To: MTB Distribution

From: T. Casey and R. Mullen

Date: 14 May 1980

Subject: Answering service performance study

1. INTRODUCTION

This memo describes some answering service performance problems that have been observed on several service systems (both internal Honeywell and customer systems). It outlines our initial approach to the study of these problems, and the search for solutions to them.

Briefly, the problem seems to be that on systems with large numbers of logged in users, the resulting frequency of logins and logouts saturates the initializer process, which is unable to keep up with the workload. As configurations grow even larger, this problem will become more serious, and it must be understood and dealt with. The study is still underway. Preliminary results have suggested new directions for the study, and we expect this feedback process to continue. Detailed results and proposed solutions will be described in later memos. At the time of this writing, three such memos are planned:

I - Modifications to the supervisor to allow the initializer to be given preferential treatment by page control and the disk dim.

II - Identification of the parts of the answering service that are the greatest consumers of resources; estimates of the benefits and costs of modifying those parts.

III- Estimates of the maximum throughput (and thus, the largest configurations) that could be supported, given various combinations of the modifications described in I and II.

The expected contents of these memos are discussed in more detail below. The reason for publishing this memo now, before the results are available, is to inform members of the project that the study is being done, and to solicit comments and suggestions that can be acted on while the work is still in progress.

2. THE PROBLEM

On heavily loaded systems, the answering service seems to be a bottleneck. Users experience annoyingly-long wait times between

Multics Project internal working documentation. Not to be reproduced or distributed outside the Multics Project.

dialing up and receiving a greeting message, between typing the login command and being prompted for a password, and between typing a password and having a process created. They experience these long waits even when they describe the response time as "not so bad" once they are logged in.

Some users experience extremely long wait times during login, while seeing other users, who started logging in later, finish logging in before them.

Logouts are also affected: there are long wait times between the typing of the logout command and the printing of the "logged out" and "hangup" messages. Some users are in the habit of hanging up the phone instead of logging out, to avoid this delay; this is a bad habit now that we have the process preservation facility.

The operators see the problem in several ways: delayed response to operator commands, inability to enter operator commands, and lack of any output on the operator console for long periods of time. The operators sometimes interpret these symptoms as evidence that the system is hung or crashed, and they return it to BOS unnecessarily.

Some of these symptoms are probably caused by straightforward bugs in the code, rather than obscure, intrinsic inefficiencies in the design. But they are observed most often on heavily loaded systems, so we will treat them as part of the performance problem until evidence proves otherwise.

3. OBJECTIVES OF THE STUDY

3.1 Understanding the Problem

The software that runs in the initializer process includes some of the most complex subsystems in Multics. These include the Answering Service, the Message Coordinator, the System Control subsystem, parts of RCP, Reconfiguration software, and probably some other things not included in this list. And that uncertainty is part of the problem.

This software has evolved over the last 12 years. It has been modified by many different people, most of whom are no longer part of the project. It continues to be modified independently and simultaneoulsy by several people. There is no program logic manual (PLM), and there is no one person who has enough overall and detailed knowledge of the software to write a PLM without spending a great deal of time reading the code to see how it actually works.

One piece of evidence of our lack of complete understanding of this software is the following: in the early stages of this study, a built-in answering service metering facility was designed and implemented. To save time in the initial coding,

only those parts of the answering service that were thought to be the greatest resource consumers were metered. The results of this metering accounts for only about 50% of the cpu time and page faults that were used by the initializer process during the metering interval. In other words, the other 50% of the resources used by the initializer process were used by programs that we are either completely unaware of, or have significantly underestimated the cost of executing.

So our first objective is to find out what really goes on in the initializer process: what processing is done, with what frequency or in response to what conditions, and what it costs.

3.2 Identifying Solutions

Our second objective is to identify processing that can be eliminated entirely, moved into some other process, or optimized to consume less cpu time and paging resources. The cost of these changes must also be estimated.

3.3 Capacity Planning

Our third objective is to produce some estimates of the maximum answering service throughput that can be achieved, and the corresponding largest configurations that can be supported, under various circumstances. These circumstances are made up of various combinations of the potential improvements identified by the second objective (each combination having a different total cost and a different net effect on throughput). The case of no improvements from the present system will be included, to provide a baseline.

Then we will make some assumptions (based on data from service systems) about the average length of a login session, and the resulting frequency of logins and logouts as a function of the number of logged in users.

From these figures we should be able to identify points, as Multics configuration sizes are increased to support greater numbers of users, where answering service throughput will become a limiting factor. For each such point, some combination of performance improvements will be identified, that will raise answering service capacity to a new, higher limit.

Various sequences of these limit points could be obtained by varying the order in which performance improvements are applied. But it seems likely that they will be applied in the order of increasing implementation cost, taking cost effectiveness into account as well. That is, the most costly to implement will be put off until last, in hopes that changing circumstances will remove the need for their ever being done.

Marketing requirements, including both configuration sizes and time frames, will also have to be taken into account in choosing the order in which improvements will be made.

It is important to recognize that the capacity and cost estimates will be derived from combinations of other estimates, each having some degree of uncertainty. The uncertainty of the final estimates will be greater because of the cumulative effect of the uncertainties of all the estimates that go into them. Obviously, the usefulness of the final estimates will vary inversely with their uncertainty. The accuracy limitations must be taken into account when using these estimates for product planning or task scheduling purposes.

In other words, if we estimate the average session length to be 45 minutes but it might actually be 30 minutes or 60 minutes, and we estimate that the effect of a given performance improvement will be a login rate of 250 per hour but it might actually be 225 or 275, and we estimate the cost of the improvement to be 2 man months but it might actually take 1 or 3, it is not useful to say that we think we can achieve a 188 user system at a cost of 2 man months when it might actually be a 112 user system at a cost of 3 man months or a 275 user system at a cost of 1 man month.

Even if out initial capacity estimates are not accurate enough to be used for planning purposes, they should be worth publishing, along with the methods used to derive them. This will serve to identify all of the parameters relevant to answering service capacity planning. We should be able to obtain more accurate and useful capacity estimates as we learn to estimate the individual parameters with increasing degrees of confidence.

4. PRELIMINARY ANALYSIS

Before beginning the current study, it was possible to make some statements and hypotheses about answering service performance, based on some earlier experiments and observations, and our understanding of how the software works.

Note that most of the numbers given in this section are typical values observed by the authors on various systems, in the course of other work. They are not the results of careful measurements in controlled experiments. (Such measurements are planned as part of this study.) The numbers are presented here to give the reader a feeling for their magnitude, and to suggest how close we may be to our capacity limits.

The priority scheduler allows for the initializer process to be allocated an arbitrarily large percentage of cpu time (up to 100%). But, being a single process, it can obviously never use more than 100% of a single CPU. So, as the number of CPUs in a configuration increases, a decreasing fraction of the total cpu time is available to the initializer process.

Further, the initializer process, like any other Multics process, can take only one page fault at a time. The system does not start to read a page into main storage from the disk until a process attempts to reference it and finds it missing. At that point, the process can do nothing until the page is available. (2) Each page fault forces a process to wait for some length of real time, equal to at least the sum of the seek, latency, and transfer times of the disk unit, before being able to resume computation. Thus the greatest rate at which one process can use cpu time is a fraction of 1 CPU. The fraction is a function of the ratio of cpu time to page faults for that process; the more page faults the computation requires, the smaller the fraction.

It seems, then, that paging is the cause of the initializer throughput problem. It is known that paging performance is very sensitive to tuning and configuration, and all of the following discussions assume a properly tuned and configured system. (3)

Page wait time is composed of I/O time and queue time. I/O time is composed of seek time, rotational latency time, data transfer time, and a possible wait time if a physical channel is not available when needed. Queue time is the time spent waiting for the device and a logical channel to be available so that an I/O operation can be started. When an I/O request is placed in a previously empty queue, it might have to wait for a previously started I/O to complete. When there are other I/Os in a queue, they are performed in an order that minimizes seek distance by choosing the one that has the shortest seek distance from the current arm position. Separate queues are kept for page writes (which no process is waiting for) and page reads and VTOC reads and writes (which processes do wait for). The latter are given higher priority.

The following are the published times for model 451 and 500 disk units, respectively (all figures are milliseconds (msec)): latency: 8.3 for both; average seek time: 30 and 25; 1-page transfer time: 5.1 and 3.4. Adding these times gives expected

---

(2) This limitation could be removed by a pre-paging facility, which would allow a page read to be started before a process faults on the page. The mechanism to read a page that has not been faulted on is available in the supervisor. The problem is to find a good algorithm for correctly anticipating the need for certain pages. This is discussed further in section 5.2.3.

(3) Tuning and configuration will be discussed in a separate memo, not yet published, tentatively titled Configuring and Tuning Multics Systems. Briefly, there must be an adequate amount of memory for each CPU, and an adequate number of disk channels and spindles to support the resulting paging traffic. Further, tuning parameters, such as max_eligible, must be set to reasonable values for the hardware configuration.

average page I/O times of 43.4 and 36.7 for 451 and 500 units, respectively.

However, the device_meters command shows typical I/O times of 30 to 35, with the lowest we have seen being about 28. This can be explained by assuming that Multics does not cause the kind of random disk arm movement that the average seek time figures are based on. The Multics average seek times of 15 to 24 are probably caused by seek optimization and the fact that successive pages of a segment are often on the same or adjacent cylinders. This hypothesis is consistent with the observation that I/O times are slightly higher (35) on heavily loaded systems, where page read priority overrides seek optimization more often, and successive page faults are more likely to be from different processes for different segments (causing arm movement), rather than from the same process for successive pages of the same segment.

The device_meters command also shows page wait times of around 50 on moderately loaded systems (with I/O times around 30), and 70 or higher on heavily loaded systems (with I/O times around 35). Thus, increased loads drive I/O time up from 30 to 35 and queue time up from 20 to 35.

An average page wait time of 70 msec means that the highest rate at which a process can take page faults is about 857 per minute. And this assumes a negligible amount of processing between each pair of page faults. As the amount of processing between page faults goes up, the maximum rate at which a single process can take page faults falls even lower.

On one particular system that was experiencing initializer performance problems, the following observations were made: logins (4) were occurring at a rate of about 3 per minute; users were experiencing very long delays in logging in (i.e., the demand was calling for a higher rate of logins); and the initializer process was taking about 833 page faults per minute, or one every 72 msec, and was hardly ever going blocked.

Three logins and logouts per minute and 833 page faults per minute gives 278 page faults for each login-logout pair. Separate measurements, obtained by flushing main memory before each login and each logout, have indicated that about 300 different pages are touched for each login, and about 200 for each logout, for a total of about 500 pages for each login-logout pair. The discrepancy between the predicted 500 different pages touched and the observed 278 pages faulted on can be explained by assuming

(4) Assume that, in the steady state, the number of logged in users remains fairly constant, and thus logins and thus logouts occur with equal frequency. Then, for convenience, we will talk about logins per hour, rather than login-logout pairs per hour.

that some pages are used by both login and logout functions, and so there are fewer than 500 different pages touched during a login-logout pair, and that some of the involved pages are pinned in memory by heavy use and are not faulted on.

To summarize, a prediction based on the required number of page faults per login and the maximum page fault rate gives a maximum login rate of slightly less than 2 per minute, while a login rate of 3 per minute is observed on a heavily loaded system where the initializer is running continuously, has some of its pages pinned in memory by heavy use, and is unable to keep up with the demand (which is for a login rate higher than 3 per minute).

Thus it appears that, with the current software and the current configuring and tuning criteria, 3 logins per minute is the limit, independent of the size of the configuration. That's 180 logins per hour. If the average session length is one hour, this means that a 180 user system is the largest that can be supported. An average session length of 45 minutes implies a 135 user system; while a 30 minute session length implies a 90 user limit. (5)

We recognize that dealing with maximum throughput and average session length results in optimistically high capacity estimates. Queueing theory shows that 100% of the theoretical system capacity can not be used, because there will be periods of relative idleness and periods of overload when the delays will be unacceptable to many users. This subject will be dealt with in more detail in Memo III. The preceding discussion serves to give the reader a feeling for the magnitude of the numbers, and how close to the limit we are.

5. SOME POSSIBLE SOLUTIONS

Since the bottleneck is based on the limited rate at which a process can take page faults, (6) the possible solutions fall into three categories: increase the rate at which page faults can be taken; decrease the number of page faults that the initializer process must take in order to complete a given amount of

---

(5) The equation for max users (maxu) is:

maxu = max_logins_per_hour * average_session_length_in_hours

(6) There is no cpu time bottleneck at present, and none is anticipated. The initializer process currently uses about 20% of 1 CPU, and CPUs in the future are expected to be many times faster than current ones. We do not expect to be able to raise the limit imposed by paging to the point where CPU time becomes the limiting factor. Waiting on some of the system locks might be the next bottleneck that we encounter; this will be investigated, but we do not see it as a problem in the near future.

processing; or split up the processing among several processes, allowing several page fault sequences to be going on in parallel. Solutions can also be categorized as hardware or software changes.

5.1 Hardware Solutions

Adding hardware to a configuration (memory, or disk channels and spindles) is always available as an effective, short term solution to this problem. But it is expensive, and our current thinking is that it is not cost effective. The assumption is that the hardware is adequate to support the user load (although perhaps just barely so), and only the initializer process is in trouble.

Adding memory will allow the initializer process (as well as other eligible processes) to keep more of its pages in memory. This will allow it to accomplish a given amount of processing while taking fewer page faults. Further, since all other processes will be taking fewer page faults, disk traffic will be lighter, and page faults will be able to be taken at a somewhat higher rate. Adding 1MB (7) per CPU to a configuration that has 3MB per CPU would probably produce a significant improvement. Adding additional 1MB increments per CPU, for a total of 5MB or more per CPU would produce successively diminishing returns. At a purchase price of $50,000 per 1MB, this is a very expensive solution, and is probably not to be considered seriously, although the throughput increase produced by each 1MB increment should be measured before a final judgement is made.

Adding disk channels and spindles will tend to decrease the I/O queue length, allowing page faults to be taken at a somewhat higher rate. However, there are limits to the use of this solution. Newer disk drives are tending toward greater storage capacity per spindle, which tends to decrease the number of spindles in a configuration. Seek times and data transfer rates are not getting significantly faster. Further, increasing the total paging transfer rates by any means (additional channels or higher transfer rates on individual devices) will tend to overload IOM capacity, requiring the addition of an IOM to the configuration. This seems to be an even more expensive solution than adding memory. This alternative will not be considered any further.

5.2 Software Solutions

Software solutions fall into the following categories:

1) Arrange for pages used frequently by the initializer process to remain in memory (either by wiring or pinning) instead of

(7) 1MB = 1 Million Bytes = 256K words

being paged out and in repeatedly.
2) Modify the supervisor so that the initializer can be given priority in page reads, and can thus take page faults at a higher rate (while other processes take them at a somewhat lower rate).
3) Add a pre-paging facility to the supervisor, to allow a process to notify the system of its anticipated needs for certain pages, so they can be brought into memory in advance of references to them; implement suitable controls on the use of this facility. Modify the answering service to use this facility for some of its data segments and perhaps also its procedure segments.
4) Modify the answering service (and other software that runs in the initializer process) so that either it touches fewer pages while accomplishing its processing or it can be run in several processes simultaneously.

5.2.1 Page Pinning

The first alternative can be accomplished today, by identifying and wiring the N pages most frequently used by the initializer process. (8)

Consider the following thought experiment: add 100 pages of memory to the configuration and simultaneously wire the 100+x pages most frequently used by the initializer process, where x is a number (<100) to be determined experimentally. With x=0, the initializer's throughput will be increased significantly because a large fraction of its working set will be wired. Further, throughput of the other processes will be increased slightly since the initializer will not be competing so heavily for the other (original) memory in the system, nor will it be contributing so much to the paging traffic. Then increase x until the throughput of the other processes is back down to what it was before the experiment started. At this point, a certain amount of additional initializer throughput has been purchased, at a cost of $20,000 worth of memory, and the performance of the other processes has not been affected.

The disadvantage of this is that the set of pages to be wired must be identified correctly (and must be re-identified whenever the answering service software changes); an appropriate size for the set must be chosen (100+x was an arbitrary number); and the page frames holding the wired pages are unavailable to other processes even during (possibly long) time intervals during which the initializer process is not using those pages.

Thus it seems that this method does not make the most effective use of the extra memory, and it is awkward to implement and

(8) One customer site is currently running with most of the text portion of bound_user_control_ wired.

maintain, possibly leading to a sub-optimal size and content of the set of pages being wired.

The paging system attempts to make the most effective use of memory by keeping the most recently used pages in memory. Rather than completely overriding it by wiring certain pages, it might be better if we could tell it to give higher preference to certain pages - that is, to allow them to remain in memory longer than normal pages, before being paged out for lack of use. This would provide some of the benefits of wiring pages, but it would retain the system's ability to adjust automatically to changing conditions. The size and content of the set of pinned pages would be continually adjusted, and the extra memory would be available to other processes when not needed by the initializer.

A number of different algorithms were considered for identifying the pages to be pinned and determining how long they should be pinned. Each had its good and bad points, including implementation cost, expected benefits, and potential for undesirable side effects. Several were implemented and tested. The details of this work will be given in Memo I.

5.2.2 Paging I/O Priority

Recall that page wait time is composed of I/O time and queue time (each about 35 msec on a heavily loaded system), and that the queue times of individual I/O operations are affected by seek optimization, whereby the next I/O operation is selected from the queue based on which one will result in the shortest arm movement. Further, seek optimization is overridden to give priority to page reads and VTOC I/Os, which processes wait for.

A change to the disk dim to give I/Os that the initializer process is waiting for higher priority than those that other processes are waiting for would reduce the queue time component of the initializer's page wait time down to that portion of queue time caused by the need to wait for a previously started I/O to complete.

If that time is, on the average, half of the average I/O time, then this change will reduce the initializer's average page wait time to 1.5 times the average I/O time. The effect that this will have on the rate at which the initializer can take page faults will depend on how much the system average page wait time exceeds 1.5 times the system average I/O time. Experiments with a disk dim containing these modifications will be described in Memo I.

5.2.3 Pre-paging

Bringing pages into memory in advance of references to them would allow a process to continue execution, rather than waiting for one page wait time, after its first reference to a given page. This would allow the process to complete its computation in less

real time. Reducing the time that processes spend waiting for page faults would reduce MP Idle and NMP Idle, and thus would tend to increase total system throughput, provided that there is enough I/O capacity to keep up with the increased paging I/O (9) and enough extra memory to hold both the set of pages currently being referenced and the set of pages being read in anticipation of their being referenced.

A mechanism to correctly anticipate all page references is clearly impossible, since it would require predicting the future with certainty. The best that we can hope to achieve is some good guesses about some of the pages that will be referenced.

It is possible to envision some page reference prediction algorithms that could be built into the system and operate automatically for all processes. (10) However, it seems clear that really good predictions can only be made by the application program (11) as it makes decisions that will determine which pages it will touch next. Building paging decision making into an application program seems to be violating the spirit of a virtual memory system, which is supposed to relieve the application programmer of the need to be concerned with memory limitations. However, until we have hardware that is both fast and cheap enough to support this ideal, it might be necessary for us to provide the option of having application programs anticipate their page references so that Multics can achieve the necessary throughput cost-effectively enough to enable us to sell systems.

We will investigate the benefits of pre-paging to the answering service by implementing a gate to request the reading of specified pages of a specified segment. We will place calls to this gate at points in the answering service where page references can be anticipated, and measure the answering service throughput increase and the impact on other processes.

---

(9) There must be an increase in paging I/O, since the same set of pages must be brought into memory in a shorter length of real time.

(10) These might be based on such things as built-in assumptions about the pattern of page references that will occur in an object segment after one of its entry points is called, or on the observation of a reference pattern such as successive pages of a segment and the assumption that the pattern will continue.

(11) From the viewpoint of the supervisor, the answering service is just another application program. While our objective is to increase answering service throughput, a pre-paging facility would be available to customers' application programs, controlled by access to a gate.

Points where page references can be anticipated include passes through entire tables, such as the 3 user tables (answer table, absentee user table, and daemon user table) at accounting update time, and decisions that cause large blocks of code to be executed, such as all the code that will be used by the login command. It might also be useful to reorganize some decision making so that page references can be anticipated far enough in advance that the pages will be in memory when they are needed.

5.2.4 Answering Service Modifications

Two classes of answering service modifications are being considered: redesigning and recoding certain functions so that they can be accomplished while touching fewer pages; and splitting up the answering service so that its various functions can be performed in different processes, in parallel.

The most cost effective of the former have already been done. Linear searches of the SAT and PDTs at login time were replaced by hash lookups in MR7.0, and a linear search of the answer table was eliminated for about 95% of all logins, in MR8.0. Further potential improvements become increasingly hard to find, more costly to implement, and less effective. A detailed study of the behavior of the answering service software, using trace, page_trace, and built-in meters, will be described in Memo II. It is not possible to predict the cost and benefit of any potential improvements that might be discovered by this study, but we consider it unlikely that any more really significant and inexpensive improvements will be found.

Splitting up the answering service into several processes is an obvious and very popular suggestion. Conceptually, there is no difficulty with this. But doing it in general (12) would require a complete redesign and rewrite of the answering service - a project that could require many man-years of effort.

The problem with the current code is that there are many locks that do not actually exist, but are implied by the facts that only the initializer process modifies certain databases, and it operates by performing a sequence of indivisible operations. Each indivisible operation is triggered by an event-call wakeup. A process can only receive event-call wakeups when it is blocked. The answering service is careful to mask off event-call wakeups before entering any code sequence that might cause it to go blocked. Failure to do this is considered a bug.

---

(12) By "in general" we mean having multiple processes, each running the entire answering service, and each able to do all of the jobs presently done by the single process answering service. Moving selected parts of the answering service into other processes is discussed below.

To allow the current code to run simultaneously in several processes, it would be necessary to identify all of these implied locks, and implement them explicitly. It is unlikely that this could be done cost effectively. The code was unstructured to start with and has become more so with each modification. It could possibly take as much time to identify and implement all of the implied locks as it would have taken to rewrite the *answering* service completely, with locks designed in. And the result would probably be buggy for years, and harder to maintain than it is now. (13)

It is possible, however, to see how certain specific functions could be split out and run in separate processes. Probably the easiest of these is process creation. The program (cpg_) that fills in a structure and passes it to the supervisor to request the creation of a process could be run in a separate daemon process, communicating with the initializer via event-call wakeups. A single lock on the answer table entry, to prevent a process destruction from being attempted in one process while a process creation is going on in the other, would probably be sufficient. However, our initial metering results lead us to believe that this could move at most 15% to 20% of the initializer's paging load to the daemon process. Process destruction (dpg_) could also be moved out of the initializer process, but we estimate that this accounts for less than 10% of the initializer's page faults.

The second most difficult function to move to another process might be the message coordinator. The costs and benefits of doing this require further study.

We see no other clearly identifiable functions that could be moved entirely to another single process. (The requirement that an entire function be moved to a single process is made so that the implied locks within the function will continue to work since the indivisible operations that make up the function will continue to be performed sequentially in a single process.)

6. CONCLUSIONS

The initializer performance problem is real. It is inherent in the current implementation of the system, and is not caused by some problem specific to an individual site. It is caused by a paging bottleneck; all of our observations and analysis agree with this. With the current software and the current configuring and tuning criteria, a login rate of no more than 3 per minute can be supported. This translates to a max users figure of 150 to

---

(13) People who tend to feel more optimistic about this, on grounds that "It couldn't possibly be that hard" are urged to consider that their optimism may be based on unfamiliarity with how bad the current code is. You have to see it to believe it.

200 users, if session lengths are around an hour, and to lower max users figures as session lengths get shorter.

We forsee a need to support larger numbers of users on single systems, but we do not know how large the numbers will get. The trend toward distributed processing, with a number of smaller systems communicating via a network, might tend to reduce the need for really large systems. We need some external input to help determine the necessity for raising our max users limits to any given level.

We see some ways to raise answering service throughput limits, but we have yet to determine the expected cost and benefits of each of these ways. The greatest throughput that we could achieve is theoretically unlimited, but the cost would be very high. (14)

We will continue to study all of the performance improvement possibilities described above, and any others suggested by readers of this MTB, to determine their costs and benefits. And we will continue to look for input to help determine the true need for single systems with max user figures significantly larger than those which we can now support.

This intensive study of initializer performance has generated, and will probably continue to generate, insights about the behavior and performance of Multics in general, which could be of benefit to other large subsystems and the system as a whole. We think it is worthwhile to continue expending resources on this study.

---

(14) This would involve complete re-implementation of the answering service to run in multiple daemon processes, each of which could do all of the jobs currently done in the initializer process. The number of daemons could be increased to support any user load, provided that the implementation is done in a way that minimizes lock contention.