where any particular file may appear. Each file type is identified by associated flags in the directory entry. Additionally, each file, regardless of type, is owned by a certain user and there are flags that specify access rights to other users. The access rights are for read, write and execute, and the owner of the file may set these rights at any time.

In figure 1, there exist two files with the name "bin", both being directories. One file resides directly under the root of the tree, while the other is a child of "usr". To specify a file uniquely, a path name is used. The path is the node traversal sequence from the root of the tree (the root being designated by "/"). For example, the file "ar" is uniquely specified by the path /bin/ar. The first "/" refers to the root node; however, the subsequent "/" are used as separators for the names.

The file "units" is specified by /usr/bin/units. Every user in the system is assigned a directory and is positioned on login at that directory. Then, a file may be specified

relative to the current position. Each directory has two additional entries--one is a pointer to the parent directory and the other is a pointer to itself (these are designated ".." and "." respectively).

All executable files have access modes set for execution. All system commands are names of executable files, and thus at the command interpreter level, a user needs only specify a file name for execution.

3. Special Files

Special files are logical names for physical devices. To a system user, there is absolutely no difference between a disk file and a special file. These files can be opened and closed for I/O just like ordinary disk files, however, when a read or write occurs, the operating system takes special action, depending on the physical device characteristics, to initiate I/O. At the time of creation of a special file, the special file type--character or blocked--is specified. In the case of I/O on character special files, there is no fixed-length data buffering, whereas in the latter case, the data is first brought into a 512-byte buffer. Typically, disk volumes are normally buffered in this manner.

Another feature mentioned earlier is the incorporation of demountable volumes. Any disk volume (accessed through logical file /dev/rk1, /dev/rk2, etc.) may be mounted on any

directory. In every volume, files exist in a tree structure. By mounting a volume to a directory, the volume's tree becomes a subtree of the parent directory to which mounted. As an example, the rk1 volume subtree (in figure 1) has been mounted on the directory "tmp". The volume's root has directly below it files named "amp", "mxs", "kbf" and "kxb". In mounting the volume, the path specification for "amp" becomes /tmp/amp from the root of the system.

What, then, is the "root of the system"? One disk volume, usually device 0 on a PDP-11, is designated as the system disk, and when UNIX is booted up, this volume's tree is brought into existence as the resident tree. Thereafter, other volumes may be mounted.

4. User Interface

From the system standpoint, this is an elegant setup. How does a user utilize the facilities provided? The user-system interface is a command interpreter called the "shell". The "shell" is not only a command interpreter, but also a programming language in a sense. This program is not part of the operating system and enjoys no special privileges. In fact, it is entirely up to the user to specify a command interpreter. By default, the widely distributed "shell" is the interpreter.

The "shell" utilizes the system primitives to set up the execution of a file specified, if executable. It also has a default search procedure for the filename specified.

CHAPTER III

CHARACTER I/O UNDER UNIX

Being a multi-user operating system, UNIX supports a number of time-sharing terminals. The PDP-11 is an interrupt driven character I/O machine. The PDP-11/34 in question has 4 KL11/DL11 single line, asynchronous interfaces. Each time-sharing terminal is connected to one of the four KL11/DL11 lines. Each interface has fixed locations (settable by on board switches) at which the processor traps when interrupts are produced by it. In addition, there are other types of hardware devices used to interface terminals with the computer (DZ11--16-line, asynchronous, buffered serial line multiplexer and DH11--16-line, asynchronous, buffered serial line multiplexer with DMA transmission are examples).

Each type of interface has its own control characteristics and, each requires a separate operating system device driver. In UNIX, the common code that is sharable between these device drivers is grouped together in one module. However, separate modules exist for each of the device types. The KL11 driver interface is always present in UNIX, since the operator console is interfaced with this driver.

As mentioned earlier, each physical device is associated with a special file which exists on the system disk. How is a special file on disk logically associated with a physical device? This question can be answered now, using the KL11 interfaces as an example.

Each KL11 interface has, on the circuit board, a set of switches that define the memory addresses at which the processor should trap in the event of an interrupt produced by it. There are two types of interrupts--the transmitter interrupt and the receiver interrupt. The transmitter interrupt occurs when a character has been successfully transmitted, while the receiver interrupt occurs when a character is received.

At the preset trap location (e.g., receiver interrupt location) are two words. The first is the address of the interrupt pre-processor, and the second is the new Program Status register value. This PS value is different for the various terminals. Thus it is at this point that the identification of the interrupting device is done. The interrupt pre-processor ultimately calls the common KL receiver interrupt routine with an argument. This argument is the device id, which is set by the pre-processor, by zeroing out all but the least significant five bits of the PS word, and then thrown on the stack. This implies that a maximum of 32 KL11 lines may be handled.

When the special file is created, a device number is supplied. This is the same number that is used to identify the device on interrupts. This number is called the device minor number.

Thus the receiver interrupt handler ("klrint") obtains the identification number of the line as an argument. This same process occurs for a transmitter interrupt, except that the transmitter interrupt handler ("klxint") is called.

1. The "tty" Structure

For each teletype, there exists a system-maintained structure (in a programming language sense). Each "tty" structure contains all the relevant information for that teletype. The information includes the modes of the teletype, the current erase and kill characters, and the information on the input character, raw and canonical, queues, and the output character queue.

The teletype mode is user-settable (although default values are built in). The mode is a set of flags indicating characteristics such as whether the input character should be echoed (echo), whether the input should be pre-processed for erase or kill up to the newline (same as linefeed) character (raw), whether tabs should be converted to spaces (tab), whether carriage return on input should be converted to a linefeed (nl), among others.

2. I/O Queues and User Access

For each teletype, there are a set of queues of characters. These are the raw queue, the canonical queue, and the output queue. When a receiver interrupt occurs, the receiver interrupt handler is called with the device id number. This routine calculates the address of the "tty" structure associated with the device, reads the character from the receiver buffer, and then calls a common routine "ttyinput" with arguments the input character and the address of the structure.

"ttyinput" then puts the input character into the raw queue, updating the information on the raw queue (such as count and pointers to first and last block of characters, each block containing six characters and a pointer to the next block). If the raw mode bit is set, it transfers the character to the canonical queue, and awakens any process awaiting a character input from this teletype. If it is not in raw mode, and if the input character is a quit or interrupt character (ASCII 034 and 177, octal, respectively), "ttyinput" signals the process that is controlled by the teletype.

Depending on whether the teletype is in echo mode or not, it places the input character on the output queue. A process trying to read an empty canonical queue is put to sleep awaiting input. The transfer of characters from the raw queue to the canonical queue in the non-raw (or cooked)

mode is done only when a newline character is input. During this transfer, all erase and kill processing is done on the input line, and only the resultant characters are put in the canonical queue.

The input queues have a maximum length of 256 characters. If the input line is longer than that or there is no process reading the input in raw mode, the buffer is flushed and 256 characters are lost.

All that has yet been described is the system character I/O management. What of the user interface--how does a user process access these queues? When a special file is created, another number known as the device major number, is specified. This number is the line number in a structure of entries. Each entry (line) contains five addresses. These addresses are of the routines that handle all I/O on devices of that category. There are two such structures--one for character oriented devices and the other for block oriented devices. The following discussion is pertinent to character devices. This structure is "cdevsw", an acronym for "character device switches".

The entries are the addresses of routines to handle opening, closing, reading and writing. In the case of teletypes, an additional entry is the address of a routine to handle setting of teletype mode flags and characteristics. Addresses that are not of importance to certain devices are set to "nodev" or "nulldev". "nodev" is

used for illegal entries and "nulldev" for insignificant aftries. This structure also contains entries for other devices such as paper tape reader/punch and line printer. An example of the usage of the "nodev" address (which returns an error) is the set device mode position for a line printer. "nulldev" is used in open and close positions of devices that are always open, like memory. (See Table on page 27)

When a user process opens a file for reading or writing, the system open routine examines the file type. If it is a special file, then depending on whether it is a character or block special file, the routine indexes into the "cdevsw" or "bdevsw" tables. It then picks up the address of the open routine, and calls it with the device minor (device id) number. The device open routine calculates the address of the device structure ("tty" in the case of teletypes), and then performs the initialization of the values.

In the case of a teletype (KL11/DL11 interface), the routines to handle the various functions are "klopen", "klclose", "klread", "klwrite" and "klsgtty". This last routine is also used to obtain device characteristics. The "klread" and "klwrite" routines, called in a similar way, simply calculate the address of the "tty" structure, and then call "ttread" and "ttwrite" respectively.

The "ttread" routine tries to get a character from the canonical queue by calling "canon" with the address of the "tty" structure as argument. If "canon" is unable to return a value, it goes to sleep, awaiting a character from the raw queue. Thus a read on an empty queue puts the process to sleep until either in raw mode a character is input or in preprocessed (buffered) mode a line is input.

The "ttwrite" routine goes into a loop, getting characters from the process' output buffer (set up in core on an open) and starting transmission if the character count is above a pre-defined "high water" mark. This transmission terminates when the count goes below the mark. The input character is then placed on the output queue by "ttyoutput". It is apparent from this that at any time, there are between zero and "high water" mark number of characters awaiting output.

There cannot be an overrun of the output buffer since the process performing output sleeps during this call. "klopen", as earlier described, sets initial values (default) for the terminal characteristics. "klclose" simply alters the flag bit that indicates that the device is open. "klsgtty" is called to set and determine (get) teletype characteristics.

Any read, write or set calls to the system on a closed line returns without any action. All that has been described so far is one level removed from the user. The actual interface is in the system entry table. There exists a number of system calls, which include "read", "write", "open", "close", "stty" and "gtty" (for set and get tty). When one of these routines is called, the actual processing is done.

The system entry table contains a set of addresses of routines that perform various functions other than I/O. This table, called "sysent", contains the addresses of all operating system service routines that are accessible to user processes. Some entries of interest are "signal" and "kill". These are process-management governing routines.

3. Signals and Signal Handling

A "signal", by definition under UNIX is the occurrence of a certain condition (e.g., quit or interrupt transmitted from the teminal). This occurrence could be fatal to a process, depending on the default actions. In the case of quit and interrupt, the default action is termination. Additionally, the former produces a core dump.

Signals are sent to a process when a memory fault occurs--a process generated address is out of bounds, a bus error occurs, or even when a system call is made with

bad arguments. All these result in termination (by default). There is a provision by which a process may set the action to occur when a signal is generated. This action is set by a system call "signal", with arguments the signal number (each type of signal is assigned a number) and the action. The action specified could be 0, 1 or an address (even value). A 0 indicates that the default action is to be taken, 1 indicates that it is to be ignored, and an even value is assumed to be the address of a routine or label that handles the occurrence of the signal--this address is in the user process.

If a signal is caught (an address specified in the "signal" call made prior to the occurrence), control in the process is transferred to that address. If the address is of a routine, return to normal processing is possible, when the execution of this routine is complete. Once a signal occurs, the action for it is restored to the default value, unless reset by the process.

One of the signals cannot be caught or ignored--the kill signal. The operating system has the provision for 20 signals, 13 of which are currently implemented.

CHAPTER IV

DECSYSTEM-10 OVERVIEW

The DECsystem-10 is a main frame computer at The University of Tennessee. It is primarily an interactive time-sharing machine, and runs the TOPS-10 monitor. It is a 36 bit-word computer with a KL-10 processor. It has 6 disk drives providing storage of up to 800 Megabytes. The memory units (two) are MG-10s of 128 K-words each. It supports about 60 time-sharing terminals.

In addition, it is interfaced with the University's IBM system consisting of a 360/65 and a 3031. The DECsystem-10 supports IBM 2741 type communication terminals (which use Correspondence Code) and ASCII terminals. There are numerous locations on campus from which the DECsystem may be accessed.

The TOPS-10 monitor is the operating system. There is also a batch job operating system called GALAXY-10 which facilitates batch job processing. The software support on these systems includes a large number of languages and utility programs. There are at least three text editors on the system, including one full-screen editor. Utility programs are used to manipulate disk files.

1. Disk Files and Directories

Disk files have a 3 digit octal protection code, with the digits establishing the privileges of the owner, others in the same project and the rest of the users. Users of the DECsystem-10 are assigned two octal numbers, which are known as the project-programmer number (ppn), and a password. The ppn is used for accounting purposes by the Computing Center (UTCC). There may be a number of users assigned the same project code. The second protection code digit refers to these users.

Each user is assigned a User File Directory (UFD) which is a child of the system maintained Master File Directory (MFD). This is indicative of a hierarchical file structure. Each directory, also, has a set of protection codes. On logging on to the DECsystem-10 by supplying the ppn and password, when prompted, the user is positioned within the UFD assigned. At this point, it is possible to create files or directories. These directories are called Sub-File Directories (SFD).

2. User Access of the DECsystem-10

The Computing Center provides remote access to the DECsystem-10 via leased lines and telephone links. When a user dials one of the numbers specified, the answering modem transmits a carrier. It is necessary to then transmit a

carriage return so that the modem can adjust itself to the speed of transmission (110, 150, 300 bauds are the standard ASCII speeds and 136.5 baud is for IBM 2741 type transmission). The user is then in System Select. All this is handled by the Communications front end, the DC-76. At the System Select level, typing the pre-set character combination would select one of the available time-sharing systems (such as the DEC-10, ATS-360, TSO and VM-370).

Typing "pd" selects the DECsystem-10. Depending on the availability of the computer, it produces a message. If a "LOGIN" message appears, the ppn and password are subsequently entered. The password is not echoed for security.

The TOPS-10 monitor handles all process initiation. It can handle only one process at a time from a terminal, unlike UNIX. The terminal is thus attached to the process in execution until termination of the process.

3. Control Characters

The user, from the terminal, may transmit control characters to the computer. On the keyboard, a control character is typed by simultaneously depressing the key designated "CTRL" and any alphabetic character key. Some control characters have special meaning to the DEC-10. Most control characters are echoed at the terminal with two

characters--the first being an upward pointing arrow and the second the upper-case alphabet (e.g., CTRL-C is echoed as "^C").

Four control characters are not echoed, two of which are the CTRL-S and CTRL-Q. These are characters that control the transmission from the DEC-10 to the terminal. CTRL-S is otherwise known as XOFF, and CTRL-Q as XON, meaning transmission (X) off and on. Thus, if the screen is filling up too rapidly, a user may type XOFF to suspend transmission until XON is sent.

Table 1 gives a description of the control characters that have special significance to the DEC-10.

Table 1

DECsystem-10 Terminal Control Characters

Keyboard Character	Echo	ASCII Value	Description
CTRL-C	^C	003	Abort the current process
CTRL-H	backspace	010	Same as backspace key
CTRL-I	tab	011	Same as tab key
CTRL-J	Linefeed	012	Same as Linefeed key
CTRL-L	Formfeed	014	Same as Formfeed key
CTRL-M	Carriage Return	015	Same as Carriage Return key
CTRL-O	^O	017	Suppress output to terminal
CTRL-Q	no echo	021	Continue transmission (XON - complement of XOFF)
CTRL-R	no echo	022	Retype partial line
CTRL-S	no echo	023	Stop transmission (XOFF - complement of XON)
CTRL-T	no echo	024	Status information on current job
CTRL-U	^U	025	Kill partially entered line
CTRL-Z	^Z	032	End of file mark from terminal

4. DECsystem-10 Communication Protocol

The computer-terminal transmission protocol involves the use of XON and XOFF. By the same token, when the system load is heavy and the computer is close to filling its input buffer for a terminal, it transmits an XOFF, and an XON when safe to continue. However, at the terminal, these characters being non-graphic, nothing is apparent. If transmission continues from the terminal after the XOFF has been sent, a few more characters are accepted until the buffer is full. At this point, any more transmission results in the BELL character (CTRL-G) being echoed, and the input is lost. This is audible indication of the transmission loss.

This protocol is the basis for the communication handler between UNIX and the DECsystem-10.

CHAPTER V

MODIFICATIONS TO UNIX

When the UNIX operating system is first booted up from a disk, through the ROM boot on the PDP-11, it performs all initialization. In the course of this, it examines the switch register on the machine to determine the version of UNIX to bring up--single user or multi-user.

If the multi-user version is selected, it determines which teletypes exist, by examining the system file "/etc/ttys". This file contains a list of all possible teletype identifications, with an indication of whether it is existent on the system or not. Each of the teletypes designated as existent has a special file associated, with path name "/dev/ttyx", where x is the terminal identifier (0-9, a-z).

During the course of process initialization, a process is created for each of the teletypes marked as existent and this is the controlling process for it. The teletype is automatically opened, and the controlling process reads from it. The terminal characteristics are default settings, with input set to raw mode. As soon as there is input at the terminal, the login program is invoked to let a user log on to the system.

1. Special File "/dev/dec10"

For the purposes of communication with the DECsystem-10, a special file, which does not have a controlling process, has to be created. This is absolutely necessary--no process but the user process that communicates with the DECsystem-10 should read from this special file. Consequently, it is necessary to dummy out the existence of a physical line (KL-11) in the "/etc/ttys" file. Another problem, then is to create a port that is automatically open with the teletype settings being raw, and with no modifications to the input.

This necessitates the creation of a special file "/dev/dec10", that has its own open routine, which sets the teletype characteristics to the requisite specifications. This routine has to be included in the teletype handler module. The only differences between this and the standard open routine are the mode setting flags (in the associated "tty" structure), and a flag to indicate that it is a DEC-10 communication port.

2. Addition to "cdevsw"

In creating the special file, the device identification number supplied has to correspond with the identification number of the dummied teletype, so that the trap vectors for I/O would remain the same. The device major number (the line number in the file of handler routine addresses), however,

is different, necessitating the addition of another entry in the table "cdevsw". This entry is the same as that of the standard KL-11 handler, except for the open routine, which is the address of the newly included open routine (decopen, klopen being the standard).

Table 2 is a partial listing of the table "cdevsw". With the port created, I/O through "/dev/dec10" is made possible, the physical line being that of the dummied terminal.

Table 2

Partial list of entries in table "cdevsw"

```
&klopen,  &klclose, &klread, &klwrite, &klsgtty, /* console  */
&nulldev, &nulldev, &mmread, &mmwrite, &nodev,   /*    mem   */
&decopen, &klclose, &klread, &klwrite, &klsgtty, /* dec link */
```

Input characters from a KL-11 line are buffered internally under UNIX, the queue (buffer) management parameters being in the "tty" structure for that line. As earlier described, the maximum queue length is 256 characters.

3. UNIX Input Buffer Losses

Since UNIX is a multi-programming time-sharing system, a process reading from a port will probably not get

continuous control of the processor. In fact, under certain circumstances, the process may be swapped out of core for over a minute. Additionally, there are system overheads. In the situation where one process reads from a port, if the transmission is sufficiently rapid, character losses are unavoidable.

Disk I/O is performed in blocks of 512 bytes under UNIX. It defers the I/O of full blocks when possible. However, every 30 seconds, it performs a disk update, processing all outstanding I/O requests and updating the volume's super-block. During this process, the processor priority is set so high that no character device (like a teletype) can interrupt the processor. This is a potential character loss situation, which proved to be the case in subsequent testing. This update is known as the "sync".

All these situations arise even at 300 baud transmission, and worsen at higher rates. Communication with the DECsystem-10 over a telephone line is at 300 baud, with each computer viewing the other as a simple terminal.

Existing device handlers cannot handle the situation wherein the queue limit is exceeded (resulting in the loss of 256 characters). They have to be modified to recognize the imminent buffer overrun and to signal the DECsystem-10 to stop transmitting. The signal itself is the XOFF

character (CTRL-S--Octal 023). A modification is necessary to the input handler, whereby it transmits this character to the DECsystem-10 when 240 characters are in the queue.

What of the restart of transmission? When the process reads the port, if the queue count is a preset low value, the XON character (CTRL-Q--Octal 021) is transmitted. A flag is also set so that this XON transmission is not performed each time, but only if an XOFF had been previously sent. Additionally, since these handlers are common to all teletypes, this XON/XOFF transmission is done only if the DEC port flag is set in the "tty" mode.

The modifications to the handlers, thus, affect only the port that is designated to handle DEC-10 communication. This solves the problem of character loss due to buffer overrun. The second problem, that of character loss during a disk update, has to be solved another way. Owing to the multitudinous levels of call in the update, and the difficulty in identifying the "tty" structure associated with the DEC-10 port, there is no straightforward patch that can be added to UNIX.

One fact, however, helps in the solution to this problem. That is the fixed inter-sync period of 30 seconds. Any process in the system can force the "sync" to occur, as can a shell level command. When a "sync" is forced by a user or a process, the 30 second count down to the next "sync" resets.

The solution to this problem, then, is to have the process that reads from the DEC-10 special file force the "sync" to occur every 25 seconds. However, before the process does this, it would have to transmit an XOFF to the DEC-10, and after the "sync", transmit an XON. This is needed as the device handler will not transmit either the XOFF or XON unless there is an imminent buffer overrun. The XON transmission cannot be left out either, since the handler does not transmit it unless it had originally transmitted the XOFF.

Yet another major problem occurs when the RX01 floppy disk drive is being used. This device is not of a Direct Memory Access type, and when operating, causes character losses even at user terminals. This problem cannot be overcome, only eliminated--the floppy disk drive cannot be used when the interface is processing.

4. DECsystem-10 Input Buffer Overrun

One way loss free communication is thus facilitated (DEC-10 to UNIX). By the DECsystem-10's communication protocol, it transmits an XOFF if its buffer overrun is imminent and an XON when safe. Output under UNIX is also buffered at the handler level. A user process has no control over this transmission. The problem, then, becomes one of recognizing that the DECsystem-10 has transmitted an XOFF.

The input handler can recognize this character, but it has no control over the output handler, and hence the output handler continues transmission. The only logical solution is to somehow indicate to the process writing to the DEC-10 port that an XOFF had been received, and let that process manage the problem.

Under UNIX, there is the concept of "signals" that a process could receive. These are technically, interrupts. The operating system has a set of default actions for the "signals" that may occur. The process could, however, override the default action of all "signals" but one, the "kill" signal. Signals are numbered, each corresponding to some condition arising. Thirteen out of a possible twenty signals are system implemented.

5. Addition of Signals

To override the default action, an operating system function, "signal", is used. This function is called with two arguments. The first is the signal number, and the second is an integer value (the action). If the value is zero, the default action on the signal (in all cases, program termination), is reinstated, and if it is one, the occurrence of the signal is ignored. If it is any other value, it is interpreted as the address of a signal (or interrupt) handler, which should be called if the signal specified occurs.

The operating system maintains a twenty element array of integers, to describe the actions on each of the signals. The default value of each element is zero (program termination). The "signal" system call replaces the appropriate element value with the supplied value. It returns the old value to the calling process.

When the signal occurs, the action specified is taken, and the value of the element is set to zero. Thus, the process has to call "signal" again to reset the value after the occurrence of the signal.

This feature proves to be a way to signal the transmitting process of the occurrence of an XOFF, and likewise of an XON. Two more signals are thus added--signal number 14, corresponding to the detection in the input of an XOFF, and signal number 15, of an XON.

The process could set the action on these two signals to handle loss-free transmission. To avoid losses, therefore, the communication has to be monitored by the operating system and the process. There certainly exist other ways in which this can been handled. However, these would involve more modifications to the operating system than seem necessary.

CHAPTER VI

INTERFACE DESIGN

1. Batch System

After all the modifications to UNIX, communication with the DECsystem-10 over a voice-grade telephone line is made possible, with the requirement that the process handling the communication also handle the system "sync" and monitor the occurrence of an XOFF.

It was envisioned that the communication would be initiated periodically. This implies that no one user would be able to use the DECsystem-10 interactively, since that would tie up two physical lines out of four. The communication facility, therefore, has to be in batch mode--with the user submitting a file of commands that would be interpreted by the communication program. These submitted files would be queued, for processing at a later point.

2. Files for User Information

To handle batch mode communication, the user's login information would have to be system maintained, with the user having to supply all the necessary information earlier. Thus, at this point, two files appear necessary--one for the queue of submitted files, and the other for the login information.

The files would have to be protected from the other users under UNIX. There is one user under UNIX known as the "super-user" or "root". This user is the only one who has total access to the system files. Each user has a specific UNIX user identification number (0 - 255). The super-user has a user id number of 0. Disk files that are "owned" by the super-user and protected properly are inaccessible to any other user on the system.

The login information file (named "passwd") and the queue file (named "queue"), hence, have to be created with the owner being the super-user, protected against read, write or execute by others.

At any time, a user should be able to add an entry to the "passwd" file. The entry contains the UNIX user id of the adding user, so that if anyone else tries to access the entry (for example, to delete it), the access is denied. A process initiated by a user normally takes on the id of the user. This would prevent the process from accessing any file protected against the user. However, if a file owned by "root" or super-user provides execute privileges to other users, the process takes on the id of "root". By setting a certain mode flag for the file, it is possible to enable the file to have super-user access rights, but make it assume the user id of the user executing it. This mode is called the "set user id" mode.

3. Access of User Information Files

Any program that accesses the "passwd" and "queue" files must have its mode set to the kind of access described above. Any user should be able to add an entry and delete his entry in the "passwd" file. These would be two separate programs. The command file submission is handled by another program that accesses both the "passwd" file (to check for the existence of a valid entry) and the "queue" file. The actual communication program accesses both these files, to pick up the submitted command file and the login information for the DECsystem-10.

From this, it is seen that four independent programs access the same data sets. Two of these programs (which add and delete entry) modify the "passwd" file, while the other two read from this file. It is apparent that to prevent scrambling the contents of the file these programs cannot simultaneously access the file. This leads to the need for a semaphore of some sort which prevents multiple access.

Under UNIX, a process may create new files, or open existing files. The creation of a file is possible even if the same file exists. With proper access rights, all this does is truncate the existing file to zero length. An open on a non-existent file returns an error to the calling process. In any case, these calls to the system (create and open) return a file descriptor value, which is used in subsequent reads and writes.

4. Exclusive Control of Files

The create system call is often used to set up a semaphore. This is done by having a process create a file with read only access rights. If another process tries to create the same file, since the access rights permit read only, the system call returns with an error value. This would be an indication to the second process that the critical resource is in use.

Testing, however, proves that this is not always the case. A process running with super-user access rights does not receive an error return, even if the file being created exists with read only access mode. This is the privilege of the super-user. This is a minor problem since all the programs that access the "passwd" file have super-user access rights.

An alternative comes about in the usage of the open system call. This call returns an error if the file does not exist. Hence, if the first process attempts an open and receives an error, it can be assured that the semaphore file is non-existent (i.e., no other process has yet created it), and can proceed to create it. The second process coming in goes through the same steps; however, since the open succeeds, it exits, with an appropriate message. This sequence of steps does not eliminate a race condition

between the programs, but only minimizes the chances of it. This is because of the two steps involved in the testing and creation of the semaphore file.

5. Format and Manipulation of "passwd" File

An entry in the "passwd" file is a fixed length record, containing a one to five character entry id, the DECsystem-10 project-programmer number, and the UNIX user-id, each piece of information being delimited by a colon. The unused space in the record (since it is fixed length) is padded with some characters. Additionally, the first character of a record is a blank, if the entry is "alive", or a non-blank if it is a candidate for deletion.

The records are made fixed length for random access. If a user deletes an entry, provided it is permitted, the entry is marked for deletion by changing the first character. Subsequently, if an entry is added, a deleted record is over-written. This method is used to avoid compaction of the file when an entry is deleted. The only disadvantage is that the file length is always the maximum length attained.

6. Deadlock in the Communication Program

In order for the communication program to perform effectively, the command file that is submitted by the user should not contain DECsystem-10 commands that deadlock the program. Assuming that any DEC-10 command is permissible in

the command file, this deadlock situation can occur if the DEC-10 expects further interaction on a command. An obvious example is the execution of an interactive program on the DEC-10. The communication program would not be in a position to distinguish between a simple command (one that returns the DEC-10 to its monitor) and a command of this type.

A read under UNIX from the DEC-10 port will not return until a character has been obtained (or end of file is reached). This can put the program in a situation where it expects a character from the DEC-10, and the DEC-10 expects an input from it. This brings about a real need to limit the range of permissible DEC-10 commands in the command file. To facilitate this, a program assists in the creation of the command file. It interacts with the user and creates commands in the command file that would be interpreted by the communication handler.

7. Command File Creator Program

This command file creator has a basic set of commands. It produces a formatted file with entries of three basic forms--directly transmissible DEC-10 commands, commands that require interpretation by the communication program, and user supplied commands. An example of a directly transmissible command is the DEC-10 directory listing command. This, when transmitted, is guaranteed to return to the DEC-10 monitor level after the generation of the

listing. A command that requires interpretation is one that transfers a file from one computer to the other. This requires interpretation since there is disk I/O to be performed at both ends. A user-supplied command is directly transmissible by the program, with the knowledge that it is a potential deadlock situation. A deadlock in this environment can be easily broken by the transmission of the abort character (CTRL-C--Octal 003) to the DEC-10.

The command file formatting convention is followed by the creation and the communication programs. The latter rejects a command file that does not conform to this format, and aborts the processing of that request.

8. Submission of the Command File

Once a command file is created, it is submitted to be added to the "queue" file. This is handled by the "submit" program. This program requires the name of the file to be submitted and the entry id to be used in the "passwd" file. The file name may be specified as a pathname. This pathname could be with reference to the root of the system (pathname begins with "/"), the current working directory, with the file being in another (begins with "../"), or with the file being in or below the current working directory (begins with the first character of file or sub-directory name).

The "submit" program determines if the record in the "passwd" file that matches the supplied entry name permits

the executing user to access the information on it. This check is performed by comparing the UNIX user id on the record with that of the executing user. If access is permitted, it resolves the supplied file name into a total path name from the root of the system, and verifies its existence.

When these have been verified, it appends an entry to the "queue" file that contains the path name of the submitted file and the entry name in the "passwd" file. This completes the user interface design aspects. The communication program, when executed at a later point, processes the requests in the "queue" file, sequentially.

9. Processing by the Communication Program

It obtains the information for logging on to the DEC-10 from the "passwd" file and performs the login. It then interprets the entries in the command file and performs the requisite interaction. A log file is created in the directory where the submitted file exists, and this file contains a log entry for every command processed, along with any diagnostics produced by the communication handler. Retrieval of a file from the DECsystem-10 causes the creation of the destination file on UNIX. The log entry for the transmission and retrieval of files indicates the source and destination file names used.

This communication handler can be run as a background job under UNIX, thus not requiring a teletype to control it.

CHAPTER VII

SYSTEM IMPLEMENTATION

This chapter discusses the implementation of the system whose design is described in the previous chapter. The communication package consists of five independent programs that form the user interface and the actual communication.

1. System Files

The user DECsystem-10 login information is maintained in the file "/tmp/dec/passwd". The queue file is "/tmp/dec/queue". The directory "dec" is protected against access by any user other than the super-user. All files having to do with this system are under the directory "/tmp/dec". Other files include the semaphore file, "/tmp/dec/semaphore", and a temporary file used to find the current working directory of a user submitting a command file to the queue.

2. Entry Format of System Files

File "passwd"

The "passwd" file contains fixed length records. This file is written on by the program that adds an entry. The entry is 50 characters in length. The first character in the entry has a special meaning. A blank signifies that the entry is current, while a non-blank indicates that it is a

candidate for deletion. Starting at the second position is a 1 to 5 character entry name. The delimiter used throughout is the colon. The next field contains the DECsystem-10 project number (in Octal) followed by the programmer number (also Octal). The password field comes next, followed by the UNIX user identification number terminated, by a colon. The remaining space in the entry is padded with dashes, making up 49 characters. The fiftieth character is the record separator, the new-line character. An example entry is shown in Table 3.

File "queue"

The "queue" file entry contains two fields separated by a colon. Since the processing is line by line, this file is in no special format. The first field contains the path name of the submitted file, followed by a colon. The first character after the colon is a blank, followed by the entry name. The blank is put in so that the entire second field can be compared with the first field of the entries in the "passwd" file. An example entry is shown in Table 3.

Table 3

Example entries in files "passwd" and "queue"

File	Entry
passwd	mxs:76141:160030:xyz:206:----------
queue	/tmp/mxs/thesis/dec.command: mxs

3. Adding an Entry to "passwd" with "addusr"

The routine that adds an entry to the "passwd" file (named "addusr") queries the user for the entry name to be used. It then performs a random-sequential scan of the entries, comparing the existing names with the one entered. In this process, if it encounters a previously deleted entry, it keeps a track of the line number of such an entry. If an entry is found that has the same name, it asks the user to try another name. If all entry names differ from the entered name, it queries the user for the rest of the information, performing a simple validity check on the entered DECsystem-10 ppn. A ppn is considered valid if it is two octal numbers, separated by a comma, of more than 5 characters and less than 13 in length. The password is limited to 8 characters. It obtains the user's UNIX identification number, converts it to character form, and completes the entry. It then displays the entry information and asks for confirmation. If confirmed, it proceeds to write the entry into the file after padding with the appropriate number of dashes.

If a previously deleted entry exists, it rewrites that entry. If a number of deleted entries exist, it chooses the one that is closest to the end of the file. (Appendix C contains a sample interaction to add an entry)

4. Deleting an Entry in "passwd" with "delusr"

Deletion of an entry is performed by "delusr". This routine queries the user for the entry name to be deleted. It does a random-sequential search until the entry is found, skipping over deleted entries. Since the records are of fixed length, the starting position of each entry is determined easily. The reading of the first six characters of an entry is sequential, but thereafter, it does a seek to the starting position of the next entry. The term "random-sequential" is used to describe this action. If the specified entry is found, it obtains the executing user's UNIX identification number and compares it with that in the entry. If they match, it marks the entry as deleted, otherwise, it informs the user of the fact, and exits.

Exclusive Control

Both these routines ("addusr" and "delusr"), at the outset, attempt to obtain exclusive control of the file "passwd". This is done by trying to open the file "/tmp/dec/semaphore". If the file exists, nothing is done, and the routines exit. However, if it does not exist, it is immediately created to block other programs from accessing it. This does not altogether ensure exclusive control, but is the closest to it.

5. Command File Creation with "creator"

A command file that is to be submitted has to be created by the program "creator". This program has a simple set of commands, including a help listing. It creates the formatted file in the directory specified. The formatting involved is the inclusion of control characters at the start of each line. These characters are CTRL-X, CTRL-Y and CTRL-Z (Octal 030, 031 and 032 respectively).

A CTRL-X at the start of a line indicates that the rest of the line is directly transmissible to the DECsystem-10. Only two DECsystem-10 commands are currently implemented--the directory listing command and the execute file(s) command. A CTRL-Y specifies that the line is to be interpreted by the communicator. This is used in two cases--transmission and retrieval of files to and from the DECsystem-10. This involves the setting up of the transmission and opening the UNIX file involved (source or destination). Entries of this form contain the names of the DECsystem-10 and UNIX files. A CTRL-Z indicates that the subsequent lines (not containing any control characters) are to be transmitted to the DECsystem-10, with the output from the DECsystem-10 being put in the log file. This set of lines is delimited by the next encountered control line.

At any time, the user can quit or end the creation of the command file. Quitting does not save the file, while ending the session produces a message indicating the name of the file that is to be submitted.

6. Submission of the Command File with "submit"

Once the command file is created, it needs to be submitted to the queue. This is handled by "submit". This is invoked with two arguments--the name of the file being submitted and the entry name in the "passwd" file. Diagnostics are produced by this program if either the submitted file does not exist or, the entry name is non-existent. This program also requires exclusive control of the "passwd" and "queue" files, and goes through the process of ensuring this by testing the "semaphore" file.

It then checks if the entry UNIX identification number corresponds with the executing user. If not, it exits with a diagnostic. It then resolves the specified file name into a total path name from the root of the system (if not already specified in that manner). This path name and entry name are appended as one line to the "queue" file, and then the program exits.

7. The Communication Program "proc.dec10"

This completes the description of the implemented user interface. When the communicaton program is executed, it

too, ensures exclusive control of the files, failing which, it exits. It then checks the existence of the "passwd" and "queue" files, exiting if either is missing. It prompts the operator to connect the designated KL11/DL11 line to an originate modem, and then dial the DECsystem-10 number. In the process of initialization, if the DECsystem-10 is unavailable, it produces a message and exits. The initialization of the line involves setting the communication rate on the answer modem on the DECsystem-10. The transmission of a carriage return does this. Then in system select, "pd" is transmitted to select the DECsystem-10.

Logging in to the DECsystem-10

Thereafter, it processes the requests in the "queue" file, line by line. It positions itself in the directory in which the submitted file exists, and creates a log file in that directory. It scans the "passwd" file for the relevant entry, and performs the login into the user's ppn on the DECsystem-10. The login is performed by a child process, created by the "fork" system call. This function call has the action of creating a duplicate image of the parent process, returning the process identification number of the child to the parent, and a 0 to the child. This facilitates alternate code to be executed by the child and parent, just by testing the returned value. The parent

process sleeps for a period, while the child attempts to perform the login. If the login fails, the child process gets deadlocked, and the parent, on awakening, detects this occurrence. If it succeeds, the child process exits.

Ensuring Successful Login

The "kill" system call is used, in general, to send a signal to a process. By sending the kill signal, a process may be aborted. It may be issued by any process, with the arguments being the signal number and the process identification number of the recipient process. The action is taken only if the issuing process has the same user identification or if it is a super-user process. Technically, this call returns an error value (-1) if the recipient process is non-existent. This would be the way to detect if the child process, in this situation, is active or non-existent. However, this function does not behave in the specified manner. An error indication is never returned.

An alternative is to have the child process create a certain file at the beginning of its execution, and have it remove it before exiting. This way, if the parent process tries to open this file and finds it existent (the open call returns -1 if the file is non-existent, else it returns a positive value), it is sure of the existence of the child process. This implies that the child is either deadlocked or that the login is taking longer than normal. The former is

assumed, and the parent kills the child process and proceeds to the next request, after writing a diagnostic in the log file.

Command Line Processing

After a successful login, the time (obtained from the DECsystem-10) is put into the log file. Next the command file is opened, and the commands interpreted line by line, with the control character at the beginning of the line indicating the type of command it is. This convention is used by the "creator" program. If, at any time, the format is not as expected, processing is aborted for the user, and it logs out of the user's ppn with the appropriate diagnostic in the log file.

User Supplied Command Processing

If a user supplied command line is to be processed, or an execution is to be initiated, it creates a child (as at login time) to perform the actual transmission. A two minute (tunable) wait is executed by the parent, in steps of ten seconds. If, at the end of two minutes, the child has not completed, the parent kills the child, and transmits the abort character (CTRL-C--Octal 003) to the DECsystem-10. This is to break a deadlock that may exist.

File Transmission

Transmission of a file from UNIX to the DECsystem-10 is handled by running PIP (Peripheral Interchange Program) on the DECsystem-10 and supplying the contents of the file from the "teletype", which is UNIX. Likewise, retrieval of a file is done by typing the DECsystem-10 file on the "teletype" and transferring the characters into the UNIX file.

The end of transmission from the DECsystem-10 on any command is the sequence of two carriage return and line feed characters followed by a dot, which is the DECsystem-10 monitor prompt.

Line Transmission

Transmission to the DECsystem-10 is handled by a low level routine ("transmit"), whose input arguments are an address of the string to be transmitted and the length of the string. This routine is called to handle one line at a time. On the DECsystem-10, a line is terminated by a carriage return. This routine is called to transmit a line, without the carriage return. Its function is to compare the transmitted characters with the echoed characters, to check the integrity of the transmission. If a character does not compare, or if an XOFF had been received, it transmits a line-kill character (CTRL-U--Octal 025). It then flushes the input up to two characters beyond the detection of the "^U" sequence, which is the echo produced on a line-kill.

The two characters following are a carriage return and a line feed. It them retransmits the line, going through the same comparison again.

The XOFF detection is handled by the system signaling the process. When this occurs, a signal handler, appropriately called "buffull" obtains control. A global flag is set to indicate the occurrence of this event, and the process sleeps for two minutes. If an XON does not occur (causing another signal), it returns to the point of interrupt, in "transmit". "Transmit" checks if the global flag ("ctrls") is set. The testing is done in the code that handles the comparison of the transmitted characters with the echo, and the flag is tested every time the echo character is compared. On finding the flag set, the action taken is to kill the line sent. The flag is immediately reset to be able to detect further occurrences of the XOFF. If, however, an XON occurs during the two minute period, another routine is called, which is a null routine. On return from it, control is passed from "buffull" to the point of interrupt in "transmit", where the previously described action is taken.

File Retrieval from the DECsystem-10

The retrieval of files is handled by a multi-purpose routine called "back_to_mon", which either flushes the input up to the monitor prompt, or saves the characters in a file,

depending on the input argument. This routine has a character counter, and every 600 characters read, it transmits an XOFF to the DECsystem-10, forces a system "sync", and then transmits the XON to the DECsystem-10, resetting the counter to 0. Thus every 600 characters read, a "sync" is forced in order to prevent character losses due to data overrun.

8. The Semaphore File

Every one of the programs that creates the semaphore file has all the signals that can be produced at the user teletype (e.g., quit and interrupt) and others that can be system produced (e.g., memory fault and bus error), set to be caught by the program. If any of these signals occur, control is transferred to a routine "abend", which merely removes the semaphore file and exits. This is to prevent leaving the semaphore existent when the program is abnormally terminated. However, this protection mechanism cannot include the "kill" signal, which cannot be caught or ignored. A potential problem exists if one of these programs is killed by a super-user from another teletype.

CHAPTER VIII

CONCLUSION AND EVALUATION

This system has been tested by the author to a limited extent. The performance has been satisfactory during the testing. However, a few points warrant mention and criticism.

The only possible manipulation of the information in the "passwd" file is through the programs "addusr" and "delusr". It would be of more facility to the user if there were a way of changing just some of the fields of an entry.

The communication program goes into a sleep state if the DECsystem-10 is mis-dialed. There is no indication of this apart from the fact that it does not produce the start of processing message, which appears after it has successfully initialized the DECsystem-10 line.

Login to a user DECsystem-10 number is assumed to have failed if the login is not completed within 20 seconds. This time period is insufficient if the DECsystem-10 is at its peak load. When this happens, the user's request is rejected and the user has to resubmit the file.

At this time, a wildcard file specification of UNIX files is not supported. If a user wishes to transfer a number of files, the file names have to be entered individually, as multiple transfer requests in the command file.

The log file that contains a record of all the commands processed is created in the directory where the submitted file exists. This log file, when created, is owned by "root". It would be an added feature if the ownership of the file is automatically changed to the submitting user name.

The time limit on a user supplied command or an execution is fixed to be two minutes. The user ought to be able to supply a time limit on the execution.

An as yet unencountered problem would arise if either computer went down during the communication. On the part of the communication program, it is incapable of recognizing the failure of the DECsystem-10. It would, sooner or later, get deadlocked, with no visual indication of its state. If, however, the PDP-11 crashed, the user whose request is currently being processed remains logged in to the DECsystem-10, until it is killed by the DECsystem-10 after 20 minutes. Alternatively, if the phone link is cut off, the job becomes "detached", and killed after 5 minutes. This is not a serious problem, but nonetheless will almost certainly occur sometime during the usage of the system.

Currently, the system uses a KL11 interface to communicate with the DECsystem-10. If it is necessary to run the system on a different type of interface (e.g., DZ11), the modifications to the device handler have to be duplicated for the new device. Additionally, the device

minor number of the special file "/dev/dec10" must be altered to correspond to this new device. There are no other modifications necessary to implement a change of this kind.

LIST OF REFERENCES

LIST OF REFERENCES

(1) Ritchie, Dennis M., and K. Thompson. "The UNIX Timesharing Sharing System", THE BELL TECHNICAL JOURNAL, Vol. 57, No. 6, July-August 1978.

(2) UNIX OPERATING SYSTEM MANUAL Bell Laboratories, 1976.

(3) PDP-11 PROCESSOR HANDBOOK Digital Equipment Corporation, 1978-79.

(4) DECSYSTEM-10 OPERATING SYSTEM COMMAND MANUAL Digital Equipment Corporation, 1974.

(5) Kernighan, Brian W., and Dennis M. Ritchie. THE C PROGRAMMING LANGUAGE.

APPENDICES

APPENDIX A

SYSTEM IMPLEMENTATION NOTES

1. Additions to the UNIX Source Code

Additions have been made to a number of source files. The additions are marked with bold dark lines in the file listings that follow. The modifications are briefly described, file by file.

File "tty.h"

This file contains global variables, structures and symbol definitions for the Character I/O handlers. Additions to this file are the definitions of the following symbols:

Symbol	Value	Description
CBUFFUL	023	ASCII value of XOFF
CBUFRDY	021	ASCII value of XON
TTCTLS	240	Input buffer count upper bound
TTCTLQ	5	Input buffer count lower bound
DECTTY	0100000	Flag to indicate DEC-10 special file

A global variable "ctrlqsent" has been added. This is used as a flag.

File "param.h"

This file contains system global declarations. The additions are the following symbols:

Symbol	Value	Description
SIGFUL	14	Signal definition for XOFF
SIGEMPT	15	Signal definition for XON

File "proc.h"

This file contains global declarations for process management routines. Two variables (integers) have been added. These variables, "ctrlspid" and "ctrlqpid", are set to the process id of processes that set signals 14 and 15, respectively.

File "kl.c"

This file contains the KL11/DL11 interface handlers. One routine has been added--"decopen". This routine is the same as "klopen", the one that precedes it, with differences in the flag settings in the "tty" structure. In particular, the flag "DECTTY" is set to identify the "tty" structure as that associated with the special file for the DEC-10 link ("/dev/dec10").

File "tty.c"

This file contains all the buffer management routines for the character I/O interface. Additions have been made to the routines "ttyinput" and "ttread".

The code that produces a signal on the input of an XON or XOFF is located in "ttyinput". Automatic transmission of an XOFF is also handled in this routine. The XOFF is sent only if the raw queue buffer count exceeds "TTCTLS". The flag "ctrlqsent" is set to zero, to be later checked before sending an XON. The XON is sent by "ttread", when the buffer count goes below "TTCTLQ". These actions take place only if the "tty" structure has the "DECTTY" flag set (on open).

File "c.c"

This file contains the block and character device switch structures ("bdevsw" and "cdevsw"). One entry has been added to the "cdevsw" structure, which defines the addresses for the special file "/dev/dec10". This entry is the seventeenth line (or sixteenth, counting from zero).

2. Special Files "/dev/dec10" and "/dev/tty3"

The special file "/dev/dec10" is created by the super-user, using the "mknod" command. The arguments used with this command, for this creation, are "c" (for character special file), "16" (for line number in "cdevsw") and "3"

(for device id number). The last number is the same as the id number of the special file "/dev/tty3", which "/dev/dec10" maps.

The special file "/dev/tty3" is nullified by altering its entry in the file "/etc/ttys". The entry is "13-", and is changed to "03-", denoting that it is not an active teletype.

3. Batch Job Submission Interface

This package consists of five programs--"addusr", "delusr", "creator", "submit" and "proc.dec10". The directory "/tmp/dec" is created, with no access rights to anyone but the super-user. This directory contains all the files that are used by these programs. Two files, "passwd" and "queue", which reside in this directory, are critical to the functioning of the programs.

The programs "addusr", "delusr", "submit" and "proc.dec10" access "passwd", while "submit" and "proc.dec10" also access "queue". Each program attempts to ensure exclusive control of the files by detecting the absence of the file "semaphore", also in the directory "/tmp/dec". A brief description of these programs is given below.

Program "addusr"

This program is executed to add a user to the list of users of the interface with the DECsystem-10. The program

interactively obtains all the entry information--an entry id, the DEC-10 ppn and the DEC-10 password. This information is stored in the file "passwd".

Program "delusr"

This program is executed to remove a user from the list of users. It removes the entry specified by the entry id (which it interactively obtains), as long as the user is authorized. Permission is denied if the user deleting did not create it.

Program "creator"

This is a program that aids the user in creating a file of commands to be submitted to the batch interface. It obtains the name of the file to be created, and formats it to suit the requirements of "proc.dec10", which carries out the actual processing. It produces an asterisk as a prompt. A help listing is produced if "help" is typed.

The commands used in creating a file are of three categories--directly transmissible commands ("dir" and "exec"), interpretive commands ("get" and "trans"), and special format commands (user supplied DEC-10 command lines).

When the command is entered, the program writes a line into the command file. The first character on the line is a control character that identifies the category of the line.

The control characters used are Octal 030, 031 and 032, for special, direct and interpretive commands, respectively. In the case of special format commands, the first line of the group contains only the control character. The group of commands (multiple lines) themselves do not contain any control characters. The group is delimited by the next line that contains a control character, or the end of file.

Program "submit"

This program is invoked with two arguments--the name of the file to be submitted (produced by "creator"), and the entry id of the user's entry in "passwd" (as supplied to "addusr"), in this order. It appends a request entry to the file "queue".

Program "proc.dec10"

This does the actual processing of the communication and batch requests. It performs a login to the user's DECsystem-10 ppn and processes the commands in the submitted file, line by line. It produces a log file ("dec10.log") in the submitting user's directory. The log file contains completion messages for each command processed.

```
/*
 * A clist structure is the head
 * of a linked list queue of characters.
 * The characters are stored in 4-word
 * blocks containing a link and 6 characters.
 * The routines getc and putc (m45.s or m40.s)
 * manipulate these structures.
 */
struct clist
{
        int     c_cc;           /* character count */
        int     c_cf;           /* pointer to first block */
        int     c_cl;           /* pointer to last block */
};

/*
 * A tty structure is needed for
 * each UNIX character device that
 * is used for normal terminal IO.
 * The routines in tty.c handle the
 * common code associated with
 * these structures.
 * The definition and device dependent
 * code is in each driver. (kl.c dc.c dh.c)
 */
struct tty
{
        struct  clist t_rawq;   /* input chars right off device */
        struct  clist t_canq;   /* input chars after erase and kill */
        struct  clist t_outq;   /* output list to device */
        int     t_flags;        /* mode, settable by stty call */
        int     *t_addr;        /* device address (register or startup fcn) */
        char    t_delct;        /* number of delimiters in raw q */
        char    t_col;          /* printing column of device */
        char    t_erase;        /* erase character */
        char    t_kill;         /* kill character */
        char    t_state;        /* internal state, not visible externally */
        char    t_char;         /* character temporary */
        int     t_speeds;       /* output+input line speed */
        int     t_dev;          /* device name */
};

char partab[];                  /* ASCII table: parity, character class */
int  ctrlqsent;                 /* flag for ctrl-q having been sent.     */

#define TTIPRI  10
#define TTOPRI  20

#define CERASE  '#'             /* default special characters */
#define CEOT    004
#define CKILL   '@'
#define CQUIT   034             /* FS, cntl shift L */
#define CINTR   0177            /* DEL */
#define CBUFFUL 023             /* ctrl s - DEC-10 protocol */
#define CBUFRDY 021             /* ctrl q - DEC-10 protocol */
```

```
/* limits */
#define TTHIWAT 50
#define TTLOWAT 30
#define TTYHOG  256
#define TTCTLS  240             /* buffer length cut-off - DEC-10 ctrl-s */
#define TTCTLQ  5               /* buffer low mark -- DEC-10 ctrl-q */

/* modes */
#define HUPCL   01
#define XTABS   02
#define LCASE   04
#define ECHO    010
#define CRMOD   020
#define RAW     040
#define ODDP    0100
#define EVENP   0200
#define NLDELAY 001400
#define TBDELAY 006000
#define CRDELAY 030000
#define VTDELAY 040000
#define DECTTY  0100000                 /* flag to define DEC-10 line */

/* Hardware bits */
#define DONE    0200
#define IENABLE 0100

/* Internal state bits */
#define TIMEOUT 01              /* Delay timeout in progress */
#define WOPEN   02              /* Waiting for open to complete */
#define ISOPEN  04              /* Device is open */
#define SSTART  010             /* Has special start routine at addr */
#define CARR_ON 020             /* Software copy of carrier-present */
#define BUSY    040             /* Output in progress */
#define ASLEEP  0100            /* Wakeup when output done */
```

```
/*
 * tunable variables
 */

#define NBUF    15              /* size of buffer cache */
#define NINODE  100             /* number of in core inodes */
#define NFILE   100             /* number of in core file structures */
#define NMOUNT  5               /* number of mountable file systems */
#define NEXEC   3               /* number of simultaneous exec's */
#define MAXMEM  (64*32)         /* max core per process - first # is KW */
#define SSIZE   20              /* initial stack size (*64 bytes) */
#define SINCR   20              /* increment of stack (*64 bytes) */
#define NOFILE  15              /* max open files per process */
#define CANBSIZ 256             /* max size of typewriter line */
#define CMAPSIZ 100             /* size of core allocation area */
#define SMAPSIZ 100             /* size of swap allocation area */
#define NCALL   20              /* max simultaneous time callouts */
#define NPROC   50              /* max number of processes */
#define NTEXT   40              /* max number of pure texts */
#define NCLIST  100             /* max total clist size */
#define HZ      60              /* Ticks/second of the clock */

/*
 * priorities
 * probably should not be
 * altered too much
 */

#define PSWP    -100
#define PINOD   -90
#define PRIBIO  -50
#define PPIPE   1
#define PWAIT   40
#define PSLEP   90
#define PUSER   100

/*
 * signals
 * dont change
 */

#define NSIG    20
#define         SIGHUP  1       /* hangup */
#define         SIGINT  2       /* interrupt (rubout) */
#define         SIGQIT  3       /* quit (FS) */
#define         SIGINS  4       /* illegal instruction */
#define         SIGTRC  5       /* trace or breakpoint */
#define         SIGIOT  6       /* iot */
#define         SIGEMT  7       /* emt */
#define         SIGFPT  8       /* floating exception */
#define         SIGKIL  9       /* kill */
#define         SIGBUS  10      /* bus error */
#define         SIGSEG  11      /* segmentation violation */
#define         SIGSYS  12      /* sys */
#define         SIGPIPE 13      /* end of pipe */
#define         SIGFUL  14      /* DEC-10 buffer full - mxs */
```

```
#define         SIGEMPT 15      /* DEC-10 buffer ready - mxs */

/*
 * fundamental constants
 * cannot be changed
 */

#define USIZE   16              /* size of user block (*64) */
#define NULL    0
#define NODEV   (-1)
#define ROOTINO 1               /* i number of all roots */
#define DIRSIZ  14              /* max characters per directory */

/*
 * structure to access an
 * integer in bytes
 */
struct
{
        char    lobyte;
        char    hibyte;
};

/*
 * structure to access an integer
 */
struct
{
        int     integ;
};

/*
 * Certain processor registers
 */
#define PS      0177776
#define KL      0177560
#define SW      0177570
```

```
/*
 * One structure allocated per active
 * process. It contains all data needed
 * about the process while the
 * process may be swapped out.
 * Other per process data (user.h)
 * is swapped with the process.
 */
struct  proc
{
        char    p_stat;
        char    p_flag;
        char    p_pri;          /* priority, negative is high */
        char    p_sig;          /* signal number sent to this process */
        char    p_uid;          /* user id, used to direct tty signals */
        char    p_time;         /* resident time for scheduling */
        char    p_cpu;          /* cpu usage for scheduling */
        char    p_nice;         /* nice for scheduling */
        int     p_ttyp;         /* controlling tty */
        int     p_pid;          /* unique process id */
        int     p_ppid;         /* process id of parent */
        int     p_addr;         /* address of swappable image */
        int     p_size;         /* size of swappable image (*64 bytes) */
        int     p_wchan;        /* event process is awaiting */
        int     *p_textp;       /* pointer to text structure */
} proc[NPROC];

        /*      mxs patch: two variables containing the process ids
         *      of processes that request a signal when a cntrl s or
         *      cntrl q comes from the DEC-10. Later used by signal()
         *      to send signal.
         */

int ctlspid, ctlqpid;

        /*      end patch.
         */

/* stat codes */
#define SSLEEP  1               /* sleeping on high priority */
#define SWAIT   2               /* sleeping on low priority */
#define SRUN    3               /* running */
#define SIDL    4               /* intermediate state in process creation */
#define SZOMB   5               /* intermediate state in process termination */
#define SSTOP   6               /* process being traced */

/* flag codes */
#define SLOAD   01              /* in core */
#define SSYS    02              /* scheduling process */
#define SLOCK   04              /* process cannot be swapped */
#define SSWAP   010             /* process is being swapped out */
#define STRC    020             /* process is being traced */
#define SWTED   040             /* another tracing flag */
```

```
#
/*
 */

/*
 *   KL/DL-11 driver
 */
#include "../param.h"
#include "../conf.h"
#include "../user.h"
#include "../tty.h"
#include "../proc.h"

/* base address */
#define KLADDR  0177560 /* console */
#define KLBASE  0176500 /* kl and dl11-a */
#define DLBASE  0175610 /* dl-e */
#define NKL11   3
#define NDL11   1
#define DSRDY   02
#define RDRENB  01

struct  tty kl11[NKL11+NDL11];

struct klregs {
        int klrcsr;
        int klrbuf;
        int kltcsr;
        int kltbuf;
};

klopen(dev, flag)
{
        register char *addr;
        register struct tty *tp;

        if(dev.d_minor >= NKL11+NDL11) {
                u.u_error = ENXIO;
                return;
        }
        tp = &kl11[dev.d_minor];
        if (u.u_procp->p_ttyp == 0) {
                u.u_procp->p_ttyp = tp;
                tp->t_dev = dev;
        }
        /*
         * set up minor 0 to address KLADDR
         * set up minor 1 thru NKL11-1 to address from KLBASE
         * set up minor NKL11 on to address from DLBASE
         */
        addr = KLADDR + 8*dev.d_minor;
        if(dev.d_minor)
                addr =+ KLBASE-KLADDR-8;
        if(dev.d_minor >= NKL11)
                addr =+ DLBASE-KLBASE-8*NKL11+8;
        tp->t_addr = addr;
```

```
        if ((tp->t_state&ISOPEN) == 0) {
                tp->t_state = ISOPEN|CARR_ON;
                tp->t_flags = XTABS|LCASE|ECHO|CRMOD;
                tp->t_erase = CERASE;
                tp->t_kill = CKILL;
        }
        addr->klrcsr =| IENABLE|DSRDY|RDRENB;
        addr->kltcsr =| IENABLE;
}

decopen(dev, flag)
{
        register char *addr;
        register struct tty *tp;

        if(dev.d_minor >= NKL11+NDL11) {
                u.u_error = ENXIO;
                return;
        }
        tp = &kl11[dev.d_minor];
        if (u.u_procp->p_ttyp == 0) {
                u.u_procp->p_ttyp = tp;
                tp->t_dev = dev;
        }
        /*
         * set up minor 0 to address KLADDR
         * set up minor 1 thru NKL11-1 to address from KLBASE
         * set up minor NKL11 on to address from DLBASE
         */
        addr = KLADDR + 8*dev.d_minor;
        if(dev.d_minor)
                addr =+ KLBASE-KLADDR-8;
        if(dev.d_minor >= NKL11)
                addr =+ DLBASE-KLBASE-8*NKL11+8;
        tp->t_addr = addr;
        if ((tp->t_state&ISOPEN) == 0) {
                tp->t_state = ISOPEN|CARR_ON;
                tp->t_flags = RAW|DECTTY;
                tp->t_erase = CERASE;
                tp->t_kill = CKILL;
        }
        addr->klrcsr =| IENABLE|DSRDY|RDRENB;
        addr->kltcsr =| IENABLE;
}

klclose(dev)
{
        register struct tty *tp;

        tp = &kl11[dev.d_minor];
        wflushtty(tp);
        tp->t_state = 0;
}

klread(dev)
{
```

```
        ttread(&kl11[dev.d_minor]);
}

klwrite(dev)
{
        ttwrite(&kl11[dev.d_minor]);
}

klxint(dev)
{
        register struct tty *tp;

        tp = &kl11[dev.d_minor];
        ttstart(tp);
        if (tp->t_outq.c_cc == 0 || tp->t_outq.c_cc == TTLOWAT)
                wakeup(&tp->t_outq);
}

klrint(dev)
{
        register int c, *addr;
        register struct tty *tp;

        tp = &kl11[dev.d_minor];
        addr = tp->t_addr;
        c = addr->klrbuf;
        addr->klrcsr =| RDRENB;
        if ((c&0177)==0)
                addr->kltbuf = c;       /* hardware botch */
        ttyinput(c, tp);
}

klsgtty(dev, v)
int *v;
{
        register struct tty *tp;

        tp = &kl11[dev.d_minor];
        ttygtty(tp, v);
}
```

```
#
/*
 */

/*
 * general TTY subroutines
 */
#include "../param.h"
#include "../systm.h"
#include "../user.h"
#include "../tty.h"
#include "../proc.h"
#include "../inode.h"
#include "../file.h"
#include "../reg.h"
#include "../conf.h"

/*
 * Input mapping table-- if an entry is non-zero, when the
 * corresponding character is typed preceded by "\" the escape
 * sequence is replaced by the table value.  Mostly used for
 * upper-case only terminals.
 */
char    maptab[]
{
        000,000,000,000,004,000,000,000,
        000,000,000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        000,'|',000,'#',000,000,000,'`',
        '{','}',000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        '@',000,000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        000,000,000,000,000,000,'~',000,
        000,'A','B','C','D','E','F','G',
        'H','I','J','K','L','M','N','O',
        'P','Q','R','S','T','U','V','W',
        'X','Y','Z',000,000,000,000,000,
};

/*
 * The actual structure of a clist block manipulated by
 * getc and putc (mch.s)
 */
struct cblock {
        struct cblock *c_next;
        char info[6];
};

/* The character lists-- space for 6*NCLIST characters */
struct cblock cfree[NCLIST];
/* List head for unused character blocks. */
struct cblock *cfreelist;
```

```
/*
 * structure of device registers for KL, DL, and DC
 * interfaces-- more particularly, those for which the
 * SSTART bit is off and can be treated by general routines
 * (that is, not DH).
 */
struct {
        int ttrcsr;
        int ttrbuf;
        int tttcsr;
        int tttbuf;
};

/*
 * The routine implementing the gtty system call.
 * Just call lower level routine and pass back values.
 */
gtty()
{
        int v[3];
        register *up, *vp;

        vp = v;
        gtty(vp);
        if (u.u_error)
                return;
        up = u.u_arg[0];
        suword(up, *vp++);
        suword(++up, *vp++);
        suword(++up, *vp++);
}

/*
 * The routine implementing the stty system call.
 * Read in values and call lower level.
 */
stty()
{
        register int *up;

        up = u.u_arg[0];
        u.u_arg[0] = fuword(up);
        u.u_arg[1] = fuword(++up);
        u.u_arg[2] = fuword(++up);
        stty(0);
}

/*
 * Stuff common to stty and gtty.
 * Check legality and switch out to individual
 * device routine.
 * v  is 0 for stty; the parameters are taken from u.u_arg[].
 * c  is non-zero for gtty and is the place in which the device
 * routines place their information.
 */
```

```
sgtty(v)
int *v;
{
        register struct file *fp;
        register struct inode *ip;

        if ((fp = getf(u.u_ar0[R0])) == NULL)
                return;
        ip = fp->f_inode;
        if ((ip->i_mode&IFMT) != IFCHR) {
                u.u_error = ENOTTY;
                return;
        }
        (*cdevsw[ip->i_addr[0].d_major].d_sgtty)(ip->i_addr[0], v);
}

/*
 * Wait for output to drain, then flush input waiting.
 */
wflushtty(atp)
struct tty *atp;
{
        register struct tty *tp;

        tp = atp;
        spl5();
        while (tp->t_outq.c_cc) {
                tp->t_state =| ASLEEP;
                sleep(&tp->t_outq, TTOPRI);
        }
        flushtty(tp);
        spl0();
}

/*
 * Initialize clist by freeing all character blocks, then count
 * number of character devices. (Once-only routine)
 */
cinit()
{
        register int ccp;
        register struct cblock *cp;
        register struct cdevsw *cdp;

        ccp = cfree;
        for (cp=(ccp+07)&~07; cp <= &cfree[NCLIST-1]; cp++) {
                cp->c_next = cfreelist;
                cfreelist = cp;
        }
        ccp = 0;
        for(cdp = cdevsw; cdp->d_open; cdp++)
                ccp++;
        nchrdev = ccp;
}

/*
```

```
 * flush all TTY queues
 */
flushtty(atp)
struct tty *atp;
{
        register struct tty *tp;
        register int sps;

        tp = atp;
        while (getc(&tp->t_canq) >= 0);
        while (getc(&tp->t_outq) >= 0);
        wakeup(&tp->t_rawq);
        wakeup(&tp->t_outq);
        sps = PS->integ;
        spl5();
        while (getc(&tp->t_rawq) >= 0);
        tp->t_delct = 0;
        PS->integ = sps;
}

/*
 * transfer raw input list to canonical list,
 * doing erase-kill processing and handling escapes.
 * It waits until a full line has been typed in cooked mode,
 * or until any character has been typed in raw mode.
 */
canon(atp)
struct tty *atp;
{
        register char *bp;
        char *bp1;
        register struct tty *tp;
        register int c;

        tp = atp;
        spl5();
        while (tp->t_delct==0) {
                if ((tp->t_state&CARR_ON)==0)
                        return(0);
                sleep(&tp->t_rawq, TTIPRI);
        }
        spl0();
loop:
        bp = &canonb[2];
        while ((c=getc(&tp->t_rawq)) >= 0) {
                if (c==0377) {
                        tp->t_delct--;
                        break;
                }
                if ((tp->t_flags&RAW)==0) {
                        if(bp[-1]!='\\') {
                                if (c==tp->t_erase) {
                                        if (bp > &canonb[2])
                                                bp--;
                                        continue;
                                }
```

```
                                if (c==tp->t_kill)
                                        goto loop;
                                if (c==CEOT)
                                        continue;
                        } else
                        if (maptab[c] && (maptab[c]==c || (tp->t_flags&LCASE))) {
                                if (bp[-2] != '\\')
                                        c = maptab[c];
                                bp--;
                        }
                }
                *bp++ = c;
                if (bp>=canonb+CANBSIZ)
                        break;
        }
        bp1 = bp;
        bp = &canonb[2];
        c = &tp->t_cang;
        while (bp<bp1)
                putc(*bp++, c);
        return(1);
}

/*
 * Place a character on raw TTY input queue, putting in delimiters
 * and waking up top half as needed.
 * Also echo if required.
 * The arguments are the character and the appropriate
 * tty structure.
 */
ttyinput(ac, atp)
struct tty *atp;
{
        register int t_flags, c;
        register struct tty *tp;

        tp = atp;
        c = ac;
        t_flags = tp->t_flags;

        /*      MXS patch -- check if tty is a dec-10 terminal (flag
         *      set on open by decopen). If so and the input character
         *      is a ctrl-s, sent by the DEC when its buffer is full,
         *      signal the controlling routine by way of a signal.
         */

        if ((t_flags&DECTTY) && ((c &0177) == CBUFFUL || c == CBUFRDY)) {
                signal(tp, c==CBUFFUL? SIGFUL:SIGEMPT);
                return;
        }

        /*      end patch.
         */

        if ((c &= 0177) == '\r' && t_flags&CRMOD)
                c = '\n';
```

```
        if ((t_flags&RAW)==0 && (c==CQUIT || c==CINTR)) {
                signal(tp, c==CINTR? SIGINT:SIGQIT);
                flushtty(tp);
                return;
        }
        if (tp->t_rawq.c_cc>=TTYHOG) {
                flushtty(tp);
                return;
        }
        if (t_flags&LCASE && c>='A' && c<='Z')
                c += 'a'-'A';
        putc(c, &tp->t_rawq);
        /*
         *      MXS patch to send CTRL-S to DEC-10 if buffer (size set
         *      by TTCTLS (=240)) is full. CTRL-Q to be sent by ttread.
         */
        if (t_flags&DECTTY && tp->t_rawq.c_cc >= TTCTLS) {
                ctrlqsent = 0;
                ttyoutput(CBUFFUL, tp);
                ttstart(tp);
        }
        /*      end patch.      */
        if (t_flags&RAW || c=='\n' || c==004) {
                wakeup(&tp->t_rawq);
                if (putc(0377, &tp->t_rawq)==0)
                        tp->t_delct++;
        }
        if (t_flags&ECHO) {
                ttyoutput(c, tp);
                ttstart(tp);
        }
}

/*
 * Put character on TTY output queue, adding delays,
 * expanding tabs, and handling the CR/NL bit.
 * It is called both from the top half for output, and from
 * interrupt level for echoing.
 * The arguments are the character and the tty structure.
 */
ttyoutput(ac, tp)
struct tty *tp;
{
        register int c;
        register struct tty *rtp;
        register char *colp;
        int ctype;

        rtp = tp;
        c = ac&0177;
        /*
         * Ignore EOT in normal mode to avoid hanging up
         * certain terminals.
         */
        if (c==004 && (rtp->t_flags&RAW)==0)
                return;
```

```
        /*
         * Turn tabs to spaces as required
         */
        if (c=='\t' && rtp->t_flags&XTABS) {
                do
                        ttyoutput(' ', rtp);
                while (rtp->t_col&07);
                return;
        }
        /*
         * for upper-case-only terminals,
         * generate escapes.
         */
        if (rtp->t_flags&LCASE) {
                colp = "({)}!|^~'`";
                while(*colp++)
                        if(c == *colp++) {
                                ttyoutput('\\', rtp);
                                c = colp[-2];
                                break;
                        }
                if ('a'<=c && c<='z')
                        c =+ 'A' - 'a';
        }
        /*
         * turn <nl> to <cr><lf> if desired.
         */
        if (c=='\n' && rtp->t_flags&CRMOD)
                ttyoutput('\r', rtp);
        if (putc(c, &rtp->t_outq))
                return;
        /*
         * Calculate delays.
         * The numbers here represent clock ticks
         * and are not necessarily optimal for all terminals.
         * The delays are indicated by characters above 0200,
         * thus (unfortunately) restricting the transmission
         * path to 7 bits.
         */
        colp = &rtp->t_col;
        ctype = partab[c];
        c = 0;
        switch (ctype&077) {

        /* ordinary */
        case 0:
                (*colp)++;

        /* non-printing */
        case 1:
                break;

        /* backspace */
        case 2:
                if (*colp)
                        (*colp)--;
```

```
                break;

        /* newline */
        case 3:
                ctype = (rtp->t_flags >> 8) & 03;
                if(ctype == 1) { /* tty 37 */
                        if (*colp)
                                c = max((*colp>>4) + 3, 6);
                } else
                if(ctype == 2) { /* vt05 */
                        c = 6;
                }
                *colp = 0;
                break;

        /* tab */
        case 4:
                ctype = (rtp->t_flags >> 10) & 03;
                if(ctype == 1) { /* tty 37 */
                        c = 1 - (*colp | ~07);
                        if(c < 5)
                                c = 0;
                }
                *colp =| 07;
                (*colp)++;
                break;

        /* vertical motion */
        case 5:
                if(rtp->t_flags & VTDELAY) /* tty 37 */
                        c = 0177;
                break;

        /* carriage return */
        case 6:
                ctype = (rtp->t_flags >> 12) & 03;
                if(ctype == 1) { /* tn 300 */
                        c = 5;
                } else
                if(ctype == 2) { /* ti 700 */
                        c = 10;
                }
                *colp = 0;
        }
        if(c)
                putc(c|0200, &rtp->t_outq);
}

/*
 * Restart typewriter output following a delay
 * timeout.
 * The name of the routine is passed to the timeout
 * subroutine and it is called during a clock interrupt.
 */
ttrstrt(atp)
{
```

```
        register struct tty *tp;

        tp = atp;
        tp->t_state =& ~TIMEOUT;
        ttstart(tp);
}

/*
 * Start output on the typewriter. It is used from the top half
 * after some characters have been put on the output queue,
 * from the interrupt routine to transmit the next
 * character, and after a timeout has finished.
 * If the SSTART bit is off for the tty the work is done here,
 * using the protocol of the single-line interfaces (KL, DL, DC);
 * otherwise the address word of the tty structure is
 * taken to be the name of the device-dependent startup routine.
 */
ttstart(atp)
struct tty *atp;
{
        register int *addr, c;
        register struct tty *tp;
        struct { int (*func)(); };

        tp = atp;
        addr = tp->t_addr;
        if (tp->t_state&SSTART) {
                (*addr.func)(tp);
                return;
        }
        if ((addr->tttcsr&DONE)==0 || tp->t_state&TIMEOUT)
                return;
        if ((c=getc(&tp->t_outq)) >= 0) {
                if (c<=0177)
                        addr->tttbuf = c | (partab[c]&0200);
                else {
                        timeout(ttrstrt, tp, c&0177);
                        tp->t_state =| TIMEOUT;
                }
        }
}

/*
 * Called from device's read routine after it has
 * calculated the tty-structure given as argument.
 * The pc is backed up for the duration of this call.
 * In case of a caught interrupt, an RTI will re-execute.
 */
ttread(atp)
struct tty *atp;
{
        register struct tty *tp;

        tp = atp;
        if ((tp->t_state&CARR_ON)==0)
                return;
```

```
        if (tp->t_canq.c_cc || canon(tp))
                while (tp->t_canq.c_cc && passc(getc(&tp->t_canq))>=0);
        /*
         *    MXS patch to send CTRL-Q to DEC-10 if buffer has been
         *    read. CTRL-S sent by ttyinput.
         */
        if ((tp->t_flags)&DECTTY)
                if (tp->t_rawq.c_cc == TTCTLQ && ctrlqsent == 0) {
                        ctrlqsent = 1;
                        ttyoutput(CBUFRDY, tp);
                        ttstart(tp);
                }
        /*      end patch       */
}

/*
 * Called from the device's write routine after it has
 * calculated the tty-structure given as argument.
 */
ttwrite(atp)
struct tty *atp;
{
        register struct tty *tp;
        register int c;

        tp = atp;
        if ((tp->t_state&CARR_ON)==0)
                return;
        while ((c=cpass())>=0) {
                spl5();
                while (tp->t_outq.c_cc > TTHIWAT) {
                        ttstart(tp);
                        tp->t_state |= ASLEEP;
                        sleep(&tp->t_outq, TTOPRI);
                }
                spl0();
                ttyoutput(c, tp);
        }
        ttstart(tp);
}

/*
 * Common code for gtty and stty functions on typewriters.
 * If v is non-zero then stty is being done and information is
 * passed back therein;
 * if it is zero stty is being done and the input information is in the
 * u_arg array.
 */
ttystty(atp, av)
int *atp, *av;
{
        register  *tp, *v;

        tp = atp;
        if(v = av) {
                *v++ = tp->t_speeds;
```

```
                v->lobyte = tp->t_erase;
                v->hibyte = tp->t_kill;
                v[1] = tp->t_flags;
                return(1);
        }
        wflushtty(tp);
        v = u.u_arg;
        tp->t_speeds = *v++;
        tp->t_erase = v->lobyte;
        tp->t_kill = v->hibyte;
        tp->t_flags = v[1];
        return(0);
}
```

```
/*
 */

int     (*bdevsw[])()
{
        &nulldev,       &nulldev,       &rkstrategy,    &rktab, /* rk */
        &nodev,         &nodev,         &nodev,         0,      /* rp */
        &nodev,         &nodev,         &nodev,         0,      /* rf */
        &tmopen,        &tmclose,       &tmstrategy,    &tmtab, /* tm */
        &nodev,         &nodev,         &nodev,         0,      /* tc */
        &nodev,         &nodev,         &nodev,         0,      /* hs */
        &nodev,         &nodev,         &nodev,         0,      /* hp */
        &nodev,         &nodev,         &nodev,         0,      /* ht */
        &nulldev,       &nulldev,       &rxstrategy,    &rxtab, /* rx */
        0
};

int     (*cdevsw[])()
{
        &klopen,    &klclose,   &klread,    &klwrite,   &klsgtty,   /* console */
        &nodev,     &nodev,     &nodev,     &nodev,     &nodev,     /* pc */
        &nodev,     &nodev,     &nodev,     &nodev,     &nodev,     /* lp */
        &nodev,     &nodev,     &nodev,     &nodev,     &nodev,     /* dc */
        &nodev,     &nodev,     &nodev,     &nodev,     &nodev,     /* dh */
        &nodev,     &nodev,     &nodev,     &nodev,     &nodev,     /* dp */
        &nodev,     &nodev,     &nodev,     &nodev,     &nodev,     /* dj */
        &nodev,     &nodev,     &nodev,     &nodev,     &nodev,     /* dn */
        &nulldev,   &nulldev,   &mmread,    &mmwrite,   &nodev,     /* mem */
        &nulldev,   &nulldev,   &rkread,    &rkwrite,   &nodev,     /* rk */
        &nodev,     &nodev,     &nodev,     &nodev,     &nodev,     /* rf */
        &nodev,     &nodev,     &nodev,     &nodev,     &nodev,     /* rp */
        &tmopen,    &tmclose,   &tmread,    &tmwrite,   &nodev,     /* tm */
        &nodev,     &nodev,     &nodev,     &nodev,     &nodev,     /* hs */
        &nodev,     &nodev,     &nodev,     &nodev,     &nodev,     /* hp */
        &nodev,     &nodev,     &nodev,     &nodev,     &nodev,     /* ht */
        &decopen,   &klclose,   &klread,    &klwrite,   &klsgtty,   /* dec link */
        0
};

int     rootdev ={(0<<8)|0};
int     swapdev ={(0<<8)|0};
int     swplo   4000;   /* cannot be zero */
int     nswap   872;
```

APPENDIX B

INTERFACE USER'S MANUAL

1. Introduction

The DECsystem-10 interface is a batch job processing system. At any time, you may create a file under UNIX, containing a sequence of operations to be performed by the system. In creating this file, it must be noted that the operations are performed in order of entry in the file.

To be able to use this system, it is first of all necessary to add yourself to the list of system users. This is done by the program "addusr". You may delete your name from this list at any time. A command file is created by using the command file creating program, "creator". Once a command file is created, it may be submitted to the batch job processor by using the program "submit".

2. Adding Yourself to the System

The command "addusr", given at the shell level, invokes the program to do this. This program asks for an entry id. This entry id (one to five characters long, not containing a colon) has to be used in all subsequent usage of the system. If the entry id supplied already exists for another entry, a new one has to be supplied.

Next, the DECsystem-10 information--the ppn and password--has to be given, when prompted for. It will display all the entered information, with one addition, your UNIX id number. Each piece of information is separated by a colon. It requires verification of the information, before writing it out into a system maintained file.

3. Deleting your Entry

The command "delusr" invokes the deletion program. It asks for the id of the entry to be deleted. This id given should be the same as that supplied to "addusr". If you are authorized, that is, you created the entry earlier, it will remove the entry from its file.

4. Creating a command file

The program to create the command file is "creator". Note that you cannot produce a command file on your own. It asks for the name of the file to be created. It then produces an asterisk prompt.

The commands currently supported are:

1. dir - to obtain a directory listing from the DEC-10. No switches are supported.
2. exec - to execute file(s) on the DEC-10. The file name(s) should be supplied when prompted. (A two minute limit is implemented for the execution of this command by the batch processor.)

3. get - to transfer a file from the DEC-10 to UNIX. Source and destination file names are prompted for. Wild card specification is not supported.

4. trans - to transfer a file from UNIX to the DEC-10. Source and destination file names are prompted for. Wild card specification is not supported.

5. spec - to supply a set of DEC-10 commands, otherwise not implemented by this program. The commands are entered line by line, ending with a CTRL-Z. (A two minute time limit is imposed on each command.)

6. quit - to quit the current session, deleting the command file.

7. finish - to end the current session normally.

8. help - to get a list of supported commands.

You would normally end the session with an "f" (for "finish"). The concluding message specifies the name of the command file.

5. Submitting a command file

The file created can be submitted to the batch processor, by invoking the "submit" program. This command has two arguments--the name of the command file, and the

entry id, to be supplied in this order. Diagnostics are produced if it fails to queue the request.

Appendix C contains a sample usage of each of the above programs.

APPENDIX C

SAMPLE RUNS OF THE SYSTEM PROGRAMS

This appendix contains sample usages of the four user interface programs ("addusr", "delusr", "creator" and "submit"), described in Appendix B.

```
% addusr


Enter id for entry (limit 5 characters): mxs
Enter DEC-10 ppn: 76141,160030
Enter password (limit 8 characters): xyz
File entry is:
 mxs:76141:160030:xyz:206:
Confirm (y or n): y
Done...



% addusr


Enter id for entry (limit 5 characters): mxs
Duplicate entry - try again
Enter id for entry (limit 5 characters): root
Enter DEC-10 ppn: 76141,18765
Invalid format for ppn - reenter
Enter DEC-10 ppn: 76141,160030
Enter password (limit 8 characters): xyz
File entry is:
 root:76141:160030:xyz:206:
Confirm (y or n): y
Done...
```

```
% delusr


Enter DECid: xroot
Permission denied
%


% delusr


Enter DECid: mxs
Deleted entry mxs:76141:160030:xyz:206:
%


% delusr


Enter DECid: root
Deleted entry root:76141:160030:xyz:206:
%
```

```
% creator

File: dec10.commands

*help

Commands are:
        direct - directory listing
        exec   - execute *
        finish - end this session & save file
        get    - transfer file(s) from DEC to UNIX *
        help   - this text
        quit   - quit this session: file not saved
        spec   - user supplied command line(s)
                 ended with a CTRL-Z *
        trans  - transmit file to DEC *
Any of the commands marked * may be aborted by a
CTRL-U.
*dir

*exec

File(s): dska:psm1
Error in input -- reenter: dskb:psm1

*get

From: alpha,beta
To: unix.dest.file

*trans

From: unix.source
To: unix.lst

*spec

path[,,sfd1]

*
```

```
*
Syntax error
*dir

*q

Quit: are you sure? n

*f

Finish: are you sure? y

File name for submission to DEC: dec10.commands
%
```

```
% submit mxrx dec10.commands

Usage: submit command-file DEC-id
%

% submit dec10.commands mxry

nonexistent DEC-id
%

% submit dec10.commands xroot

Permission denied
%

% submit dec10.commands mxs

%
```

VITA

Mukesh Sundaram was born in New Delhi, India, on April 28, 1956. He attended St. Joseph's Boys' High School in Bangalore, India, graduating in December, 1971. The following July, he began the five-year Bachelor of Technology degree at The Indian Institute of Technology, Madras. He received his degree in Electrical Engineering (Electronics) in July 1977.

After being admitted to the Graduate School at The University of Tennessee, in September, 1977, he accepted a teaching assistantship with the Computer Science department. He was awarded the Master of Science degree in Computer Science in August, 1979. He will begin work on the Ph.D. in Computer Science at The University of Illinois.