0. INTRODUCTION

The file system of TENEX provides a means of storing programs and data on various peripheral devices and providing access to such stored data. A file is more or less an ordered set of data which has a name or for some devices simply an unnamed stream of data. All files are handled uniformly with some operations unavailable for the more restricted devices.

File names are kept in directories with each entry in the directory relating the name to the location of information in the file. Directories are also named, and the names of directories are kept in a directory index. Each entry in the directory index relates the name of the directory to a number which can be used to determine the location of the directory. Separate directories are kept for each device in the system. So-called disc files may be kept on the swapping drum but still appear in the same directory as real disc files. Files may be shared in a very general way with explicit access protection. Furthermore, as many system tables and data bases as possible (consistent with efficient operation) are kept as files so that ordinary programs (with special status where necessary to protect the system integrity) may examine and process these tables for extracting information about the state of the system or performing routine activities like providing file backup.

1. FILE DIRECTORY STRUCTURE

All directories for the main file system of TENEX are contained within one large file. This large file is subdivided into regions of 4K (K = 1024) words each, one region for each directory in the system. The position of each directory in the large file is computable with simple arithmetic from a directory number which is associated with a directory name as described below in section 2. The file directory file is permanently open and referenced by a fixed file number. After a user logs in, the eight pages which constitute the portion of the directory under which LOGIN is done are mapped into a 4K region of the monitor address space of every process in the job.

1.1 Individual Directory Format

The portion of the file directory file devoted to an individual directory is divided into three areas. The area at the low end of the directory has a fixed allocation and contains all the bookkeeping information for the directory. The second region contains many different kinds of information and is described in section 1.3. The last region is an ordered symbol table which associates name strings with "value" information in the free storage region.

1.2 The Fixed Allocation Region

The contents of the fixed allocation region are listed and described below:

1.2.1 Directory Lock and Use Indicators

These two words are used to arbitrate various kinds of access to an individual directory and prevent simultaneous references from losing. The directory lock must be set before the directory is modified in any way which could affect other processes. The directory lock must also be set when a process which is examining the directory could be affected by another process modifying the directory. The use indicator serves to prevent unnecessarily locking the directory for long periods of time. Whenever the directory is in use, the use indicator is incremented. Before the directory can be locked for an extended period of time, the use indicator must show no uses are being made of the directory. Manipulating the use indicator requires the lock to be set.

1.2.2 Directory Number

The directory number is used to identify the directory currently being mapped for a particular process. This is done to eliminate unnecessary map switching. Because the information is redundant, it also aids in detecting system failures.

1.2.3 Symbol Table Bounds

Two words define the region of the directory which currently contain the symbol table.

1.2.4 Beginning of Free Storage List

The left half of this word points to the beginning of the free storage chain described below in section 1.3.

1.2.5 Default File Protection Word

The contents of this cell are used to initialize the protection word of a file descriptor block in the absence of an explicitly specified protection given by the user or program.

1.2.6 Directory Protection

This word is used to determine who may reference this directory, and how. The following kinds of

protection are afforded:

a. list the directory
b. open files from the directory
c. add new files to the directory
d. attach to the directory
e. ownership rights (delete, rename, change account number)

1.2.7 Default Automatic Backup Protection
This word points to a block of bytes which specifies how many backup versions of various groups of files are to be retained.

1.3 Free Storage Region

The free storage region is used to contain all of the variable length data which is needed for the directory. Examples of items kept in the free storage region are name strings, file descriptor blocks, and protection strings. Space in this region is dynamically allocated and deallocated and incrementally garbage collected. This is done to reduce the need for total garbage collection. (Total garbage collection may be occasionally necessary due to fragmentation of the space.) Use counts are kept where necessary and at the time an item becomes unnecessary, its space is returned to the free storage pool.

All blocks in the free storage region which are in use, have similar formats. The first word contains in the left half a (negative) block type and in the right half the length of the block. The rest of the words in the block contain the data for the block. Free blocks contain in the

left half a forward chain pointer to the next free block. The right half of a free block again contains the length of the block. The end of the free chain is marked with a zero in the left half of the first word of the last free block.

Space is allocated in this region by scanning the free storage chain for a block which is:

1. Exactly the right length.
2. Greater than the length needed by the size of the most common block.
3. Greater than the length needed by the smallest amount.

The above criteria are applied in the order given, and if a free block is found satisfying any of them, the space needed is extracted from the block and made into a new used block. If a block cannot be found which satisfies one of the above criteria, an attempt is made to expand the free storage area by moving the symbol table to the end of the next page and adding the 512 words thus gained to the end of the free storage list. If the symbol table cannot be moved then there is no room for this entry. Note that when this happens, it may still be possible to allocate blocks of smaller length.

Deallocation is done by scanning the free list for the free block just preceding the block being deallocated. The block is either linked into the chain, or if it is adjacent to either or both of the preceding or following blocks, it

is merged with that block.

1.3.1 Block Formats

The formats of the various block types which may exist in the free storage region are given below.

1.3.1.1 Free block

word	contents
0	XWD pointer to next free block,length
rest	ignored

1.3.1.2 String block

word	contents
0	XWD 4000xx*,length in words(n)
1...	ASCIZ /the string/

(* xx are used to identify what the string is for)

1.3.1.3 File descriptor block

word	contents	
0	XWD	400100,25 (block type and length)
1	XWD	control bits,name pointer
2	XWD	other extensions,SIXBIT extension
3		file address (including disc vs. drum) and class
4		protection word
5		first creation date and time
6	XWD	last writer,retention count
7	XWD	other versions,version number (job number is used for version number in the case of temporary files. See section 4.1.)
10		account number (positive) or pointer to account string (negative)
11	XWD	last byte size,number of versions to retain
12		length in bytes
13		this version creation date and time
14		last write date and time

15		last reference date and time
16	XWD	write count, reference count
17		last incremental backup date and time
20		last medium term backup date and time
21		last archival backup date and time
22		backup bits
23		backup location (tape number)
24		user settable word

1.3.1.4 File indirect pointer block

word	contents	
0	XWD	400101,6
1	XWD	control bits,name pointer
2	XWD	other extensions,SIXBIT extension
3	XWD	directory number pointed to, pointer to file name string of file pointed to (includes name, extension, and version)
4		protection
5		creation date and time

1.3.1.5 Backed-up file block

word	contents	
0	XWD	400102,3
1	XWD	control bits,name pointer
2	XWD	other extensions,SIXBIT extension

1.3.1.6 Account string block

word	contents	
0	XWD	400200,length of block
1		retention count
2...		ASCIZ /string/

1.3.1.6 Protection string block

word	contents	
0	XWD	400201,length of block
1		retention count
2...	BYTE	(indefinite number of bytes specifying protection)

1.4 Symbol Table Region

The symbol table region is used for associating a string or symbol with a block of information. Each entry in the table consists of one word containing in the left half a pointer to a string block and in the right half a pointer to a block of other information. Three bits in the top of the right half are used to indicate the type of information. Three types of information blocks are pointed to from the symbol table. There are: file names, group names, and protection names. Entries are ordered in the table alphabetically; i.e. the comparison used to order the entries is "SUB JCRY0" on successive words of the strings. The entry type is also used for ordering, and has the highest weight.

Entry type 0 is used for file names. The left half of the word addresses a string block in the free storage region which contains the file name string. The right half word addresses one of three types of blocks. The block addressed is either a file descriptor block, a file indirect pointer block, or a backed up file block.

Entry type 1 is used for group names. The left half of the word addresses a string block containing the group name string. The right 15 bits addresses a group descriptor block.

Entry type 2 is used for protection names. The format of the symbol table entry is parallel to the above two entry types.

2. DIRECTORY INDEX STRUCTURE

The directory index is also a file as is the file directory. The directory index is also divided into subindices of 4K words each in order to avoid map switching while searching. The correct subindex can be found on the basis of the first character of the directory name by dispatching into a table at the beginning of the directory index. Each subindex is divided into 3 regions which serve the same functions as the three regions of an individual file directory. The first 4K region of the directory index is not a subindex, but contains directory index global information including a directory number table for performing the inverse translation from number to string. This 4K region will be mapped as part of the swappable monitor. A 4K subindex will be mapped in the same 4K region of the process monitor map that is used for directories. To distinguish whether a directory subindex or a directory is currently mapped, each subindex will have the negative of its subindex number in the same position as the file directory number in a file directory.

The fixed allocation region for each subindex of the directory index contains the following items of information.

1. Directory subindex lock used to lock the directory to prevent its use while in a transient state.

2. Directory subindex use indicator serves to indicate the subindex is in use and cannot be locked.

3. Negative subindex number identifies which subindex is currently mapped in this process.

4. Symbol table bounds delimit the current limits of the symbol table.

5. A pointer to the beginning of the free storage chain is used to allocate and deallocate space in the free storage region.

The free storage region is managed in the same way as the free storage region of the file directory. The blocks in the directory index consist of mainly string blocks and directory descriptor blocks. The directory descriptor block describes all the attributes of a directory (and the associated user) which are necessary to identify it to the system. The items contained in a directory descriptor block are the following.

The password string pointer points to a string block which contains the password which must be typed by a user on a terminal in order to login under the particular directory name associated with this descriptor block. If this pointer indicates that no password exists, then login may not be done under this directory name. Files may be referenced by explicitly naming the directory, or attaching to the directory.

The directory name string pointer addresses the string pointed to by the left half of the symbol table entry whose right half addresses this directory descriptor block.

The maximum permanent disc allocation specification determines how much file storage the files in the directory associated with this block are allowed to occupy on a long term basis. Files may be stored on the disc in excess of this number, but at the time the user logs out, he is advised of the fact that he has exceeded his

storage allotment and given the opportunity to reduce his storage requirements by deleting files or requesting certain files to be backed up. If he fails to do this, files may be moved to backup storage at the discretion of the system in order to reduce the storage used below this level.

The absolute maximum disc allocation specification determines a stronger upper limit on the amount of file storage a directory may occupy. If this allocation is exceeded, a user attached to this directory may not open any files for writing until he has deleted or requested a backup and delete operation for enough files to reduce his total usage below the absolute maximum.

The date and time of last LOGIN are saved to determine which (if any) login messages the user should see. This prevents the user from seeing a lot of junk that he has seen before.

A word of privilege bits is allocated for specifying special privileges available to a user logging in under this directory name.

A word of mode flags is allocated to control any options which the user has enabled for this directory.

The directory number gives the internal system handle for this directory. It is used to directly find the location of the directory corresponding to this name.

User identification information provides information about the user responsible for this directory which might be necessary to identify him to people. Address and telephone information would be kept here.

Information about special system resource guarantees are kept in the directory index so that such requirements can be stated to the system when a user logs in under this directory name.

The symbol table region of the directory index is similar to that of the file directory. The left half of each entry points to a string block containing the directory

name. The right half contains the directory number.

The directory number table is used to translate from a directory number to the location of a directory descriptor block. Since the directory number space is not dense, the directory number table is a hash table. Each entry in the table contains in the left half the location of the directory descriptor block and in the right half the directory number.

3. FILE NAMES

File names in TENEX are composed of five identifiers. These are device, directory name, file name, extension, and version. These five items uniquely identify any file accessible to a user on the system. The device name identifies on what device in the system the file is contained. The directory name gives the directory under which the file appears. The file name and extension and version identify a particular file in the directory given by the device and directory name.

The character set from which these identifiers may be constructed is composed of the upper case letters, digits, and punctuation marks found in the range 40-137 (octal) in the ASCII set with the exception of the characters period : ; < > space and comma. These punctuation marks may be appear in an identifier if they are preceded by a control-V. The control-V is not part of the identifier. Lower case letters may be used, but they are equivalent to the corresponding upper case letters. This is done so that all file names may be typed on upper case only terminals such as model 33 Teletype terminals. The punctuation marks listed above as exceptions are used as delimiters for various fields of the file name and while their use within a particular field may not be ambiguous from a syntactic view, a blanket restriction on their use is made for simplicity.

The general form for a file name is:

device:< directory name> name.extension;version number ;T;Pprotection specification;Aaccount string

The underlined characters are real, the rest is representative.

A file name may either come from the memory of a process, or it may come from a file (including a terminal) or it may come from both memory and then a file. If any of the fields of the name are omitted, the field is supplied by the program, or from a standard set of default values. The standard default values are:

device	DSK
directory	currently attached directory
name	no system default, must be specified by program
extension	null string
version	no system default, program must specify usually most recent for reading, and next higher for output. Special ways of representing these cases are provided for the program.
T	file is assumed not temporary
P	as specified in directory (usually read all and write self only)
A	account number of login.

Recognition is done on file names in a uniform manner regardless of the source of input, or the intended use of

the file. The program can control certain aspects however. Whenever an alt-mode is input from memory or file, the portion of the field input prior to the alt-mode is looked up according to which field is currently being input. A match is indicated if the input string either exactly matches an entry in the appropriate table, or is an initial substring of exactly one entry. In the latter case, the portion of the matching entry not appearing in the input string is output on the output file. The field terminator is output also, and recognition is done on successive fields with a null string as input, or if not possible, the fields are defaulted. If the file name cannot be uniquely determined, as much as possible is recognized and a bell is output signifying that more input is required. If the string input cannot possibly match any existing file name by appending more characters, an error return results.

Control-F behaves like alt-mode except recognition is not carried out past the current field. This allows the name to be recognized for example, but not the extension.

If an alt-mode is not used, then each field is delimited as indicated above, and the name so specified must exactly match some existing file name unless the program specifies that new file names are allowed (i.e. an output file). The complete file name is specified whenever all fields have been recognized, or a comma, space, or carriage

return is input.

Confirmation may be required if the program has so specified in the call. In this case, one of three messages is output following termination of the name. The messages are: "New File" if no versions of the indicated file exist, "New Version" if other versions exist but the indicated one does not, and "Old Version" if the indicated file name and version already exists. Following the message, one character is input and if it is one of the characters alt-mode, space, carriage return, or comma a normal return occurs. Otherwise, an error return results and no JFN (job file number) is attached to the file name. The confirmation character may be read by the program by using the "back up one character" operation.

Editting characters are recognized while typing file names as follows:

↑ A deletes one character from the current. If no characters remain in the current field, a bell is output.

↑ W deletes the current field. If the current field is null, a bell is output.

↑ X causes the file name gathering operation to be aborted, and an error return given to the program.

Rubout deletes the entire input, and starts over.

↑ R retypes the entire name as specified so far and awaits further input.

4. FILES

There are several types of files in TENEX. These are all handled in a uniform way as far as possible. The exceptions are for operations which have no meaning for certain devices. Such operations are treated as illegal.

4.1 Ordinary Files

So-called ordinary files are maintained on disk/drum/core and provide the heart of the TENEX file system. When the unqualified term "file" is mentioned, it usually refers to an ordinary file. Each file has at least one index block which is essentially a page table in which each entry gives the location of one page of information in the file. There are variants of ordinary files which differ in the way the system handles them, but not in their basic structure.

Files longer than 256K have more than one index block and are called "long files". The location of index blocks for a long file is given in a higher level index block. Long files provide a means for storing up to 128KK or over 128,000,000 words. The management of long files is invisible to the user.

Frequently programs use scratch files to store intermediate results. The names for such files are usually built into the program and therefore if that program is run under under more than one job, under the same directory name, a conflict in the use of the name occurs. It is necessary that these programs reference different files. This is done by making the default version number for these so-called "temporary files" be the job number. Temporary files also have the feature that they nominally disappear when logout occurs and are therefore useful for storing data of a temporary nature.

It is occasionally desirable that the name of a file and its protection and accounting information be permanent even though its contents be deleted. Such a file is a user's message file which must be accessible to other users for appending messages, but must be deletable by the owner. For this purpose, a file may be declared "permanent". Note that a file may be permanent and temporary independently.

The existence of version numbers for files has been alluded to in the above discussion. Version numbers are nothing more than a further extension of the file name which is used for two purposes. First version numbers allow environments to be saved which refer to particular versions of files and subsequently resumed even if incompatible newer versions have been created in the meantime. Second, it

provides an automatic backup version of a file which is rewritten because the usual default version number for files opened for writing is one greater than the most recent version number.

4.2 Subroutine files

A subroutine file is a mechanism for making a program look like a file. That is, file operations instead of transferring data to or from a device, make calls to a program. This allows special processing to be interposed between the data the main program sees and the data appearing on an ordinary file, or the generation of data by the subroutine file from internal strings or whatever.

Subroutine files will be implemented in a more general way than was done on the SDS-940 system. First, subroutine files will be processes and have all the capabilities of processes. Subroutine files may be called from any other processes in the job if access is permitted (as for other files). Subroutine files may be named and appear in file directories. Subroutine files will have multiple entry points so that the operations that can be performed on ordinary files can have their counterparts for subroutine files. Subroutine files can generate any of the special signals which can occur when using an ordinary file such as end-of-file, data error, etc. The only restriction placed

on a subroutine file is that it may not be called recursively.

Opening a named subroutine file is equivalent to setting up a procedure type file as a process, and then declaring it to be a subroutine file. Opening an unnamed subroutine file is simply declaring a process to be a subroutine file. When a process is opened as a subroutine file, it is started once at its "open" entry. Subsequent data transfers cause the process to be restarted at its "input" entry for input operations, "output" entry for output operations etc. A byte size is associated with a subroutine file and the system packs or unpacks data to match byte sizes between the subroutine file and the program calling it. Operations for which no entry point is provided are treated as illegal.

4.3 Terminals

Under most circumstances, terminals are used like files. That is, they provide a stream of input characters, or accept a stream of output characters. The TENEX Exec will open the terminal for input and output so that programs will not have to do so. Terminals are discussed further in TENEX-5.

4.4 Mailbox Files

Another special kind of file is a so-called "mailbox file". Opening a mailbox file establishes a depository in the monitor for exchanging information between two or more processes anywhere within the system. Mailbox files are discussed more fully in TENEX-10.