TO: F. Corbató
J. Saltzer
S. Webber
N. Morris
J. Gintell
C. Clingen
V. Voydock
R. Roach
R. Frankston

FROM: C. Doyle

DATE: December 7, 1971

SUBJECT: Core Usage Metering Proposal

There will be a meeting to discuss the attached proposal on core usage metering on Monday, December 13, at 10:00 a.m. The meeting will be held at 545 Technology Square in the 5th floor conference room.

:cpd

Core usage metering proposal

For distribution:

F. Corbató
J. Saltzer
S. Webber
N. Morris
J. Gintell
C. Clingen
V. Voydock
R. Roach
R. Frankston

* *

Table of Contents

Introduction

This proposal presents an implementation of core usage metering on Multics. An attempt is made to find an equitable means of metering usage, distributing the cost in proportion to the usage of primary storage by each process. There are both intrinsic and pragmatic problems in implementing the metering scheme and the proposed solution is at best a compromise.

The purpose of resource metering in a system such as Multics is two-fold. First, it provides a basis for distributing charges for usage of system resources (the pricing policy) and secondly, it provides a basis for adjusting the system and its parameters. The pricing policy itself has two goals: to recover the cost of the computer system and to provide incentives to the user community to adjust its behavior. The former is a necessary function, the latter is more open to discussion. This proposal provides a means of achieving both of these goals and a better understanding of the use of core as a Multics resource.

Historically, the usage of core and of the CPU have been coupled in the scheduling and pricing algorithms of timesharing computer facilities. This is in contrast to the policy of systems such as OS/360 that have explicit charges for the use of core as a resource. In this system core is not normally a preemptable resource. (1)The reasons for this coupling differ from system to system.

The scheduling of a computer system has generally been viewed as a problem of scheduling processor time. The requirement for the use of main store is just a byproduct of the fact that most CPU's require that data to be fetched for an instruction reside in core. In the simplest systems each user is brought into core to be run and is then swapped out to make room for another user, and his use of core is directly proportional to the amount of processor time used. As systems became capable of keeping more than one process in memory simultaneously, this proportionality was no longer valid though it was normally considered to be a sufficiently good approximation.

In paged systems where users can have a fair number of pages in core at the expense of other users this assumption must be questioned. This is not simply a technicality, because as the complexity of data processing in information system increases, the limitation on a computer system becomes that of the availability of data in the primary store, rather than processor availability. This is already obvious in systems such as CP/67 and Multics where the installation CPU-Core costs are heavily weighted toward storage. The costs for storage are three to four times that for the processors in these systems.

(1) Even here the charge for core is keyed to the charge for the processor.

Introduction

It seems unfair, not just inaccurate, to charge for core at a rate proportional to CPU usage. (1)Charging in this way tends to provide a false evaluation of the resource usage costs of processes, thereby encouraging poor use of these resources. Since the sharing of core by users is not normally reflected in the charges (except through a reduction in page faults), users are provided with neither benefits nor incentives for sharing it, other than possible secondary storage savings. Again, detailed metering may show that the amount of actual sharing of programs in core is neglibible.

(1) On the other hand, all this metering might show that it is really proportional!

Approach on Multics

As the name implies, core usage metering, should provide a measure of actual usage of core. The problem is that the term "usage" is very vague. A page of core can be said to be in use if it is assigned to the address space of at least one user. That is, any time a page is resident in core it is "in use".

It is easy to meter the actual residency of a page, but this measure does not reflect the goals stated above for core usage metering. If the charges are to reflect the value of the resource so as to regulate competition for it, they should not vary inversely with this competition. If the measures are of basic system performance characteristics, they should be able to provide data which is invariant under changes of load characteristics. The first method of measuring usage -- core residency -- fails under both of these tests because residency increases in the absence of competition for core and thus does not provide a true measure of the "usage" of the page.

Neither does a measure of the number of actual references to a page (assuming the hardware were available to do so) provide a valid basis for metering. What we are trying to do is measure the value of having the page of data in core. This can be approximated by determining whether a page has been used (or referenced) in a small interval. If it has, then the page is considered to be "in use". The length of this interval should be less than the mean residency time for a page. This would make the interval short enough so that the value should not change too much with varying loads. It is long enough to reflect the actual time the physical page is unavailable to other users because it is in use.

If there were no sharing, it would only be necessary to add the usage of a page of core to the cumulative usage for the process whenever the usage is computed for a page. Since there is sharing, it is necessary to distribute the charges among the processes using the page. Since the basis for sharing in Multics is the segment, we estimate the sharing of the page by assuming that it is poportional to the sharing of the segment as a whole and measure only sharing of the segment.

In measuring usage for the segment we make an additional assumption that all processes share the segment equally. Neither assumption is correct, but their degree of validity can be determined by measuring the effect of user behavior on the metering. Only in pathalogical cases should the approximations be greatly inaccurate.

For any interval, the processes referencing a segment can be determined. For each active segment, the processes having the segment in their address space are listed in the trailers for the segment. Each active segment has an entry in the Active Segment Table (AST). The AST contains a fixed number of entries and only those segments currently being used (and their superior directories) have entries in the AST. For each AST entry, there is a list of trailers, one for each address space containing this segment. Each trailer tells how to find the Segment Descriptor Word (SDW) used to map the segment into a particular address space. The current trailer format can be modified to tell which process corresponds to a given address space.

Thus we have a basis for establishing an interval and determining which processes have used each segment in that interval. At the end of each interval we simply count the number of processes on the trailer of a segment and charge each processes with its portion of the usage. One more modification is needed to make the scheme viable since we want to count only those processes that have actually referenced the segment in the interval. We can do this by using the SDW to simulate a reference flag for the segment. At the beginning of each interval, the SDW is marked to cause a fault on the first reference to the segment in the interval. The fault handler can then reset the SDW to a non-faulting SDW. At the end of the interval, only those processes whose SDW does not have the fault set would be charged for the use the corresponding segment. The fault handler itself can run in less than 100 microseconds on the 645 and proportionally faster on the follow-on hardware. The cost of initializing a new interval can be kept relatively low, as described below in the implementation details.

The interval lengths should be system parameters which can be adjusted as necessary. The interval for measuring page usage should be on the order of one second, while the interval for measuring segment usage may be as long as several minutes.

Implementation Details

The first effect of the implementation of the core metering algorithm is on the system wide data bases -- the APT (Active Process Table), the AST (Active Segment Table) and the AST trailers (STR or segment trailers). This requires recompiling all programs that reference these tables. It is also necessary to make sure that all references are properly parameterized, particulary with regard to the size of the entry. These changes do not have any functional effect on the system which should be released as a service system to facilitate coordination with other ongoing projects.

Next, the two orthagonal measurements must be put in the system. The first is the calculation of core usage. This will be done by examining each page after a short interval and adding the length of the interval to the cumulative usage for the segment being maintained in the AST entry, if the page was referenced in that interval. For system metering, it is useful to maintain a separate counter of pages in core that are not being referenced (i.e. those not being charged for).

This can be done by having a procedure which updates the information at the end of an interval by scanning the core map (a table containing an entry for each page in core) and adding the interval length to the usage value being accumulated in the AST for each page that has been referenced. The reference bits would be cleared to initialize for the next interval. This method is straightfoward and relatively simple to implement. There is some interference caused by the fact that the paging algorithm uses these reference bits, but this can be resolved by mapping the physical reference bits in the Page Table Word (PTW) into software reference bits in the core map.

A simpler algorithm with the same effect can be used by taking advantage of the fact that the paging algorithm already performs a similar function of dividing the usage of a page into intervals determined by the time a lap through the core map takes.

Briefly, the paging algorithm cycles through the core map advancing an entry at a time when a page must be selected for removal from core. Each time it tests the reference bit for a page, it can perform the update of the usage information in the AST entry for the segment containing the page. As long as the interval is short, the effect is the same (perhaps providing even better resolution). The only difficulty is that the interval length may become long.

It would take no change to the paging algorithm to guarantee that it advances through the core map at a minimum rate. This minimum rate can correspond to the sampling rate in the first algorithm for measuring page usage. The rate can be enforced by a mechanism such as the traffic controller which can check whether the next entry in the core map to be examined has been updated within the minimum time. If it hasn't, a call can be made to page control to advance the counter enough steps. In order to implement this it is necessary to maintain a time of last update in the core map entry.

This is necessary in any case to be able to compute the length of the interval for a given page.

The other part of the metering is the distribution of charges to each process according to the usage of the segment. This is also done periodically, but with a much longer period than the usage calculation. It is therefore feasible to drive it from a process such as the initializer process.

Each update involves determining which processes are using each segment and the distribution of charges to these processes. All of the computation is done with the AST locked and unavailable to other processes. First the descriptor segments of each process are examined to determine which segments have been referenced by this process. For each standard SDW (in contrast to SDW's with the fault flag set), a flag is set in the trailer on the given segment for the process currently under examination. Some segments (superviser segments) do not have trailers so that they would not be affected. Then each SDW (other than those for supervisor segments) would be set to the special fault.

After all of the descriptor segments have been so processed, the AST is scanned and the charge for each segment is distributed to all processes that have been flagged as having referenced it. The charge for each process is stored in the APT entry and used by the Multics accounting programs in the same manner as processor charges.

The special fault is implemented by setting the flags for a directed fault in the SDW for each segment. In order to process this fault fast, the fault handler should not need to look at data other than the faulting SDW itself. It is therefore necessary to encode enough information in the faulting SDW in order to be able to recreate the proper SDW. This is done by recognizing that there are less than eight valid access modes permitted on normal SDW's. We can therefore use three bits which do not have meaning in the faulting SDW, to encode an index into a table of valid access modes. The fault handler looks up the mode in this table and stores it in the SDW.

In many cases another fault is used in the SDW to require the recalculation of access.

Setting null access will cause an access violation fault which will permit access to be recalculated the next time the segment is referenced. The routine that sets this null access ("setfaults") must be modified to set one of two forms of null access; the normal and the encoding in the faulting SDW. When the directed fault occurs, the SDW is set to a normal SDW with null access. The retry of the operation referencing the segment will immediately cause an access violation fault and access will be calculated in the normal matter. It is therefore not necessary for the directed fault handler to consider the null access as a special case.

When a process is removed from the trailer of a segment it is necessary to charge it for the portion of time which it was using the segment. The solution is to keep track of the number of processes referencing the segment by having the directed fault handler increment a reference counter in the AST entry. When a trailer is deleted, the process, if it has referenced the segment, is charged for its portion of the usage, which is then subtracted from the cumulative usage. The reference counter is, of course decremented since this program is no longer referencing the segment. The counter must also be zeroed at the beginning of each interval when the fault flag is set in the SDW's.

When a segment is deactivated (that is, removed from the AST) it is necessary to perform the computations for the distribution of the charges immediately. This would differ from the computation at the end of the interval in that the trailers are scanned, not the descriptor segments of the processes. The SDW for the segment in each process is found from the trailer and is reset. A table is made during the scan of processes which have referenced the segment so that these can be charged for the usage. For convience in the scan, it would be useful to have a count of the number of trailers a segment has so that table space for the processes (the APT pointers) can be reserved.

This would prevent the need for a second scan, thereby avoiding the possibility that the number of processes referencing the segment may change between scans.

It is possible for a segment to be deactivated with no processes referencing it, but a nonzero usage value since the intervals for the two meters are different.

The loss due this type of error is miniscule, but should be recorded.

Additional metering

Once the mechanism is put in for measuring core usage for processes, it is then easy to add additional metering to the system. The simplest is to keep track of the cumulative core usage for each segment in its entry in the directory. This could provide useful system metering as well as feedback on segment usage. A slight modification to this would provide another measurement which could provide insight into system performance -- a page-process meter. This would measure the amount of time the pages of a segment is in use and multiply it by the number of processes referencing it. The ratio between these measures would indicate the degree of in core sharing of the segment.

Of more immediate concern is the effective utilization of system resources. At all times it is necessary to assign core usage to classes of usage. These may be charged verses uncharged, referenced verses not referenced, or shared verses unshared. This can also provide a basis for measuring the effect of sharing, by providing an estimate of the number of pages being shared in core.

Other Considerations

It is possible that extremes in user behavior might adversely affect the validity the of the charging. For example, if one user consistantly references a single page in a segment, and another user the other 255 pages, there would be an inequity in the charges if both were running simultaneously. It is unlikely for such behavior to persist over a long period of time, but this possibilty should be considered and effect measured by test programs.

It should be noted that currently the user bringing in a page is charged even if there are other processes making use of the page. In addition the number of page faults is highly dependent on the system load (the behavior of other users). This is an inequity we are currently willing to accept, though this can be corrected with more accurate assignment of processor charges.

The initial implementation does not include the measurement of the cost of using supervisor segments because these do not normally have trailers associated with their APT entries. If the cost is considered reasonable, the trailers should be easy to add for those segments on which the special directed fault may be taken. Care must be taken in charging for supervisor segments so as not to charge the currently running user for the core used in handling faults for another process.