home 494-8273

Massachusetts Institute of Technology
Cambridge, Massachusetts
Proposal for Thesis Research in
Partial Fulfillment of the Requirements for the
Degree of Bachelor of Science

Title: Hardware estimation of a process' primary memory requirements

Submitted by: David K. Gifford
H417 MacGregor House
450 Memorial Drive
Cambridge, Massachusetts 02139

Date of Submission: October 21, 1975

Expected Date of Completion: January, 1976

Brief Statement of the Problem: In large multiplexed virtual memory computer systems it is difficult to assess how much memory a process needs to run efficiently. The intent of this thesis is to implement a hardware extension to the associative memory on a 6180 processor to enable Multics to approximate a process' primary memory requirements. The Multics scheduler will be modified to make use of this information to optimize overall system throughput as defined by Sekino[1].

Supervisor Agreement: The program outlined in this proposal is adequate for a Bachelor of Science. The supplies and facilities required are available, and I am willing to supervise the research and evaluate the thesis report.

Fernando J. Corbató
Professor of Computer Science and Engineering

I. Introduction

To efficiently control the level of multiprogramming in a multiplexed virtual memory computer system it is essential to have a measure of each process' primary memory needs. Multics, CP/67, and MTS all attempt to provide such a measure by software means.

Using software it is difficult without excessive overhead to obtain accurate estimates of the amount of memory a process needs to run efficiently. This thesis proposes a simple modification applicable to any large scale virtual memory processor to provide hardware assistance in calculating a process' primary memory requirements.

The amount of memory a process needs to run is a function of the efficiency one desires it to operate at in a virtual memory environment. Thus, we shall represent a process' primary memory requirement (pmr) as pmr=M(efficiency).

If we assume the mean headway between page faults (mhbpf) is a function of the number of pages a process can use we find:

$$mhbpf = f(M)$$

Given that the page fault handling time (pfht) has an extremely small variance we find the efficiency of a task as:

$$eff = \frac{mhbpf}{mhbpf + pfht} = \frac{f(M)}{f(M) + pfht}$$

What is this efficiency? How does it relate to what the users actually see? To system efficiency?

With simple algebra we find:

$$f(M) = \frac{eff * pfht}{(1 - eff)}$$

Knowing f^{-1} we can find:

$$M = f^{-1}\left(\frac{eff * pfht}{(1 - eff)}\right)$$

Alternatively:

$$M(eff) = f^{-1}\left(\frac{eff * pfht}{(1 - eff)}\right)$$

Thus knowing f^{-1} for each process allows us to approximate its pmr given the per task efficiency with which we desire it to operate.

II. Dynamic derivation of f^{-1}

If we imagine a per task LRU stack model of primary memory the rate of references past level i shall be represented as R_i. Thus, if we have M pages that can be utilized by a process, R_M is the page fault rate. It is assumed that primary memory is managed under an LRU replacement algorithim.

The hardware extension this thesis proposes to the 6180 processor is the maintenance of R_{16} in a program accessible register. R_{16} is easily determined due to the LRU management of the sixteen word page table word associative memory in the 6180.

Knowing R_{16} we can approximate R_K. Saltzer has suggested[3] that R_K is linear in memory size, making the following a tentative approximation for R_K:

$$R_K = \frac{16}{K} * R_{16}$$

Now:

$$mhbpf = \frac{1}{R_M}$$

Alternatively:

$$mhbpf = \frac{M}{16 * R_{16}} = f(M)$$

Allowing us to deduce:

$$f^{-1}(mhbpf) = mhbpf * 16 * R_{16}$$

Thus, for this first order model:

$$M(eff) = \frac{eff * pfht * 16 * R_{16}}{(1 - eff)}$$

III. Implementation

A seperate board (645HK) containing the additional logic for maintaining R_{16} in program accessible form will be added to the 6180 in a spare board slot. The Multics scheduling algorithim will be modified to use the information it provides in the following ways:

1) At a call to block or at timer runout time the R_{16} register will be read and cleared. The initial hardware implementation will return the absolute number of page table word associative memory misses. Thus a virtual timer will be maintained, allowing R_{16} to be calculated as:

$$R_{16} = \frac{\text{PTW AM misses}}{\text{elapsed virtual time}}$$

2) Software compensation will be provided for associative memory clearing caused by fault and interrupt handling. The number of page table words refaulted on will be calculated at fault or interrupt time by a compare of the current page table word associative memory with a previously saved copy from the last fault or interrupt.

only if the associative memory is not full!

overhead?

3) A linear multiplier will be applied to the raw pmr estimate, allowing the pmr estimator to be tuned as follows:

pmr = pmr_coff * M(eff)

4) A moving average of a process' pmr will be maintained at pmr calculation time. Tentatively the following algorithim will be used:

$$\text{pmr\_average} = \frac{\text{pmr\_average} + \text{pmr}}{2}$$

5) At each new interaction the process' moving pmr average will be used as its pmr.

6) Upon task creation each process assumes a preset pmr until it goes blocked or incurs a timer runout.

7) The level of multiprogramming will be solely determined by the availability of primary memory for processes pmrs.

IV. Research plan

Most of the research effort will be spent analyzing the effect my pmr estimator exhibits on Multics under simulated operating conditions.

As described in section III, Multics hardcore will be changed considerably to accommodate my pmr estimator. A set of experiments will be constructed to attempt to factor out any unintended performance differential introduced by software changes.

Experiments will be run varying the linear multiplier pmr_coff in the pmr estimator to determine its optimum setting to target the per task efficiency in the system to the system administrator specified level. Experiments are planned varying the system administrator specified efficiency parameter with primary memory sizes ranging from 192K to 1024K. On each configuration, the system administrator specified efficiency parameter will be adjusted to optimize overall system throughput as defined by Sekino[1]. Analysis of experimental variance will be done.

Finally, comparisons between the completed design and other running virtual memory eligibility control algorithims will be made.

References and Bibliography

1. Sekino, Akira, "Performance Evaluation of Multiprogrammed Time-Shared Computer Systems", Ph.D. thesis, MAC TR-103, Project MAC, MIT, September 1972.

2. Denning, Peter J., "Resource Allocation in Multiprocess Computer Systems", Ph.D. thesis, MAC TR-50, Project MAC, MIT, May 1968.

3. Saltzer, J. H., "A simple linear model of demand paging performance", Communications of the ACM 17, Number 4, April 1974.

4. Denning, Peter J., "Comments on a Linear Paging Model", ACM Performance Evaluation Review 3, Number 4, December 1974.

5. Schatzoff, M. and Wheeler, L. H., "CP-67 Paging Priority Dispatcher", IBM Cambridge Scientific Center Report G320-2088, March 1973.

6. Morris, James B., "Demand Paging Through Utilization of Working Sets on the MANIAC II", Communications of the ACM 15, Number 10, October 1972.

7. Rodriguez-Rosell, Juan and Dupuy, Jean-Pierre, "The Design Implementation, and Evaluation of a Working Set Dispatcher", Communications of the ACM 16, Number 4, April 1973.