To: MTB Distribution

From: R. Mullen, T. Casey

Date: 19 May 1975

Subject: Priority Scheduler

INTRODUCTION

This document describes the functions and proposed implementation of a scheduler for Multics which will allow more flexible administrative control of the allocation of the cpu time resource to system users and groups of users.

It is not an objective of this proposal to attempt to achieve greater throughput in any numerical sense. However, it is an explicit objective that throughput of jobs deemed most valuable by a system administrator will be increased. To that extent, the value of Multics as a computer utility is enhanced. Of course, every effort will be made to ensure the efficiency of the design and implementation of the priority scheduler.

THE PROBLEM

Currently, the Answering Service provides a mechanism (load control) for classifying users into groups, and giving each group a specified share of the system (by limiting the number of users from each group that may be logged in concurrently).

However, except for the setting of the per-process parameter, timax, no control over the rate of consumption of cpu resources by any user or group of users is provided. (Briefly stated, there is a parameter, ti, associated with each process, which is roughly proportional to the amount of cpu time used by the process since it last interacted. If the value of ti for a process ever exceeds timax, it is set to timax. The process with the lowest value of ti is always selected for eligibility.) In practice, the considerable advantage given to a process by a lower-than-normal value of timax has prevented all processes but the Initializer (and sometimes Backup) from being given timax's lower than the default value.

THE SOLUTION

This MTB proposes that the scheduler allow the grouping of processes into work classes, and provide each work class with a guaranteed percentage of available cpu time. Conceptually, each work class will be assigned a virtual processor of

---

administratively defined computational power, available to members of the appropriate work class on demand. Any cpu time not needed by a work class will be made available to other work classes, and cannot be reclaimed at a later time. In this respect each virtual processor is like a real processor: time unused is time lost forever.

In its idealized form, the scheduler proposed here provides each work class with a specified computational power on an instantaneous basis. The idealized scheduler has a time constant (or integrating time) approaching zero seconds. The service and function provided by the idealized scheduler are known constants, not subject to being bent out of shape by previous transients in per-workclass loads.

The actual scheduler will for reasons of system efficiency, scheduler efficiency and response, necessarily have a time constant on the order of several seconds. As an example, consider the time constant required to smoothly provide service to a work class which has been assigned 20 percent of a single cpu configuration and whose members are generally provided with an eligibility quantum of 2 seconds. If the scheduler functions correctly, some process in the work class will be given a two virtual second quantum every 10 virtual seconds, or approximately every 20 real seconds. This implies that in some way the scheduler must be integrating over the past 20 real seconds for such a work class. Averaging over a considerably shorter period would require significantly shorter quanta and result in increased scheduler and paging overhead. Averaging over longer periods of time moves away from the idealized scheduler and toward a scheduler whose behavior is more dependent than necessary on the past history of the system.

The ability to limit the number of processes in each work class is clearly desirable, if not an absolute necessity, and the ability to assign each process to a specific work class is obviously needed.

To have two separate and independently-functioning mechanisms for classifying users into groups and limiting the number from each group that may be logged in concurrently is at best unnecessary, and at worst, confusing and full of hidden problems.

Therefore, there must be a close relationship between work classes and load control groups, and a single algorithm must be used to determine a process's membership in both classifications. For example, there could either be a one-to-one correspondence between work classes and load control groups, or else the work class of a process could be a function of its load control group, with possibly more than one load control group belonging to one work class. We have chosen the latter, more general, alternative.

It will be possible for the system administrator to specify the number of work classes (a limit of 16 will be imposed by the scheduler), and the guaranteed percentage of each work class.

The administrator will be able to define the membership of each work class. It will be possible to define such work classes as: all IO daemons, the Backup daemon, all users on a certain project, or one individual user. In each of those groupings, it will be possible to assign absentee and interactive processes either to the same or to different work classes.

The set of work class parameters, and the membership of each will be able to be changed automatically (at each shift change) and manually (by the system administrator, who may install a new table at any time). Thus, the work classes of existing processes can change.

HARDCORE SCHEDULER

The new scheduler will maintain an eligible queue consisting of eligible processes only and will manage 16 ready queues, one for each work class. Each ready queue will be managed just as the non-eligible portion of the current ready queue is managed --- that is the queues will each be internally sorted by ti values and favor the most interactive users within the work class. The current method of maintaining a ready queue is chosen for the new scheduler for three reasons:

1. It is response oriented, and in fact has been proven to provide the minimum mean respose time.

2. If such a queue consists of processes all with ti = timax, the the queue is largely run as a pushdown stack. This leads to very desirable paging behavior in that the most recently run process (the process most likely to have its working set still in core or on the paging device) will often be the next process to be run.

3. Use of already existing code will simplify the implementation effort required.

To contain information pertaining to each work class, tc_data will contain 16 work_class_table_entries (WCTE's). Each WCTE will contain a thread-word for accessing the members of the work class which are ready, and all parameters and metering data relating to the work class. This will include the total amount of virtual cpu time used by the work class, the total number of times eligibility was granted to a member, the fraction of virtual cpu time which the work class is to receive, and the response time seen by its members.

The actual algorithm used to enforce the proper sharing of the cpu resource will be as follows. Imagine the existance of a system virtual clock which increments as virtual time is used by non-idle processes. Imagine also that each work class has a store of credits (in units of microseconds) which is continually growing at a rate proportional to the speed of the virtual clock multiplied by the fraction of cpu resources which the work class is to receive. Suppose further that the store of credits for the work class is decremented as members actually consume virtual cpu time. Clearly it is undesirable to allow credits to build up indefinitely for a work class with no processes ready, so a maximum value is set on the number of credits which can be accumulated. In addition the value is restricted from ever becoming negative. The algorithm for chosing the next work class from which to choose a process to which to award eligibility may then be as simple as choosing that work class which has accumulated the maximum number of credits.

A worthwhile refinement would be to choose the work class for which the ratio of the number of credits to the quantum to be awarded (ie. to the top member of the given ready queue) is a maximum. This tends to favor the prompt scheduling of the most interactive users across all work classes. It does not cause non-interactive work classes to fall far behind since eventually the interactive work classes choke off. This is because they are temporarily using credits faster than they are gaining them, and will eventually have a ratio which is arbitrarily low --- and not be chosen.

It follows that the maximum build up of credits to be allowed must be greater than the maximum quantum allowed. It should probably be at least double that amount.

The computation required for such an algorithm will amount to about 300 microseconds per eligibility granted, less if fewer than 16 work classes are defined. If eligibility is awarded 10 times per second (a high figure) on a one cpu configuration, the loss in system throughput may be about .3%. This is somewhat reduced by the fact that all sorting operations into the ready queue will be replaced by sorts into shorter queues.

HARDCORE INTERFACE

The interfaces to the hardcore scheduler will be the following:

1. A gate to define (or redefine) the set of work classes and their guaranteed percentages of cpu time. This gate is tentatively called hphcs_$define_work_classes. The target of this gate will be a new procedure (tc.pl1) which will check the consistancy of its arguments, use existing subroutines to wire and mask, and lock the APT before modifying the work class table. Because this procedure will

not be heavily used it will call wire_proc$wire_me rather than being permanently wired. It will be illegal to undefine a work class that currently has processes in it. If that is attempted, the processid and work class number of one of the "offending" processes will be returned, in order that appropriate action can be taken.

2. A gate to reassign one existing process to a different work class. It will refuse to change the work class if the new one is not defined. It is tentatively called hphcs_$set_process_work_class. The target of this gate will be pxss$set_work_class.

3. An additional parameter in the create_info structure passed to hphcs_$create_proc: the initial work class. It will be illegal to specify a work class that is not defined. It will be necessary for act_proc, the target of hphcs_$create_proc, to call pxss$set_work_class, to insure that the work class being assigned to the new process currently exists.

A primitive to simultaneously redefine the work classes and reset the work class of each process is neither required by logical considerations nor justified by efficiency considerations. Furthermore such a primitive would not be able to handle an arbitrarily large number of processes.

In order to redefine the work classes in the general case, it will be necessary first to define a transitional set of work classes and percentages (including both old and new work classes), then to reset the work class of each process to the new value, and finally to define the new set of work classes. A procedure to do this will be implemented in the answering service.

SUMMARY OF CURRENT LOAD CONTROL SOFTWARE

Since work class membership will be a function of load control group membership, work class definitions will be stored in the MGT, and the implementation of the answering service and administrator interface to the priority scheduler will consist mainly of modifications to the current load control software, a summary of that software, as it now exists, is presented here.

Load control group membership is specified in the SAT entry for each project. In addition, each project's SAT entry contains an absolute max user figure for that project that is enforced independently of the load control group limits.

Absentee and daemon processes are not subject to load control. They are always logged in on request. They are assigned to the load control group corresponding to their projects, but their group membership is ignored by everyone.

Load control groups are defined in the master_group_table (MGT), which is a binary table maintained by an editor (ed_mgt), and is not subject to the install discipline. (1) This table contains limit parameters for each group, set by the system administrator, and it is also used to hold current load figures for each group, during a session.

The group limits are defined in units of user weight, rather than number of users. (There are, however, limits in units of users, for the system as a whole, and for each project.) By default, each user has a weight of 10, so max_units is ten times max_users. Weight is a function of the process overseer, and is determined by an array of weights kept in the SAT header.

There are two sets of limit parameters per group, one used to compute primary_max_units, the other, to compute absolute_max_units. Each set contains three parameters: a constant (which may be zero), and a numerator and denominator of a fraction. The formula for absolute_max_units for a group is:

```
absolute_max_units = absolute_constant +
(available_max_units*absolute_numerator)/absolute_denominator
```

where available_max_units is the system_max_units less the units used by the absentee and daemon processes who are not subject to load control. The formula for primary_max_units is the same, but using primary_constant, primary_numerator, and primary_denominator.

These calculations are performed for all groups each time a user attempts to log in, so changes to units used by absentee or daemons, changes to system_max_units, or changes in the MGT made by the system administrator are all taken into account immediately.

The system_max_units figure is either:

1. taken from the SAT header, for a special session, or

2. set by the operator, using the maxu command, in which case automatic maxunits setting is turned off, or

3. set automatically at each shift change and whenever the maxu auto command is given by the operator. The automatic setting looks up the current shift and configuration in the config array in installation_parms, and chooses the corresponding

---

(1) The install discipline is a method used for installing certain critical tables, whereby the Answering Service installs the table, when requested by a system or project administrator, ensuring that the Answering Service will not attempt to reference the table while it is being updated.

values for: system_max_units, max_absentee_users, max_absentee_queue, and response_high and response_low. (The latter two figures are used by the load leveler (when it is enabled by the maxu level command), which readjusts system_max_units at every 15-minute accounting update, to keep response between the high and low figures.)

The load control decision is rather complex, when special privileges like guaranteed login, the nobump attribute, and protection from preemption for a specified grace time are taken into account. But basically, if the system is full (as measured by system_max_units or system_max_users) then someone must be bumped or else the user is refused login. If the system is not full, but the group or the project is full (as measured by the group's absolute_max_units or the project's max_users), then someone in the group or project must be bumped, or else the user is refused login. If the group's primary_max_units are all allocated but its absolute_max_units are not, then the user is logged in as a secondary user, subject to preemption. Secondary users (in any group) are the first to be bumped (oldest first) when some primary user wants to log in, followed by primary users (in the same group) whose grace time has expired, followed by practically anybody, when a user with the guaranteed login attribute is trying to log in.

The load control group membership of a process never changes, but both the proportion of the available_max_units that each group gets, and the number itself, can vary with the available_max_units (which varies with shift, configuration, and absentee and daemon load), because of the max_units formula described above.

NEW ANSWERING SERVICE AND ADMINISTRATOR INTERFACE

The MGT will be reformatted to hold work class definitions as well as load control group definitions. Since there will be a maximum of 16 work classes, but there is currently no restriction on the number of load control groups, the new MGT will consist of a header, followed by a fixed-length array of 16 work class definitions, followed by a variable-length array of load control group definitions. The header and the load control group definitions will remain essentially unchanged, except that each load control group definition will contain two additional 8-element arrays, specifying the work classes to which interactive and absentee users in that load control group belong on each shift.

One or more load control groups can belong to each work class. The max_users and max_units figures for each work class will be the sum of the corresponding figures for the load control groups that make up the work class. The work class maxima will not actually be computed and stored anywhere by the answering service, but they will be displayed by ed_mgt to assist the

system administrator in assigning reasonably consistent percentages to the work classes and max user and unit figures to the load control groups. The normal operation of load control, as described above, will limit the number of processes in each work class.

The ed_mgt command will be modified to be able to store and modify the work class parameters, to verify, on request, the correctness, reasonableness, and consistency of work class parameters and the corresponding load control group definitions, and to print work class definitions and a cross reference showing the correspondence between load control groups and work classes. The changes to ed_mgt are described in detail in a later section.

The install command will be modified (and a new procedure, up_mgt_, will be written) so that the MGT can be installed while the system is up and users are logged in. up_mgt_ will have to, in general, reset the work class of all existing processes to the work class specified in the newly-installed MGT.

The answer table (and daemon and absentee user tables), and the create_info structure passed to hphcs_$create_proc, will have a new variable, work_class, added.

load_ctl_$load_ctl_init will call hphcs_$define_work_classes, to define the work classes to be used by the scheduler, during answering service initialization. (This call must be made before the daemons are logged in.)

load_ctl_$set_maxunits, which is called at each shift change (as well as during the second half of answering service initialization, and whenever the operator command "maxu auto" is given) will redefine the work classes, as specifed for the current shift in the MGT, and will reset the work class of each existing process as required.

Since the function of redefining the current work classes and resetting the work classes of all existing processes must be performed both at shift change and whenever a new MGT is installed, it will be implemented as a separate procedure, called in both situations.

To support the assignment of work classes on the basis of person as well as project, the SAT and PDT, and the procedures which compile, edit, and install them, will be modified to allow a load control group to be specified for an individual user rather than just for a whole project.

A new attribute, igroup (individual group), will be created. When that attribute is on in the SAT entry for a project, it permits the load control group for users on that project to be specified in the PDT entry for any user on that project. (If that attribute is not on in the SAT entry, then all users on the project will

continue to belong to the load control group specified in the project's SAT entry.) When the lgroup attribute is on in the PDT entry for an individual user, it indicates that a load control group is specified in that user's PDT entry. (If lgroup is not on in a user's PDT entry, that user will continue to be a member of the load control group specified in the project's SAT entry.) The name of the individual user's load control group will be stored in a presently unused pad field in the PDT entry. This, plus the use of lgroup in the PDT entry as a positive indication that a group is specified, will allow this change to be installed without requiring that any existing PDT's be reformatted or reinstalled.

lg_ctl_ will be modified to assign a load control group on the basis of person as well as project, as described above.

load_ctl_ will be modified to use the load control group of each process to assign it to the work class specified in the MGT for absentee or interactive users in that group, on the current shift. (Daemon processes will be treated as interactive, for the purpose of work class assignment.) The assigned work class will be stored in the answer table (or absentee or daemon user table) entry for the process.

cpg_ will copy the work class from the answer table entry into the create_info structure, before calling hphcs_$create_proc.

INSTALLATION PROCEDURE

The hardcore system containing the priority scheduler, and the answering service containing the above modifications, can be installed in either order. If hphcs_$define_work_classes is not called, a single work class (work class 1) will exist by default, and will have a percentage of 100%. The version number of the create_info structure will allow act_proc (the ring zero procedure called via hphcs_$create_proc) to determine if the new version, containing the work class, has been passed. If the old version of create_info is used, act_proc will assign processes to work class 1 by default. This allows the hardcore system to be installed first.

The new answering service will check for the old format MGT, and if it is found, none of the new gates will be called, and every process will be assigned to work class 1, independent of their load control group membership. Further, a switch in the reformatted MGT, settable by ed_mgt, will allow this mode of operation to be specified by the system administrator after the MGT is reformatted. Finally, a check for the existence of hphcs_$define_work_classes will be made during answering service initialization, and if it is not present, the old mode of operation will be used. This will make the new answering service compatible with older system tapes. It will also cause the system to run as it does now, with only one work class, when both

the new answering service and the new hardcore system are installed. The new scheduler will not be turned on until the MGT is reformatted, and the system administrator explicitly enables it.

The system administrator will, of course, be informed of all this in release documentation. The first time he uses the new ed_mgt, it will recognize the old format MGT, reformat it automatically, define a single work class (work class 1) with a percentage of 100%, make all load control groups members of it, and then invite the system administrator to define more work classes and reassign the load control groups to them. It will not be required that the administrator do so, but ed_mgt will keep reminding him, every time he uses it, until he does.

Since the MGT will now be subject to the install discipline, the reformatted copy can not be put back in >sc1. When the "w" request is given, the reformatted MGT (named MGT.mgt) will be written in the working directory of the administrator (which should be >udd>sa>admin). The administrator will be told about this by ed_mgt. Except for the instance when the MGT is reformatted, the edited MGT will be written back into the input MGT, as is done now. However, the system administrator will not be editing the >sc1 copy any more. As a convenience, after writing the edited copy back into the original, ed_mgt will always ask "Install?", and if the answer is yes, it will invoke the install command. The administrator will of course be able to invoke it directly.

The installation procedures described above will make testing and initial installation of the system very convenient, and it will also allow the system administrator at each customer site to turn on the priority scheduler at his convenience, instead of forcing him to define some (possibly ill-considered) work classes, just to get the new system release to run.

CHANGES TO ed_mgt

Summary of Current ed_mgt

The following summary describes only those features that are being changed. The MGT, as seen by a user of ed_mgt, is an array of load control group definitions. The find (f) request positions the current pointer to the specified group. The next (n), top (t), and - (minus sign) requests move the current pointer forward or backward in the array. The change (c) request changes parameters of the current group. The print (p) request prints all information about the current group. The pall (pa, p*) request prints all information about all groups. Only the find and change requests take arguments. Their formats are:

```
find      <group name>
change    <code> <new value> [<code> <new value> ...] *
```

where <code> is the name of the parameter to be changed. Typing these requests without arguments causes ed_mgt to prompt the user for them. The change request puts ed_mgt into the change subcommand, in which <code> <new value> pairs are accepted. The asterisk at the end of the line exits from the change subcommand and returns to ed_mgt request level.

Summary of New Features

The find request will be modified so that a <group name> consisting of one of the integers 1 through 16 will refer to the corresponding work class.

The next, top, and - requests will be modified to print the name and type of the entry being pointed at after the pointer is moved.

The change request will be modified so that the set of codes accepted will be different, depending on which type of entry (work class or load control group) the current pointer is pointing at. New codes and other arguments will be added, to allow parameters of work classes, and the work class membership of load control groups, to be edited.

A new request, global_change (gc), will be added, to allow the same (set of) change(s) to be made to all work classes or to all load control groups.

A new request, verify (v), will be added, to request that ed_mgt check all the work-class-related parameters in the edited MGT, and report any errors or warnings that would be received if the MGT were to be installed.

The print and pall requests will be changed to print the new parameters, and the pall request will take arguments, requesting that all work classes, or all load control groups, or both, be printed, or that a cross reference of work classes and load control groups be printed.

Detailed Descriptions of New Features

Two new formats for the change request will be added:

change <code> [<shift specification>] <one or more values>

change <code> [<shift spec.>] <interactive|absentee> <value(s)>

The first is used when editing work class parameters; the second, while changing the work class membership of a load control group.

The following new <code>'s can be used in the above requests:

```
percent     (pct, %)
absentee    (abs)
defined     (def)
work_class  (wc)
```

The first three are used with the first form of the change request, to edit work class parameters. The fourth is used with the second form, to change the work class membership of a load control group.

The format of the <shift specification> is the word "shift" followed by a shift number or a range of shift numbers (two numbers separated by a hyphen, the second greater than the first):

```
shift <number>|<number>-<number>
```

The shift specification is optional. If it is omitted, the default is a function of how many values are supplied. If one value is supplied, it is assigned to all 8 shifts. If a list of values is supplied, they are assigned to shifts 0, 1, ..., respectively, and shifts for which values are not supplied are not changed.

The following relationship exists between the shift specification and the list of values: when a range of shifts is specified, a single data value is expected, and is assigned to all shifts in that range; when a single shift is specified, one or more values may be supplied, and they are assigned to shifts, in order, starting with the specified shift.

<interactive|absentee> can be:

```
interactive (int)
absentee    (abs)
```

This argument is used when setting the work class of a load control group. Separate work classes may be specified for interactive and absentee processes in the load control group, on each shift. If this argument is omitted, but the work class value(s) are given, the default is interactive.

The work class parameters "defined" and "absentee" can have values of "off" or "on" (or "0" or "1"). They are per-shift switches, that indicate respectively, whether the work class is defined on the given shift, and whether absentee processes are permitted in it on that shift.

The format of the global_change request will be:

```
gc <type> <arguments acceptable to the change request>
```

where <type> can be:

```
load_control_group (lcg)
work_class         (wc)
```

The effect of this command will be to make each change (specified by a change subcommand) to all entries of the specified type.

The format of the pall (print all) request will be:

```
pall <type>
```

where <type> can be:

```
load_control_group (lcg)
work_class         (wc)
cross_reference    (cref, xref)
```

If <type> is omitted, the default will be to print all three sets of information.

Examples:

```
change % 10 .
change % shift 0 10 10 10 10 10 10 10 10 .
change pct shift 0-7 10 .
```

The above are equivalent ways of assigning 10% to the current work class on all shifts.

```
change % shift 1 50 % shift 2-4 30 .
```

The above request is equivalent to the following two requests:

```
change % shift 1 50 .
change % shift 2-4 30 .
```

```
c wc int shift 1 3 wc shift 2-4 int 2 wc abs 1 .
```

The above sets the interactive work class of the current load control group to 3 on shift 1 and to 2 on shifts 2-4, and the absentee work class on all 8 shifts to 1. Notice that the shift specification and the interactive|absentee indicator may appear in either order.

```
gc wc defined shift 0 off defined shift 5-7 off .
```

The above will set all work classes to undefined except on shifts 1-4. This would be useful at an installation where only shifts

1-4 were in use, to simplify the output of the print and pall commands, since information about undefined shifts is not printed.

Note that, in the examples, a period is used to terminate the change request lines instead of an asterisk. In the new ed_mgt, a period will be accepted for that function, in addition to an asterisk, for compatibility with other Multics editors.

The examples show all required arguments supplied on a single line. ed_mgt will prompt for missing values. The example above, in which the percents for shifts 1 and 2-4 were changed, would look like this if the user typed only what was requested (! indicates prompting messages):

```
!     type:
      change
!     code:
      %
!     shift:
      1
!     value(s):
      50
!     code:
      %
!     shift:
      2-4
!     value:
      30
!     code:
      .
!     type:
```