To: Distribution

From: Steve Webber

Subject: New Multics Scheduler Proposal

Date: 04/19/74

Overview

As part of the Multics performance analysis effort, the idea of a new scheduler has been raised more than once. This memo is being written to describe a scheduler which hopes to solve many of the problems we can recognize as well as provide for several new features which have been proposed in the past, but which have little bearing on the performance issue itself. In particular, it is claimed that this proposed scheduler will:

1. provide more efficient scheduling of processors and main memory

2. provide more information to system analyzers thru better and more relevant meters

3. provide for the capability of process "priorities" which are assigned at log-in time and which govern the price of the process as well as the response of the process relative to others on the system.

4. provide for system administrator controlled process classes such that the processes of a process class can be guaranteed a certain percentage of the CPU capacity.

5. provide for dynamically changeable priority characteristics under system administration control.

6. provide for the capability of system software performing automatic tuning of the system.

7. provide for a realistic, useful interface to page control to communicate necessary information.

What is currently called the traffic controller is subdivided into the following areas:

1. process management and synchronization,

2. process scheduling, and

3. process dispatching.

The process management features of the traffic controller consist of the callable interfaces to block, wakeup, stop and the like. The process scheduler function assigns a process to a given queue and awards the process a given "quantum" of time for its next execution period. The dispatching function of the scheduler does process switching. It awards "eligibility" and chooses which process is next to be given the processor.

Process Management

The process management features of the traffic controller are not being changed by this proposal except for the addition of a few administrative interfaces for controlling process priorities and scheduling tables.

Scheduler

The scheduler is to be completely replaced by a table driven mechanism that attempts to guess a process's future behavior and then award the process relative priority with respect to the other processes on the system.

There are three decisions the scheduler must make about a process in order for the dispatcher to decide when to run the process. The first is the relative delay the process will observe before it is next run; the second is the quantum the process will be allotted, and the third is a measure of the core requirements that the process will be allowed to use.

Queues

The relative delay is implemented via a series of queues. There are two classes of queues in the system. The first, the priority queues are an ordered sequence of, say, N separate queues each managed individually on a round robin bases. If a process is assigned to one of these queues, it will be threaded in at the tail of the queue and will not be run until after it trickles up to the head. If a process is not assigned to a priority queue it will be assigned to one of M special queues. As in the priority queue case, a process is threaded to the tail of a special queue at scheduling time. The difference between the priority queues and the special queues will be described when the dispatching mechanism is given. The only way a process will be assigned to a special queue is if the process has been awarded (penalized) by the system administrator the appropriate special queue attribute, typically at process creation time but not restricted to it. If a process is not special in this sense, the scheduler uses its heuristics to select one of the priority queues and threads the process to the tail of the selected queue.

Queue Selection

The queue selection process consists in evaluating a given function with several inputs based on the past performance of the process. In particular the inputs under consideration are:

1. estimate of previous working set
2. previous quantum
3. interaction behavior
4. time weighted average of mean time between page faults
5. previous core use limits

These inputs will be distilled into the two variables below:

1. quantum characteristics (Q)
2. paging characteristics (P)

The actual technique of mapping of the 5 variables into the 2 variables is not described here (it has not been chosen yet) but it will supposedly be very simple, easy to compute and easily (although not dynamically) changed.

The two measures of a process's recent behavior (Q and P) are used to answer the three scheduler questions. This is done by a simple table lookup in 3 two-dimensional arrays which yield as output: the new queue number, the new quantum and the new core use limits. The diagram below shows how the array might be built for queue number selection.

Q \ P	4	8	16	64	128	256	512
0	1	1	1	1	2	4	6
1/2	2	2	2	3	4	5	6
2	2	3	3	4	5	6	6
4	2	4	5	5	6	6	6
8	2	5	6	6	6	6	6

N=6

queue selection

Note that for interpretation of the table the Q should be thought of as the previous quantum modified somewhat by the 5-to-2 distillation. Similarly the P should be thought of as the previous working set estimate.

Quantum Selection

The diagram below shows a typical quantum selection table.

Q \ P	4	8	16	64	128	256	512
0	1/2	1/2	1	4	8	8	8
1/2	1/2	1	2	8	8	16	16
2	2	2	4	8	12	16	16
4	4	4	6	12	12	16	16
8	8	12	12	12	12	16	16

N=6

quantum selection

Core Use Limits Selection

The core use limits decision is analogous and would select minimum and maximum page pool sizes for the given process. (See MTB-XXX).

Note that it is intended that these selection tables be changeable while the system is running, both from explicit system administrator request and from dynamic system overload or underload detection mechanisms.

Dispatcher

The dispatcher function of the traffic controller is invoked whenever a processor is made available. The dispatcher selects a process to run on the processor unless no acceptible process exists in which case the dispathcer runs an _idle process_.

Input to the dispatcher consists of the (ordered) priority queues, the (unordered) special queues, process characteristics and variables for each process in the queues and the _queue selection list_.

Queue Selection List

The queue selection list consists of an apparently random sequence of queue numbers from which the dispatcher will select another process to run. This list is inspected whenever the dispatcher can not run any of the _eligible_ processes (i.e. those processes which are allowed to compete actively for core memory). The queue selection list index points to the next entry in the queue selection list (with wrap-around). If the process at the head of the queue selected by the queue selection list can not be awarded eligibility because of core considerations, the dispatcher will run an idle process. Otherwise it will award eligibility (thread the process into the eligible queue) and initiate the _loading_ of the process, i.e. getting enough of the process into core so that the process can page in the rest of what it needs.

The awarding of eligibility is done only if there appears to be enough core available to run the new process. This is determined by comparing the sum of the minimun core pool sizes (see MTB-XXX) with the amount of core that is in the paging pool and determining if the new process's minimum core pool size will cause an overflow.

Some Observations

One of the prime reasons for rethinking the scheduling is to attempt to reduce thrashing. Thrashing seems to occur primarily under heavy loads and is caused primarily because the degree of multiprogramming (i.e. the current number of eligible processes) is too high. The current system runs with a fixed eligible list size of 6 and with current working sets we have a great deal of thrashing.

The new proposal attempts to attack this problem at two stages. First, by reintroducing the variable eligibility which we once had, we can dynamically adjust to local changes in system load and avoid too high a degree of multiprogramming. Second, by using the queue selection list technique, we are fairly safe in assuming that 2 large jobs will not both be run at the same time (the large jobs would supposedly be in queues which are rarely selected) and hence we need not fear giving these large jobs a large quantum. The advantage of giving a large job (larger working set job, that is) a large quantum is that system overhead is reduced because we are not continually required to reestablish the working set of the large job whenever it wants to run.

After what the superviser considers an interaction takes place, the given process will be given special threatment by the scheduler. There are several options but a reasonable one would be to assign the process to the tail of the highest priority queue and somehow choose an appropriately small initial quantum.

Meters

One of the meters that will be kept for each process (and for the system) is the mean (virtual) cpu time between page faults. This meter is useful both in scheduling a single process as well as in testing the system for underload or overload.

Other meters that will be taken are

1. average (virtual) CPU time between interaction for a process

2. the average quantum selected by the system as a function of queue size

3. the mean (virtual) CPU time between page faults as a function of working set size and as a function of core use limits

4. average response time after an interaction.

The current plan is to attempt to implement this new shceduler (with appropriate modifications resulting from reader feedback) and to run several test systems using all the available metering tools we have. If the new scheduler seems to be better and easily workable an MCR will be submitted for the installation of the scheduler. Until that time the study work will be done under the authority of an MCR requesting the study of the new scheduler alone.