TENEX Scheduler

An

Overview

Technical Memo

Wrenwick Lee
June 7, 1974

TM.74-19

Comment

The following is the skeleton of a more comprehensive document which will analyze the scheduler more deeply. Each of the various points will be expanded to greater detail. Specifically, the emphasis of the later document will be on the problems of having a separate management processor do scheduling. Encapsulation will then be considered also.

However, it is felt this document will be of interest and use to those who want/need a basic understanding of the scheduler.

TENEX Scheduler - An Overview

The TENEX Scheduler's primary function is to run processes for quantums of time; to allocate the processor (according to some policy) among various processes. Processes compete for the processor resource; some with higher priorities than others.

The actual implementation of a scheduler, however, involves various gyrations that tend to conceal the basic simplicity of the scheduler.

The scheduler will be described in terms of the following functions it performs:

(1) Time-dependent system services
   a) Terminal handling
   b) Status checks (IMP, disk, magtape)
   c) Job averages

(2) Blocks and Wakeups

(3) Scheduler Requests

(4) Reservation constraints

(5) Clocks

(6) Balance Set Control

(7) Flies in the ointment

(8) Scenario: a user process being scheduled, run, rescheduled, etc.

Time-Dependent System Services

Periodically, the scheduler must run certain system service code.

The first of these is the terminal handler. Currently, a PDP-11 acts as a concentrator for various terminals. The PDP-11 receives the characters and loads them into a ring buffer. Along with the character is a tag that tells which line it came from. The terminal handler routine that the scheduler must run periodically (between 15-33 ms) retrieves the characters from the ring buffer (affectionately known as the big buffer), and packs them into the appropriate individual line input buffers. Similarly, it takes characters from the line output buffers and stores them into PDP-10/PDP-11 interface buffers. The PDP-11 takes these characters and sends them to the appropriate terminals. The reader is referred to the terminal I/O document for further details.

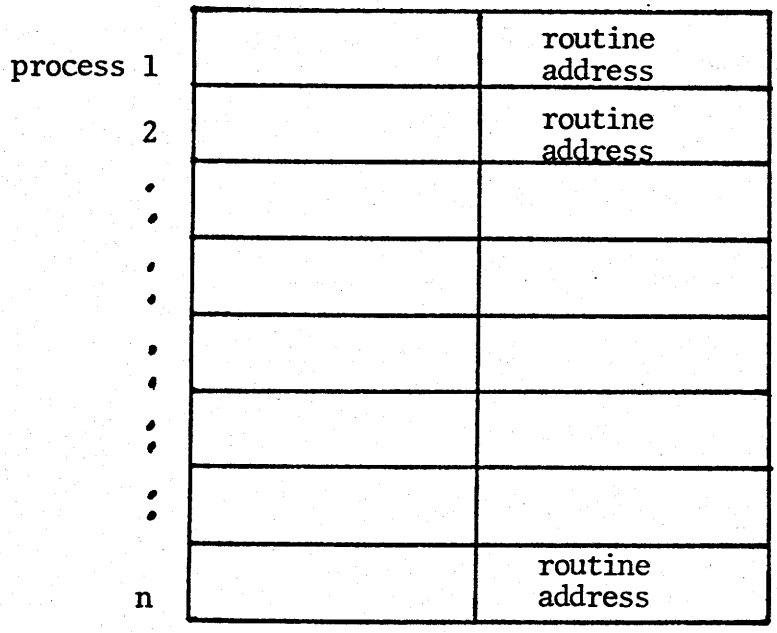
Status checks on certain devices are also run periodically. The IMP is checked frequently (between 15-33 ms), having the same periodicity as the terminal handler. If it is found that things are not well with the IMP, various corrective actions are taken. Similar checks are done on the magnetic tape units and disk, but these are at intervals of 100 ms.

Runnable job averages are taken every second. Information such as the number of jobs that are runnable at any given moment, etc. are updated.

Blocks and Wakeups

Ideally, it would be easiest to think of giving a process a quantum of processor time and that the process would then fully execute in this amount of time, whereupon we would do the same for the next process in line. Realistically, this is not usually the case. Processes have to wait for pages to be swapped in, wait for input, and wait for critical cells to be set. It is inefficient to have the processor wait along with the process. So we block the process and run a different process. There are two routines that set up the blocking. These are the routines EDISMS, and DISMSE (located in the scheduler), which are called from many different parts of the operating system. Corresponding to blocking the process, we must at a later time unblock or wakeup the process. Wakeup does not mean allocating the processor to the process, it means the processor can be allocated to the process. Testing for wakeup in TENEX is done by routines which decide if the process can now be unblocked. The address of a test routine is given to EDISMS or DISMSE when a process is blocked. These addresses are saved in a system fork (process) table: FKSTAT. There is one entry in the table for each fork (process) in the system.

FKSTAT



The entry is actually in use only when the fork is blocked. When it is not blocked, there is a slot reserved for it. Periodically, the scheduler will execute the wakeup tests for all the blocked forks. (TENEX 1.32 has a modified version of this scheme, see Appendix on changes to scheduler in TENEX 1.32) The flag ISKED is the switch used to signal the scheduler that it should go through the test routines. At worst it is set every 100 ms by a system clock (CLKS) but other events may have it set also. e.g. a pseudo-interrupt request will cause ISKED to be set.

The test routines take the form of testing if a bit has been set, or a certain time elapsed, or for positive or negative, etc. For a routine that is blocked indefinitely (e.g. by a HALT or FREEZE in the fork control mechanisms), the test defaults to an immediate failure to meet the wakeup conditions. There is also a gallery of wakeup test routines provided by the scheduler for the other parts of the operating system. However, many parts of the operating system

use their own tailored tests.

Scheduler Requests

Various requests for service can be made to the scheduler. These come in at random from various forks, terminals, etc. These are placed on a request queue and are serviced (if there are any) at worst, every 33 ms when the basic 33 ms scheduler clock interrupts. (This is described in more detail in the section on clocks). The priority of servicing these requests is low. All other time-dependent system services are done first. These requests are:

JOBSRT	LOGIN, start a new job
ASSFK	Assign a Fork
P70V	Overflow, Floating Overflow
P7FOV	Floating Overflow Channel
P7POV	PDL Overflow
P7PI1	
MPEINT	Gives I/O error interrupt

Reservation constraints

The ILLIAC version of TENEX imposes a reservation system upon the normal TENEX scheduling scheme. At present, time is split up into two minute intervals. Within each interval, a process (more correctly the job of which the process is a part) has a maximum percentage of the processor time that it can use. This is given in variable JOBQNT (msec) corresponding to that job. Every two minutes, regardless of whether the job has used all its processor allocation, the processor is redistributed again. Under the reservation system, two classes of users are designated: Heavy and light. Heavy users can be allocated up to 50% of the processor time, light users approx. 5%.

Clocks

In order to properly initiate the time-dependent routines, time the running process, and maintain various functions involving time, the scheduler is given the responsibility of maintaining certain system clocks.

The basic system clocks are derived from two hardware clocks: a 1 ms clock, and a 16 2/3 ms (60 cycle) clock. These cause processor interrupts at their respective periods. The primary system clock is

TODCLK (to-date clock)

which is incremented every 1 ms (when the 1 ms hardware clock interrupts)

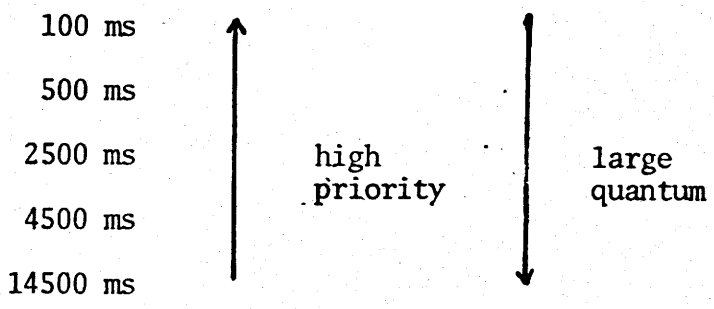
The scheduler is given control every 33 ms to update specific other system clocks. At every other tick of the 16 2/3 ms (60 cycle) clock, an interrupt to the scheduler is generated. This is a forced transfer of control to the scheduler via the interrupt system. Of course, the process running may block first, and thus the scheduler will regain control before the 33 ms are up.

The following clocks are updated at least every 33 ms or earlier if the running process blocks. These clocks are initialized (in appropriate places) with a negative ms value. The scheduler will add the appropriate elapsed time (since the last update). When one of these clocks turns positive, it is time to perform the time-dependent service or whatever function needs to be performed. These clocks, and their descriptions are:

1) RJQNT - running job quantum. The TENEX scheduling policy uses 5 queues with different priorities. A job is put on the highest priority queue (lowest numbered) to begin with. This queue has the smallest quantum. It is believed that interactive jobs will tend to migrate to this

queue. If a job shows need for greater amounts of computation (compute-bound characteristics), it is moved to successively higher numbered queues which are of lower priority but have larger quantums. This clock is loaded when the process is started on the processor. Initially it is given a full quantum from one of the priority queues. RJQNT itself will not time out unless the process uses up all its compute time. If, however, for some reason the process is temporarily blocked (e.g. the scheduler had to run some system service routine) then when it returns, the unused portion of the quantum is put into RJQNT. A compute bound process will use up the quantum on a high priority queue, and move to queues with succeedingly larger quantums but lower priorities. Heuristics have been added to allow movement back up to higher priority queues.

The quantums for the queues are:



2) ISKQNT - Running job remaining interval quantum. Under the reservation system, a job cannot have used more than a given percentage of the processor resource in a given interval of time. Currently this interval is set at two minutes. Notice that this restriction is over and above any TENEX quantum policies. Regardless of what RJQNT has in it, if a job has used up its allotted percentage in the two minute interval, it will not be granted any more processor resource. At two minute intervals, ISKQNT is reset to allow a new full percentage for the job.

3) ISKCLK - Related to the clock in 2), this clock is the principal clock of the reservation system. It has a time out every two minutes. At this time out, all the job clocks are reset to allow a full percentage quantum for the next two minute interval. The percentage is taken from JOBISK, while JOBQNT is loaded with the then computed quantum.

4) TTBTIM - This is a 15 ms clock that initiates two actions. The first is to check on the status of the IMP.

This is one of the time-dependent system services mentioned earlier. The other is to dispatch <object?> to the terminal handling routine that retrieves characters from the ring buffer loaded by the PDP11, etc. Note that this is a 15 ms clock while the forced control to scheduler is a 33 ms clock. If there are no blocks of processes, then the actions controlled by this clock will take place 33 ms apart. If there are blocks by processes, then the various clocks will be updated in shorter than 33 ms intervals (depending on how quickly the processes block) and the actions to be performed will be closer to the 15 ms time out period.

5) TIM2 - This is a 100 ms clock that periodically checks the status of the disk and magnetic tape.

Balance Set Control

The scheduler must implement a policy concerning which process, out of the many possible, will be allocated the processor. (According to the basic policy mentioned earlier in the description of the RJQNT clock.) It also must be aware of the state of core memory regarding

fullness. The word balance indicates an attempt to balance the number of processes with pages in core. If core gets too full, then a process may have to be removed to make room for one with higher priority. Thrashing occurs if the processes get swapped too much, while poor response results if certain processes hog the memory.

Starting or stopping processes involves overhead such as updating certain process-related clocks, changing status entries in fork tables, setting up the pager to accomodate this process, etc. These routines are part of the scheduler code.

Flies in the ointment

There are certain system quirks that deserve mention. Previous to this section, the functions of the scheduler have been given with a specified raison d'etre. The following is given as a "Well, that's the way it is."

1) NOSKED, OKSKED - Certain critical sections of monitor code are sandwiched between NOSKED, OKSKEDs. This prevents the scheduler from scheduling another process. This is because critical structures are currently being handled on

behalf of a single process. However, updating of the process clocks (RJQNT, ISKQNT, etc.) may take place.

2) Pager Simulation Code - If, when the clocked interrupt to the scheduler (the 33 ms clock) occurred, we had happened to be in the pager simulation code, we immediately return to the pager simulation code. Not even the process clocks (RJQNT, ISKQNT, etc.) are updated. We do, however, set a flag to tell the pager simulation code that there was an attempt to schedule, and want to do so as soon as we are out of the simulation code.

3) RSKED - Similar to the Pager Simulation Code situation, if there was a clocked interrupt to the scheduler (the 33 msec clock) when we were in a NOSKED situation, no scheduling decisions were allowed to be made , but process clocks do get updated. When the process comes out of the NOSKED situation, it then has the option to execute RSKED which will cause us to enter the

scheduler if there had been an attempt to schedule. Otherwise RSKED is like a no-operation (NOP).

4) Pager Trap Starting - A pager trap has been initiated for the current process, then we can be considered to be in a NOSKED situation. RSKED can similarly be used to enter the scheduler at a later opportune time. (Process clocks are updated similar to RSKED.)

5) INSKED - The clocked interrupt to the scheduler may have come while we happened to be in the scheduler already. The clocks are updated, and we continue the scheduling as if no interrupt had occurred. INSKED is a flag that tells us whether we are in the scheduler.

6) Priority Interrupt system - Somewhat hidden, but nonetheless vying for the processor resource, are various hardware interrupts. These are associated with various devices such as the disk, drum, displays, etc. Evidence of this shows up when we find that not only is NOSKED invoked, but the interrupts are turned off for some or all devices in some critical data structure handling.

System Clock Periods

1.	1 ms:	hardware clock
2.	1 ms:	TODCLK (to-date clock)
3.	15 ms:	TTBTIM (IMP check, terminal handler)
4.	16 ms:	60 cycle hardware clock
5.	33 ms:	Interrupt to scheduler clock
6.	100 ms:	TIM2 (check on disk, mag tape)
7.	100 ms:	ISKED flag; (max) time before wakeup tests performed.
8.	100 ms:	RJQNT queue quantum clocks (maximum for each queue)
	500 ms	
	2500 ms	
	4500 ms	
	14500 ms	
9.	1 sec:	RJTIM runnable job averages
10.	2 min * n%:	ISKQNT (per job) per cent of 2 minutes that a job under the reservation system can run within the 2 minute interval.
11.	2 minutes:	ISKCLK time to reset quantums for jobs under the reservation system

Scenario: A user process being scheduled, run, rescheduled, etc.

In the following scenario, we shall follow a user process as it is being scheduled, run, and rescheduled. The scenario is intended to illustrate the various workings of the scheduler and does not claim to show the typical behavior of the scheduler.

The process, being newly introduced to the system, is put on the highest priority scheduler queue (with a short quantum). It has a (maximum) specified amount of time (with respect to the reservation system) that it can run within the reservation system's two minute intervals.

The process being of high priority, it is decided by the scheduler balance set control that the process should be run. Then comes a series of page faults, as the process accesses virtual addresses that aren't in core. For a given page fault, the process will block. It will give the scheduler a test routine to check a word that will be set (by the page manager) when the page is read in. The scheduler then can update the process clocks, possibly having to do some terminal handling, checking the status of various devices, etc. When the scheduler executes the wakeup test for the process, and finds that the page is in, then the process is awakened, and available to run. It must wait for the scheduler to let it run, however. When it is again running, it is being interrupted (really quite unnoticeably to the process) each msec because TODCLK (to-date clock) is constantly

being updated. Assuming the process did not block by itself, the system clocks now indicate it is time to interrupt to give the scheduler control (this is the 33 msec scheduler clock). The scheduler finds it must do the terminal handling service, and perhaps some scheduler requests.

At this point we see that the process hardly ever uses the processor in an uninterrupted continuity. The scheduler regains control often and frequently. When we return to the scheduler, it always looks to see if some time-dependent service must be done.

Now we have a picture of a process running, interrupted at times by the scheduler to see if time-dependent services need to be done. If the process needs input, it will block until the input comes in. Then when the input comes in, it will be awakened again. If the process times out (i.e. runs out of its quantum) then it is placed on a higher queue with larger quantum but lower priority. It may also time out if it has used up its quantum with respect to the reservation system and can't run until the next two minute interval.

The process remains in core as long as there is enough core to accomodate other processes coming into the balance set. If, however, another process coming into the balance set is of higher priority, and there is not enough core to fit all of the processes' requirements with respect to the balance set, then it is possible that our subject process might be removed from the balance set. If it is the lowest priority among the processes in the balance set, it will be removed.

There are other intricacies that can be mentioned here, but it suffices to stop at this point. The reader can easily add other complications described in the previous sections.