TENEX Functional Parts and Formalization of their Interfaces

Wrenwick Lee January 29, 1974

1. Introduction

TENEX Documents

TENEX Memos

TENEX Monitor Manual

Documentation vs Implementation

2. Brief Functional Descriptions

Scheduler

Memory Management

Peripheral Control

File System

Interrupt, Trap Handler

System Services

3. Entry into a Functional Part

Data Structure Functions

Data Structures are manipulated in the following ways:

Domain, Local, Nonlocal

Uses of data structures

4. Functional Part Interface Identification

5. Movement of Functional Parts into Separate Processors

Security Effects

Interrupts

Local data structure protection

Nonlocal data structure protection

Processors

Memory management

- 6. Interface Formalization
- 7. Modularization
- 8. Random Comments

How does having two modes (user and monitor) affect interface specifications? Probably Reorganization

File System, access checking stuff - how could they go onto a separate processor?

(Anonymous Notes from an Anonymous Source)

9. TENEX SOURCES Classification

TENEX Functional Parts and Formalization of their Interfaces

1.1 Introduction

The TENEX operating system can be considered to be comprised of six <u>functional</u> parts:

- 1) Scheduler
- 2) Memory Management
- 3) Peripheral Control
- 4) File System
- 5) Interrupt, Trap Handler
- 6) System Services (JSYS, Monitor Calls)

Taking the set of TENEX operating system files, one can logically apportion them among the six functional parts.

1.2 TENEX Documents

1.2.1 TENEX Memos - The TENEX memos give a nice general overview of several parts of the operating system:

Scheduler - TENEX Memo 12

Memory management - TENEX Memo 12

File System - TENEX Memo 4

Peripherals control and interrupt, trap handling are not described very clearly. Information pertaining to them are found in:

Terminal Service TENEX Memo 5

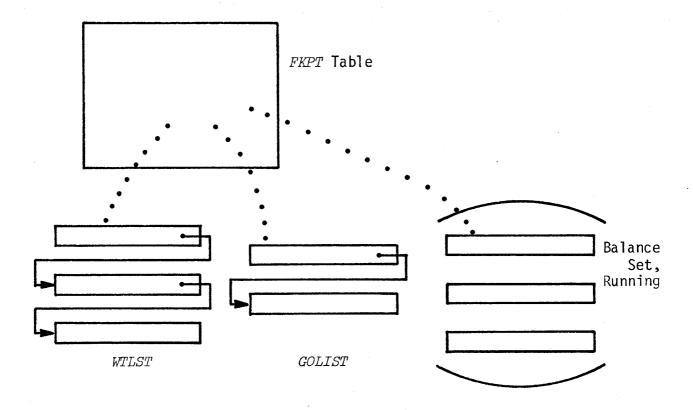
Calls & Interrupts TENEX Memo 8

Pseudo-Interrupts TENEX Memo 7

- 1.2.2 TENEX Monitor Manual This manual gives a brief description of the functions of the different source files of the monitor. It is incomplete and sketchy at points. However, it does provide another overview of the operating system in addition to the memos.
- 1.2.3 Documentation vs Implementation There are important differences between the documentation and the implementation. The documentation is basically for DEC PDP-10 KA processor with the BBN processor modifications. The implementation we are working with is a DEC PDP-10 KI processor without the BBN processor modifications.
- 2.1 Brief Functional Descriptions
- 2.1.1 Scheduler The scheduler controls the running of processes. In TENEX, process $\equiv fork$. The data structures and functions used by the scheduler are related to process handling.

The basic data structure for a process is a PSB (Process Storage Block), containing various parameters and attributes of the process. In addition there are various system fork (\equiv process)

tables to which the scheduler refers to in scheduling processes.



The FKPT structure shown above is a table of active processes.

Within it are scheduler queues. WTLST is a queue of processes in wait mode. GOLIST processes are ready to be loaded and run. Balance set processes are mostly loaded except for a page or two. The actual running process must belong to the balance set.

The scheduler makes decisions as to which process to load and run. There are other system process structures that the scheduler uses.

The operating system must perform certain real-time tasks. Echoing terminal characters and display refreshing are two of these.

The operating system is interrupted by the system clock at regular intervals and control is sent to the scheduler. The scheduler thereupon dispatches to the task handlers.

2.1.2 Memory Management

TENEX is a virtual storage system. Pages are swapped into core and out. This is the responsibility of memory management.

The data structures and functions used are related to page management.

The basic data structures used are the CST (Core Status Table) and the DST (Drum Status Table). These tables contain information about the status of pages as they move about. A PT (Page Table) is used to record the location of all pages of processes whether in core or secondary memory.

Parameters and characteristics of the rotating devices are known by the memory manager as it queues data transfers between these devices and core.

It communicates mainly with the scheduler module. The scheduler module tells it which process to swap in. At times, it tells the scheduler that it should reschedule swaps.

2.1.3 Peripheral Control

TENEX documentation does not consider peripherals control as a separate functional area as we are doing in this report.

The peripheral control in TENEX is found in a combination of a part of the file system, a part of the interrupt handling, and numerous device control modules.

Peripheral control is in two parts: terminal peripherals and "others". Others include magnetic drum, disk, paper tape, etc.

The data structures and functions used are related to input / output control and buffering. There are various buffers for data transfers to / from peripheral devices. PTR (Paper Tape Reader) and MAGTAP (Magnetic Tape) are two of the many drivers

used in peripheral control.

Peripherals are the primary cause of hardware interrupts. Devices interrupt the processor to signal I/O completion. The processor is regularly interrupted by a clock to service terminal peripheral data transmission. Terminal peripheral handling involves an interactive element. Echo modes, break characters and wakeup are several functions that must also be handled.

2.1.4 File System

The file system organizes pages into functionally useful ordered sets. It maintains, organizes and controls accesses to files in the system. The data structures and functions used are related to maintenance, organization and access control of files.

A FDB (File Descriptor Block) contains information describing the file. A pointer in the FDB points to an IB (Index Block) which contains locations of the pages of the file. FDB's are combined in FD's (File Directories) to form a group of associated files.

2.1.5 Interrupt, Trap Handler

The data structures and functions used involve interrupt and trap handling.

TENEX has two modes, user and monitor. Hardware <u>interrupts</u> are handled by monitor mode. Fixed locations in the monitor *EPT* (Executive Process Table) are used to dispatch to different interrupt

routines. The routines are of three types:

- 1) APR (Arithmetic Processor) errors
- 2) Device I/O completion
- 3) Clock

The clock interrupt causes the scheduler to be invoked to service real-time processes.

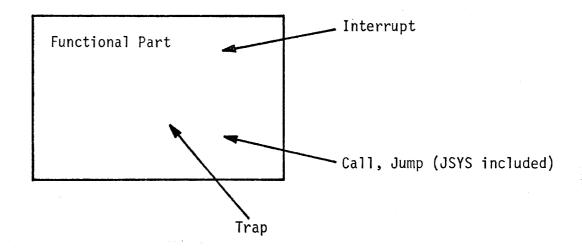
There are two classes of traps. The first is APR originated. The second is Pager originated. The Pager maps the virtual page of a process to a physical page, looking in associative registers and PT's.

The APR traps are page failure (page not in map), and overflows. Fixed locations in the UPT (User Page Table) for user mode and EPT for monitor mode, are used to dispatch to different trap routines.

The Pager traps set the TSW (Trap Status Word) in the PSB and then jump to a trap routine.

2.1.6 System Services - These are implemented as JSYSes (Monitor Calls). Some JSYSes are contained in the other functional parts of the operating system. Thus the JSYSes do not form an entirely independent part. Both parts of the operating system and the user may make JSYScalls. The JSYS manipulate various data structures. There are many file system JSYSes that read and set different parts of the file descriptor block. The RESET JSYS initializes many things about the current fork. In providing the user these services, the system must protect against the user's gaining illegal and damaging access to system structures.

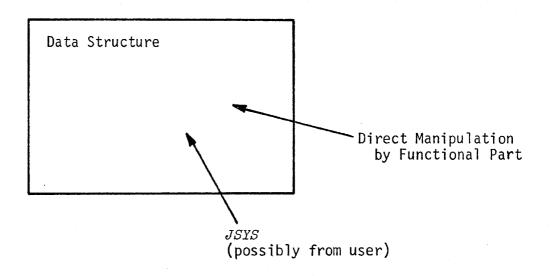
3.1 Entry into a Functional Part There are three main entries into a functional part



- . Interrupt e.g. The scheduler is entered whenever a periodic clock interrupt occurs. Peripheral control is entered when device interrupts occur.
- . Call, Jump e.g. File system routines dispatch to peripheral controls when it finds out that a file it is to perform data transfers on is actually a physical device. The scheduler calls the memory manager to swap in processes.
- . Trap e.g. The CPU, not finding a page in the map, traps to the memory management part to locate (and perhaps swap in) the page.

3.2 Data Structure Functions

3.2.1 Data Structures are manipulated in the following ways:



3.2.2 Domain, Local, Nonlocal

- . A data structure manipulated by a functional part is said to be in the <u>domain</u> of that functional part (FP).
- . A data structure manipulated by only one functional part is said to be <u>local</u> to that functional part. If it is manipulated by more than one FP, it is <u>nonlocal</u>.

Note: a data structure manipulated by a user via a $\it JSYS$ is said to be in the domain of the user.

Note: Later, we may further want to define domains of different types.

i.e. whether a FP reads, writes, modifies, etc the data structure.

3.2.3 Uses of data structures

A local data structure is used primarily for state information needed only by a single functional part. Data structures in the domain of several functional parts are often used to communicate between the functional parts.

Examples: The memory manager is the only FP concerned with the DST (Drum Status Table). This data structure is local to memory management. The BALSET (Balance Set of Pages in Core) is manipulated by both the scheduler and memory manager, being in both domains. The PSB (Process Storage Block) is in the domains of most FP's because the process is a fundamental element of the operating system and manipulated by most FP's

4. Functional Part Interface Identification

Having our different FP's, the first task is to identify the interfaces between them:

- 1) Identify the entries into each functional part (3.1)
- 2) Identify the domains of each data structure (3.2) Identifying the entries and domains will give an exact picture of the interfaces of the FP's. Currently, this is being pursued, data structures domains and FP entries are being charted out.

5. Movement of Functional Parts into Separate Processors

It has been proposed that certain functional parts be moved to separate processors. Prime candidates are the memory manager and peripherals controller.

- 5.1 Security Effects
- 5.1.1 Interrupts If the peripherals control is placed in a separate processor, then the interrupts due to Device I/O completion (2.1.5) may be eliminated. Similarly, one would not need to cause a clock interrupt to the scheduler to do I/O (real-time) processing because the peripheral control processor would be dedicated to this task. The remaining APR (CPU) error interrupt can be changed to a trap of some sorts. The elimination of interrupts on TENEX would simplify the operating environment a great deal. The operating system will be easier to define (# easy to define) and install protection mechanisms.
- 5.1.2 Local data structure protection See 3.2.2 Local domains can be protected from illegal access. Techniques may be:
 - 1) putting it into a local memory
 - 2) hardware enforcement
 - 3) object protection a la Anita Jones'* style Details are to be worked out at a future date.

^{*} Jones, Anita "Protection in Programmed Systems" Dept. of Computer Science Carnegie-Mellon

5.1.3 Nonlocal data structure protection - See 3.2.2 Nonlocal data structures present a more difficult protection problem.

With more than one processor manipulating it, access must be given to only one at a time. This can be controlled by a PROTECT mechanism as used for the BCC 500.

A problem regardless of whether the functional part goes into a separate processor or not, is control of manipulation of the data structure. With the BCC 500, there were times a nonlocal data structure was modified incorrectly but it was uncertain who did it. Suggestions have been made to the effect of using a processor to receive requests to modify a data structure, ensure the validity of the request, and then do it. More thought needs to be put in this area.

5.2 Processors

The processors to be used would probably be of the nature of Al Goodrich's LOLO processor.

5.3 Memory management

A question arises as to whether memory management should have its own channel (built to specifications) or continue with the standard DEC interface.

6. Interface Formalization

Whether the FP goes into a separate processor or not, the interface (both entries and data structure manipulations) must be formalized. It can take the form of request nodes passed

between FP's, or controlled access to certain data structures, etc.

7. Modularization

The FP's might be physically separated in separate processors or in core, running on the same APR (CPU). Barriers are erected around each so that one FP cannot communicate with another except through formally defined entries and controlled access to data structures in the domain of both FP's. One cannot branch into the other's code arbitrarily, nor modify some of his data areas arbitrarily. Hardware or software fences are erected.

Security is then thought of in terms of the security of the individual modules. A secure module cannot be breached from one of the other modules. Nor can it breach other modules.

Note: The importance of control over nonlocal data structures is shown here. Module 1 which shares a data structure with module 2 cannot be said to be secured as long as module 2 is not controlled in what it can do to the data structure. A formalization of how the data structure can be manipulated would probably best reside in a module separate from 2. In that way, module 2 can't "cheat".

8. Random Comments

8.1 How does having two modes (user and monitor) affect interface specifications?

8.2 Probably Reorganization

Indiv Device Modules

Part of File System

Interrupt System (most of)

have to find and decide how to slice peripheral control stuff out of file system.

- 8.3 File System, access checking stuff how could they go onto a separate processor?
- 8.4 (Anonymous Notes from an Anonymous Source) (See Section 5)

 Some thoughts about moving parts of the Tenex O/S into separate

 "management" processors.
 - I. All the modules moved out of Tenex will obviously have to be re-written. This is unfortunate in a way but has the advantages that
 - a) if we have our way, the new code will be written for a processor which enforces with hardware the separation of the code into small tasks limited in their ability to screw things up.
 - b) an opportunity will exist to enforce by administrative means the construction of programs in a way...

It will probably be desirable to design a language which will strongly "suggest" the proper organization of code and allow for the use of the new hardware of (a) above. Hopefully, this can just be a modification to an existing language.

- II. A large problem to be solved is the design of the interface between user programs running on Tenex and the services which have been moved out of Tenex into the other processors. Some points:
 - 1. The interface as seen by user programs musn't change.

 This means theymust be able to continue using the JSYS instruction (i.e., opcode 140, no matter how this gets handled by the hardware), with arguments being passed in the central registers. This has a bunch of implications. Remember that in some cases arguments are pointers into arbitrary places in the user programs virtual address space. As the JSYSes now work, this causes no difficulties (well...) because the JSYS code runs on the CPU and has the benefit of all the elaborate mapping and page-fault handling stuff implemented there.

What I'm trying to do is consider the possibility of implementing (at least some of) the JSYSes entirely on a separate processor. This has been given some thought from time to time. It seems most feasible (though maybe not reasonable) if the JSYS processor is another KI-10 processor. One can imagine that when a program running on a "user KI-10" executes a JSYS, this causes it to be kicked off that processor and scheduled to run on the JSYS processor.

9. TENEX SOURCES Classification

The following is a first cut at classifying the various source files of the TENEX operating system as belonging to functional parts. As more information is known, reorganization will occur.

(Compiled by Alan Kam and Wrenwick Lee)

1) Scheduler

SCHED, SWPMON

2) Memory Management

DISK11, DISC, DRM, DSK, PAGEM

3) Peripherals Control

DECTAP, FILTTY, IMP11, IMPDEV, MAGTAP, NETWORK, PTP
PTR, TTYSRV

4) File System

DEVICE, DIRECT, FILINI, GTJFN, IO

LOOKUP, STRING

5) Interrupt, Trap Handler

KISRV, PISRV

6) JSYS

FREE, FUTILI, IH, SWPMON

Others:

*) Macro, Data Structures

FILE, PARAMS, PROLOG, STENEX

*) Miscellaneous

ERRMES, SYSNAM, SYSSAV, VERSIO,

POSTLD, MR, MFEIN, MFLOUT, NATIME