SUPERCP

A High-Performance Version of CP-67

Robert g. Munck
Robert L. Abraham
Steven T. Carmody
Richard J. Harrington
Richard M. Kogut
Edward D. Lazowska

The Computing Laboratory
Brown University
Providence, R.I. 02912

January 9, 1972

ABSTRACT

SUPERCP is a redesigned version of IBM's CP-67/CMS providing improved system performance and better administrative control over the allocation of resources. It replaces the standard CP-67 dispatcher and paging system with a table-driven scheduler and a different configuration of dispatching queues, and with a "slot-sorting" pager. The resultant system allows the specification of the machine requirements and political importance of each user, giving greater flexibility in the administration of the system. The present paper describes the algorithms and implementation of SUPERCP.

Key Words and Phrases

Time-sharing, multiprogramming, virtual machines, CP-67, schedulers, table-driven dispatchers, paging systems, slot-sorting, working set, interactive systems, batch systems.

CR Categories

4.32, 4.30, 2.43

INTRODUCTION

SUPERCP is an extension of IBM's CP-67/CMS (Control Program - 67/Cambridge Monitor System) with the intent of providing better system performance and increased flexibility in the allocation of resources. It was developed at Brown University during the first half of 1971 as an outgrowth of the course Applied Mathematics 103, "Design of Multiprogramming Systems."

CP-67 is a time-sharing system for the IBM/360 model 67 which supports a number of "virtual machines," simulations which appear to be real /360's to their users. This paper must perforce assume a familiarity with CP-67/CMS as detailed in the IBM SRL "CP-67/CMS User's Guide," GH20-0859, and a limited familiarity with the internals of CP-67 as detailed in "CP-67 Program Logic Manual," GY20-0590.

CP-67 is used at Brown for three major purposes: interactive programming using CMS (five to ten virtual machines during average periods), batch processing using O.S./360-P.C.P. (two or three virtual machines), and interactive problem solving using APL (a single virtual machine running DOS/APL, five to ten users). Batch is the traditional method of computing at Brown, and still accounts for about 75 percent of computer use, but CMS and APL are both growing in usage, with APL made available to the entire student body in the fall of 1971.

These three methods of computing put quite different demands on the computer in terms of the characteristics of the service that they require. CP does not provide for different "types" of users, and so could not provide ideal service to any one of the types in the presence of the others. In addition, Version 3 of CP did not provide for some users being more important than others, in the way that a batch machine is more important than a CMS machine or a professor is more important than a student (Version 3.1 does provide a priority scheme). There was therefore little flexibility possible in the management of the system.

The different service requirements of the various virtual machines can be characterized as a two-parameter committment of service. The first parameter is the percentage of total available CPU time which the machine requires. This figure would be high for batch machines (for example, .25), medium for the APL machine (.10), and low for most CMS machines (.05). The second parameter is the time frame over which the first parameter is to

be applied, that is, how often the system must check that it is meeting the CPU percentage requirement. This figure, called the "slot length," may be thought of as the response time desired, and would therefore be low for CMS machines (for example, one second), very low for the APL machine since it must respond to several users (.5 seconds), and could be quite high for the batch machines (20 seconds). A machine with service parameters of (.02,1), for example, would be guaranteed at least 20 milliseconds of CPU time within 1 second of requesting it, and one with parameters of (.25,20) would be guaranteed five seconds of CPU out of every twenty, but may not get the CPU at all until fifteen seconds after requesting it (and should thereafter get 100 percent of the CPU for five seconds).

These parameters are thought of as being sufficient to satisfy the user under all conditions, that is, an important user has no right to expect more than his committment, even if the other users on the system are less important than he. However, when the system is unable to meet its total committment, it should use an external priority, the "political factor," to determine which committments it should try hardest to meet. Each virtual machine should therefore have associated with it in the directory three numbers, the CPU percentage, the slot length, and the political factor.

DESIGN GOALS

Based on the above requirements and the general principles of multiprogramming systems design, the following design goals were set for the SUPERCP project:

The Dispatcher

1). Support explicit requirements for CPU percentage and response time.

2). Use an explicit political factor when not meeting these requirements.

3). Follow Denning's suggestion in "A Working Set Model of Program Behavior"(1) to attempt to run only those tasks which have their "working set" in core.

4). Make the dispatching algorithm easy to change within wide limits, to allow for tuning, experimentation, and unforeseen requirements.

The Paging System

5). Provide "slot sorting" of page requests and chaining of drum I/O operations to reduce interrupt traffic.

6). Provide fully dynamic drum allocation to eliminate the possibility of virtual machines being unfairly relegated to disk-only paging.

7). Be able to handle large clusters of page requests efficiently and quickly, as the dispatcher will be requesting the paging-out of all pages belonging to a virtual machine at once.

8). Maximize use of the drum by migrating lightly-used pages from drum to disk.

Miscellaneous

9). Make the conversion of a CP system to SUPERCP as easy and release-independent as possible. Use either UPDATE files keyed to the original CP source or complete replacement of modules, with the replacement being transparent to the rest of the system.

10). Put sophisticated measurement and evaluation tools into the original design, and allow the system itself to use them in adjusting its algorithms.

11). Build debugging tools into the system.

12). Provide flexible shared segment capabilities.

These design goals were met by the replacement of the CP Dispatcher with a "table driven" dispatcher and queue system quite similar to those in TSS(2), and a replacement of the CP paging system by one similar to that found in TSS and the University of Michigan's UMMPS(3).

THE PAGING SYSTEM

THE PAGING DRUM HANDLER

The paging system of SUPERCP may be looked upon as an independent task rather than a subroutine called by other tasks. The paging drum handler (PDH) takes as its input requests placed on queues by other tasks, starts the I/O operation, "goes to sleep" when it has nothing to do, and is awakened by I/O interrupts from the drum. Tasks having paging requests that they wish serviced need only place them on the pager's request queues and, if the pager is not waiting for an interrupt from the drum (a rare case), "shoulder tap" it so that it will check its input queues.

There are several advantages to this type of structure: It allows faster response to drum requirements since the PDH is concerned only with running the drum and is given immediate control when a drum interrupt happens; It reduces overhead since the PDH will attempt to empty out its input queues whenever it runs, thus handling a large number of requests (up to 18/drum) in one invocation instead of one invocation per request; And it provides a high degree of isolation of the paging mechanism from the rest of the system, making checkout and modification easier.

SLOT-SORTING

The PDH uses a slot-sorting technique similar to that found in TSS, UMMPS, and CP-67 Version 3.1. The 200 tracks on the drum are divided into 100 pair, and each pair is formatted into nine slots, as shown in figure 1. Note that slot 5 consists of a record written half on the first track and half on the second, an situation acceptable to the hardware. In actual fact, small dummy records are written between slots, to allow time for switching from one head to another between slots. Thus it is possible to read slot 1 from track 2, slot 2 from track 198, slot 3 from track 100, etc., and in fact to read nine pages in two revolutions of the drum, with each page possibly coming from a different track. The only restriction is that each page must be in a different slot, although this restriction may be relaxed under certain conditions explained below.

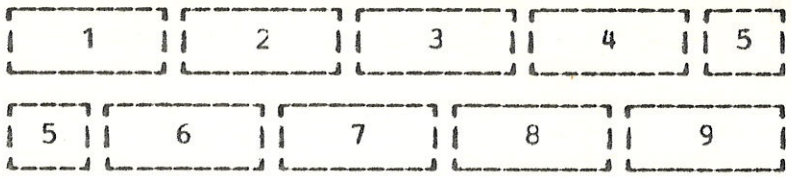

Figure 1. Format of a pair of drum tracks.

The PDH contains for each drum in the system two channel programs consisting of nine SEEK/NOP sections chained together into a single program. It has as input nine queues for page read requests and one queue for write requests, with the read requests placed on the queue corresponding to the slot in which the page requested resides on the drum. Write requests do not specify particular slots, since the PDH will allocate drum space to them as available. The PDH then follows the following algorithm:

1). Removes the top request from each read queue, changes the NOP in the corresponding SEEK/NOP section to a READ, and fills in the appropriate core addresses.

2). If any of the slots are not filled in (corresponding read queue was empty), checks to see if there are free pages on the drum for that corresponding slot. If so, dequeues the top request in the write queue, changes the corresponding NOP to a WRITE, and fills in the appropriate core addresses.

3). Starts the channel program as channel program 1.

4). If there are requests left on the queues, repeats steps 1 and 2 for the second channel program.

5). Changes the NOP which is at the end of channel program 1 to a TIC (Transfer In Channel) to channel program 2, with PCI (Program Controlled Interrupt) set so that an interrupt will occur when channel program 1 is finished.

6). When the PCI comes in, marks all the requests in channel program 1 as finished, changes it back to SEEK/NOP sections, and changes the TIC at the end back into a NOP.

7). If there are more requests, repeats steps 1 and 2 for channel program one and changes the NOP at the end of channel program 2 to a TIC/PCI.

The PDH will therefore keep the drum running continuously under a moderately heavy paging load, and will be able to service up to 270 requests per second, with only thirty invocations of the PDH per second.

A unique modification to the above algorithm consists of the following: After step 2, the PDH checks to see if there are any additional read requests for a slot and if the two slots which overlap that one on the other track are empty. Thus if there were two requests for slot 7, it would check if slots 2 and 3 were filled in. If they were not, it changes one of them to a read which satisfies the additional slot 7 request. The channel program will now read (in order) slots 1, 7, 4, 5, 6, 7, 8, and 9. This modification allows reading of more than one page from the same slot under restricted conditions.

There are a number of important advantages to slot-sorting in this fashion. It produces a much lower interrupt traffic than would a request-at-a-time system, since there is one interrupt for every chain of up to nine requests, but conversly delays notification of the completion of a request until all nine have been done. The PDH runs most efficiently when page traffic is heaviest, and would in fact have the minimum overhead per request when request density is greater than or equal to the maximum it can handle. Finally, it makes write requests very inexpensive to the system, since they are handled in what would otherwise be idle time, and "cost" only the trivial overhead of changing a NOP to a WRITE.

PENDING QUEUES

SUPERCP follows the rather vague policy of attempting to perform operations as soon as possible and defering having that operation be irrevocable as long as possible. This is best illustrated by the system of pending queues used by the system to handle pages.

A core page may be in one of three states:

1. Owned by CP; contains either CP code or control blocks.

2. Owned by a Virtual Machine (VM); contains user data.

3. FREE; contains no useful data.

User pages may be in one of four states:

1. In Core; Page is occupying a core page and is in active use.

2. Core Pending; Page has been written to secondary storage but has not yet been changed in core. It is not available to the user, but can be made available without the necessity of a read operation.

3. Drum Page; Page has been written to secondary storage and no copy exists in core.

4. Drum Pending; Page has been read from secondary storage and is in active use, but has not yet been modified by the user. The copy on drum is valid, and the page can be made Core Pending or a Drum Page without the necessity of a write operation.

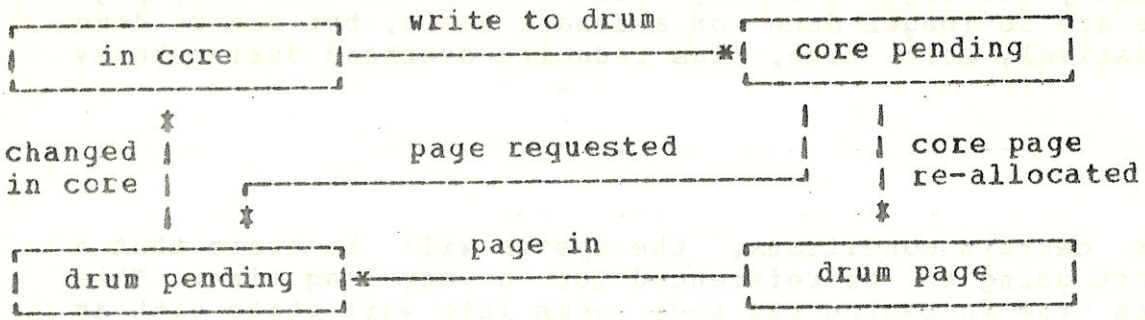

Figure 2. Possible states of a user page.

When the system needs a page, it will choose in the following order:

1. A Free page.

2. A Core Pending Page, converting it into a Drum Page without having to write it out.

3. A Drum Pending Page, converting it into a Drum Page without having to write it out.

4. Do something else, explained later.

Using the "pending" concept, the system can write out user pages at any time and as often as it wants, since they will be retained as Core Pending until the core is actually needed and since write operations are cheap. Pages are only irrevocably "paged out" (moved from Core Pending to Drum Page) only when necessary and, since the page which has been on the Core Pending queue for the longest time is choosen, the algorithm resembles the highly-efficient Least-Recently Used (LRU) algorithm for simpler systems.

This setup increases the apparent core size of the machine, since the set of core pages immediately available to the system consist of Free pages plus Pending pages, while the set of core pages immediately available to the user consist of In Core pages plus pending pages. The same is true of secondary storage space,

and the total effect is to provide better multiplexing of primary and secondary storage. A very nice effect is that many page operations are no longer done on a demand basis, but can be done during relatively idle time, thus reducing overhead during heavy loads.

MIGRATION

Under certain conditions, the system will be aware that a page is not going to be referenced for a very long time. This may be when the VM owning the page goes into wait state with an outstanding read request to the console, when a batch machine waits on the card reader and there are no more spooled files for it, or when a page on drum has not been referenced for a very long period. Pages of this type should not be kept in core (obviously) or on drum, but should be on disk, disk being the slowest and cheapest storage available. It is therefore possible to set a "disk preference bit" in the write request for such a page. The PDH will check if the disk channel is heavily loaded and, if not, create a CP Request Block to write the page to disk in the same way that normal CP would. If the channel has a heavy load, the PDH will ignore the disk preference bit. Also, if a page has been on drum for a long time, it is eligible for "migration" to disk. Such a page would be "Disk Pending" as long as a valid copy exists on both drum and disk, and a "Disk Page" when the drum copy is wiped out.

This would result in a true three-level store, using a LRU algorithm to determine the appropriate level for a page, and would have the effect of freer page movement, with less likelihood of "choking," but it does have some disadvantages. Since it is not possible to move a page directly from drum to disk, the PDH must decide if it can afford to use a core page for the time involved, if there is sufficient idle time on the drum channel to read the page in, and if there is sufficient idle time on the disk channel to write the page out. Because these decisions are difficult for the PDH to make, and because there is a fair chance that migration will result in more overhead than it saves, the migration function has not yet been implemented, pending further study.

THE DISPATCHER.

The SUPERCP dispatching algorithm closely resembles that of ISS, but with certain simplifications made possible by the simpler design of CP. All virtual machines in the system will reside in one of four queues, as follows:

THE DEAD QUEUE.

The Dead queue contains, in no particular order, all virtual machines which are in long term wait. Since it is not always possible to tell exactly why a virtual machine has entered wait state, i.e. what it's waiting for, "long term wait" has been defined as being in wait state with no selector channel I/O outstanding. For an interactive VM, this will normally happen when it is waiting for I/O from the console, and for a batch machine when its virtual card reader is empty.

THE READY QUEUE

The Ready Queue contains those VM's which are ready to use the CPU, that is, those not in wait state, and whose committments either are or are not currently being met. Each VM has associated with it a Scheduled Start Time (SST) computed from its service parameters. SST is defined as the time when the system must give the VM the CPU if it is to meet its time slot and CPU percentage requirements. For example, if a VM with parameters (.25,20) becomes ready to use the CPU, its SST will be 15 seconds in the future, since it must be started by then if it is to meet the committment. VM's whose SST is in the past, i.e. has come and gone, are said to be behind schedule.

The Ready Queue is in priority order, where the pricrity is a rather complex function of the political factor (importance), the amount behind schedule, and the Scheduler Table Priority (explained below). This priority is used only to determine the initial position of the VM in the queue, and thereafter its priority increases only as a constant function of the amount behind schedule, that is, increases at the same rate for all VM's in the queue. This eliminates the necessity of constantly re-ordering the queue as virtual machines become further and further behind schedule. VM's which are ahead of schedule have a negative priority, and are therefore at the end of the queue.

THE DISPATCHABLE QUEUE.

Virtual machines in the Dispatchable queue are the only ones which are eligible for the CPU, and therefore the only ones being multiprcgrammed. These tasks will either currently have the CPU, be ready for the CPU, or be in short term wait. It is a design goal of SUPERCP that tasks in the dispatchable queue should have their working sets in core (for a definition of "working set," see Denning(1)). Obviously, SUPERCP cannot follow Denning's definition exactly in determining working set, as this would involve intolerable amounts of overhead, but it does make an attempt to determine the working set size (number of pages in the working set, WSS) and limits membership in the dispatchable queue such that the sum of WSS's of dispatchable VM's is less than the number cf available core pages.

The Dispatchable queue is ordered by priority, where that priority is again a function of the amount behind (or ahead of) schedule, the political factor, and the Scheduler Table Priority. It should be noted that this priority will decrease as the VM receives service. Simply stated, the Dispatcher gives the CPU to the highest VM in the dispatchable queue which is not in short-term wait.

THE SCHEDULER TABLE.

Virtual machines in the dispatchable queue (and, to a lesser extent, in the ready queues) are contrclled through the use of the Scheduler Table. The Scheduler Table is an array of up to 256 entries called STE's (Scheduler Table Entries), each STE containing a number of parameters. At any time, a VM is associated with a particular STE, and may move from one STE to ancther for varicus reasons.

Each STE consists of 12 one-byte parameters, with names and meanings as follows:

Priority. The STE priority, to be used as part of the calculation of the VM's priority.

Quantum. The amount of CPU time, in units of 50 milliseconds, to be given the VM while in this STE.

Page Max. The maximum number of page reads allow the VM in this STE. This corresponds to Maximum WSS.

Page Max WTG. "Where To Go" The next STE to be used if Page Max is exceeded.

Page Min.	Minimum number of page reads allowed the VM in this STE.
Page Min WTG.	Next STE if Page Min is not exceeded.
SIO Max.	Maximum number of SIO (Start I/O) operations to a device on the selector channel allowed the VM while in this STE.
SIO Max WTG.	Next STE if SIO Max is exceeded.
SIO Min.	Minimum number of SIO's allowed the VM.
SIO Min WTG.	Next STE if SIO Min is not exceeded.
NTSE WTG.	Next STE if the quantum ends with Page reads and SIO's within limits (Normal Time Slice End).
FTSE WTG.	Next STE if the virtual machine is forced to Time Slice End.

Each VM has specified (in its directory entry) an initial STE to be used when it leaves the Dead queue. These initial STE's are a special format, containing only an initial Working Set Size (WSS) estimate and a normal STE number to be transfered to immediately. Currently there are three initial STE's, for CMS, OS, and APL.

The Scheduler Table is probably best thought of as a tabular representation of the state diagram of the dispatching algorithm, although actually drawing the state diagram would be messy since each state has six outbound edges and there may be up to 256 states. A VM will move around in the Scheduler Table in a way determined by its execution characteristics. For example, a CPU-bound VM which stays within Page and SIO bounds might move through a chain of STE's with successively more quanta and lower priority, while a VM which consistantly exceeds SIO Max might move into a ring of STE's giving a single quantum and high priority.

It can be seen that it is possible to have a fairly complex dispatching algorithm specified in the table. As a bonus, it is possible to have several distinct algorithms in concurrent use, simply by having "unconnected graphs," i.e. distinct sections of the table with no transisions between them. This is the case in the current system, where we have distinct (and quite different) algorithms for CMS, OS, and APL.

THE ACTIVATE DECISION.

The decision to move a VM into the Dispatchable Queue, called activation, is an important one since it determines the degree of multiprogramming of the system. This decision is made by the module ACTIVATE, which is called when (1) a VM moves from Dead to Ready; (2) a VM moves from Dispatchable to Dead; (3) 200 milliseconds elapses since the last call; or (4) if all VM's in Dispatchable are in short-term wait (the CPU has nothing better to do). ACTIVATE resets the priorities and calculates the WSS of all VM's in the Dispatchable Queue and subtracts the total of the WSS's from the number of pages in the system to get the number of available pages. It then checks each VM in the Ready Queue, starting at the top, to see if that VM's WSS is smaller than the number of available pages. If it is, ACTIVATE moves that VM into Dispatchable, re-calculates its priority, subtracts theVM's WSS from its estimate of the number of available pages, and goes on to the next VM on the Ready Queue.

If a VM will not fit in core, ACTIVATE checks if the Ready VM's priority is higher than any of those in Dispatchable, that is, if it can "bump" any dispatchable VM's. If the Ready VM can bump enough dispatchable VM's so that sufficient pages will be freed to hold its WSS, those VM's will be forced to Time Slice End (FTSE), moved from Dispatchable to Ready, and their in-core pages queued for writing out. The bumping VM is then activated, and ACTIVATE continues to check the Ready Queue. Only one bumping operation is allowed for each call to ACTIVATE.

WORKING SET SIZE CALCULATION.

Working Set Size is calculated by maintaining an exponential average of the number of pages the user has in core (NUMPAGES). This average is related to WSS because of the following mechanisms:

1). A VM in the Dispatchable Queue does not have pages stolen from it by the paging mechanism. When a page is needed, and there are no free or pending pages, the lowest priority dispatchable task is forced to Time Slice End, and all of its pages written out.

2). A module called FLUSH is called for each VM in the dispatchable queue at frequent intervals. FLUSH checks the referenced bit for each in-core page the VM owns. If the page has not been referenced, it is queued for writing out (if necessary) and made unavailable to the user. If it has been referenced, the referenced bit is turned off.

FLUSH is called for a VM only if it has had at least eight milliseconds of CPU since the last call to FLUSH. This means that pages not referenced within 8 ms. will be gotten rid of and NUMPAGES will be pages referenced in the last 8 ms., a good definition of working set. It should be noted that FLUSH is much less drastic than it seems to be, since the pages it writes out will remain available on the PENDING queue for quite some time.

The routine which calls FLUSH is entered whenever the CPU has nothing else to do, that is, just before loading a wait state PSW, and it momentarily enables interrupts often during its operation. In this way it is guaranteed that FLUSH will not reduce the capacity of the system, since it simply fills up some of the unavoidable wait time.

An interesting and desirable result of this algorithm is that thrashing should be self-correcting. As the system becomes page-bound, there will be more idle time available for calling FLUSH. Working Set Size estimation will therefore become more accurate, the degree of multiprogramming will be reduced by ACTIVATE, and thrashing will cease.

IMPLEMENTATION.

SUPERCP was written in such a way that it will hopefully be adaptable to future versions of CP. The bulk of the code is in complete replacements of CP modules (mainly DISPATCH and PAGTRANS) and in new modules (ACTIVATE, DRUMIO, etc.). There are a few minor changes to existing CP modules, and these are done using the UPDATE facility of CMS. The adaptation of SUPERCP to CP 3.1 took less than one man-week of work, despite massive IBM changes in precisely the areas where SUPERCP is different.

Use of SUPERCP by other installations should be quite simple, with one reservation. Brown has a quite elaborate accounting system in both CP and MVT/65, including such features as temporary CMS machines, individual accounting for DOS/APL users by CP, and running account files on disk for about 4,000 users. This system has interacted with SUPERCP in several places, the major one being that virtual machines no longer have passwords, the password being instead associated with the account which owns the VM. SUPERCP uses the UFDPASS field in the directory for slot-length, guarantee, and initial STE values.

Intuitively, SUPERCP should be even more effective in a large configuration than it is in Brown's 512K , one-drum /67. We would be very interested in seeing it run on different configurations and in different environemnts. If you are interested in SUPERCP, contact Richard Kogut or Robert Munck at 401-863-2221.

REFERENCES.

(1) Denning, P.J. "The Working Set Model for Program Behavior," CACM 11:5 p. 323 (May 1968)

(2) IBM Corporation, System/360 Time Sharing System Resident Supervisor, PLM Y28-2012

Doherty, W.J. "Scheduling TSS/360 for Responsiveness," Proc FJCC, p. 97 (1970)

(3) Alexander, M.T. Time Sharing Supervisor Programs, The University of Michigan Computing Center, May, 1971

Bernstein, A.J., and Sharp, J.C., "A Policy Driven Scheduler for a Time Sharing System," CACM 14:2 p. 74 (Feb. 1971)