AD729261

BOLT BERANEK AND NEWMAN INC

CONSULTING · DEVELOPMENT · RESEARCH

BBN Report No. 2180 15 August 1971

TENEX, A PAGED TIME SHARING SYSTEM FOR THE PDP-10

by

Daniel G. Bobrow, Jerry D. Burchfiel, Daniel L. Murphy and Raymond S. Tomlinson

This research was supported by the Advanced Research Projects Agency under ARPA Order No. 1967; Contract No. DAHC-71-C-0088.

53

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Bolt Beranek and Newman Inc. 50 Moulton Street Cambridge, Massachusetts 02138	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

TENEX, A PAGED TIME SHARING SYSTEM FOR THE PDP-1Ø

4. DESCRIPTIVE NOTES *(Type of report and, inclusive dates)*

Scientific

5. AUTHOR(S) *(First name, middle initial, last name)*

D.G. Bobrow D.L. Murphy
J.D. Burchfiel R.S. Tomlinson

6. REPORT DATE 15 August 1971	7a. TOTAL NO OF PAGES 47	7b. NO. OF REFS 8
8a. CONTRACT OR GRANT NO DAHC15 71 C 0088 b. PROJECT NO. ARPA ON 1967 c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) BBN Report No. 2180 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*	

10. DISTRIBUTION STATEMENT

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce for sale to the general public.

11. SUPPLEMENTARY NOTES This research was sponsored by the Advanced Research Projects Agency under ARPA Order No. 1967.	12. SPONSORING MILITARY ACTIVITY

13. ABSTRACT

TENEX is a new time sharing system implemented on a DEC PDP-10 augmented by special paging hardware developed at BBN. This report specifies a set of goals which we feel are important for any time sharing system. It then describes how the TENEX design and implementation achieves these goals. These include specifications for a powerful multiprocess large memory virtual machine, intimate terminal interaction, comprehensive uniform file and I/O capabilities, and clean flexible system structure. Although our implementation required some compromise to achieve a system operational within six months of hardware checkout, TENEX has proven to be a good interactive system with flexible multi-process facilities and reliable operation.

Unclassified

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
TENEX						
Paging						
Virtual Machines						
Time sharing system						
Scheduling algorithm						
Process structure PDP-10						

DD FORM 1 NOV 65 1473 (BACK)

S/N 0101-807-6821

Security Classification

A-31409

TENEX, A PAGED TIME SHARING SYSTEM FOR THE PDP-10*

D. G. Bobrow, J. D. Burchfiel, D. L. Murphy, and
R. S. Tomlinson

Bolt Beranek and Newman, Inc.
Cambridge, Massachusetts

- - - - - - - - - - - - - -

*This paper has been accepted by the Association for Computing Machinery and will be presented at the Third Symposium on Operating Systems Principles, October 18-20, 1971. In addition to the authors, significant contributions to the design and implementation of TENEX were made by T. R. Strollo, who led the technical staff, and J. R. Barnaby who implemented the EXEC subsystem. Others who contributed include T. Myer, E. Fiala, D. Wallace and J. Elkind.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

BLANK PAGE

1.0 INTRODUCTION

TENEX is a new time-sharing system implemented on the DEC PDP-10 processor augmented by special paging hardware developed at BBN. In this introduction we specify a set of goals which we feel are important for any time-sharing system, and which governed the design of TENEX. The following sections describe how TENEX tries to achieve these goals. The constraints on the implementation required that minimal changes be made to the PDP-10 processor, including no change to the basic address computation. In addition, the system had to be in service for users within six months of the operation of the hardware, including standard subsystems, such as FORTRAN, editors and assemblers.

The first class of user oriented goals is centered around the virtual machine seen by the user.

1. Process Memory. A user should see a memory space which is independent of the system configuration of core memory. The memory space for a particular process should allow independent read, write, and execute protection for units smaller than the total memory space. No part of a user process's memory space should be used for system functions (e.g. buffers).

2. Multiprocess interaction. A single user job should be able to have multiple processes with independent memory spaces and computational power. Facilities should be provided for several kinds of communication among such processes, including memory sharing, interrupt capabilities, and direct control.

3. Extended Instruction Set. In addition to the basic instruction repertoire of the PDP-10, the user should be provided with an extended instruction repertoire for common operations. We attempted to make monitor functions and other monitor operations general and uniform while retaining the degree of economy and efficiency appropriate for the power of the system. This balance was achieved in many cases by the specification of default arguments, simple-case calling sequences, and by the coding of special routines to handle frequent cases most efficiently.

Another set of criteria is aimed toward maximizing the ease and efficacy of terminal use.

1. Terminal Character Transmission Protocol: The system should make effective use of full duplex terminals (with system-originated echos) to improve interactive capability and control output in a consistent manner. Intimate interaction requires that the system and user be able to conduct a dialogue with each typing only a few characters

before being prompted by the other.

2. Terminal Interrupts: A number of characters should be available on the terminal which can interrupt the program for various actions.

3. User-User Communication: Facilities should exist to enable two or more users on the machine concurrently to communicate with each other via the system, and thereby cooperate on various activities.

4. Command Language Form: The user command language interpreter should provide an easily understood, mnemonic interaction facility, with common operations available as simple commands (e.g. list a file, report resources used). The language should be adaptable to the skill and experience of the user.

5. The command language should provide quick access to all of the general functions of the system (those not a part of specific subsystems), provide access to the subsystems, and always be available as the final "fall-back" in case of a program malfunction or user panic.

A third set of user oriented goals is focused on the file system.

1. Uniformity of Access: All byte stream devices should be treatable uniformly and symbolically, e.g. storage files and terminals. All file media which could provide effective user random access storage should allow it. All external devices and byte streams should be accessible through the file system, and file operations should be able to invoke general (user or system) programs to absorb or generate data.

2. Lack of Small Arbitrary Limits: No arbitrary small limit should be placed on individual file size, e.g. a file should be able to contain at least several full address spaces. File names should be large enough (with enough parts) to distinguish important classes of files, with efficient lookup so as to provide file name completion.

3. Efficiency of I/O: File operations should be implemented efficiently so that the user is not tempted to go around the file system on special devices.

4. File Protection: File access ought to be limited in a number of ways, and default protection rules should provide the user with adequate protection.

A number of system oriented goals governed the structure, design process, and implementation of the TENEX monitor.

1. Adjustable Scheduling: Scheduling should be equitable with good response time for short interactions. The scheduling procedures should be dynamically adjustable by operators so as to provide regulated services to selected jobs.

2. Measurability: Measurements of system, module, and user program performance should be available so that dynamic adjustments can be made or system changes could be noted.

3. Modularity. The system had to be built in separated modules with well defined interfaces to device dependent programs so that it could be easily adaptable to a number of different configurations in the ARPA community, and new modules could be added easily.

4. Reliability and Ease of Debugging: The system had to be built with internal redundancy checks to maintain high reliability, to make crash recovery with file integrity easy, to allow bugs to be found before having destructive consequences, and to allow intermittent hardware failures to be detected.

5. Backward Compatibility. Useful operation within six months of hardware operation required that manufacturer supplied standard subsystems should be made operational

without need for special changes.

The set of goals specified above have been met in the TENEX system, which has been operational for users since June 1970. Although none of them is entirely new, a currently operational system which meets all these specifications is a significant accomplishment.

2.0 HARDWARE DEVELOPMENT FOR TENEX

Hardware development and modification for TENEX was minimized as much as possible consonant with achieving the goals specified above. These included an address mapping (paging) box to be implemented with then current DEC modules (to speed development time), and some changes to the PDP-10 processor. A hard limit on the latter changes was imposed by the physical room available in the processor. Both projects were constrained to be upward compatible from the standard PDP-10 so that the standard DEC time sharing software and diagnostics could be run.

2.1 The BBN Pager

The BBN pager is an interface between the PDP-10 processor and the memory bus. It provides individual mapping (relocation) of each page (512 words) of both user and monitor address spaces using separate maps for each. The pager uses "associative registers" and core memory tables to store the mapping information. On each memory request from the processor, the 9 high-order bits of the address, and the request-type level (read, write, execute) are compared in parallel with the contents of each associative register. If a match is found, the register containing the match also contains eleven high-order address bits to reference up to one million words of physical core.

If no match is found, reference is made to a 512 word "page table" in physical core memory. The word selected in this page table is determined by a dispatch based on the original 9 high-order address bits. In the simple case of a private page which is in core, the 11 high-order address bits and protection bits are found in this word and are automatically loaded into an associative register by the pager.

There are three other cases:

A. The page is not in core, is protected from the requested type of access, or is non-existent; in this case a page fault (trap) will occur.

B. The page is shared; in this case the map contains a "shared" pointer to a system table which contains the location information for the page.

C. The page belongs to another process; in this case, the entry contains an "indirect" pointer to an entry in another page table from which the location information is obtained.

The goal of program (code and data) sharing was given extensive consideration in the design of the BBN Pager. The indirect and shared pointer mechanism allows pages to be actively shared (exist in more than one memory space) but still have the current address (core or secondary storage) stored in only one place. This means the memory management software may move pages between core and secondary storage by changing only one address thereby avoiding the overhead and programming difficulties which would result from finding and changing many copies of the same address. The pager permits individual pages to be shared for write as well as read references, so two or more processes may communicate effectively and efficiently by sharing a common page into which any or all may write.

An independent per-page status bit is available to users which will produce a trap on a write or read-modify-write reference. This bit marks a shared page which _may_ be changed by a running program and for which a private copy should then be created. This permits shared programs to be prepared with pre-constructed data areas which will be kept shared if not modified, and put in private storage _if changed_. This capability also permits some sharing of programs which are not completely pure (usually older programs), since an overlooked impure reference will cause no malfunction, only a loss of storage efficiency.

One final unique feature is that the pager maintains a record of the activity of the pages in core memory in a "core status" table. The pager notes when a page has been referenced, which processes have used that page, and whether the page has been written into. This information is particularly useful for the evaluation of working set states and sizes by the memory management software.

2.2 Processor Modifications

The hardware modifications to the PDP-10 processor added a new system call instruction, JSYS, and two ways of accessing a calling memory context (user or monitor) from the monitor. Of the latter, the first is an execute instruction which allows current or previous context to be specified for each memory reference of the object instruction, and the second is a group of move instructions in which the references are done in the previous context. The previous context can be either user or monitor, and a bit in the state word of the processor is set accordingly at each monitor call. The pager also contains an 'AC base register' which specifies the location of the stored AC's (general registers) of the previous context.

Except for the pager trapping facilities, all of the TENEX virtual machine facilities (monitor calls) are reached via the JSYS instruction. It provides a new and independent transfer mechanism into the monitor which does not conflict with "UUO" system calls used by DEC software. The JSYS instruction meets a number of desirable objectives for a monitor call instruction. It accomplishes a transfer from the user program to the specified monitor routine in one instruction time. Further, it stores the state of the processor, including the return PC, in a location specified by the monitor so that it is suitable for reentrant code. The transfer vector occupies exactly one page in the monitor space and could be mapped independently for each process, but this is not needed in the current system.

3.0 THE TENEX VIRTUAL MACHINE

A user process running under TENEX operates on a virtual machine which looks something like a PDP-10 arithmetic processor with 256K of attached memory. This _virtual memory_ is a creation of the paging hardware and swapping software which traps processor references to any data not in core, and performs the necessary I/O operations to second- or third-level storage to make the referenced page available. Such traps are invisible to the user process.

The virtual processor does not make available to the user the direct I/O instructions of the PDP-10, but through instructions which call monitor routines, the virtual machine provides facilities that are considerably more powerful and sophisticated than typical hardware configurations used directly.

3.1 Virtual Memory Structure

The TENEX virtual memory may be viewed as a single block of 256K words, and programs may use it in this fashion. However, the existence of the paging hardware means that the monitor must deal with memory in pages of 512 words, and some of the power which the mapping hardware provides is accessible to a user program.

The contents of the virtual memory at any time are specified by the virtual memory map of 512 slots, which the user may read or write. The contents of each slot specify the page in that position in the virtual address space, and the type of access allowable (read and/or write and/or execute) for that page. A page contents may be unique, a particular page may be specified in more than one slot of a single process, or in several processes, or it may appear in a combination of processes and files.

In the simplest case, a map slot may contain (a pointer to) a private page. By the term private, we mean shared with no other processes in the system. A private page is automatically created whenever a process makes a reference to a page and the map word for that page is empty.

A slot may also contain an indirect pointer to a page in this or some other process. A memory reference to a location in such a page will be executed just as though the instruction had directly addressed the page pointed to. Any change made to the page by either process will be seen by both processes. If the owner of the page changes the contents of his memory map, the the process with the indirect pointer will see the change. A virtual memory slot may also contain a pointer to a page from a file in the file system, as discussed later.

3.2 Job Structure

A job is a set of one or more hierarchically related processes, and it has the following attributes.

1. The name of user who initiated the job
2. An account number to charge costs associated with use of system resources.
3. Some open files.
4. A hierarchy of running and/or suspended processes.

A job may also have one or more terminal or other devices assigned and attached.

3.2.1 Process Hierarchy

TENEX permits each job to have multiple simultaneously runnable processes. The relationship among them is defined by a structure which looks like an inverted tree. A process always has one superior process and may have one or more inferior processes. Two processes are said to be parallel if they have the same superior.

Although not completely general, a tree structure process heirarchy implicitly provides the protection and reference facilities that are wanted in most applications. These include referencing inferior forks as a class for freezing, killing, and resuming, fielding of interrupts and special conditions by a superior fork, and protection of the superior fork from inferiors.

In TENEX, a process may create processes inferior, but <u>not</u> parallel or superior in the structure. A fork can communicate with other members of the structure by (a) sharing memory (b) direct control (superior to inferior only), or (c) pseudo (software simulated) interrupts as described in 3.3.

Some examples of current use of multiple processes in TENEX include:

1. The EXECUTIVE (command interpreter program) is a user program which resides in the top fork of every job. To RUN another program, the EXEC creates an inferior fork, places the other program in it, and causes execution to begin. The program cannot reference or change the EXEC so it is perfectly protected. If the program malfunctions, control returns to the EXEC which prints a diagnostic message for the user.

2. A program wishes to wait for terminal input but

only for a specified length of time, after which it will prompt the user. An inferior fork is started which invokes a monitor call to wait for the specified time, and the main process waits for input. If the input arrives first, the inferior fork is disabled and the program continues. If the time elapses, the inferior wakes up and signals (see PSI below) the main process. By using this technique, a program can wait for any one of an arbitrary set of events.

3. Invisible DDT (debugging program). The debugging program and symbols reside in a fork with the program under test in an inferior fork. This allows the debugger to detect and interpret various types of malfunctions such as illegal instruction execution. It also protects the debugger from erroneous references by the program.

3.3 Pseudo-Interrupt

TENEX provides a facility for a process to receive asynchronous signals from other processes, from terminals, or as the result of its own execution. The various processes in a job may explicitly direct interrupts to each other for purposes of communication. A process may enable an interrupt which will occur whenever the user hits a particular key on the controlling terminal. Finally, a process may use the pseudo-interrupt system to detect any of a set of unusual conditions, including illegal references to memory, processor overflow conditions, end-of-file and data errors.

The fork and pseudo-interrupt features of TENEX have been found to be a general purpose capability suitable for solving a number of diverse problems which would otherwise require a less general facility to be added to the system.

3.4 Other Monitor Functions

Other functions which form a part of the virtual machine include:

a) Functions which provide information to the program about the state of the system or job. (Time of day; runtime used, name of user, etc.)

b) Functions which save and restore the environment of a fork.

c) Functions which provide frequently needed forms of I/O conversions such as fixed or floating point number input and output, and date and time to string conversions.

3.5 Backward Compatibility (DEC 10/50 monitors)

Since TENEX was being implemented on a machine for which a large useful program library existed, mostly for use under the DEC 10/50 time sharing monitor, we felt it was highly desirable to be able to run such programs under the new monitor system. We felt it should be possible to run binary images of old programs, i.e. without reassembling.

Toward this end the following steps were taken. First, none of the instructions which were used by the 10/50 monitor for user-to-monitor communication were used by TENEX. All of the TENEX monitor calls were implemented with the JSYS instruction.

Secondly, routines were designed which implemented all of the existing 10/50 monitor calls in terms of the available TENEX monitor calls. This set of routines implements all of the functions available in the 10/50 monitor except those specifically intended for the maintenance of the system. Assembled together as the compatability package, they occupy slightly less than 2.5K of core. The package is kept as a core image file and is never seen by programs which use only TENEX monitor calls. However, the functions are automatically made available to 10/50 type programs by the monitor. When a program makes its first 10/50 type monitor call, the TENEX monitor maps the compatability package into a remote portion of the process address space. Subsequent 10/50 type monitor calls cause a transfer to the compatability package which then interprets the call.

The compatibility routines are placed in the user space for several reasons: a) regular use can be made of the pseudo interrupt system, b) the compability package (which requires constant maintenance) can be maintained as a separate module, totally independent from the monitor, and c) the monitor is protected from malfunction by the compatiblity routines.

4.0 USER INTERACTION WITH TENEX

Users at terminals communicate and work with TENEX primarily through a command language interpreter called the TENEX Executive, or EXEC. The EXEC is an interactive, well human engineered program which can accept commands from a user's teletype or from a file. It is implemented as a reentrant, shared program which runs in user mode, usually as the top level process in the structure.

The EXEC provides the user with a multitude of facilities which are activated by simple, easy-to-learn commands. These facilities allow access to the system (e.g. LOGIN); utility operations on files and file directories; initiation of private programs and subsystems; limited debugging aids; initiation of batch (detached operations); printout of user information and system statistics; and system maintenance.

4.1 Terminal Interaction Capabilities

The TENEX EXEC is primarily intended to be used with a full duplex terminal, and when so used, its interactions are typically on the order of one or a few characters. However, a number of modes of echo and wakeup are available to the programmer which allow use of half duplex and/or line-at-a-time processing. Upper/lower case terminals are not required, but may be used to no disadvantage.

4.2 Human Engineering

The EXEC was designed with two primary objectives--ease of learning and ease of use. To ease the learning process, all commands are English words which are descriptive of the facility being activated. (e.g. COPY to copy information from one file to another, STATISTICS to obtain a listing of current system statistics). In order to help novice users, two special assistance features were incorporated. First, when the EXEC requires input from the user during a command interaction, (for instance, to collect arguments of that command) a cue is typed to indicate to the user what is expected. For example, an interaction which renames a file might be:

```
@RENAME$ (EXISTING FILE) ALPHA$ (TO BE) BETA
```

The user's input has been underlined. The $ indicates a typed ESC (ASCII escape, code 33(8)) which invokes the EXEC's verbose cueing responses in parentheses. If the novice user still doesn't understand what is expected in his response, he may type the character '?' at any time. This causes the EXEC to type out a list of all options available to the user at that point, then request a response.

For example,

```
     @AVAIL$ABLE ? ONE OF THE FOLLOWING:
LINES
DEVICES

@AV$AILABLE D$EVICES
MTA0, MTA1, MTA2, MTA3, DTA0, DTA3, PTR, PTP
```

In the above example, the user typed ESC after 'AVAIL'. This invoked command completion by the EXEC, a feature which makes the language particularly easy to use. An ESC after any initial substring of a command or argument (such as a file name) invokes completion. If the substring is insufficient for unique identification of the intended input, the EXEC rings the teletype's bell and awaits additional characters. If the initial substring cannot be recognized the EXEC types '?' to ask the user to retype that input.

The EXEC also provides editing characters to permit the user to correct typing errors in his input. These editing characters permit the user to delete the last character of his typed input, the last word, or all of it. He can also ask for his edited input to be retyped for clarity.

All these features contribute to making the TENEX executive very easy to learn and use.

4.3 General Form of Commands

Each command begins with a keyword. Depending on the command, the initial keyword may be followed by arguments such as file names, numbers, and additional keywords, and/or "noise words" to make the command more readable. The noise words are enclosed in parentheses to distinguish them from the arguments. The initial keyword usually identifies the command function. Some commands include optional arguments or argument lists of indefinite length. A few commands, such as that for file directory listing, take optional "sub-commands", each with arguments, to specify options.

Any initial word not recognizd as a command keyword is taken as the name of a subsystem to be started.

4.3.1 Command Input

Three general styles of input may be used. The styles are distinguished by syntactic analysis and by input terminators; hence they do not require different input modes and thus may be intermixed freely within a session or even with a statement.

1. Complete Input. A complete command may be typed in, with all keywords and noise words given in their entirety, and without use of any non-printing characters. This style is good for novices who are copying a typescript, command file, and terminals without the full ASCII character set (e.g. ESC).

2. Abbreviations. The user may shorten a command in two ways: he can omit noise words completely, and he can shorten keywords. Any keyword may be abbreviated with any initial substring (terminated with space) long enough to distinguish it from the other keywords acceptable in that context. Keywords have been made unique in three characters or less insofar as possible without producing very non- english-like words.

3. Completion. The user types the same characters as in abbreviated input, except he terminates each field (keyword or argument) with the ESC key. This produces a print-out of the complete command--each ESC causes the rest of the field (if an abbreviated keyword or file name) and any following noise words (with enclosing parentheses) to be printed.

Most commands are confirmed with a carriage return, but some which only print information are executed as soon as they are recognized.

4.4 Interrupt Characters

ASCII Control-C is the EXEC's attention character. When typed by the user, it causes any running program to be stopped and control to be given to the EXEC via the pseudo interrupt system. The user may then continue his program or take any other action.

Another terminal interrupt character, control-T is serviced by the EXEC. It interrupts a user's EXEC process to type out total CPU and console time used, and status of the fork being run under the EXEC.

5.0 THE TENEX FILE SYSTEM

The TENEX file system provides a general mechanism for obtaining information from and sending data to external devices attached to the TENEX system. Write only and read only devices are included in the file system so that all TENEX I/O may be handled uniformly. The first major function of the TENEX file system is to provide symbolic file name management. This includes two separate but related activities. The first involves translation of a symbolic name into an internal pointer associated with that name, which we call a file descriptor block. Second, it involves checking information concerned with 1) if the file exists, and if so information related to the file as found in the descriptor block; and 2) the process requesting access to the file. This information is used to determine if this process should be allowed to know about the existence of this file, and if so, what accesses are allowable. This activity is known as File Access Protection.

A symbolic name for TENEX files consists of up to five fields and thus conceptually represents a tree of maximum depth five. Not all nodes of this tree go down to maximum depth. This scheme was chosen rather than a full tree to simplify the problem of compatibility with existing DEC

PDP-10 software and name lookup and recognition. We are currently considering the feasibility of implementing a full tree directory structure. At each level there would be set of information which is related to access rights, and media dependence of the data access for this node. Each node represents a collection of related information, with the terminal nodes being files.

The fundamental unit of storage in a TENEX file is a byte which may be from 1 to 36 bits in length. A stream of bytes constitutes a file which is the basic named element in thd file system. Programs may reference files byte by byte in a sequential manner or, if the device permits, at random. Files may also be referenced by byte strings. No structure other than bytes and files is imposed on the user; and byte and string input and output are the basic operations. Of course, additional structure and other operations may be implemented by the user programs.

5.1 File Names

A TENEX file is named by a file descriptor composed of 5 fields some of which are omitted for certain devices. The five fields are device name, directory name, file name, extension, and version number.

The file name field is intended to designate a class of files which are related in some way. This convention is not enforceable of course but most users of TENEX tend to follow the convention since it facilitates management of a users files. The extension field is intended to designate variously processed forms of the same information. A file's extension is frequently specified by a program. For example, FFT.MAC, FFT.REL, and FFT.SAV would be used to indicate the assembly code source, relocatable file, and binary image of a single program.

The version number of a file enumerates successive versions of a file. Normally each time a file is written a new version is automatically created by making its version number be one greater than the highest existing version. This protects a user from loss if he accidentally writes on the wrong file. Excess versions may be deleted by the user or automatically by the system when they have been put on a backup storage medium.

Any of the fields of a file description may be abbreviated except for device and version.The appearance of an ESC in the file descriptor causes the portion of the field before the ESC to be looked up. In this case the system will supply the omitted character and/or fields. Abbreviation without this output is not provided in order to insure that the typescript reflects exactly what was done. The system provides default values for each field except the file name. A default value is used for a field if the user omits any input for that field, e.g. the device and directory. This simplifies references to files in most common cases.

5.2 File Access Protection

Because TENEX must service a diverse user community, it is essential that access to files be protected in a general way. Generally, access to a file depends on two things: the kind of access desired, and the relation of the program making the access to the owner of the file. Presently, a simple protection scheme is implemented in which the only possible relationships a program may bear to the file's owner are:

1. The directory attached to the job under which the program is running is the same as the owning directory.

2. The directory attached to the job under which the program is running is in the same group as the owning directory.

3. Neither 1 or 2.

Five kinds of access are distinguished for a file; directory listing; read; write, execute and append. The above three relationships and five protection types are are related by 18 bits (a 3 by 6 binary matrix) in which a one indicates that a particular access is permitted for a particular relationship. If directory listing access is not permitted, the process requesting access is given an error return which is indistinguishable from the error for nonexistent file. Other access restrictions cause errors only when an attempt is made to open a file, as described below.

For purposes of determining group access, a 36 bit word is administratively associated with each directory and each user. If the bitwise "and" of the user group word of the accessor and the directory group word of the accessee is non-zero, the group access permission is used.

Provision has been made for a more general file protection system in which more general access relationships may be expressed in a special file protection language. For example, access may be allowed only to an explicitly named set of users.

5.3 File Operations

Using a file in TENEX is basically a four step process.

First a correspondence is established between a file name and a Job File Number (JFN) which is a small index into a job table for files. Next the file is opened, establishing the mode and access permission and setting up monitor tables to permit data of the file to be accessed. Third, data is transferred to or from the file; and finally the file is closed fixing up the directory information and releasing the space occupied in system tables for the file.

For purposes of file sharing, all instances of opening a particular file reference the same data. Data written in a file will be immediately seen by readers of the file. To protect against confusion resulting from multiple uncooperating simultaneous writers and readers of a file, a file can be opened with either thawed or unthawed access. With unthawed access, a file may have either exactly one unthawed writer or any number of unthawed readers thus preventing any potentially conflicting operations. With thawed access, a file may have any number of thawed writers and/or thawed readers. Simultaneous accessors of a file must be all thawed or all unthawed.

The contents of a disk file are always accessed by mapping pages of the file into an address space. Monitor calls are provided which transfer single bytes (1-36 bits) or strings of bytes sequentially to and from files, with no user buffering required. These are simply monitor calls which reference pages of the file which have been mapped into regions of the monitor map of the process. These pages are called window pages into the file.

6.0 THE MONITOR

6.1 Scheduler

The TENEX scheduler is designed to meet a set of potentially conflicting requirements. The first and most fundamental requirement of any time sharing scheduler is to provide an equitable distribution of resources, principally CPU service, to various jobs that are competing for such resources. Secondly, because TENEX is designed to be a good interactive system, the scheduler must identify and give prompt service to jobs making interactive requests. Thirdly, because use of the CPU is intimately tied to the allocation of core memory, and because TENEX is a paged system, the scheduler must be sensitive to the changing memory and swapping requirements of the various running processes, and work with the core memory management routine to provide efficient use of core memory. Finally, the scheduler should have provision for administratively controlling the allocation of resources so as to obtain other than equal distribution.

6.1.1 Scheduling CPU Priorities

To implement the basic scheduling function, a scheduling algorithm was chosen which groups processes together on a number of separate queues each with associated runtime quantum, similar to algorithms described by Corbato (2) and BBN (1). Lower queues in general have lower priorities but longer runtimes. A common problem with many schedulers of this type is that processes are placed on the highest priority queue after any interaction. Under conditions of heavy load or with poorly behaved interactive processes, it may happen that the interactive processes succeed in using all of the available time and so lock out the compute bound processes which have fallen to the lower queues.

In TENEX, priority is based on a long term average ratio of CPU use to real time, and a process's priority after an interaction is determined by its priority before the interaction and the length of the interaction. Specifically, a process's priority is decreased while running at a constant rate, C, and increased while blocked at a rate of C/N, where N in the number of runnable processes in the system. This ensures that equitable service is given both to compute-bound and interactive jobs.

To improve response characteristics, an interactive "escape clause" is included in the scheduling algorithm. After a block wait of greater than a minimum time of 100 ms., a process is given a short quantum at maximum priority. Priority and queue position after this burst are determined by the long term average. The effect of this provision is to ensure quick service to very short interactions, even when requested immediately after a long computation.

6.1.2 Balance Set

The algorithm described above provides at any time a priority ordering of all the runnable processes in the system. However, some additional structure is imposed on the scheduler in order to optimize the use of core memory and reduce swapping overhead. For each process in the system, the core manager maintains an estimate of the size of the working set (5). The scheduler uses this information to maintain a balance set, that is, a set of processes whose working sets can co-exist in core. These processes will be chosen in order from the first N processes in priority ranking as determined above.

The scheduler periodically monitors the state of the balance set and performs the following operations. If the sum of the working sets of the processes in the balance set has increased above the maximum, then the lowest priority process is removed. Secondly, if there is a runnable process not in the balance set, it is moved into the balance set if there is room for its working set, or if it is higher in priority than the lowest priority process now in the balance set.

This scheme provides a coherent way of dealing with page faults of running processes. A running process page faults if it makes a reference to page which is not currently in core. To maximize system efficiency, the scheduler must have something else to do when this occurs. That is, it should have one or more other processes to run while the requested page is being swapped in. This is best accomplished if the scheduler has under consideration a small set of processes which can co-exist in core. At one extreme, if the scheduler considered only one process, then it would have nothing to do when that process page faulted. At the other extreme, if the scheduler considered as equivalent all runnable processes, then it would be likely that no process would have enough pages in core to run efficiently and so a great deal of thrashing(6) could result.

6.1.3 Resource Guarantees and Limitations

The overall result of the above scheduling algorithms is to provide to each of the processes demanding CPU service at least 1/N of the available service, which is our definition of an equitable distribution of resources. In some cases other distributions are desired. These might include running a demonstration which requires significant CPU time during a period of medium or heavy load, or a user who is willing to pay extra for premium service which does not degrade as the load on the machine increases. A facility is implemented in TENEX to handle these situations.

An operator or other person with appropriate administrative access can assign to any job or user of the system a fraction, F of guaranteed CPU service. For any job so designated, the scheduler will attempt to ensure that:

$$C/T > F$$

where C is the CPU seconds used by the process, and T is real time. For example, if the parameter is set to 30%, the scheduler will provide at least 18 seconds of CPU service to the specified job during each minute of real time.

This parameter acts as a ceiling as well as a floor for CPU service. That is, if there are other runnable processes on the system which are not declared special, then the scheduler will ensure that the special process receives no <u>more</u> than the stated fraction of CPU service. A process with this sort of resource guarantee will display very consistent interactive behaviour despite widely varying loads on the time sharing machine.

6.2 <u>Core Management</u>

The information provided in the core status table by the paging hardware is essential to the proper management of core memory in TENEX to avoid thrashing and other forms of inefficient operation. Paging is done on demand. No ordinary pages are preloaded before a process is run, and in general, a process will not have all the pages of its virtual memory in core at once.

The only basis for predicting the future use of a set of pages is the history of the recent use of those pages and a logical assumption (without further information from the user) is that those which have been used most recently will be used again. The paging hardware stores a 9-bit age field for each page when a reference to that page causes a pager reload. This does not happen on every reference, but does happen often enough to record any change in the age register. The software uses a 9-bit register in the pager as a logical clock so that any set of pages may be ordered according to the time at which they were referenced by comparing the 9-bit age fields. This allows the core manager to remove the oldest pages of a process when core must be reallocated.

Certain other information is also used in the core managing algorithm. When a process references a page which is not in core, a pager trap occurs and a first level core management routine is invoked. The run time since the last page fault is used for a running average of page fault times. A process is considered to have enough of its working set if the average page fault time equals PAV, a system parameter currently set to 67ms (or 2 drum revolutions). If the process is faulting more often than PAV, it is below its working set size, and a swap to bring in the requested page initiated. Control returns to the scheduler so that it can run another process until the swap is complete. If the process is faulting less often, a second core managing routine is invoked to reduce the size of the process, i.e. remove some of its pages from core if space is needed. This operation uses the age field from the core status table word for each page belonging to the process.

6.3 System Measurements

In order to observe and improve the performance of TENEX in regular service, various measuring functions were built into monitor routines. Quantative measurement is usually the only way to truly ascertain the performance of a time sharing monitor; subjective information such as reactions of users does not provide comparable results day-to-day and week-to-week.

Some values serve to indicate the efficiency of scheduling, the core/CPU balance, and the nature of the various processes running on the system. The scheduler maintains a set of integrals over time which give (as a fraction of real time):

IDLE - time when no processes are requesting CPU service

WAIT - time when all runnable processes are waiting for completion of page fault

CORE - overhead time spent in core management

TRAP - time spent handling pager traps

The various relationships among these are:

IDLE + WAIT = total time spent in scheduler

1 - IDLE - WAIT - TRAP = time spent running user processes

Also maintained by the scheduler is an integral over time of the number of processes in the balance set, the number of transfers between core and secondary storage, and number of terminal interactions.

One measure is of interest on a recurring basis to all users of the system. The scheduler maintains three exponential averages (with time constants of 1 minute, 5 minutes, and 15 minutes) of the number of runnable processes on the system. This indicates the true current load on the system better than the number of jobs logged in. Users often choose on the basis of these load figures what they do on the system at a particular time.

6.4 Debugging Aids

Certain debugging procedures used in the development of the system contributed greatly to the speed of development and integrity of the system. We felt initially that debugging facilities were important and that we should not skimp on efforts to provide them. However, the facilities that we did use did not cost significant effort.

The principal debugging aid is a program called DDT, available in several forms in the system. DDT is a program which allows memory locations to be examined and modified and breakpoints to be placed in a running program. All interactions with DDT are symbolic using the symbols defined in the source program and obtain from the assembler.

The form of DDT first used and still necessary for debugging basic level code is a stand alone version which resides in core memory along with monitor. It is used for debugging the scheduler, portions of the core manager, and other basic routines.

The second form of DDT was added as soon as the basic monitor could support demand paging and create a virtual memory. This DDT exists in the monitor map and may be used as an ordinary program at a system terminal. It is capable of examining and changing the running monitor and all of the associated tables and other contents of the monitor virtual memory. Use of this form of DDT actually allows several persons to work on debugging portions of the monitor simultaneously.

A third form of DDT is used with user programs and is cognizant of the access status (execute or write protected, etc) of pages of the user program.

One additional debugging facility is actually a coding convention. Early in the coding, an entry point to a routine was defined to handle cases that were not implemented or which were logically impossible. It is called BUGHLT and a jump there indicates a situation so anomalous as to suggest that the system can not continue to run. This routine takes two different actions depending on the setting of a switch. If the system is attended by system personnel, the the routine enters a DDT breakpoint and the state of the monitor can be examined to determine what has gone wrong. If the system is unattended, (e.g. at night), a system restart procedure is invoked.

Later in the development a second entry point was added which indicates an inconsistency which is not considered fatal to the system. This routine also stops with a breakpoint if the system is attended, but continues if the system is unattended. The occurrence of either of these calls is reported on a logging teletype in the computer room so that attention is drawn to developing problems. As the system developed, consistency checks were added to many of the most critical monitor routines, and calls made to one of these two routines when trouble is indicated. This procedure was very significant in enabling us to find obscure or infrequent bugs in the software. They also serve to prevent hardware or software failures from cascading and causing great loss of information.

7.0 CONCLUSION

TENEX was built with the knowledge of a number of other time sharing systems, including the DEC PDP-1 systems designed at BBN, the Berkeley System for the SDS-940, MIT CTSS, the DEC 10/50 System and MULTICS. We stole freely from the good design ideas of all these systems, and tried hard to avoid problems of operation and implementation we saw in these systems. We attempted to dominate all but MULTICS which had even grander goals than ours, and there we attempted to get a system operational for users much more quickly, still meeting what we considered the most important goals for a system. We conclude here with a brief summary of the implementation strategy which allowed us to get a good state of the art time-sharing system operating reliably within a very short time frame.

7.1 Design

Virtually all of the work on TENEX from initial inception to a useable system was done over a two year period. There were a total of six people principally involved in the design and implementation. Approximately the first six months were spent in discussion and thought aimed at producing a design for the paging hardware. Actual hardware technical design drawing of prints, and wirelists was begun at that time and took a total of approximately 9 months. The construction and a checkout of the pager was completed in another three months, that is approximately 18 months after the start of the project. During this latter 12 months, an increasing amount of effort was spent on software design, and this effort culminated in a series of documents which describe in considerable detail each of the important modules of the system. These documents were carefully and closely followed during the actual coding of the system, and in retrospect, it is our judgement that they contributed significantly to the overall integrity of the system.

7.2 Implementation

The actual coding of the system was begun approximately 18 months after the start of the project. The first stage of coding was completed in 6 months. At this stage, the system was operating and capable of sustaining use by non-system users for work on their individual projects. The efforts of five full-time people were involved over this six months, and were distributed as follows:

Two people worked on the monitor (including scheduler, core manager, file system, etc.). One person was involved full time on the EXEC as a user program. An average of one person was involved on a number of other projects including the 10/50 compatability routines and one person was involved almost full time in documenting the system as it grew and evolved. This consisted mainly of preparing the JSYS manual, the document which describes all of the calls that user programs can make on the monitor. The concurrent development of this documentation was necessary not only so as to have it available when users came on the system, but also to provide essential communications among the implementers of the system.

We felt it was extremely important to optimize the size of the tasks and the number of people working on the project. We felt that too many people working on a particular task or too great an overlap of people on separate tasks would result in serious inefficiency. Therefore, tasks given to each person were as large as could reasonably be handled by that person, and in so far as possible, tasks were independent of each another or related in ways that were well defined and documented. We believe that this procedure was a major factor in the demonstrated integrity of the system as well as in the speed with which it was implemented.

8.0 BIBLIOGRAPHY

(1) BBN Medical Information Technology Department: "The Hospital Computer Project Time Sharing Executive System" - BBN Report Number 1673, April 1968

(2) Corbato, F. J., et al: "An Experimental Time-sharing System" - AFIPS Conference Proceedings Vol. 21 (1962 SJCC)

(3) -- : "An Introduction and Overview of the Multics System" - AFIPS Conference Proceedings Vol. 27 (1965 FJCC)

(4) Digital Equipment Corp.: "PDP-10 Reference Handbook" - DEC, 1971

(5) Denning, P.: "Working Set Model for Program Behavior" - Communications of the ACM, Vol. 11, No. 5, May, 1968

(6) -- : "Thrashing, It's Causes and Prevention" - AFIPS Conference Proceedings Vol. 33 (1968 FJCC)

(7) Lampson, B., et al: "A User Machine in a Time Sharing System" - Proceedings of the IEEE, Vol. 54, No. 12, Dec. 1966.

(8) Spier, M. J. and Organick, E.: "The Multics Interprocess Communication Facility" - Proceedings of the Second Symposium on Operating System Principles,