NOTES ON:

PLACING THE FUNCTIONAL MODULES OF TENEX INTO MANAGEMENT PROCESSORS

Wrenwick Lee

February 15, 1974

Technical Memo. TM.74-7

Notes on:

Placing the functional modules of TENEX into management processors.

Qualification

The following notes are not necessarily organized in a logical manner. They represent many observations and thoughts concerning the above topic. Several purposes are intended for these notes. A first gathering of the various "loose" thoughts is intended. This will be worked into a more formal memo or report. A means of communication to other members of the task as to the nature of moves to management processors as I see it. In conjunction with that, to receive comments from others as to

1) disagreements with my observations and thoughts

2) further development or insights into the topics mentioned

3) as a booster to others to pursue or plan towards certain aspects of the topics mentioned.

Periodic Routines

There are many cyclic, time-dependent routines that must be invoked by the scheduler in its main loop. The scheduler uses the system clock to update routine clocks to determine when to run these routines. These routines can be placed in the management processors which maintain their own querying cycles. Some of these routines are

1) TTCH7 to check for terminal transmission, etc. echoes.

2) IMP

3) DSKCHK (?) specific function needs to be specified.

4) MTACHK (?) specific function needs to be specified.

Blocks and Wakeups

Presently the following scenario occurs for blocks and wakeups. To block a process, a function in some other part of the operating system makes a well-defined entry into the scheduler. State is saved, and the process is blocked. Blocking involves setting a word in a system fork table for the fork being blocked. This word containing a test condition and a transfer address of a test routine. The scheduler in its main loop runs through these tests to see if a fork should be waken (periodically depending on the ISKED flag). Upon wakening, the scheduler moves the fork from the wait to the go list. Many of these tests are in the scheduler (the DISMISS tests, BLOCK tests) but there are many in the other parts of the operating system.

These blocks and wakeups should be handled at the local management processors. The tests should be maintained at the local processor (As for modules that will continue to share the CPU with the scheduler, assuming the scheduler is not in a separate processor, the current scheme might be maintained). Instead of the scheduler testing for wakeups, the separate processors will request wakeups on a ring queue. Presently jumps are made to certain routines in the scheduler (JSYS EDISMS, SCHEDP etc.) to cause a block. These should be changed to requests to block.

We must find these routines and list them.

Re-Entrant Considerations

Currently the TENEX system is re-entrant (to the greater extent). Explanation is given by the following example. When a process blocks for a page fault, it might be in the memory manager. It calls the scheduler via SCHEDP and the scheduler remembers the current location in memory

management that the call came from. Also the state of the system necessary for that specific fork is saved in that process PSB (Process Storage Block). Another fork with a different PSB is then started up. Later on, when the page comes in for the original fork, and it reaches sufficient priority, the fork is continued where it left off in the memory manager.

There is no re-entrance into the management processor itself since it is running independently of the CPU. Observations of memory management imply that certain parts of the code must remain outside of the processor itself. Thought must be given as to how re-entrantness (as currently implemented) affects the cut of code into processor and that remaining in the monitor. A rule of thumb is that the more fork related, the more it must remain in the monitor. Actually a good idea is to push the re-entrantness out of the memory manager as much as possible. This may involve more state keeping in the local tables or in the PSB. This problem and solution is a good one to study for one module (e.g. memory management) because it is a case of the problem one encounters among all the modules (i.e. peripherals module, file system).

Currently Judy is working on looking at PAGEM, the memory manager. Since I have studied the scheduler pretty thoroughly already, when she is fairly familiar with PAGEM, then we can address ourselves to this problem in actual implementation detail. From this we will probably obtain guidelines with respect to the other management processors.

Interrupts

The interrupt scheme presently employed is complicated. It would be hard to convince oneself much less someone else of the security of the code involved. With the addition of management processors, much of the interrupts can be removed. The (ISB SCDCHN) which is a form of programmed interrupt is used perhaps too freely (from a security standpoint).

Many times it is used to set up rescheduling and starting up of forks. Because much of the complexity of the scheduler will be removed (e.g. all the WAKEUP tests, periodic routines) it may be that the scheduler need not be interrupt driven.

The clock does need to be updated though but this may be in an encapsulated function. The scheduler might be request driven more than interrupt driven.

To the extent that time-dependentness is less a factor for the CPU, the less role interrupts will take.

Fork Tables

Forks being the key processing unit, there are important tables that are maintained for active forks. These will probably come under protection. Protection mechanisms should be considered for these tables.

Pseudo-Interrupts

A fork table FKINT, and also FKINTB are definitely to be shared among various processors. These perhaps would be important elements to use in thinking of domain protection and control. (The fork tables in general provide a good case). The terminal handling processor will probably set these tables as well as the scheduler. E.g. ↑C received on a teletype will cause FKINT and FKINTB entries for that fork to be set. Later the scheduler will see and handle the pseudo-interrupt accordingly. Similarly, the APR processor overflows will generate such interrupts.

Macro-scheduler and Micro-scheduler

It seems feasible to have a macro-scheduler (part of monitor) and a micro-scheduler (in a processor). The micro-scheduler can handle the more time-dependent stuff (i.e. update clocks) while the macro-scheduler

handles the larger more general scheduling tasks such as LOGIN, LOGOUT. In fact, the micro-scheduler might perform another task currently relegated to the macro-scheduler, that of determining when processes can be removed from the balance set.

Domains, Data Structures

The domains and data structures must be well chosen and the functions or accesses to them well-defined in a multi-processor environment. As mentioned earlier, hardware or software fences must be defined to limit access (of various types) to these domains.

Relation to Critical-Non critical Code, Insertion of Protection Mechanisms

These two topics are firmly tied in with that which is the subject of this discourse. The right hand must know what the left hand is doing. Discussions should be set up to coordinate efforts. e.g. What goes into a separate processor need not be worried as to whether it is critical or non-critical.

JSYS Considerations

Movement into separate processors must be cognizant of the effect on JSYSes. One of the inherent problems is the same as that discussed in re-entrantness. The code the JSYS may want to execute may be in the processor. Thus the value of the above proposed discussions. A memo should be written concerning major problems of implementing JSYS caused by separate processors. Several bases of information are needed. A good understanding of the various JSYSes' implementation, an understanding of the probable partition into processors which is being described here, and ability to combine JSYSes into logical groups so that major properties and difficulties can be foreseen and thus worked on.

Details

The actual implementation involves overwhelming detail. But for the system to run practically, all this detail must be gleaned. A comprehensive base of knowledge is an important requirement. This task of moving TENEX into separate processors should include several subtasks. Some of these have been already mentioned. But for example,

1) concentrating on removing the peripheral stuff from the standpoint of the peripherals control processor. This is perhaps the most immediately fruitful and practical area.
2) similar for the memory manager
3) general coordination, and problems to be encountered with all processors. Setting up protection mechanisms and domain.
4) concentrating on simplifying and formalizing the interfaces between processors. Creating feasible protection mechanisms.

Protection Mechanisms

Perhaps the least developed, but most interesting aspect is to be able to protect the important structures. This topic will be pursued soon. Various ideas (general in nature) have been proposed. At this point it is feasible to consider some of these proposals and discuss them. Already, simplicity is a key. By making TENEX simpler, we can define it, formalize it better. Protection mechanisms to be inserted into this new design can be considered.

File System

I haven't given much thought to the file system. But the following should be pursued. Should we, and how could we put the file system under separate control? Various terminal things have to be broken out of the file system. Rainer mentioned a high level cut. I would appreciate a memo on this.

The file system stuff perhaps will be more involved in critical-non critical than separate management processor. But lets not assume a priori that this is so.

Protection Processor

Really this relates to the protection mechanisms mentioned earlier. But it is worthwhile to consider the possible role (also economics) of having such a processor. What kind of things could it do (both to prevent as well as signal violations of security). Actually it might be in the nature of a SYSDDT type processor except with different functions. Or like the HISMIMP process that checks queues, etc for validity, some of this can be in microcode. From others experience and our ideas, such a processor might be very important.

Mapping, SPT, User/Monitor Modes, Fast Monitor Routine, Slow Monitor

This whole area seems like a bag of worms resulting i many gory special case checks and unbelievably messy code. We must analyze this area carefully because there's a lot of broken glass around. Judy is initially working on this stuff.

LOLO Processor

What anticipated problems can we expect with the LOLO processor? The word size is 16 bits while TENEX addresses in general are 18 bits.

Startup, Initialization

The startup code will be different in the new scheme since there is a major reorganization. Conventions need be set up to COLD START the various processors, and the necessary tables. Similar problems must be considered for system checkpoint/crash recovery. How is security

maintained in such an instance?

Formal Definitions, Procedures

As part of securing the system, sloppiness must be avoided. It is a cardinal sin. Procedure must be set up that clearly state, i.e., how data structures are accessed, who accesses them, etc. Further, constructions such as that of the protection domains (Spier) which force implementation to follow strict procedures will be used. Part of the implementation of security means prevention of sloppy programming.

Seminar Proposition

It is a good time to have a seminar on how the TENEX operating system functions. This would probably help many people grasp the system. I have some knowledge from the scheduling standpoint. Combine this with the file system and memory management, we can perhaps present the activity going on when a job is run on the system.