Memorandum TENSYS 6

To: Ten-Sys Group

From: D. Murphy

Subj: Some Thoughts on Scheduling and Core Managing

Date: April 3Ø,1969

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

Note: Rather than representing a final design, this memo is one of a series which should evolve and converge to a complete design.

* * * *

During discussion of possible algorithms of core managing, it has become clear that there are two separate problems to be considered. The first is managing the pages within a process, such decisions as what pages to bring in when a process is about to be run, and what page to throw out if a process has exceeded its working set size and requires another page. The second problem is the management of the pages of a process as a whole, such decisions as what pages to throw out when core becomes full and which to retain.

The algorithms used to decide these two questions must be very different as will be shown. To decide the first question, a reasonable algorithm would use the ages of pages

of the process (where age is defined as the time since the last reference, or the time of the last reference or the like). To make space for a new page, the core manager should probably throw out the page least recently referenced. This is the sort of algorithm presently in use by the LISP system on the 94Ø.

In deciding the second question however, the age of the page is probably not a useful piece of informaion. To see why, consider the case of a process which has just exceeded a time quantum and therefore been dismissed. Although its pages have been referenced quite recently, they will not be referenced again soon (assuming there are other jobs to run), so the desirability of keeping them is fairly low. After a period of time, the scheduler will again decide to run this process. However, by this time, the pages of the process will all be quite old and they may have just been thrown out. Thus, it is not appropriate to extend the algorithm used for aging pages within a process to managing pages within the system as a whole. Therefore, it seems necessary to discover an algorithm that will provide for the appropriate movement of the page processes in and out of core. One such algorithm is discussed here.

Scheduler Operation

It appears that the management of pages must be tied closely

to the operation of the scheduler. To manage the usual class of interactive and compute jobs, the scheduler will have some algoritms which determine when a job shall run and for how long. Oe likely algorithm is to have a set of queue's numbered 1 through n in decreasing priority. Each queue would have associated with it a quantum of time, increasing from queue-level to queue-level by some convenient ratio. When a job makes its first request for CPU time, it is placed in the highest priority, shortest quantum queue. If it exhausts that quantum it is moved down to the next queue and given the correspondingly longer period of time. Whenever a job exhausts the quantum on a particular queue, it is moved down to the next queue.

The scheduler selects a job to run by scanning through the queues. It selects the highest priority queue on which there is a job ready to run, and if there is more than one runable job on this queue, it selects the job with the least remaining time on this queue. This algorithm may be modified by the existence of jobs with guaranteed CPU percentages and the like, but in general it provides a convenient way of scheduling jobs and providing reasonable console response time.

With this scheme, there are three situations which call

for the dismissal of a job. The first case is when a job exceeds its quantum. In this case, the job is moved from its current queue to the next lower queue and given the longer quantum for this new queue. If there are no other jobs on this queue and if no jobs appear on higher level queues, then, when the scheduling algorithm is again run, it will select this same job for CPU service.

The second case is the appearance of another job on a higher priority queue. In this case, the currently running job will be stopped but placed on the front of the queue to which it currently belongs. If the new job then blocks for I/O or computes until it falls to the queue level of the first job, the first job will again be run, completing the time quantum that it was originally given. In the third case, when a program must be dismissed for I/O wait, it is removed from the queues until its I/O activity has completed.

In the first two of these dismissal cases, the scheduler can make a reasonable estimate of how long it will be before the job is again run. In the first case, when the job overflows its quantum and is placed on the end of the queue, the length of time until it will be run again is determined by how many other jobs are on this same queue, and the expected arrival of jobs on higher priority queues

as determined by the general load on the system. Similarly, when a job is dismissed because a higher priority job has in fact been entered in the queues, the current job will be run again after a time no greater than the sum of the quanta for the higher priority queues.

Scheduling Core

We will now see how this scheduling algorithm will affect the management of pages in core. As stated above, the scheduler should be able to make a reasonable estimate of the length of time before any job in the system will be activated for running. It is precisely this information which the core manager must take into account when deciding which pages to remove from core. Obviously, if core becomes full and some pages are to be removed, those pages should be selected which will be needed furthest in the future. These will be pages of processes which are to be run furthest in the future.

For example, consider the case of three compute bound jobs, all of which are on the lowest priority, longest quantum queue. Job 1 will run for its specified quantum after which the scheduler will dismiss it and call for running job 2. At this point, there may be pages of all three jobs in core. However, pages may be needed for job 2 which are not in core, and the core manager may have to make

core space available to bring in these pages. At the start of the job 2 quantum, the scheduler contains the information that job 3 is to be run after one long quantum, and job 1 is to be run after 2 long quanta. Therefore, the core manager will elect to eliminate as many pages of job 1 as may be necessary to acquire the needed core. Since the choice of pages to eliminate is between job 1 and job 3, it is obviously better to retain the pages of job 3 since it is to be run immediately after job 2 is run. Examples of increased complexity could be shown, but in all cases it seems that the core managing algorithm should call for the elimination of those pages needed furthest in the future.

Summary of Scheduler Structure

As an aid in understanding the inter-relationship of the scheduler and core manager (perhaps "time-scheduler" and "core-scheduler" would more aptly suggest these functions), consider the following simlified scheduler scheme. It is not presented as a complete description, but rather as a general picture of one of several possible, practical schemes which could be adopted-completely, partially or as modified.

Assume first a multi-level queue structure as mentioned above. Shown below, is one set of possible queue and quantum values. These "typical" quantum values are ones that seem appropriate to a high computing power machine. Considering the extremes, it may be seen that these parameters will tend to give very fast response time to jobs which use little CPU time (one can do a lot in 4 milliseconds of PDP-1Ø CPU time), while minimizing overhead (swapping demands particularly) for long running compute-bound jobs.

This structure is not meant to suggest any particular implementation nor should it be assumed that there is one obvious implementation (e.g. a list structure of jobs on

each queue).

Queue Values

QUEUE LEVEL	QUANTUM	LENGTH OF RUN TIME WHEN ENTERING QUEUE
1	4 MS	Ø MS
2	32 MS	4 MS
3	256 MS	36 MS
4	2.Ø48 S	288 MS
5	16.38 S	2.336 SEC

Scheduling Procedure

1. Find highest priority non-empty queue
2. Run job on front of queue, set alarm clock to amount of remaining quantum.

Scheduler Operation - Events

1. Job exhausts quantum (alarm clock signal). Place job at end of queue n (n = m + 1, one higher than last queue, or highest queue if job was on highest queue). Give job full quantum for new queue. For core manager, calculate expected time to next running as:

T = ((number of jobs on Qn)-1)
(quantum of Qn)(load factor)

That is, this job will be run again after all other jobs on this queue have run their full quantum. The load factor is the ratio indicating how much real time it will take to provide one unit of compute time to jobs on this queue. It reflects the expectation that some jobs now dismissed for I/O will become runable and must be given preference over jobs on this queue. It will usually be greater than 1. It could possibly be less than one if there were no jobs expected to terminate before using their full quantum. It is approximately equal to:

AVNNJ * AVJRT

where

AVNNJ = average number new jobs/sec

AVJRT = average job run time before dismissing

Then enter scheduler.

2. Job dismisses for I/O. Remove job from Q, set alarm to go off when I/O is completed. For core manaer, if possible, calculate expected time for completion of I/O. Reasonable estimate can be made for drum, disc, paper tape, dectape, and probably magtape. Estimate might also be made for terminal (teletype). Otherwise, use default estimate somewhere between several seconds

and infinity. Enter scheduler.

3. I/O alarm (some job formerly blocked for I/O is now runable). Place currently running job on front of queue m (current queue), save amount of remaining quantum. For core manager, calculate expected time to next running as:

AVJRT*LDFCTR

That is, the new job is expected to run some average length of time. It can run a maximum time equal to the sum of the quanta for the queues less than m. The load factor is again included to reflect the expectation that still other jobs will become runable while this job is using its AVJRT.

The job for which the I/O alarm was given, is now runable and must be placed somewhere in the Q's.