file - Frankston

To: F. Corbató
W. Burner
C. Clingen
R. Daley
J. Gintell
N. Morris
R. Roach
J. Saltzer ✓
T. Van Vleck
D. Vinograd
V. Voydock
S. Webber

From: R. Frankston

Subject: Charging for memory services.

Date: April 11, 1972

Abstract

The current Multics pricing scheme determines a user's charges as a function of the processor time used. By computing the charge as a linear sum of the processor time and the number of demand page faults, the actual costs can be more accurately assessed. Pricing for memory service provides a basis for determining the cost-benefit tradeoffs for differing amounts of primary memory.

Current Status

The supporting mechanisms are currently implemented in an experimental version of Multics for testing of the system modifications and the charging formula. Preliminary measurements tend to confirm the assumptions made, but more testing is necessary.

1. Why charge for primary memory services?

Primary memory services form a major portion of the computational services of a computer system. In the case of Multics, the cost of primary storage is $63,000 a month versus $30,000 a month for the processors. Charging for processor use alone can distort the actual costs of providing the computational services.

There are currently no strong incentives for users to decrease their load on the system's primary memory. Charging for the memory services enables the installation to evaluate the benefits of purchasing more memory and compare them to the costs and the potential revenue.

2. A suggested charging scheme.

The charge for primary memory services should be proportional to the load on the system memory and should not be unduly influenced by factors a users cannot control -- changes in configuration and the activities of other users.

The charge is to be proportional to the number of paging units. The number of paging units is equal to the number of demand page faults times the amount of memory available to the user. The amount of memory available to each user is computed by considering that memory is shared equally by the users currently eligible for memory services. For example, in system 15 at MIT, the normal configuration has 315 available pages shared among up to 6 users. Each demand page fault would then be equivalent to 52 paging units.

The formula for charging is to be:

$$tc = (cu*cr+pu*mr)*sf$$

where

tc is the total charge
cu is the cpu usage
cr is the cpu rate
pu is the number of paging units
mr is the memory service charging rate
sf is the factor for the shift

$$pu = demand\_page\_faults * \frac{\text{total number of pages available}}{\text{average number of eligible users}}$$

3. A justification for the charging scheme.

Demand page faults form the basis for calculating the memory service charge because they are easy to count and provide a measure of memory usage. This appears to be a reasonable measure of memory usage since in general page faults are well-correllated with memory occupancy, even when there is considerable sharing occuring.

Pre-paging is a technique currently used to improve performance by the system attempting to predict which pages a user is going to require and bring them in before they are demanded. It benefits a user with predictable paging activity by decreasing his demand paging. Since the user has no direct control over the amount of prepaging that is done for him, prepaging is not included in the charging formula.

Configuration		usable time per second	idle time per second	page faults per usable processor-second
CPU	Pages			
1	187	.55	.05	73
2	315	.65	.05	46

Table 1: Measurements on system 15

We can evaluate the effect of changing the configuration by examining the figures in Table 1, provided by Roger Roach. The "good" time is the time spent doing computations for the user as opposed to overhead functions. We can compute the number of demand page faults by assuming that the overhead is proportional to the number of page faults and considering the total overhead

per page fault (including the fault processing itself) to be 10 milliseconds. The page faults per usable second are computed by first finding the number of page faults per processor second:

faults = (1.00-usable-idle)/.010

For the one cpu configuration there are 40 page faults per real ("wall clock") second and on the normal, two cpu configuration there are 30 page faults per processor per second. The number of page faults per usable second is then the page faults per second divided by the usable time, giving the figures of 76 faults for the one cpu and 46 for the normal, two cpu configuration. The ratio, 73 to 46, between the paging levels on the two configurations is 1.59, which is close to the ratio between the number of available pages (315/187 = 1.68). These empirical results demonstrate that the charge for a given computation, as computed by the formula, will not vary greatly with changes in configuration.

In the formula for calculating the number of paging units, the number of available pages is divided by the average eligibility. The average eligibility represents the number of users that are currently sharing the memory. Therefore the number of pages available to a single process is the total number of available pages divided by the number of eligible users.

4. An initial suggestion for pricing.

We can compute a charge for the full configuration based on the current charge of $.13 per processor per real second. The cost of the configuration is $21K per core box (128 pages) per month and $15K for a processor. Therefore two processors with 315 available pages costs $81.7K, 36.7% of this is for the processors and 63.3% of this is for the available core. Since only 95% of each processor is utilized, the processor charge would be .13*.367/.95 which is $.05 per second. There are 30 page faults per processor second which implies that the memory charge should be .13*.63/30 which is $.0027 per page fault on the normal two cpu configuration.

For system 15 the normal configuration is considered to have 315 available pages shared among six users. Therefore each demand page fault is considered to be equivalent to 52 paging units. The memory charge is then $.000052 per paging unit.

To summarize:

Rate for processor usage - $.05 per processor-second
Rate for paging - $.000052 per paging unit.
($.0027 per page fault normally)

Shift factors:

Shift 1 - 1.00
Shift 2 - 0.75
Shift 3 - 0.31
Shift 4 - 0.50

5. The implementation of memory service charging

The implementation of this scheme is straightfoward. There are two new interfaces to return the metering information, one for the accounting system (hphcs_$process_status) and another for the user to determine his usage (hcs_$get_process_usage).

Page control increments the paging unit count in the active process table entry (APTE) for the process responsible for the page fault in the same manner as it now updates number of demand page faults, except that the paging unit count is multiplied by the number of available pages and divided by the average number of eligible users. The additional entry in the APTE requires recompilation of routines that make assumptions about the size of the APTE. Normally the size can be found in tc_data$apt_entry_size, so that the actual number of recompilations required is minimal.

The following programs are affected by the changes:

Routine	Change
pxss.alm	return paging measurement
get_process_usage.pll	return new usage information
tc_data.alm	recompile for apt change
emergency_shutdown.alm	"
dump.alm (bos)	"
fdump.alm (bos)	"
page_fault.alm	increment paging measure

Once the paging measurements are being recorded we can more accurately determine the effect of the charging upon users and adjust the policies accordingly. For the first month or two of its usage, the charge for memory services can be provided to users for comparison with their charges under the current pricing structure.

A procedure has been written that types a ready message consisting of the charges under both the old and the new pricing structures and the number of paging units used. This procedure may be used in place of the standard ready message program to observe the effect of these changes on individual commands.