MIT/LCS/TM-81

HARDWARE ESTIMATION OF A PROCESS' PRIMARY MEMORY REQUIREMENTS

David K. Gifford

January 1977

MIT/LCS/TM-81

HARDWARE ESTIMATION OF A PROCESS' PRIMARY MEMORY REQUIREMENTS

by

David Kenneth Gifford

This Technical Memorandum is a minor revision of a thesis submitted on May 7, 1976 in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering at the Massachusetts Institute of Technology.

The work reported here was performed in the Computer Systems Research Division of the M.I.T. Laboratory of Computer Science, an interdepartmental laboratory. The work was supported in part by Honeywell Information Systems Inc., and in part by the Computer Science Department of Ford Motor Company Car Engineering. Publication of the work was supported by the Air Force Information Systems Technology Applications Office (ISTAO) and by the Advanced Research Project Agency (ARPA) of the Department of Defense under ARPA order No. 2641 which was monitored by ISTAO under contract No. F19628-74-C-0198.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LABORATORY FOR COMPUTER SCIENCE
(formerly Project MAC)

CAMBRIDGE MASSACHUSETTS 02139

ACKNOWLEDGMENTS

I would like to thank my thesis advisor, Professor Fernando J. Corbato, for the many helpful suggestions he provided over the course of this thesis.

Many thanks are due to Ron Bierman of the Ford Motor Company, for first providing me with the opportunity of pursuing this thesis research.

I would like to thank Bob Montee and Charlie Clingen of Honeywell for arranging for permission to modify the 6180, machine time, and my transportation which enabled the experiments in this thesis.

This is for Katrina.

HARDWARE ESTIMATION OF A PROCESS' PRIMARY MEMORY REQUIREMENTS

by

David Kenneth Gifford

ABSTRACT

It is shown that a process' primary memory requirements can be approximated by use of the miss rate in the Honeywell 6180's page table word associative memory. This primary memory requirement estimate was employed by an experimental version of Multics to control the level of multiprogramming in the system, and bill for memory usage. The resultant system's tuning parameters were shown to be configuration insensitive, and it was conjectured that the system would also track shifts in the referencing characteristics of its workload and keep the system in tune. The limitations of the assumptions made about a process' referencing characteristics are examined, and directions for future research are outlined.

THESIS SUPERVISOR: Fernando J. Corbato

TITLE: Professor of Computer Science and Electrical Engineering

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

1.1 Overview

Virtual memory systems are now enjoying increasing popularity due to the automatic management of memory they provide a programmer. Because of their high cost and the desirability of information sharing, they are generally time shared, which introduces a host of technological problems.

In this thesis we will be examining the task of correctly assessing a process' primary memory requirements. This assessment is important for two reasons. First, an approximation of each process' primary memory requirements is needed to strike the balance necessary to insure optimal operation of a time shared system. Secondly, an approximation is required to equitably charge for usage of primary memory, one of the most expensive components of a virtual memory system.

1.2 The Resource Balance

Before virtual memory the task of multiplexing processors among the processes competing for them was rather straightforward. Jobs had a fixed size, and primary memory was filled until it could hold no more. The jobs that were wholly contained in primary memory were eligible for processor cycles in

the multiplexing process.

However, virtual memory has complicated this scheme. As virtual memory presents an arbitrarily large virtual space to a process by multiplexing primary memory, processes will run in any amount of primary memory. Naturally with less real memory the simulation overhead of a large virtual memory increases, and it is much more efficient to have a larger real memory.

Somehow the operating system must determine which processes to make eligible in the time division multiplexing of the central processors. Inherent in this selection process is a trade off. Multiplexing too many processes simultaneously creates an excessive demand for the finite primary memory available, and each process can not obtain enough pages. This results in each process spending a high percentage of its time in paging overhead, instead of doing useful work.

If too few processes are made eligible then the time when all processes are waiting for page I/O to complete and are incapable of receiving processor time rises. This "multiprogramming idle time" represents unrealized processing capability.

Thus when too many processes are made eligible paging overhead increases and the fraction of the processor available for users' useful computation drops off. Likewise when too few processes are made eligible multiprogramming idle time increases, and users' useful computation drops off. Figure 1.2.1 shows the typical relationship between the number of eligible processes,

paging overhead, and multiprogramming idle.

Sekino [S2] has shown that the fraction of the processing capability of a system available for users' useful computation is linearly related to system throughput. After Sekino we shall call the fraction available "percentile throughput", and it is this quantity that we would objectively like to maximize.

The needed information then is an approximation of how much primary memory a process requires. With this information we could rationally select a subset of the ready processes to make eligible, with the knowledge they would not cause a performane collapse of the computing system by collectively demanding more primary memory than is configured.

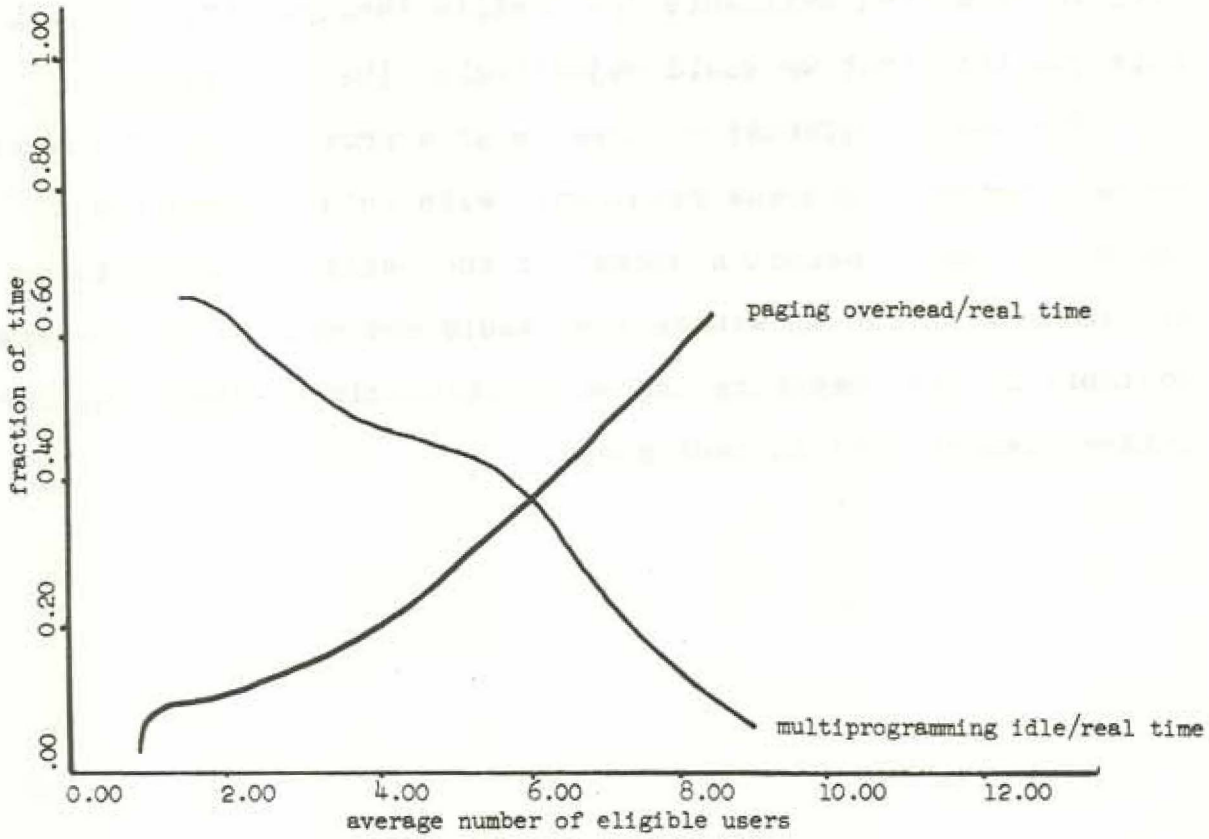


Figure 1.2.1

Effect of number of eligible processes on paging overhead and multiprogramming idle time

1.3 Previous Work

Various algorithms have been devised to approximate how much real memory a process needs to run in a virtual memory environment. Peter Denning [D1, D2] has suggested that each process has a "working set" of information. This set of information is represented by W(t,d), the collection of information referenced from t-d to t. Denning's model has been popularized as the set of pages that a process needs in primary memory to run optimally, and the amount of primary memory that needs to be set aside for a process is often called its working set size. To avoid confusion in this thesis we define the term primary memory requirement to represent the number of pages of primary memory a process requires.

Most operating systems that support virtual memory attempt to approximate through software each process' primary memory requirements. VM/370 [V1], CP/67 [S3, R1], The Michigan Terminal System [A1], MANIAC II [M2], and Multics [O1] all maintain primary memory requirement (pmr) estimates which are used in the process of deciding what processes to make eligible in the time division multiplexing of the central processor.

The estimation schemes used by these systems suffer from a number of common ailments. The behavior of a process is colored by the behavior of the processes it is being multiplexed with, as global page replacement algorithms are employed. The observable behavior of a process is noisy and generally does not accurately depict the true characteristics of the process. In addition, the

information available to the estimation scheme is limited as the hardware supporting virtual memory systems is generally minimally augmented traditional equipment.

To compensate for the limited noisy environment they operate in, primary memory requirement estimation algorithms have grown in complexity. Some provide reasonable estimates but they require careful tuning and are sensitive to system change.

Reed [R3] explored the problems of using a model similar to the one proposed by this thesis without hardware assistance for pmr estimation. He demonstrated the potential usefulness of the model, and provided a very rough measure of a process' primary memory requirements.

1.4 Scope of the Thesis

This thesis proposes a simple model of process behavior that can be utilized to estimate a process' primary memory requirements. It is shown in Section 2 that this model can be fitted to a process with information provided by a simple hardware extension to the associative memory of the Honeywell 6180 processor. The estimates generated with this scheme are load and configuration independent.

A prototype implementation for Multics is proposed in Section 3, with the results of numerous experiments on this implementation reviewed in Section 4. Multics was chosen for the prototype implementation due to the availability of equipment,

however the thesis could have been implemented on a number of different systems. Section 5 summarizes the utility of the thesis in light of the results of Section 4.

2. Estimation of a process' primary memory requirements

2.1 Functional representation of a process' primary memory requirements

As outlined in Section 1.2 the amount of primary memory a process is allowed to utilize is related to the amount of paging overhead it will experience. We define the mean headway between page faults (mhbpf) to be the mean time a process runs before it causes a page fault and is forced to incur a page fault handling time (pfht) in addition to the time it takes to retrieve the needed page from secondary store, the page fetch time (pft).

We now can formalize the concept of a process' primary memory requirements. Imagine a function f relating the mhbpf for a process to the number of pages it requires to obtain this mhbpf. That is:

$$M = f(mhbpf)$$

$$mhbpf = f^{-1}(M)$$

Naturally f will not be a static function. The next section deals with the problem of dynamically determing f for a process. Inherent in this discussion is that the scope of f is limited to the time frame in which process behavior was observed to determine it.

2.2 Dynamic determination of f

The problem of determing a process' primary memory requirements is now reduced to the problem of determing what f looks like for the given process. With f we can analyze what type of performance return to expect from an additional commitment of primary memory to a process.

To determine f for a period of time in a process' life we can observe the process for this period and then use the information collected to estimate f. The naive approach is taken by this thesis, that is that f for the next period is equivalent to f for the period just measured.

If we imagine a per process least recently used (LRU) stack model of primary memory [M1] the rate of references past level x shall be represented as R(x). 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 algorithm, as it essentially is in Multics [C1].

The hardware extension this thesis proposes to the 6180 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 associative memory in the 6180.

Knowing R(16) we can approximate R(K). Saltzer [S1] has shown through measurements that R(K) is roughly linear in memory size at the system level for Multics, making the following a tentative approximation for R(K) at the process level:

$$R(K) = \frac{16}{K} * R(16)$$

Now:

$$mhbpf = \frac{1}{R(M)}$$

Alternatively:

$$mhbpf = \frac{M}{16 * R(16)} = f^{-1}(M)$$

Allowing us to deduce for this first order model:

$$M = f(mhbpf) = mhbpf * 16 * R(16)$$

2.3 Establishment of a desired mean headway between page faults

Having been able to determine f, we must specify the performance level we desire a process to operate at in terms of its mhbpf to estimate its primary memory requirements. Direct specification of the desired mhbpf of a process as a tuning parameter did not seem to provide an eloquent solution. It was felt that specification in terms of the maximum fraction of time that could be devoted to paging overhead would be a much easier to conceptualize tuning variable than an absolute value of mhbpf in microseconds.

As each page fault incurs a page fault handling time (pfht), and this pfht has a low variance, we can characterize one aspect of process efficiency, the fraction of time the process is not

spending in paging overhead as:

$$eff = \frac{mhbpf}{mhbpf + pfht}$$

For historical reasons in the implementation of this thesis the term "efficiency" was retained for this application. Thus the reader must not confuse it with a representation of total system efficiency. Note that

$$1 - eff = \frac{\text{paging overhead in system}}{\text{busy time}}$$

is an approximation that could be made on the gross expenditure of time at the total sytem level.

To obtain a specified "efficiency" for a process the above expression allows us to calculate its mhbpf as:

$$mhbpf\_wanted = \frac{efficiency\_wanted * pfht}{(1 - efficiency\_wanted)}$$

Thus with a system administrator specified value of efficiency_wanted it is straightforward for Multics to compute the value of mhbpf_wanted, as the page fault handling time is currently metered by the system.

2.4 Estimation of a process' primary memory requirements

From Section 2.2 we have the approximation:

$M = mhbpf * 16 * R(16)$

and from Section 2.3 we have:

$$mhbpf\_wanted = \frac{efficiency\_wanted * pfht}{(1 - efficiency\_wanted)}$$

Thus M_wanted, a process' primary requirement estimate, can be expressed as:

$$M\_wanted = \frac{efficiency\_wanted * pfht * 16 * R(16)}{(1 - efficiency\_wanted)}$$

To allow for error in the approximation process a simple linear multiplier is provided, ws_coff, allowing us to replace M_wanted by M. The name ws_coff is also historical in origin. The final pmr estimate is then:

$$M = \frac{ws\_coff * efficiency\_wanted * pfht * 16 * R(16)}{(1 - efficiency\_wanted)}$$

2.5 Inaccuracies inherent in the estimate

In Section 2.2 we made the assumption that R(K) was roughly linear in memory size. If we let p(x) be the stationary probability distribution of referencing level x in the LRU stack on any reference, and generalize p to be continuous, we know that:

$$\int_{1}^{inf} p(x)\,dx = 1$$

In addition the probability of referencing past level y on any reference is:

$$F(y) = \int_{y}^{\infty} p(x)\,dx$$

Now for R(K) to be linear in memory size:

$$F(y) = 2 * F(2 * y)$$

Or:

$$\int_{y}^{\infty} p(x)\,dx = 2 * \int_{2*y}^{\infty} p(x)\,dx$$

With the constraints presented for p(x) we find:

$$p(x) = \frac{1}{x^2}$$

Thus, for the assumption that R(K) is linear in memory size the static probability of referencing level x in the LRU stack must be as shown above.

The accuracy of the estimate then depends on how a process' referencing characteristics differ from the ideal characteristics assumed by this first order model. A serious departure from p(x) by a process will degrade the estimate provided. The more serious the departure, the worse the estimate will become.

Greenberg [G1] discusses possible representations of p(x) in view of his experimental observations of Multics. More

information about the typical form of p(x) would allow R(16) to be used in higher order approximations of R(K), and provide improved accuracy.

3. Implementation

3.1 Virtualizing the raw associative memory miss rate

The associative memory for the page table words in the 6180 holds sixteen page table words, and is managed under an LRU replacement algorithm. Thus the number of non-matches, or misses, in the associative memory is the same as the number of page faults that would be incured in a sixteen page real memory being managed under LRU replacement.

The number of raw misses in the associative memory is relatively easy to obtain through hardware. However Multics frequently clears the associative memory to reflect changes in the in core page tables, causing superfluous misses. In addition page fault and interrupt handling cause a substantial number of misses that are counted by the hardware.

The problem of taking a per processor raw miss count and maintaing a per process virtual miss count is very similar to billing processes for virtual cpu time, although more complex. As indicated above the virtualized miss count represents the number of page faults the process would have incured running in a sixteen page LRU managed primary memory.

A separate board (645HK) containing the additional logic for maintaining the raw page table word associative memory (PTW AM) miss count in program accessible form was added to the 6180 in a spare board slot. Software compensation is provided for

associative memory clearing and page fault and interrupt handling. The number of page table words missed on while refilling the associative memory is calculated at page fault or interrupt time by a compare of the current PTW AM with a previously saved copy from the last page fault or interrupt. A subroutine, called adjust_vpfs, was added to the system to perform this compensation.

3.2 Maintenance of R(16)

A virtual cpu timer is maintained for each process, allowing R(16) to be calculated as:

$$R(16) = \frac{\text{virtual PTW AM misses}}{\text{elapsed virtual time}}$$

3.3 Mechanics for computing a process' pmr estimate

To facilitate varying experimental approaches the computation of a process' primary memory requirement was made a subroutine of pxss [S4], the scheduler of Multics. To compute a process' pmr one calls compute_working_set (once again called this for historical reasons), and a pmr is estimated using information collected since the last compute_working_set call according to the equation derived in Section 2.4.

3.4 When a process' pmr is computed

For the experiments descired in Section 4 a process' pmr was computed at three times: timer runout, process block, and at preemption for a higher priority process. Before a process' pmr is computed a check is made to insure it has amassed min_vcpu microseconds of virtual time since its last compute_working_set call. If it hasn't, compute_working_set returns without calculating a new estimate.

The min_vcpu parameter was a safeguard after early experiments showed erratic estimation due to the systems tendency to look through very small windows at a process' behavior. Fifty milliseconds was the standard setting of min_vcpu.

3.5 Approximating a new interaction's pmr

When using a process' pmr for control purposes, i.e. deciding whether to let a process become eligible or not, there is an underlying assumption that the previous behavior of a process is indicative of what its future behavior will be. What then can be said of a process' behavior after it has interacted with a human? Different types of requests generate greatly differing resource requirements.

The assumption made by this thesis is identical to the traditional scheme used in estimating a running process' future

requirements: resource requirements for a new interaction can be estimated from resource requirements of previous interactions.

In the case of primary memory requirements, a moving average is maintained on each call to compute_working_set:

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

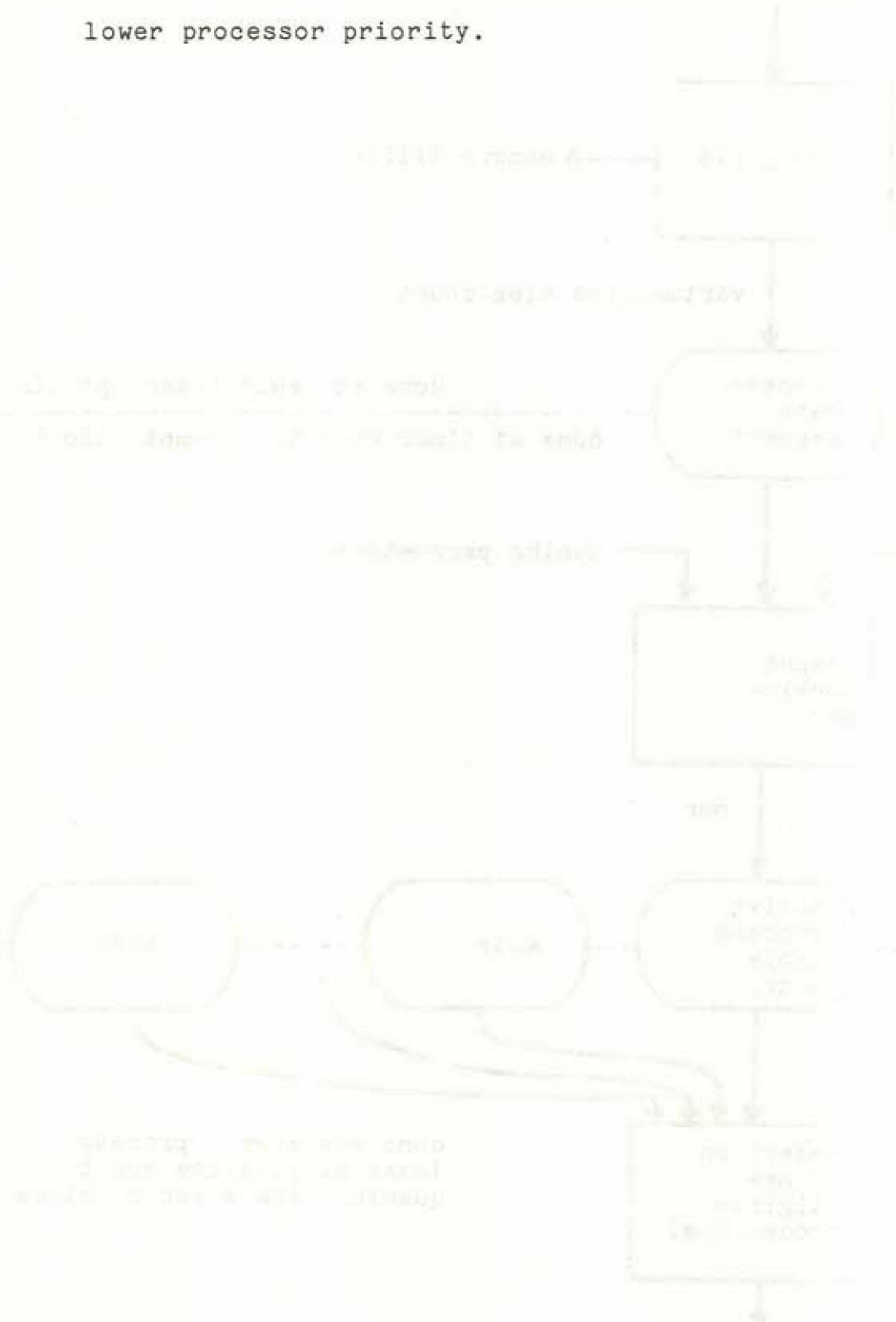
The average is used as the process' pmr on each new interaction. On process creation each process assumes a preset pmr until it goes blocked, is preempted, or incurs a timer runout.

3.6 Determination of level of multiprogramming

Figures 3.6.1 and 3.6.2 outline the flow of information and decision process used for the implementation of eligibility control in this thesis. They are simplified representations of the actual implementation but they characterize the decision process well as all of the logic from the standard system was removed.

In both the standard system and the one proposed in this thesis a preemptive priority discipline is used for processor scheduling within the eligible processes. This discipline mitigates the effect of lower priority processes with large pmr's displacing pages of higher priority processes. Thus the decision in the control logic for the pmr system to make processes with extremely large pmr's eligible concurrently with other processes

seemed reasonable in view of the fact they would always be of lower processor priority.



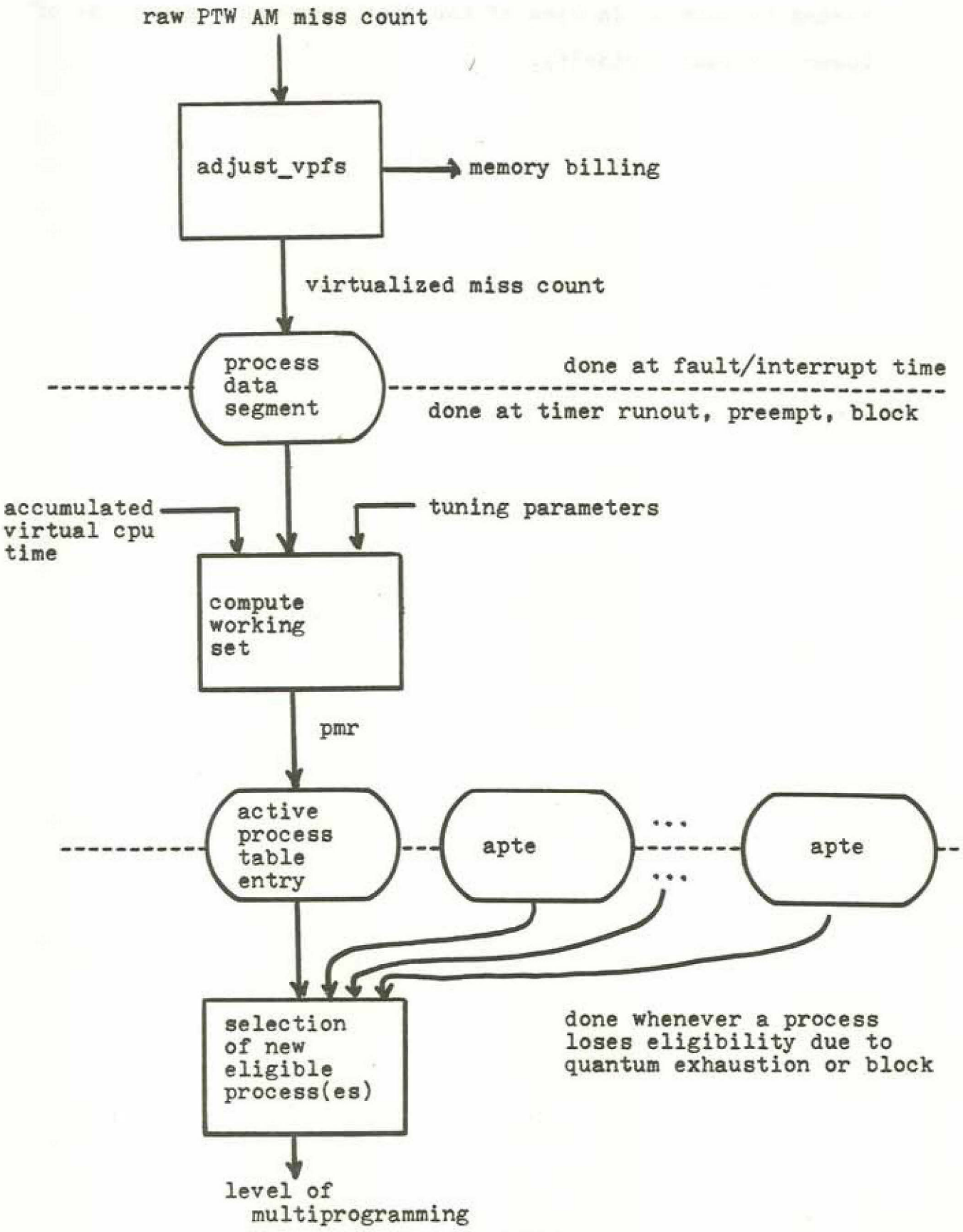


Figure 3.6.1
pmr estimation information flow

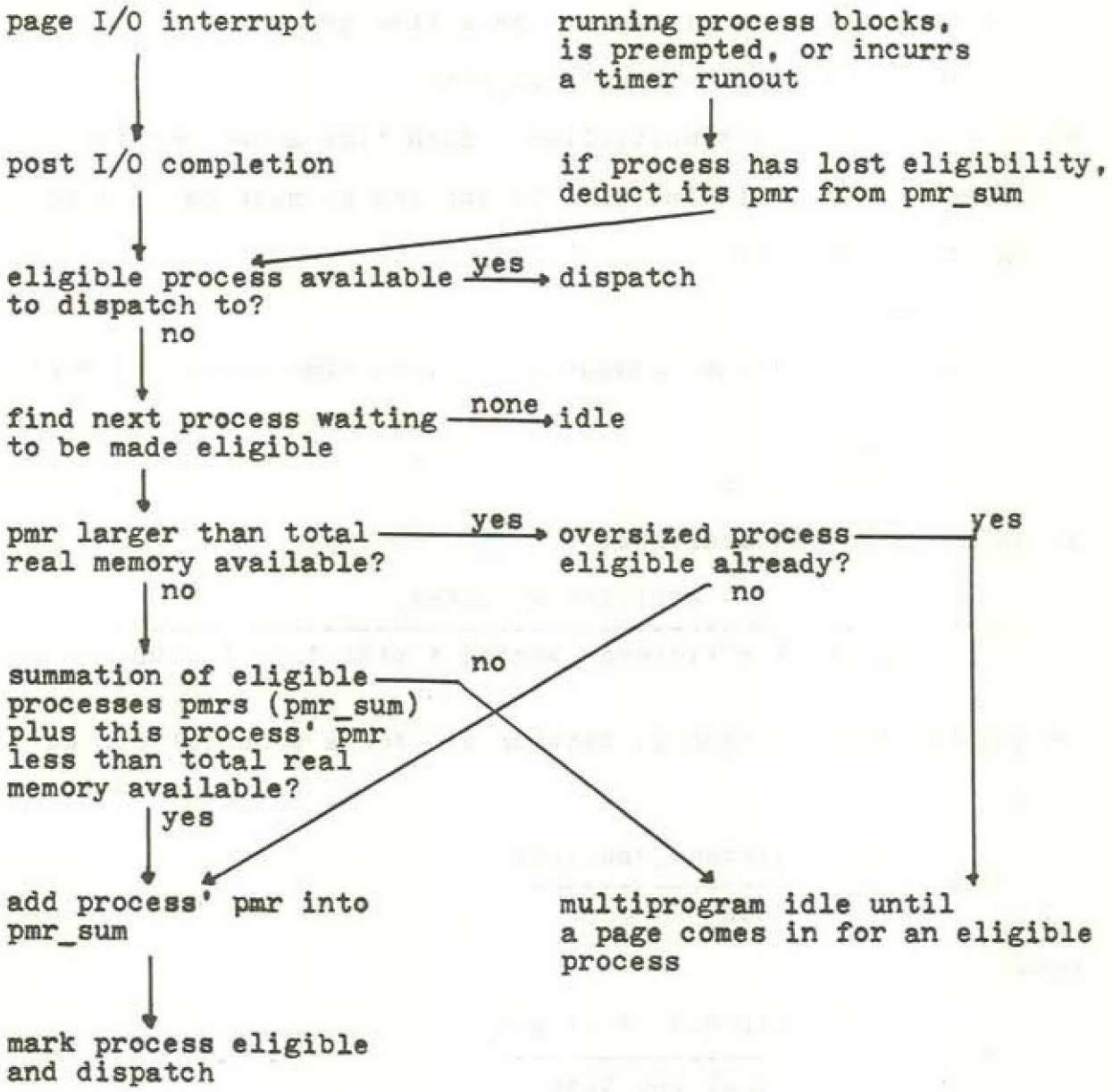


Figure 3.6.2

Eligibility control logic

3.7 Billing for memory usage

Memory usage is billed for as a time product:

$$U(M) = a * pmr * virtual\_cpu\_time$$

Where a is a constant multiplier. Each time a new pmr is computed the virtual time span it applies to must be used to integrate U. That is

$$U = \sum_{\text{all pmr computations}} a * (\text{pmr computed}) * (\text{vcpu time computed over})$$

If we let a be the constant:

$$a = \frac{(1 - efficiency\_wanted)}{ws\_coff * efficiency\_wanted * pfht * 16 * 1000}$$

Then using the equation in Section 2.4 for a process' pmr we find:

$$U = \frac{R(16) * virtual\_cpu\_time}{1000}$$

Knowing:

$$R(16) = \frac{\text{virtual PTW AM misses}}{\text{virtual cpu time}}$$

We find:

$$U = \frac{\text{virtual PTW AM misses}}{1000}$$

Thus with the estimation algorithms employed in this thesis a time product memory charge is simply the absolute number of

references past level 16 in the LRU stack. The factor of 1000 was added to reduce the magnitude of the charge for user consumption. The subroutine adjust_vpfs maintains the virtual PTW AM miss count for memory billing in the implementation of this thesis.

3.8 Self tuning of the system

As indicated in Section 2.5 the first order model proposed in Section 2.2 was not expected to be very accurate. To help determine the correct value of ws_coff several experiments were run with the self tuning algorithms in Figure 3.8.1 enabled.

As depicted in the flow chart the algorithm attempts to adjust ws_coff such that the system administrator specified efficiency_wanted is maintained by the system. The parameters delta_to_ws_coff, tune_interval, and min_busy control the dynamic response of the feedback mechanism.

In addition to the self tuning of the pmr estimator, self tuning of scheduler quanta was implemented. The system administrator is allowed to specify what percentage of interactions should complete without exhausting their time quanta. The system will automatically tune the scheduler quanta to reach the "task switching ratio" specified by the system administrator.

Note that the stability problems encountered in any feedback system are present here. Figure 3.8.2 is a block diagram of the

feedback loop. In Section 4 the stability of this feedback system is examined.

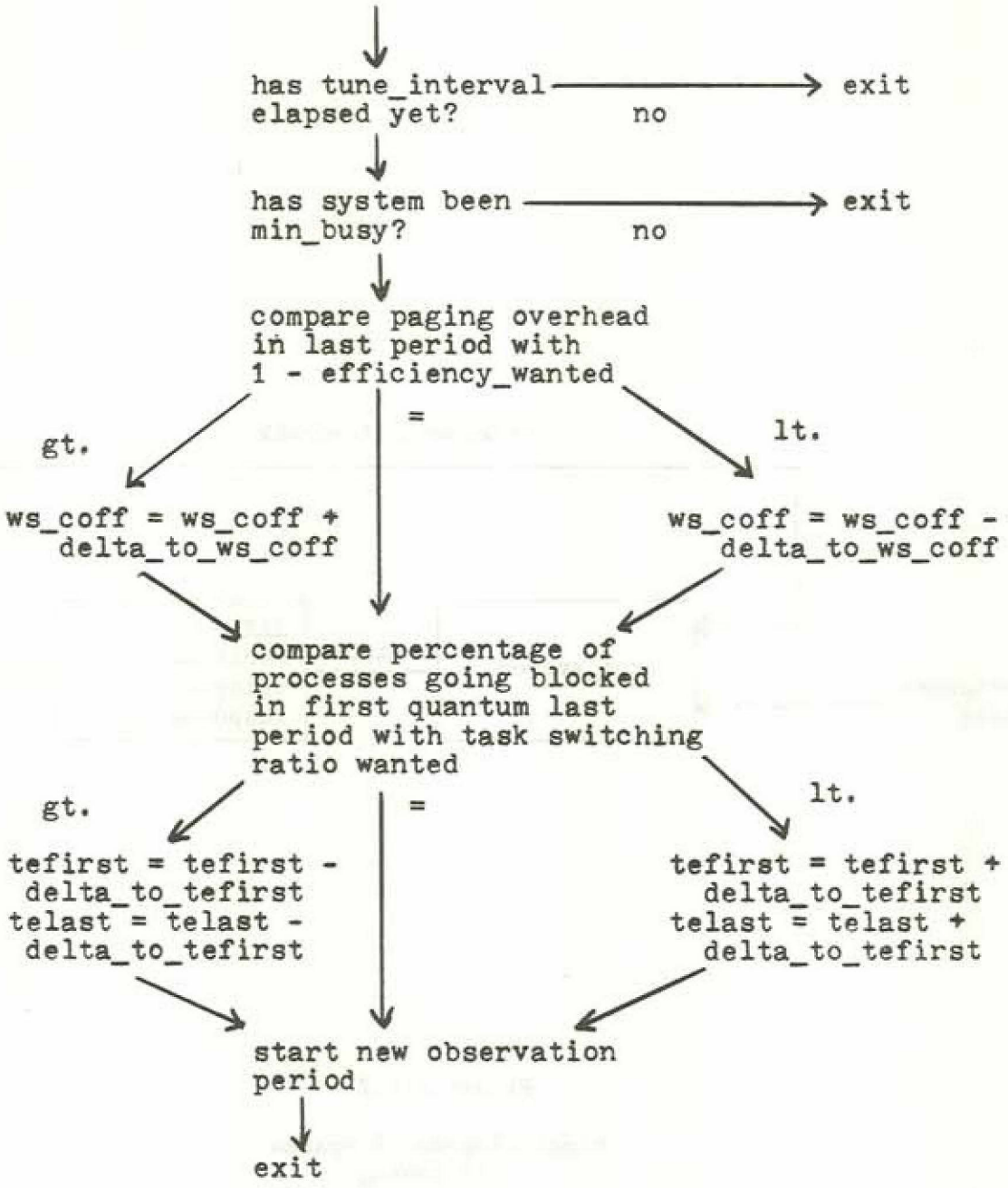


Figure 3.8.1

tune_system subroutine
flowchart

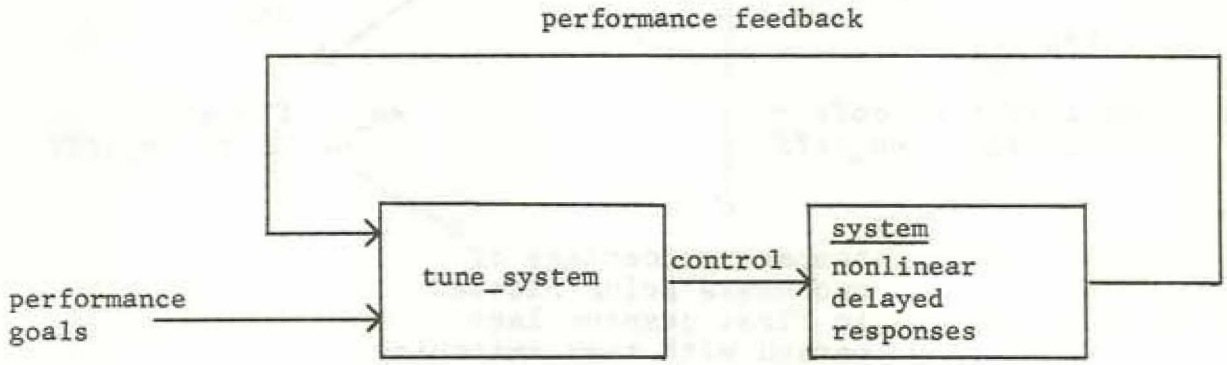


Figure 3.8.2

Block diagram of system
self tuning

4. Experimental Results

4.1 Design of the experiments

A total of 67 controlled experiments were run varying both software design and system configuration. The experiments were conducted at three different sites: the Ford Motor Company Research and Engineering Center in Dearborn, Michigan; Honeywell Information Systems Inc. in Phoenix, Arizona; and Honeywell Information Systems Inc. in Cambridge, Massachusetts.

Each experiment consisted of applying a synthetic script driven load to a dedicated system and measuring the systems resultant behavior. The scripts were designed by M.I.T.'s Information Processing Center, and were used in a packaged environment they designed to initiate, terminate, and collect statistics on performance experiments [R2]. The so called "M.I.T. performance test" allows the specification of an arbitrary number of users to be simulated with random based think times. Ten simulated users were used for each thesis experiment.

Multics is typically operated with a small capacity paging device of bulk core (usually a few million words). As the page fetch time from the "bulk store" is negligible no multiprogramming occurs on page faults that can be statisfied from the bulk store. The experiments for this thesis were run without bulk store to magnify the effects of the pmr estimator.

During each experiment the software monitor Aware [M3] developed by the Ford Motor Company was run to record system performance statistics. The graphs reproduced in this section representing individual experiments were produced by Aware.

Both the standard version of Multics and the one modified to incorporate the ideas in this thesis were doctored to make the simulated users appear more like a real user load. This was accomplished by providing interaction credit on timer wakeups.

4.2 Validity of the pmr estimator

Two experiments were run on different configurations with the system's self tuning turned off. The experimental load was identical, thus affording an opportunity to examine how the pmr estimator behaved on vastly different configurations.

Table 4.2.1 outlines the results of the experiments, along with the configurations employed. Figures 4.2.2 and 4.2.3 show the histograms of pmr sizes calculated for the 512K and 256K experiments, respectfully.

The first observation that can be made is that the pmr estimator does indeed discriminate between the varying memory requirements of processes. The histograms show that the miss rate in the PTW associative memory is far from constant, and probably contains significant information about a process' primary memory requirements.

The second observation that can be made is that the pmr

estimates are fairly configuration independent. The averages are resonably close across configurations, and the distribution of pmr's as depicted by the histograms are very similar.

The amount of virtual cpu time that elapses on the average before a process' pmr is computed is significantly less in the 512K experiment. This can be attributed to a higher number of preempt interrupts, caused by a higher degree of multiprogramming.

The parameter ws_coff was estimated to be .157 by an experiment where the system was allowed to self tune ws_coff. The experiment found that .157 was value that most accurately targeted the specified efficiency_wanted into system performance.

The reason ws_coff is so low is possibly due to the high percentage of time the system spends in so called "wired core". This memory is paged, and figures in the PTW associative memory miss rate. However it is never considered for removal and its contribution to a process' pmr can essentially be considered as overhead.

	Configuration A	Configuration B
Effective memory size (pages)	440	187
efficiency_wanted	.50	.50
user time / busy time resulting	.62	.46
pmr_coff	.157	.157
page fault handling time (msec)	3.35	3.224
average virtual cpu time between pmr computations (msec)	103.2	141.9
average virtualized misses between pmr computations	1048	1475
average R(16) (misses/msec)	10.15	10.39
average pmr (kept by compute_pmr)	83.98	77.05
Configuration information		
total primary memory (words)	512K	256K
disk channels	2	2
bulk storage device for paging (ICS)	no	no
experiment number	37	58

Table 4.2.1

Typical values for a

pmr system on two different

configurations

pages	count	pcnt	
0- 16	122	2.08	*****
16- 32	112	1.91	*****
32- 48	588	10.04	**********
48- 64	552	9.42	**********
64- 80	917	15.65	**********
80- 96	1454	24.82	**********
96-112	1099	18.76	**********
112-128	654	11.16	**********
128-144	293	5.00	**********
144-160	57	0.97	**
160-176	8	0.14	
176-192	2	0.03	
192-208	0	0.00	
208-224	0	0.00	
224-240	0	0.00	
240-256	0	0.00	

efficiency_wanted = .50 ws_coff = .157

Figure 4.2.2

Sample pmr histogram
512K
system

Experiment 37, 10

script load

pages	count	pcnt	
0- 16	91	2.14	****
16- 32	100	2.35	*****
32- 48	469	11.01	**********
48- 64	434	10.19	**********
64- 80	942	22.11	**********
80- 96	1344	31.55	**********
96-112	535	12.56	**********
112-128	269	6.32	**********
128-144	69	1.62	***
144-160	7	0.16	
160-176	0	0.00	
176-192	0	0.00	
192-208	0	0.00	
208-224	0	0.00	
224-240	0	0.00	
240-256	0	0.00	

efficiency_wanted = .50 ws_coff = .157

Figure 4.2.3

Sample pmr histogram
256K
system

Experiment 58, 10

script load

4.3 Performance of the pmr system

A series of experiments was conducted to compare the performance of the system with and without the pmr estimator proposed in this thesis. The configurations and loading used were the same as outlined in Secions 4.1 and 4.2.

The standard system (without the pmr estimator) was tuned by locking the tuning variables min_e and max_e together, fixing the level of multiprogamming at the specified level. The pmr system was tuned by varying efficiency_wanted through its range from 0.0 to 1.0.

Figures 4.3.1 and 4.3.2 depict the results of the experiments. The x-axis is the efficiency_wanted specified, and the y-axis is the resulting performance of the system as measured by three variables. Note the horizontal lines represent an optimally tuned locked eligibility standard system, as it does not have an efficiency wanted variable.

The top graph in 4.3.1 and 4.3.2 is the elapsed real time of the ten scripts. The gaps between the pmr system and the standard system are within the measurment noise, and are not significant. Note that the 256K configuration has a much sharper null as shown in Figure 4.3.2 due to its smaller primary memory size. Raising min_e and max_e to 3 for a run of the standard system on the 256K configuration drove the elapsed time to over 70 minutes. The 512K configuration was much less sensitive to such changes.

From the information in Figures 4.3.1 and 4.3.2 it is

obvious that the pmr estimator does not increase percentile throughput as hoped. The best the pmr system can do is match the performance of the locked eligibility system.

Also note the relationship between the ratio of user time to busy time and efficiency_wanted. The relationship is linear over the range measured in the 256K experiments due to the control afforded by the small memory size.

For both configurations the optimum performance of the pmr system was found at the same value of efficiency_wanted. This seems very significant, as ideally one would like to tune a system once using configuration insensitive parameters. Most user sites do not have the inclination or expertise necessary to tune their system. Thus tuning a system once "at the factory" would allow user sites to enjoy optimum performance without burdening them with the difficult task of system tuning.

The standard system could be tuned for an optimal level of multiprogramming by taking the number of available pages, dividing by a constant, and fixing the level of multiprogramming at the resulting quotient. However the constant would contain information about the referencing characteristics of a specific site's workload, and could not be expected to be applicable to other sites.

The pmr estimator described in this thesis would probably track a workload characteristic shift due to its actual measurement of the needs of processes. We have already seen evidence that it tracks over configuration changes. Thus even

though no performance gain was demonstrated, it looks like it is much easier to achieve optimum performance with the proposed pmr estimator.

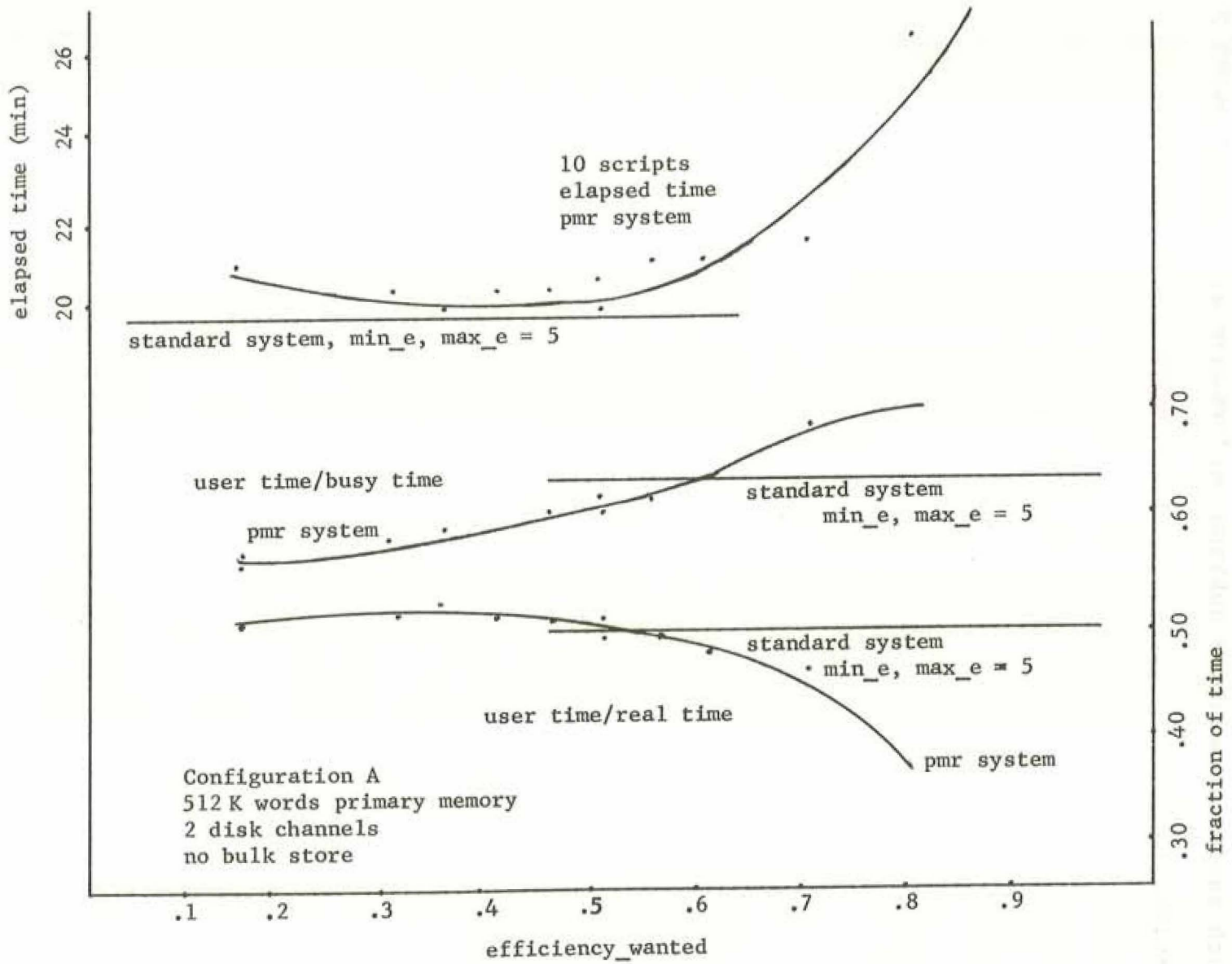


Figure 4.3.1
pmr system performance on configuration A

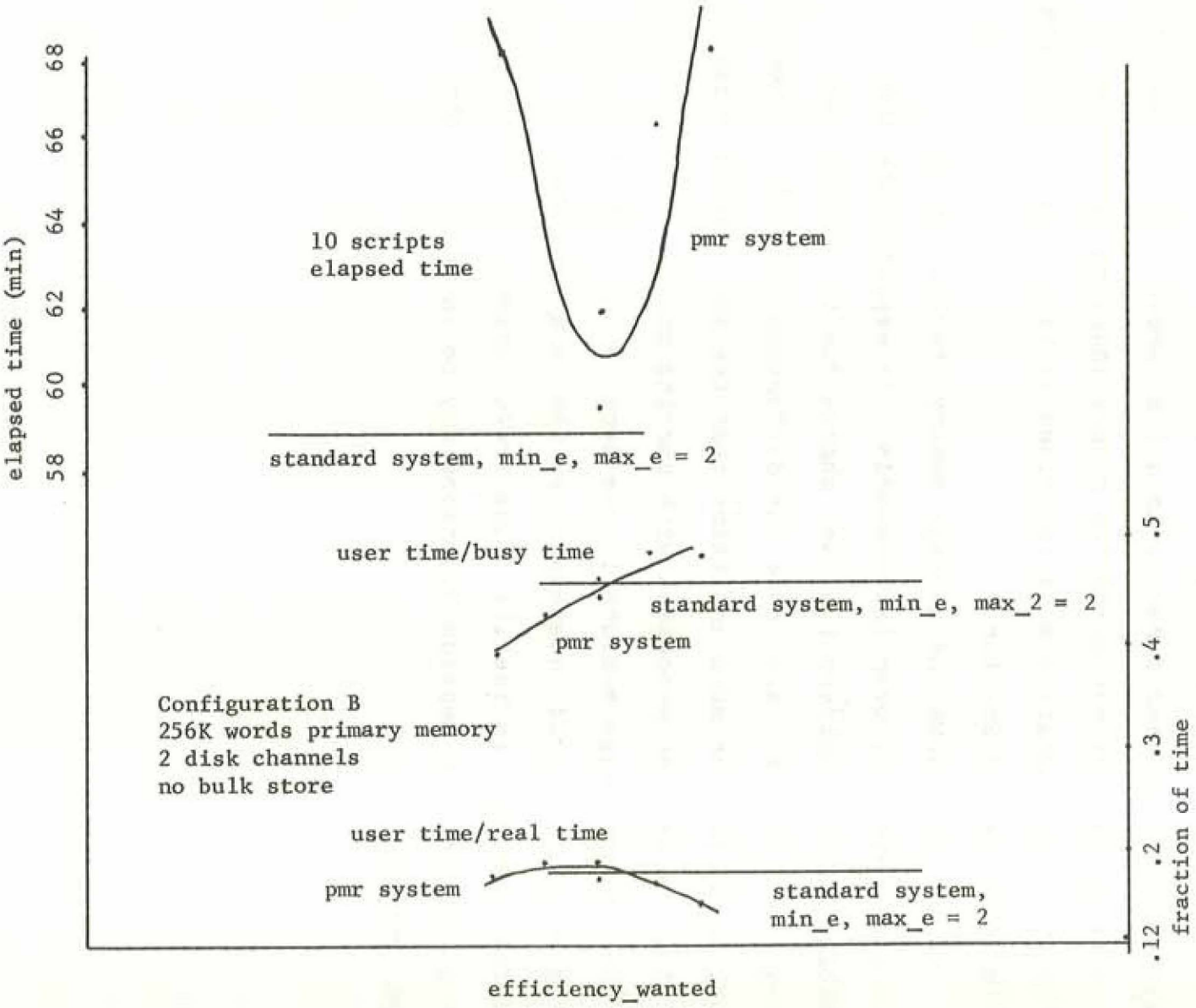


Figure 4.3.2
pmr system performance on configuration B

4.4 Memory usage charging

In Section 3.7 it was shown that the number of virtual PTW AM misses could be used directly as a time product memory charge. Two experiments were run using the memory charging algorithm proposed on configuration B as described in Table 4.2.1, with the loading outlined in Section 4.1.

Table 4.4.1 shows the average memory charges for two translators averaged over ten commands. As expected the ALM assembler has a considerably lower charge due to its smaller memory requirements. Also note the difference in R(16) between translators, further substantiation that the pmr estimator can discriminate between processes with varying pmr's.

The memory usage measure in use here is theoretically configuration and load independent as the data from the associative memory is ideally noise free. However for the reasons outlined in Section 2.5 there may be inequity in the usage charge.

translator	average virtual cpu time (sec)	memory charge: average virtualized PTW AM misses (/1000)	R(16) (misses/msec)
ALM	4.39	48.16	10.96
PL/1	4.96	123.65	24.91

Table 4.4.1

Memory usage of PL/1 compiler and ALM assembler averaged over 10 typical commands

4.5 Self tuning results

Figures 4.5.2, 4.5.3, and 4.5.4 depict typical system behavior with the self tuning algorithm in Section 3.8 enabled. Experiment 12 was conducted on configuration C as described in Table 4.5.1, with the load described in Section 4.1. The settings for the feedback parameters and average observed behavior are also given in Table 4.5.1.

Figure 4.5.2 shows the pmr self tuning algorithm trying to keep efficiency_got, the observed ratio of user time to busy time, the same as efficiency_wanted by adjusting ws_coff. Figure 4.5.3 shows ws_coff plotted against time of day.

Figure 4.5.4 shows the system trying to maintain tsr_got, the percentage of processes going blocked in their first time quantum, to percent_complete, the system administrator specified goal.

The self tuning algorithm was developed primarily to hunt for a value of ws_coff that targeted efficiency_wanted into corresponding system behavior. It is obvious from Figures 4.5.2 and 4.5.4 that there was a great deal of error looking for the proper values of system tuning paramters.

Configuration

192K words primary memory
124 pages available to users
2 disk channels
no bulk store device

Feedback parameters
---

tsr_wanted	75%
efficiency_wanted	.30
delta_to_tefirst	.050 seconds
delta_to_ws_coff	.1
tune_interval	30 seconds

Observed behavior
---

	mean	standard deviation
tefirst	.171	.082
ws_coff	.289	.186
tsr_got	73.3	17.66
efficiency_got	.325	.108

Table 4.5.1
Parameter values and
configuration information for
Experiment 12

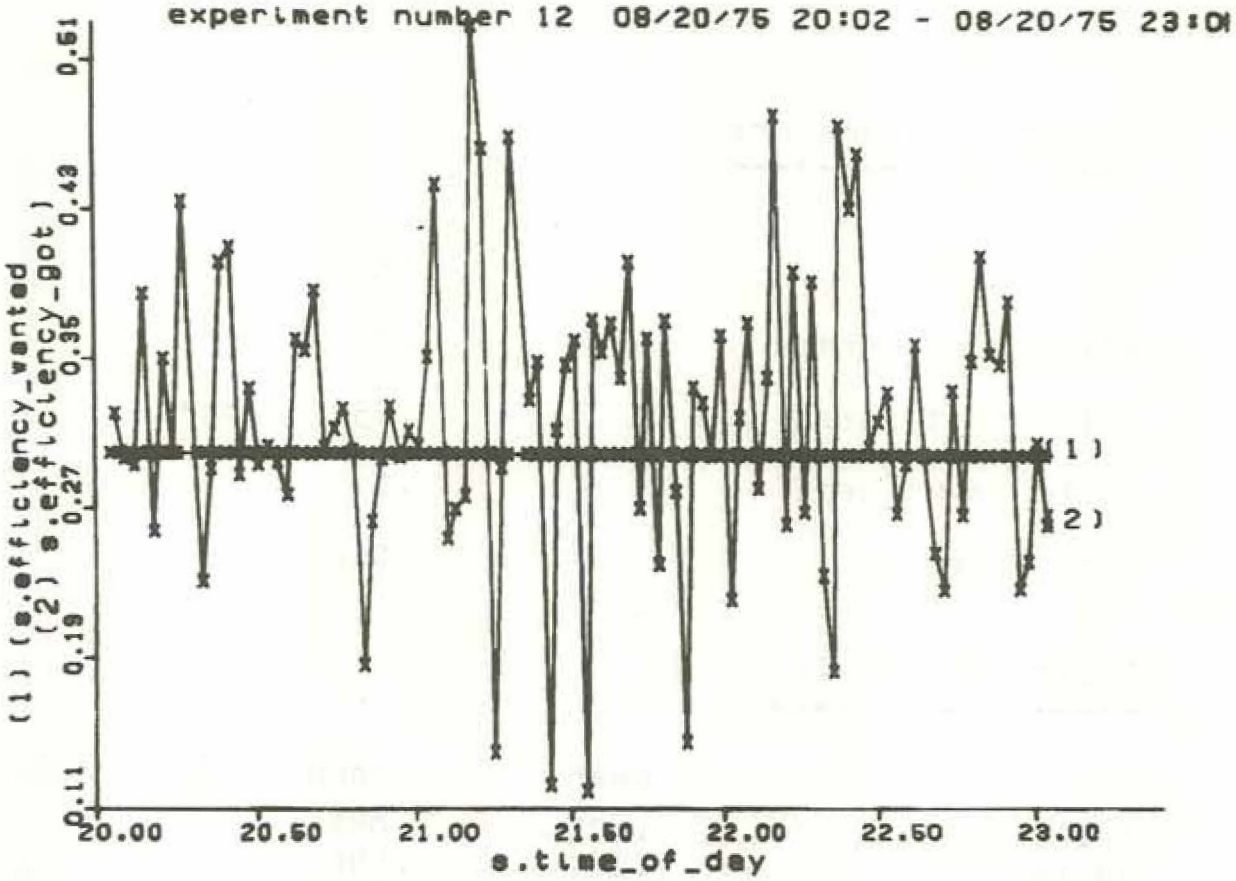


Figure 4.5.2

Dymanic response of the system to the pmr self tuning algorithim

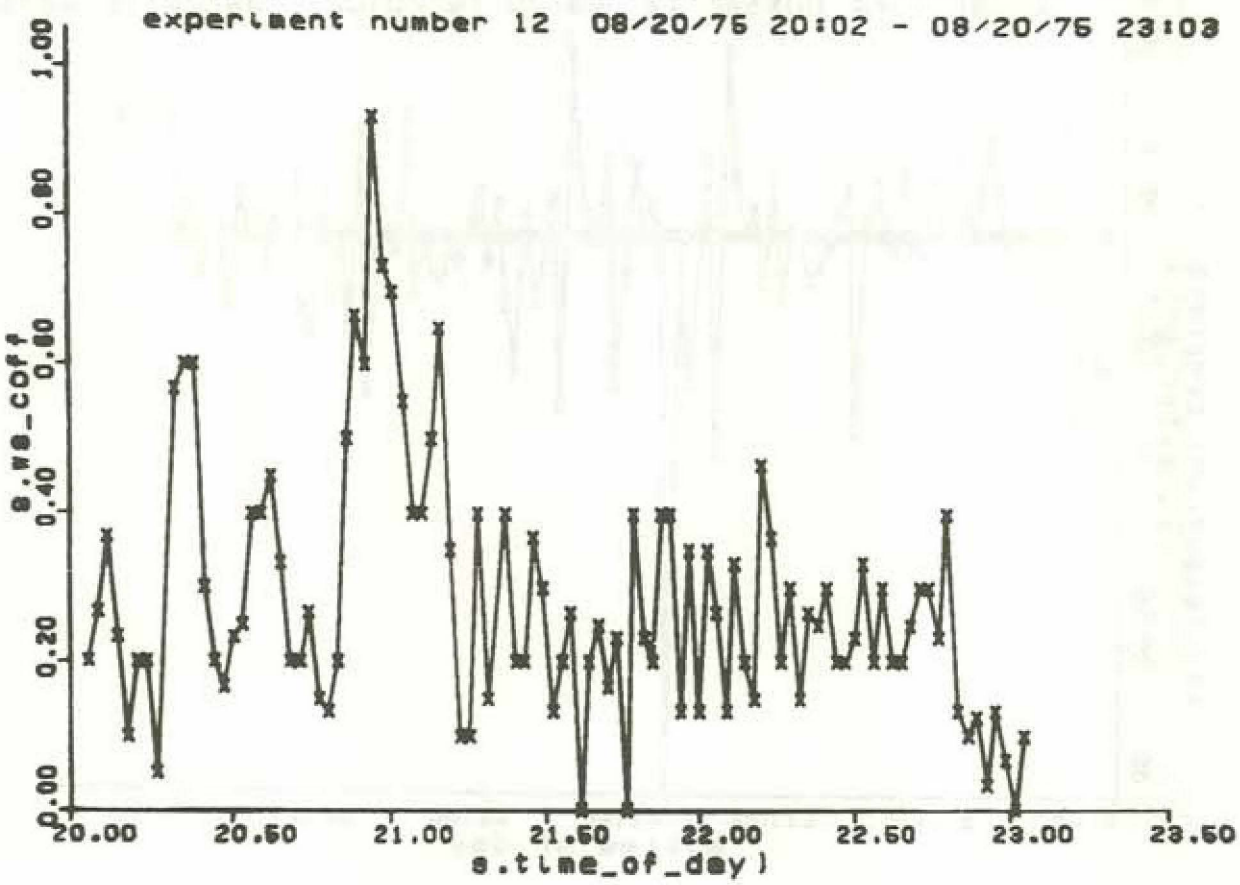


Figure 4.5.3

Control variable ws_coff
plotted against time of day

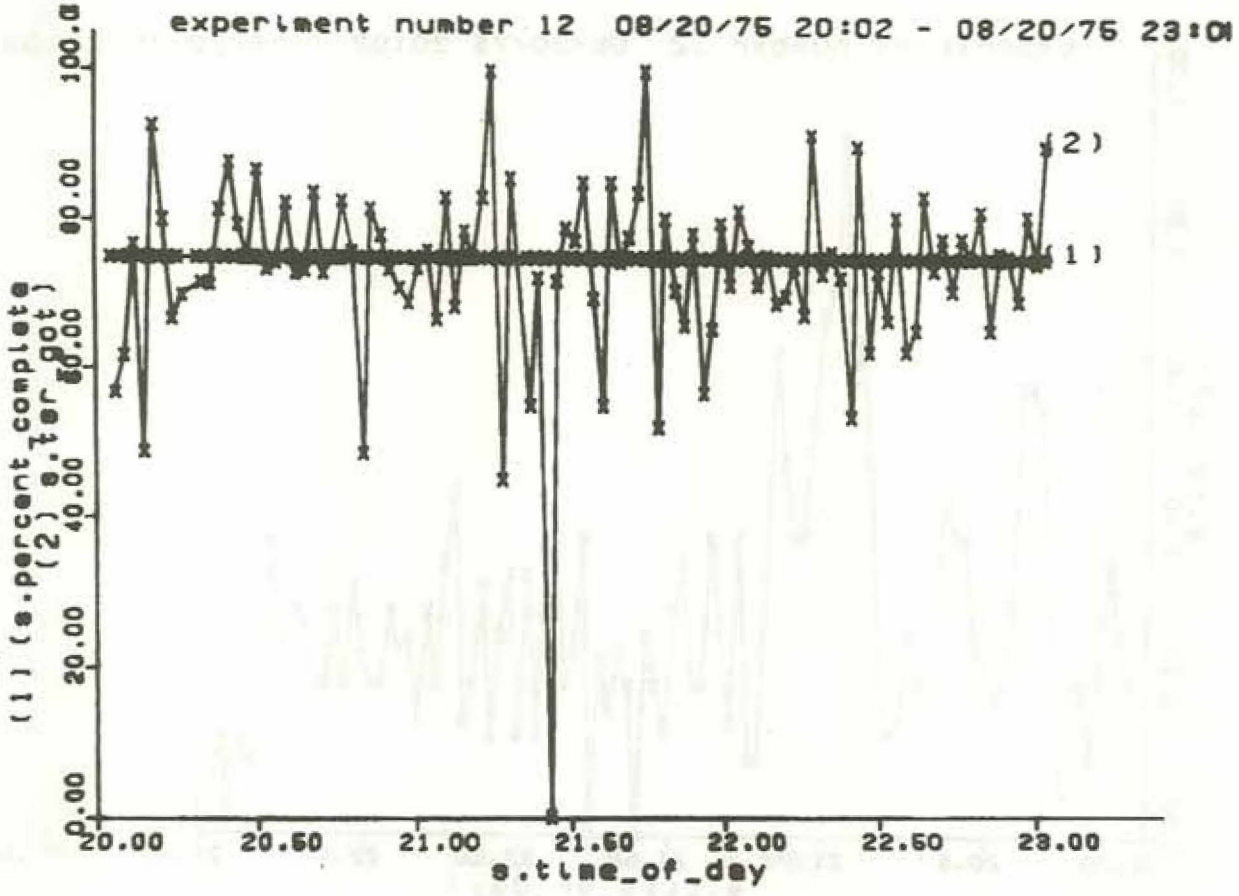


Figure 4.5.4

System task switching
ratio under the influence of
self tuning

4.6 Analysis of error

The M.I.T. performance test is far from a perfect measure of the performance of a given Multics system. For identical experimental runs variance of elapsed time of up to ten percent was noted. Thus critical experiments were run at least twice to provide a more stable measure.

The scripts determine for the most part was is being measured, as they provide as a whole a set of resoure requirements. The scripts in use for the experiments described in this thesis were designed by the M.I.T. Information Procesing Center, and were used unmodified.

Thus the results reported in this thesis must be viewed within the limitations through with they were obtained. A typical command employed in the M.I.T. scripts used 100 milliseconds of cpu time, making it difficult to assess a process' pmr before it changed. Section 5 will discuss the types of system loads that would derive greater benefit from pmr estimation for eligibility control.

5. Conclusion

5.1 Summary of thesis proposal and results

We have seen how a simple hardware addition to measure the page table word associative memory miss rate in the 6180 processor allows a process' primary memory requirements to be estimated. Assumptions were made about a process' behavior that enabled this estimation, and they were outlined.

Experimental results showed that the primary memory requirement estimator was indeed discriminating between processes with differing primary memory requirements. However, no incremental performance gain could be demonstrated using the estimated primary memory requirements (pmr) to control the level of multiprogramming in the system.

The major result of this thesis was that the pmr estimator proposed simplified the tuning of the system for an optimal level of multiprogramming. Experimental results showed that the system stayed in tune across a large change in configuration. It was conjectured that the system probably would also track changes in the profile of its load and adjust accordingly.

The problem of charging for memory usage was also explored. It was shown that the number of misses in the page table word associative memory could be used directly as a time product memory charge.

5.2 How a net performance increase might be realized

At the outset of this thesis research it was hoped that providing better estimates of a process' pmr than were currently available would increase the percentile throughput of the system. As explained in Sections 1.2 and 1.3 the problem of estimating a process' pmr has traditionally been associated with performance gains.

Most of the theoretical work in this area has dealt with a process running under steady state conditions, and does not consider transient behavior. However the load used for the experiments in this thesis was extremely transient in nature. The load was designed to reflect what a general purpose time sharing system saw under a production load at M.I.T..

It is difficult to utilize a good estimate of a process' pmr if it is changing very dynamically. Thus a contributing factor to the failure to demonstrate a performance gain using the pmr estimator proposed was the nature of the load used for assessing the estimators effect. If workload characteristics were such that processes began to reach steady state in their referencing behavior then estimates of memory requirements would be of greater utility.

5.3 Future work

Naturally this thesis has left great amounts of territory

unexplored.

As described in Section 2.5 the accuracy of the first order estimation is predicated on the form of p(x). A simulator for the the 6180 could be implemented, allowing p(x) to be calculated for various commands and programs. With typical forms of p(x) the limitations of the estimation would be better understood, and the accuracy could be quantitatively described.

Software was used to compensate for superfluous misses in the PTW associative memory, inducing additional overhead on each page fault and interrupt. Most of the compensation could be performed in hardware, making the overhead cost of calculating a process' pmr completely negligible.

The conjecture that the pmr system described in this thesis will track changes in the referencing characteristics of the system's workload and optimally set the degree of multiprogramming should be verified.

The utility of pmr information in a dynamic process environment should be investigated. If a process' past behavior can not be used effectively to predict the process' future behavior then investment in a pmr estimator only lets one control the average level of multiprogramming. No optimization would be possible that required distinguishing between processes with varying requirements.

Finally, the stability of a self tuned system should be examined. Typical feedback system problems have to be dealt with to reduce erratic system behavior.

REFERENCES

[A1] Alexander, Mike, Private Communication, (1975).

[C1] Corbato, F. J., "A Paging Experiment with the Multics System" in In Honor of P.M. Morse, M.I.T. Press, Cambridge, Massachusetts, (1969), pp. 217-228.

[D1] Denning, Peter, "Resource Allocation in Multiprocess Computer Systems", Ph.D. Thesis, M.I.T. Department of Electrical Engineering, May, 1968.

[D2] Denning, Peter, "The Working Set Model for Program Behavior", Communications of the ACM 11, 5 (May, 1968), pp. 323-333.

[G1] Greenberg, Bernard, "An Experimental Analysis of Program Reference Patterns in the Multics Virtual Memory", S.M. Thesis, M.I.T. Department of Electrical Engineering, January, 1974.

[M1] Mattson, R. L., et. al., "Evaluation Techniques for Storage Hierarchies", IBM Systems Journal 9, 2 (1970), pp. 78-117.

[M2] Morris, James B., "Demand Paging Through Utilization of Working Sets on the MANIAC II", Communications of the ACM 15, 10 (October, 1972), pp. 867-872.

[M3] Morenc, Rodger S., "Aware", MJDM-5.0, Computer Sciences Department, Ford Motor Company, Dearborn, Michigan.

[O1] Organic, E. I., The Multics System: An examination of its structure, M.I.T. Press, Cambridge, Massachusetts, 1972.

[R1] Rodriguez-Rosell, Juan and Dupy, Jean-Pierre, "The Design, Implementation, and Evaluation of a Working Set Dispatcher", Communications of the ACM 16, 4 (April, 1973), pp. 247-253.

[R2] Roach, Rodger A., "Revision of Multics Performance Tests", Internal Honeywell Memo MTB-126, Cambridge Information Systems Laboratory, Honeywell Information Systems, Inc., Cambridge, Massachusetts.

[R3] Reed, David, "Estimation of Primary Memory Requirements of Processes in Multics", S.B. Thesis, M.I.T. Department of Electrical Engineering, June, 1973.

[S1] Saltzer, J. H., "A simple linear model of demand paging performance", Communications of the ACM 17, 4 (April, 1974), pp. 181-185.

[S2] Sekino, Akira, "Performance Evaluation of Multiprogrammed Time-Shared Computer Systems", Ph.D. Thesis, M.I.T. Department of

Electrical Engineering, September, 1972.

[S3] Schatzoff, M. and Wheeler, L. H., "CP-67 Paging Priority Dispatcher", Report No. G320-2088, International Business Machines Corporation, Cambridge Scientific Center, Cambridge, Massachusetts (March, 1973).

[S4] Saltzer, J. H., "Traffic Control in a Multiplexed Computing System", Sc.D. Thesis, M.I.T. Department of Electrical Engineering, July, 1966.

[V1] IBM Virtual Machine Facility/370, Introduction, Form No. GC20-1800, IBM Data Processing Division, White Plains, New York (1972).