A PERFORMANCE EXPERIMENT ON A UNIX SYSTEM

by

RODERICK LANE DOWNING

B.Sc., The University of Western Ontario, 1975

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

August 1979

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date Aug 8/79

ABSTRACT

A performance experiment on a PDP 11/45 running under UNIX is described. The purpose is to discover the major influences in the system and their relationships in an attempt to analyze the performance of the system and predict the effect on performance of possible hardware and workload changes.

A suitable performance parameter is developed and the workload, hardware and internal system parameters are determined. Tools are constructed to record these parameters. A controlled experiment, using a synthetic workload is then conducted. The results are analyzed using regression analysis and suitable equations are obtained. Sample applications are given. The merits of the relationships, as well as the suitability of the tools developed and methods used, are discussed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENT

First of all, I would like to thank Sam Chanson for his ideas and guidance throughout this work, for his careful readings of the thesis, and for his financial support through research assistantships. His knowledge of the subject was a great asset. I also greatly appreciate the efforts of Dr. Ito in both his suggestions and his reading of the drafts, as well as his slightly different perspective on the work.

This project would not have been possible if it were not for the generous use that was granted me of the computer facilities at the Institute of Animal Resource Ecology. For this I am most grateful. In particular, I wish to thank Bill Webb, whose knowledge of UNIX is truly amazing. He extricated me from many a baffling predicament.

Finally I wish to mention my wife, who has always given me the freedom to explore my goals and yet also keeps me from taking myself too seriously.

1. INTRODUCTION.

1.1 OVERVIEW

The evaluation of a system, product or activity is an essential step for its sound development. Thus the emergence over the past several years of such a discipline in the computer field is a healthy sign. While initially the discipline suffered many growing pains, in the last couple of years there has been much evidence of its maturing and consequently of its increasing contribution to computer science. A good synopsis of the whole field can be found in [FE78]. An overview of the more limited area of measurement and evaluation, under which this thesis falls, can be found in [SV76]. As well, the reader is referred to section 3.1 which further illuminates the current state.

Computer performance studies usually help to give insights in two ways. First of all, the studies simply answer whatever questions were posed, such as whether one design or modification is better than another, whether the addition of hardware or change in software justifies the cost involved, etc. Secondly, and probably of more importance in the long term, performance studies require or eventually produce a model of that which is being investigated. The process of establishing these models, because they cause the evaluator to seek out a simple unified picture of what often appears complex and disjoint, often produces insights which have far greater implications than the actual study itself.

Juxtaposed with the above optimism and importance of the area, is the need to relate some of the current obstacles. The performance evaluation of computers, like many areas in computer science, suffers because of two phenomena: the relative infancy of the field, and the continuing explosion of new technology. Because performance evaluation is a recently established discipline, there has been insufficient time to explore and uncover many underlying principles. This problem is compounded in that part of the scope that influences the area involves other almost as young areas of study. For instance, operating systems are one of the prime considerations in investigating the performance of a system, and yet it has only been recently that some basic principles in design methodology have started to emerge (for example see [HA70], [DE72], [HO70], [HA76]). Until that area is able to solidify its domain and functionings, the implications of the findings in performance evaluation will remain limited.

The rapid technological advances contribute to the everchanging nature of computer hardware and software and thus to the proliferation of new configurations and systems. This again makes it very difficult for any insights gained in a performance study to have very wide application. To develop any encompassing theories, it will have to transcend this level.

It should be noted that there is a way of viewing computer systems performance evaluation which states that the discipline

will have matured when, for the most part, it has disappeared as a separate field of study [FE78, p. xviii]. This view comes from the engineering perspective where the evaluation of performance is an integral part of design and implementation, not a distinct after-the-fact endeavour. Currently this is not the case in computer systems design. The fundamental principles of workload characterization and of the effects of hardware on them and the relationships between them and performance just don't exist yet. So any studies that are done on performance as affected by the eventual workload, are conducted after the hardware has been designed and implemented and layers of software placed on top. This places the evaluation much too far from the foundational layer of design and decision making(*).

This brief introduction hopefully provides enough background on the role and current state of the discipline so that the reader may be able to place this work in its proper context.

1.2 BASIC GOAL, APPROACH, AND MOTIVATION.

The basic goal of this thesis is to discover, on the PDP 11/45 running under UNIX at the Institute of Animal Resource Ecology at the University of British Columbia, the major

---

(*) Other related areas, such as operating systems [LI72], programming languages [FO78], and microprogramming [CH76, p.961] report a similar disatisfaction with the current role of hardware, again, probably indicating the immaturity of the whole spectrum of computer design.

influences in the system and their relationships, in an attempt to analyze the performance of the system and predict the effects of possible hardware changes.

The approach chosen is, first of all, to determine probable workload parameters and pertinent hardware and internal system parameters and to develop a suitable performance parameter. Tools to extract this information will be constructed. Then a controlled experiment, using a synthetic workload, is to be conducted varying the parameters and recording the values of the performance index. The results will then be analyzed to try and establish some strong relationships among the parameters. If this is successful, then the effect of changes in the workload and/or system on performance can be studied. For example, one could then determine how much the response would be degraded by adding more processes, or how much it could be improved by adding more main memory. As well, the relationships could be used to help answer comparative questions such as whether the addition of new disk hardware is more cost-effective than the addition of more main memory.

The motivation for this project is basically twofold. First, as mentioned in the previous section, the whole field of computer performance evaluation is very young and in much need of research in even the most basic issues. It is hoped that the results of this effort will contribute to the understanding of the area. And secondly, the target system is a fairly popular

one. Thus it is hoped that any practical insights or methods used may be beneficial to those who find themselves in similar circumstances.

Finally, a comment on the style and contents of this report is in order. It contains in many places, a fairly detailed account of decisions and methods employed. This will hopefully increase one's ability to judge the usefulness and appropriateness of this work. In chapters 2 and 4 a basic understanding of operating systems is assumed and a knowledge of UNIX will be helpful though not essential. Chapter 5, where an analysis of the results is conducted, requires a basic understanding of some statistical tools, most notably of regression techniques.

2. DESCRIPTION OF SYSTEM.

2.1 THE HARDWARE.

The host machine for this project is a PDP11/45. It has 56K words (16 bits per word) of main memory, of which approximately 24K words are used by the operating system. On-line secondary storage consists of two RK05 disk cartridges. These have small capacity (1.2 Mega words each) and slow transfer time (about 90,000 words/sec.). They operate under the same controller and do not currently have the capability to overlap seeks. Because of the small on-line storage, there are also two Dectape drives and one magtape drive, which are used to store the bulk of the user files and data not immediately being needed. Other peripherals include a printer, plotter, and a card reader (which is used to submit one batch job at a time). Access is gained to the system either through the card reader, or on an interactive basis through the console, eight other terminals, and one dial-up port.

2.2 OPERATING SYSTEM.

The operating system in use is UNIX (a fairly early version of V6). For a comprehensive description of UNIX, including source code, the reader is referred to [LI77]. As well, a good survey of its I/O , process control, and implementation can be found in [RI78] and [TH78]. The following paragraphs summarize a few of the areas which are of particular concern to this project.

2.2.1 Processes.

The basic unit of manipulation in UNIX is the process. While the basic concept is easily understood, a rigorous definition of a process in UNIX involves the system data structures [LI77, p.7-1]. For the purposes of this thesis, it is simply defined as a program in execution.

One of the chief features of a process in UNIX is its ability to spawn descendants via a fork mechanism. Thus one user may have several processes executing simultaneously. This realization is needed later when describing the selection of parameters for the experiment (section 4.2) and in the discussion of the synthetic workload.

The structure of a process will also be briefly described. All processes consist of three entities: program code, data, and a stack. Generally, all three of them are placed together in main memory, thus requiring only one pointer to keep track of them, and making their manipulation easier. However, it is possible to separate the program code and make it reentrant and sharable among other users. Thus many users can run the same program (such as a compiler) and only one copy of the program code needs to be in main memory, saving considerable space. As well, because it is pure code and thus never changes, the text never needs to be swapped out, which reduces the swapping load.

2.2.2 Scheduling Policies.

With most algorithms in UNIX, when a design choice must be made between simplicity and power, the former was usually favoured. This is very evident in the scheduling policies, which will be briefly described here (see Appendix A for a more detailed explanation of the swapping and cpu policies).

The cpu scheduling algorithm incorporates a loose round robin policy. There is no time quantum as such, though it will generally be at most one sec., and will be reduced as the amount of I/O increases. The decision of which process to run next is based solely on the priority of a process, which for competing user processes, is based only on the cpu time intervals between I/O's (exclusive of context switches and swaps), - the longer the interval, the lower the priority. After a process sleeps due to I/O, it's priority is reset to the highest possible user priority. Thus for instance, a process that continually does less than one second of cpu time followed by an I/O will be considerably favoured over a process which is totally compute bound. Thus we begin to see possible indications that the cpu or I/O requirements of a process may have some effect on response.

UNIX uses a swapping algorithm to handle loads which exceed the amount of available main memory. To summarize the policy used, swapping is only done when necessary, that is, only when there is at least one process out on disk that wishes to use the cpu. Processes are swapped into main memory as long as there is

room or it is decided that it is time to boot someone from main memory out to disk. The decision as to which process(es) is to be brought in is based only on its elapsed time out on disk. Similarily, the decision as to which process(es) gets swapped out is based only on its elapsed time in core. It can be seen from this (because the policy ignores such process characteristics as size), that especially with a slow disk and little main memory, swapping likely contributes significantly to poor response, to a point where thrashing is likely to occur when there are a few very large I/O bound processes in the system.

3. EVALUATION METHODS.

3.1 OVERVIEW

Once it has been decided that an evaluation is to be conducted on a given system, one must then determine which approach to use: measurement (where the desired information is obtained directly from the system), or modelling (where a model of the actual system is used to obtain the desired information). The possible methods shall now be discussed and then the reasons for the approach chosen in this project shall be given.

One of the chief advantages of modelling is its ability to provide results when the target system is unavailable for direct control. This would be the case, for instance, in a highly used commercial installation where the system must always be available for the customers, or in certain improvement studies or in the design of a new system where the target system doesn't exist yet.

Another area of strength of modelling is where the study can be conceptualized using fairly straightforward models and where simplifying assumptions can be made about the input parameters. This is particularly true of analytic techniques, which could loosely be described as employing paper and pencil models (for example, queueing models [GR78] or non-queueing models [CH75]). They often give satisfactory results with a comparatively low cost. See [KO78] for a good synopsis. One of their main problems has been that the model did not adequately

reflect the actual functions and interactions, usually stemming from making simplifying assumptions that were unrealistic. This is especially true where there are complex interactions such as workloads and communication networks [CH77, the foreword]. However there are currently very promising indications of success with the increased emphasis on model validation [CH77]. As well, there has recently been developed the "operation analysis" approach which bases decisions on observable and measurable quantities only, thus requiring no simplifying assumptions of the type needed by queueing models [DE78]. For a discussion of the state of the art for queueing models, the reader is referred to [MU78].

Modelling using simulation techniques offers greater flexibililty. However it suffers too as complexity increases, not so much from its inability to model complex events, but from the cost usually involved. The cost can be reduced in some cases by using hybrid techniques, which is simply a combination of simulation and analytic methods. There are again problems of validation of the input parameters and the model.

The other major evaluation approach is direct measurement. It has the important advantage of accuracy since it uses the actual system and thus eliminates the need for a model and all its pitfalls. Its major drawback is simply that the system to be measured is often inaccessible for such an approach. Another disadvantage is that similar to simulation but unlike analytic

modelling, large volumes of data are generally collected which often become unwieldy. Also, for measurement experiments where the actual production workload cannot be used, a synthetic workload must be constructed which then introduces validation problems similar to those of input parameters in modelling. See [FE72] for the problems of characterizing workloads.

3.2 CHOICE FOR THIS EXPERIMENT

The approach chosen for this project was measurement. The main reason is that we have an almost ideal environment for it: a dedicated system with access to the source code of the operating system and thus the ability to change it. UNIX is written in a high level language which makes it easy to change and has the advantage of being easy to understand, which is essential to ensure that the changes made are, indeed, what is desired and causes no side effects. We were given a removable disk cartridge with a copy of the UNIX system on it and were allowed to run our own altered version after midnight when no one else was using the system.

This set-up is very appropriate for an experiment which wishes to see the influences that various parameters have on the responsiveness of the system. It allows us to conduct a controlled experiment, that is, to develop a workload in which we can control the desired parameters and hold constant any undesirable influences. While the parameters could also be controlled in modelling, it is unsure whether the model itself

would capture all the interactions which might occur among the variables.

4. DESCRIPTION OF EXPERIMENT: SETUP AND PROCEDURE

4.1 OVERVIEW

In this chapter, the experiment which was conducted shall be described in detail. As yet, there is no distinct methodology for evaluating a system, and thus it seems wise to include the procedure that was involved here, in hopes that eventually a common methodology will emerge. Things are presented basically in the same order as they occured in the development of this project.

Having decided to do a measurement experiment, the first step is to decide what performance measure to use and then to identify the major parameters which influence it. Included in that section is a discussion of the hoped-for relationships between them. The next step was to develop tools to extract the desired information from the system. An explanation of the exact type of information obtained from these tools is included. Following this, it was necessary to develop a synthetic workload which would allow for a controlled experiment. And finally came the experiment, of which the design and implementation are discussed.

4.2 PARAMETERS.

4.2.1 Identification.

The initial step is to determine which type of performance indicator to use. The three basic categories are those

concerned with the responsiveness of the system, those concerned with throughput, and those concerned with utilization. The only major concern at the host site is the responsiveness of the system. See the next section for the choice made and description.

The identification of the major influences in a system is as yet a largely subjective and intuitive process. It is aided by previous attempts on other similar systems, and more importantly by the evaluator's knowledge of the system at the user and system levels and from insights gained by the other users of the system. It is often an iterative process. Fortunately, in this case, the system is small (and comprehensible) and the problems stood out fairly clearly. As well, the operating system is well written in a high level language, making its comprehension much easier. From the details of chapter 2 and the swapping algorithm of appendix A, and also from observing the lights on the disk drive, it became fairly obvious that the disk was the major bottleneck and that the likely culprit was swapping.

With this in mind, and yet remaining open to other considerations, we chose those parameters which were felt to be possible candidates in affecting the response of the system. These parameters, which will be described after the section on the performance parameter, are divided into those stemming from the workload, those originating from the system hardware, and

those indicated internally by the system software.

4.2.2 Description & Justification: The Performance Parameter.

There are several responsiveness indices which have been used (see [SV77]). From an interactive user's viewpoint, response time is usually the quantity which one is most directly aware of and interested in. However in this experiment such a quantity is inappropriate since the experiment is controlled and there is no terminal input (see workload construction, section 4.4).

It was eventually decided to use a type of reaction time which we labelled "priority based reaction time", but for brevity will simply refer to as reaction time (see appendix B for the basis for this choice). The event which is timed is the period from when a process wakes up until it is selected as the next process to run. A process sleeps or is suspended in UNIX when for instance it must wait for some I/O to finish. The reaction time is not measured for all processes however, but only for those with low priority. Specifically, it is done for processes whose priority is that of terminal input or lower (which includes terminal output).

4.2.3 Description and Justification: The Other Parameters.

a) workload parameters.

1) the number of processes - This is a fairly obvious candidate since it is the basic unit of manipulation in

UNIX.

2) process size - It is clear that this is likely an important parameter as well. The illustration of thrashing in section 2.2.2 serves as an indication of the influence that the size of a process may have in an extreme case (though it should be recognized that the situation in 2.2.2 is also a function of the system hardware and the number of processes).

3) file sytem requirements of a process - Disk I/O is considered the major bottleneck. While it is felt that this is mainly due to swapping, there is probably a contribution brought about by processes trying to access the file system. As well, it was indicated that in the cpu scheduling algorithm, the priority of a process is strongly influenced by the amount of I/O it does.

4) cpu requirements of a process - As indicated in section 2.2.2, it appears that the length of a process's cpu request may affect its responsiveness.

Parameters 2, 3, and 4 are on a per process basis (as opposed to a per workload basis). In doing so, this keeps the parameters all within the context of the basic unit of manipulation within UNIX and enhances modularity in the deisgn of our workload.

b) system hardware parameters.

1) main memory capacity - The addition of more primary memory intuitively appears to be the most promising remedy

to the disk bottleneck. With more main memory, the system could fit more processes in at a time and thus the swapping should be reduced if not eliminated. So the choice of this quantity as a parameter is very appealing.

Fortunately within UNIX, one of the routines used to help boot itself up checks the actual amount of main memory available. And so by slightly altering that routine, systems can be created which would be "tricked" into working with less main memory than what is physically present.

2) disk configuration - The disk is also a probable hardware parameter, since a faster disk would reduce the backlog that currently occurs. Simulation could be used to study the effects that would occur with a a faster disk or multiple disks. However, to keep the experiment within reasonable bounds, this parameter will remain constant. It should be noted that the effect of a faster disk can be roughly estimated if an equation for the reaction time in terms of the internal system parameters (specifically the disk waits and queue lengths) can be obtained. This estimate would then be possible since the hardware manufacturers usually include information such as the average transfer time, etc., which could be used to estimate the shorter disk waits.

C) internal system parameters.

This is the term used to describe those quantities derived from the internal structure of the operating system (e.g. queue

lengths). While the quantities chosen will likely be valuable to observe, the procedure to select them was again arbitrary.

These parameters are included for the possible insights they may give in interpreting the results of the experiment, for possible use in determining the influence of the disk, and for possible future research. There are basically two views that can be taken towards these quantities. One can view them as indicators of how well the system is handling the workload, that is, as a reaction of the system to the workload. In this context, if one was interested, they could be candidates for performance parameters. However in this project our main concern is with the responsiveness of the system and not directly with the internal efficiency of the system.

In this thesis the internal system parameters are viewed as variables that can be related to the performance parameter. The following are those that have been selected.

1) swap rate - This together with swap size is the prime suspect for the cause of the disk congestion and so its inclusion is obvious.

2) swap wait - This includes the waiting time in the queue as well as the actual I/O time. It is included along with the swap rate since the swap rate does not differentiate between the size of swaps, which for a slow disk, could have a substantial effect on responsiveness. The swap wait does capture this effect.

3) disk queue length - This will help indicate the number of file system accesses that are being made to the

disk, - the longer the queue, the greater the number of accesses. This is not entirely true since the swapper I/O gets queued onto the same disk queue (and it gets queued according to the same algorithm, that is, swap I/O gets no special priority). The swapper influence is not very significant on the queue length however, since the number of swaps is at least an order of magnitude less than the number of file system accesses.

4) disk waits - As with queue lengths, this quantity helps indicate the number of file system accesses. However, it is much more heavily influenced by the swap I/O. This is because all file system I/O consists of a block of 256 words, whereas the size of swap I/O blocks often is several kwords. Again, for a slow disk, this difference could be significant and will be worth noting.

5) cpu intervals - this interval is the time between context switches (not to be confused with the time between I/O's, which is the quantity used by the cpu scheduling algorithm). This quantity is included to give us a detailed view of cpu usage. See section 6.2 for hindsight comments.

There are several quantities which are excluded in this experiment. For instance, I/O to other peripherals is not considered as a prelimenary experiment was performed which indicated that the disk I/O was by far the dominant quantity.

The exclusion of some of the other quantities is more difficult to justify except again on the basis that the experiment would become too large and the overhead too high. These variables would include for instance the cpu and swap queue lengths.

4.2.3 Possible Relationships.

From the results of the experiment we hope to be able to derive an equation for the performance parameter in terms of the workload parameters and system hardware parameters. This is very desirable for it would give us the ability to predict the expected responsiveness given the characteristics of the workload (in terms of the above parameters) or given a desired change in the hardware. It must be realized though, that the results cannot be assumed to be applicable to arbitrary workloads. It is not within the scope of this thesis to prove the the synthetic workload is equivalent to the production workload (see section 4.4 for the procedure that was employed). However it is hoped that the relationship will be quite similar and thus of significant practical value.

Other possible relationships of interest, though of less practical value, may be found in deriving the internal system parameters in terms of the workload parameters or the performance parameter in terms of the internal system parameters. This may be of interest to the person who wishes to change the operating system and wants to see where, internally,

are the main influences.

4.3 DESCRIPTION OF THE MEASUREMENT TOOLS

4.3.1 Background

From the previous section, it was decided to collect disk, swap, cpu and reaction time information. The first three quantities are very high volume and so we need a data collection method which can obtain this information without interfering too much with the system. There are three basic types of monitors that can be used to collect data: hardware, firmware and software (see [FE78] for the general merits of each). For this study a hardware monitor would have been ideal but was unavailable. The host computer is not microprogrammable and so firmaware tools were not applicable. This left us with the task of developing software tools which could gather high volume data without disturbing the system too much.

Prior to this project, there was developed on the host computer a software tool for low volume data. This tool was modified for our purposes and will be described now. The changes are explained in some detail as it is hoped that in an area where no clear methodologies have emerged, the sharing of such information might be an aid in discovering commonalities.

4.3.2 Operating System Changes

The following are the routines and data structures that were implanted in our modified version of UNIX.

a) a new clock. The regular clock on our version of UNIX is accurate to the nearest 1/60 sec. In some preliminary tests, this resolution was thought to be just barely adequate for our purposes and was inadequate for any type of detailed disk timing analysis. As well, by using the same clock that the whole system uses, there is the possibility of becoming unavoidably synchronized with other events in the system and thus producing distorted results.

Fortunately a faster clock was discovered in a DUP-11 synchronous line interface. The clock in this piece of hardware is used for internal maintenance and has a resolution of just over one millisecond. By writing some code to drive the clock, we solved the above problems , though at the expense of slightly greater overhead.

b) timing routines. Two routines together with a table acted as a stopwatch mechanism. When we wished to start timing an event, a call would be made to the routine START_TIMING. This routine would be passed a unique value to identify the event and would store the value and the current time in the table. As is the case in UNIX, an address is the value used to uniquely identify an event. Each time is stored in a 16 bit word, allowing us to time events up to just over one minute (65536 milliseconds). This was more than adequate for any of the quantities we wished to measure. In tests, we found reaction times never exceeded 15 seconds, and the uninterrupted cpu intervals, even on an idle system, never exceed 30 sec. due to

a background task that awakens twice per minute.

When we wished to stop the timing of an event, a call would be made to the routine STOP_TIMING. The identifying value of the event would be passed to it, and the routine would then calculate and return the elapsed time since the timing was started.

c) routine to output the statistics. When this routine (STAPUT) is called, it is passed a variable length array containing the information to be output from the operating system. STATPUT places this information plus the elapsed time since it was last called in one of the specially created output buffers. When the buffer is full, it prods the existing magtape device handler to write the buffer directly onto magtape. From testing it was found that three 512-byte buffers proved to be sufficient in preventing a buffer from being overwritten before the magtape handler was finished with it.

d) controlling the output. It is desirable to have the ability to turn on or off the output of the data. To accomplish this, we modified the mechanism that was previously implemented in the low volume measurement tool.

Basically a new device, called a "stat" device, was created. That is, open, close and read routines were written for it in the operating system, and a node was made in the file system for it. The open routine simply makes sure it is not already open, and then sets the status to "OPEN". The status is

checked by STATPUT, which simply returns if the device is not open. The close routine sets the status to "CLOSE". The read routine is not used for our modifications.

Thus when a user program issues an open system call on the "stat device"(*) the collection information starts to be written onto the magtape. Usually the user process will then sleep until it is desired to stop the output of this information, at which time the user process must then wake up and close the "stat device".

e) imbedded code. Calls to the three routines START_TIMING, STOP_TIMING, and PUTSTAT, were placed in the existing code of the operating system at the points we wished to observe. These areas included the disk handler, just before a request is queued and just after the I/O has completed; the place where the decision to swap is made; the routine that handles context switches; and for reaction times, the place where a process is put to sleep.

4.3.3 Description of Information Collected

The following describes the four basic types of information (or records) that were collected. The records contain the needed information to observe the internal system parameters and the performance parameter that were defined in section 4.2.2.

(*) This is slightly simplified. Actually the mechanism uses the minor device number to create 5 "stat devices": stat0,....stat4. STATPUT must then also be passed the minor device number and then must check that the corresponding "stat device" is open. This allows greater control over exactly which information is output.

The first two values of each record are respectively, the elapsed time since the previous record was output and a value identifying the type of record it is. See Table I for the contents of the remainder of each record.

Table I
Description of Unique Portion of Records.

Record Type	Unique Contents
disk	- disk location - size of transfer - queue length - total wait time for transfer - flags & minor device number
swap	- disk location - size of transfer - wait time in swap queue
cpu	- cpu interval
rt	- reaction time

4.4 WORKLOAD USED FOR THE EXPERIMENT

4.4.1 Possibilities and Choice.

When one wishes to measure the performance of a system, either the natural (production) workload can be used or an artificial workload can be developed to drive the system. The production workload (or a segment of it such as the peak load) has the obvious advantage of being most representative, though this can be affected by the choice of when and how the system is measured. Artificial workloads offer better reproducability, are more flexible and potentially more compact, though they are more expensive to construct, less representative and often contain a certain annoyance factor in that they require a

dedicated system.

In our situation the production workload was not avaiable for monitoring and so we had to use a synthetic workload. For the purposes of this thesis, this is the most desirable choice anyway since we need a controlled environment to establish the relationships of section 4.2.3. There are various types of artificial workloads, - from instruction mixes (see [GI70] and [FL74]) to standardized synthetic benchmarks (see [FE78, p253]). For this experiment a prototype process was constructed which became the basis for our synthetic workload.

4.4.2 Construction of the Synthetic Workload

The composition of the prototype process (or program) is an infinite loop consisting of the following:

a) compute loop. A series of statements are looped through a variable number of times according to a distribution.

b) disk I/O operations. Following the compute loop, a routine is called to create some disk I/O by reading, writing, opening or closing a file on the disk. In UNIX, however, one must be aware that simple reads or writes don't necessarily do I/O to the disk, or at least not immediately. An internal pool of buffers is resident in memory. Reads or writes place information into these buffers and, very loosely, the information will stay there until the file is closed or the

buffer is needed for other purposes, at which time the actual disk file will be updated accordingly.

c) sleep. There is a system call in UNIX which allows a process to suspend itself for the number of seconds passed as a parameter. While in this experiment a call to 'sleep' helps to simulate think time, its main use is to increase the number of reaction times that are produced by the process. These are reaction times of priority 100 or greater, which should be very sensitive to the responsiveness of the system. The decision to sleep or not after each execution of the compute loop, is distribution driven.

d) terminal I/O. The natural workload, consisting largely of interactive users, involves a great deal of terminal input. It was felt that this could not be simulated in our synthetic workload, since this would require either extra hardware to feed commands into the system, or some people sitting at terminals and typing in a set of commands at a given rate into the system. Neither the hardware nor the manpower were available. As well, it was felt that the latter technique would open the experiment to too much potential human error.

To compensate for the reaction times lost due to the omission of processes waiting for terminal input, it was decided to approximate this by writing a string of characters to the terminal via the file system. What this does is that it sufficiently fills the terminal output buffer mechanism and

causes the process to be suspended (at a priority only slightly lower than the input priority) until the characters have been sent to the terminal. The number of character strings (consisting of 25 characters) is determined by a distribution.

e) process size. There is a dummy array in the prototype process which can be statically varied to give us the desired size.

To create the synthetic workload, this prototype process can then be duplicated (on a non-sharable basis) to give the desired number of processes.

4.4.3 Justification

In the creation of the workload there was a continual struggle to maintain the balance between minimizing any undesirable influences on the one hand and making it complex enough to be representative on the other hand. If there were influences which we were not measuring and we did not try to minimize them (usually by keeping them constant), then there would be a large error in our relationship between the performance index and the workload and hardware parameters, rendering it of little value. Yet if we exert too much control so that the workload is very simplistic in nature, then the relationship will be very strong but the results will be next to useless because it pertains to a workload that is grotesquely distorted from reality.

The creation of a workload in which processes are simply clones of a prototype is simplistic yet necessary if we want to isolate the effects of the workload and hardware parameters. To help create some richness in the workload, a certain amount of variablilty was introduced. The number of consecutive times that the compute loop is performed, the decision of when to sleep, the amount of terminal output, and the type of file system operation were all distribution driven. This adds variablity while keeping their averages constant. As well, the seed for the random number generator and the file which is accessed were both program parameters and thus were made different for each process. Finally, by exchanging hunks of code, though maintaining the same order of execution, some of the processes were started in different places (i.e. some started with the compute loop, others started by doing I/O).

From this, it is hoped a healthy balance will be achieved between complexity and controllablilty.

4.5 DESIGN OF THE EXPERIMENT

The purpose of this project is to help establish a relationship between the performance parameter and the workload and hardware parameters. In cases, such as this one, where there may be interactions among the parameters it is necessary to use a factorial design for the experiment. In a factorial design all values of each parameter must be varied with all

values of the other parameters.

In section 4.2 we identified the quantities which were felt to influence the responsiveness of the system. From among these quantities, a subset was selected which became the factors of our experiment. By "factors" we mean those quantities in the experiment which we explicitly vary. The different values of a factor that are used are called the "levels" of a factor. The influences that we are not interested in and thus must hold constant are called secondary factors. We now wish to indicate the factors and levels that were selected.

In determining which factors to use, it had to be recognized that for a factorial design, each additional factor adds considerably to the size of the experiment. And so we started with those parameters deemed (intuitively) to have the greatest influence: the number of processes, process size, and size of main memory. As well, the mean cpu requirements inbetween consecutive disk I/O's of a process(PC) were included. Since the mean rate of disk I/O calls per unit time is inversely proportional in the prototype process to PC, the factor PC indicates both the cpu and disk requirements. See Table II for a summary of the factors and their levels.

In determining the levels of the factors, a program was run on the production system which calculated the average number of processes and the average size over a given length of time. It

was found that usually on a busy system, there were about two READY processes and about 12 - 14 BLOCKed processes in the system. Usually more than 30% of the processes have sizes between 100 - 200 blks (3k - 6k words), and up to 5% are over 500 blks(16k words) in size. The 50% which are under 100 blks are usually fairly idle system processes.

Table II
Factors & Levels for Original Experiment.

Factors	Levels
# of procs (PN)	2, 4, 6
size of a proc (PS)	157 blks (5Kw) 314 blks (10Kw) 470 blks (15Kw)
cpu req'ts of a proc(PC)	12, 186, 354 (msec)
size of main memory(MM)	650 blks (20.7Kw) 800 blks (25Kw) 955 blks (30Kw)

The levels for the cpu requirements were less clear to determine. Fortunately a 150 minute glimpse of the production workload was obtained. See Appendix C for the information extracted from it. The distribution of the compute loop of the synthetic workload was manipulated until the resultant uninterrupted cpu intervals closely corresponded to those of the production workload. This formed one of the levels, with an average cpu interval of 12 milliseconds. Since the production workload on which it was based was highly interactive with a lot of editing and compiling, the other two levels were chosen with larger cpu requests to hopefully correspond to a more highly

compute-bound environment. See Figure 4.1 for the distributions of the compute loops for the three levels.

Figure 4.1
Distribution of Compute Loops.
(% of occurences)

PC	# of times through loop			
	5	20	200	5000
12	3	47	49	1
186	1	1	48	50
354	1	1	1	97

The maximum level for main memory is based on the maximum amount that was available on the system. The lower level is based on providing just enough memory so that the largest size of a process for this experiment could fit (plus a little extra for some system stuff).

It should be noted that the internal system parameters cannot be factors since we have no direct control of their values, - they are all observable quantites only. Thus while no analysis of variance can be done on them, they are still candidates for regression techniques.

4.6 IMPLEMENTATION

The implementation of this experiment initially involved the acquisition of a disk cartridge with a version of UNIX on it, changing the operating system so that it contains the tools described in section 4.3, and running this version on a

dedicated basis.

Before running the experiment, the length of a session had to be determined. It is assumed that the series of reaction times are ergodic, that is, the accuracy of the mean reaction time increases as the number of observations in the series are increased. On this basis a sample workload was run several times increasing the time lengths until the variation over several runs of the same time length was acceptably small. An alternative method for discovering this length is discussed in [FE78, pp. 75-77]. This point was found to be 20 minutes with differences of 5% or less in the values of the performance parameter.

Since the original experiment was factorial in nature, it required 54 runs. As will be seen in chapter 5, this proved insufficient and so an additional 29 runs were included.

To minimize possible variations in the start-up procedure and to reduce possible error, each run was initiated via a file of commands. UNIX has a mechanism which allows a command to be processed asynchronously. And so the basic start-up sequence is to asynchronously: open the magtape for writing; execute another command file which simply starts up the desired number and type of prototype processes for the particular synthetic workload; sleep for 15 seconds while the workload stabilizes and then open the "stat device" which will enable the writing of the

information onto the magtape. After a little over 20 minutes the "stat device" is closed, stopping the information flow. An end-of-file is then written on the magtape and the synthetic workload is terminated.

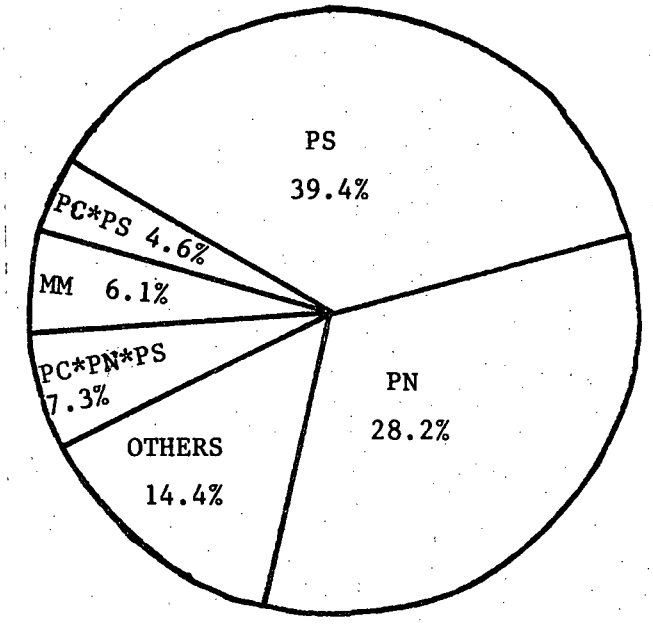
5. DESCRIPTION OF EXPERIMENT: RESULTS & ANALYSIS

5.1 INITIAL ANALYSIS.

5.1.1 First Set of Data.

The values of the performance parameter for the 54 runs are given in Table III. Since it was a factorial experiment, an analysis of variance was performed to help indicate the significant interactions among the factors. Those contributions to the total sum of squares are illustrated in Figure 5.1.

Figure 5.1
Analysis of Variance Results



It should be recognized that the design of the experiment is such that there is only one observation for each combination of levels. That does not allow us to uncover the exact amount of error within a combination. As noted in section 4.6, this error

Table III
Results of Initial Experiment.

cpu (msec.)	memory (64b blks)	# of procs	size (64b blks)	reaction time (msec.)
12	955	2	157	5.84
12	955	2	314	5.97
12	955	2	470	432.86
12	955	4	157	13.66
12	955	4	314	1175.43
12	955	4	470	2250.61
12	955	6	157	514.19
12	955	6	314	1683.09
12	955	6	470	3539.72
12	800	2	157	8.07
12	800	2	314	85.06
12	800	2	470	1728.63
12	800	4	157	83.85
12	800	4	314	1742.33
12	800	4	470	4380.06
12	800	6	157	1011.09
12	800	6	314	2617.82
12	800	6	470	7469.52
12	650	2	157	5.70
12	650	2	314	293.62
12	650	2	470	1706.85
12	650	4	157	403.68
12	650	4	314	2059.48
12	650	4	470	4341.80
12	650	6	157	1295.64
12	650	6	314	3057.06
12	650	6	470	7112.48
186	955	2	157	6.65
186	955	2	314	7.85
186	955	2	470	642.81
186	955	4	157	9.70
186	955	4	314	1303.29
186	955	4	470	1529.06
186	955	6	157	533.62
186	955	6	314	1375.00
186	955	6	470	2002.51
186	800	2	157	6.77
186	800	2	314	146.92
186	800	2	470	1182.53
186	800	4	157	128.21
186	800	4	314	1254.18
186	800	4	470	2381.16
186	800	6	157	1093.32
186	800	6	314	1717.93
186	800	6	470	3995.25
186	650	2	157	7.35
186	650	2	314	562.55
186	650	2	470	1130.04
186	650	4	157	497.56
186	650	4	314	1672.34
186	650	4	470	2215.37
186	650	6	157	1392.39
186	650	6	314	2237.34
186	650	6	470	3638.75

is thought to be in the neighbourhood of 5%.

Within the context of this experiment, the number of processes(PN), the size of a process(PS) and their combination account for almost two-thirds of the total sum of squares in the reaction time values. This is not unexpected. The size of main memory(MM) directly accounts for 6% of the variation. It must be realized though that this much smaller contribution is largely due to the smaller variation of the levels of main memory as opposed to that of PN and PS. To determine the exact relationships among the four factors, we turn to regression analysis.

Techniques have been developed to select the best independent variables for a regression equation from a list of possible candidates, given the dependent variable (see [DR66]). The candidates must of course be independent. Both the forward algorithm (also called stepwise regression) and the backward algorithm were used in our selection process. Via the statistical package called MIDAS [FO76].

The search for the best set of independent variables to form the regression equation was a lengthy one. The measure used to roughly determine whether or not one equation was superior to another was r**2, the square of the correlation coefficient and which is sometimes referred to as the coefficient of determination [WA72, p.321]. This value

indicates the proportion of the variation in the independent variable that can be explained by the regression equation.

We tried numerous equations, almost to the point of an exhaustive search, without producing any stunning results (that is, we failed to get a value of r**2 > .9). Because of an inability to represent the cpu requirements of a process (PC) in any equation, it appeared that its two levels were insufficient, and so we conducted a second experiment of 18 runs with a different value for PC. As well, we chose different values for PN and PS to help increase the number of points for those quantities.

5.1.2 An Unusual Trend.

An examination of the data had generated another concern. Some of the runs had produced results which seemed counter-intuitive. Generally this can either lead to a new understanding of the data or to a source of error. The runs in question are the six pairs which go from MM=800 to MM=650 with PS=470 and PN and PS held constant. One example(*) is the pair (12,800,6,470) with RT=7469.52 and (12,650,6,470) with RT=7112.48. It is unexpected that, with everything else held constant, a decrease in the size of main memory would produce a quicker reaction time. Since there is no apparent insight which this trend uncovers, it leaves two possibilities: (a) one of the

---

(*) Unless otherwise explicitly stated, data shall be referred to by its levels for the four factors and shall be of the following form: (PC, MM, PN, PS).

pairs of the points is being strongly influenced by another quantity which was overlooked in the design of the experiment, or (b) the trend is simply due to error. A re-examination of the results and the workloads did not uncover any recording or human errors, but recalling the rough experimental error bounds suggested in section 4.6, the difference in the values of the pairs in all but one case becomes insignificant. However the trend remains somewhat disconcerting in that it is felt there should be a significant trend in the opposite direction. While such a thought does not justify labelling these points as erroneous, a further examination needs to be made.

5.1.3 Second Set of Data.

The results of the second set of runs are given in Table IV. The table also includes 11 extra runs, nine of which were randomly chosen to help strengthen the model. The other two were run to check out the trend mentioned above and suggests that the trend is due to error, since the values of RT dropped 13% for (12,800,6,470) and 21% for (186,800,6,470) from the original runs. This shall be used later.

5.1.4 The Quantity 'PCTRMM'.

Regression analysis was then applied to the combined 83 runs. There was sufficient change to indicate that the previous model was inadequate. Various combinations of the independent variables were tried and one quantity stood out as being very strongly correlated to reaction time. I refer to this quantity

Table IV
Results of Second Experiment.

cpu (msec.)	memory (64b blks)	# of procs	size (64b blks)	reaction time (msec.)
354	955	3	220	12.00
354	955	4	220	498.67
354	955	5	220	823.36
354	955	3	576	828.35
354	955	4	576	1504.70
354	955	5	576	1918.93
354	800	3	220	196.91
354	800	4	220	797.08
354	800	5	220	787.76
354	800	3	576	728.40
354	800	4	576	1316.00
354	800	5	576	2252.27
354	650	3	220	673.61
354	650	4	220	657.31
354	650	5	220	867.20
354	650	3	576	1200.13
354	650	4	576	1470.14
354	650	5	576	2046.52
12	955	6	314	1755.37
12	800	4	314	1748.97
12	800	6	470	6530.28
12	650	2	314	303.92
12	650	5	576	6374.74
186	955	2	157	6.63
186	800	6	470	3187.56
186	650	4	314	1581.95
186	650	5	576	3249.73
354	650	5	314	1560.86
354	650	5	470	1940.98

as the percentage of remaining main memory (denoted hereafter by PCTRMM) and is defined by the following equation:

$$PCTRMM = \frac{MM - PN * PS}{MM} * 100 \qquad (5.1)$$

This quantity, which can be either positive or negative depending on whether all the processes can fit in main memory or not, can account for a large amount of the variation in reaction time(RT). However, we failed to find other combinations of independent variables to account for the remainder of the variation. The reason for this is discussed next.

5.1.5 Complexity of the PC Factor.

The nature of the PC factor appears to be quite complicated. As inferred in section 4.5, as the cpu requirements of a process are adjusted, it also affects the number of file system accesses. Specifically, as the cpu requirements, are increased, the rate of file system accesses in the prototype workload process are decreased. On an independent basis, an increase in the cpu requirements would tend to increase reaction time since, as in this experiment, for all cpu requests under one sec., the cpu will not do a context switch. Thus any other process competing for the cpu will have to wait longer. With the file system access rate however, things are not so simple and can't even be considered independent of the cpu requests in the context of this experiment. For in the prototype process, the only way to increase the rate of file system accesses is to decrease the cpu requirements between

accesses.

Generally when a file system access occurs a context switch will be made while that process waits for its I/O to complete. Thus any processes waiting for the cpu will have a greater chance of getting its request serviced and thus the reaction time will generally be less. This trend would start to reverse however if most of the processes were doing a large amount of disk I/O, since disk congestion would start to take its toll by increasing the time to complete any individual request. As well, there are other things to consider. If user processes are trying to access the disk then they will be interfering with the swapping if there is any. This will tend to increase the reaction time. Futhermore, processes which have just completed doing a file system access retain for a short period in this UNIX version, the priority of a disk request (which is much higher than a normal user priority). In this case then, as the number of file system accesses increases, there is an increasing chance of such processes getting the cpu before a process, which because it is at a lower priority and thus will produce a reaction time, will obtain the cpu. This would tend to increase reaction time values. Regardless, there will generally be a longer wait, since the cpu must attend to disk I/O first, which will add slightly to overhead.

Finally,the swapping algorithm needs to be examined for its indirect contribution in this matter. It should be noted that

when swapping must occur, the swapper initially looks for blocked processes to swap out. Hence whenever a process does I/O, it is quite likely that it will get swapped out before the I/O is completed. This probability increases with the number of processes and with shorter cpu requests. The result is that as the PC factor decreases, less and less work is getting done by a process before being swapped out. This will be reflected in significantly larger reaction times and is due to the simplistic nature of the swapping algorithm.

In summary, it is sufficient to say that a change in the value of the PC factor may have a significant and complex effect on the reaction time. Examining the data, it appears that the overriding effect is that as the PC value is increased, RT drops, indicating the file system access rate has the dominant influence. This holds true for most, but not all, of the cases in this experiment. The complexity of the issue, though, helps explain the difficulty in developing a good regression equation.

A hypothesis can be made that in the general case the PC factor affects the size of the cpu and disk queues. For small PC values it is felt that the length of the disk queue on the average will be longer than that of the cpu queue but that this will reverse as PC increases. When the disk queue is substantially longer, the file access rate will increase the value of RT. As PC increases, however, this influence will drop and eventually when the cpu queue becomes substantially larger

than the disk queue, RT will again be increased except that this time it will be due to the longer cpu requests. Clearly if this is the case, then it would be desirable to find the transition point, where neither influence has much effect.

5.2 FINAL MODEL AND INTERPRETATION.

The final model was derived by partitioning the 83 runs into three categories according to their PC value and doing a regression analysis separately for each category. This allowed us to ignore the influence due to changes in the PC level, which as indicated in the previous section was too complex a quantity to adequately incorporate into an equation. The results are given in Table V. We refer to this model as model M1. It is noted that the PC=12 level was constructed to closely correspond with the general characteristics of the production workload.

Table V
Results of Initial Regression.

PC level	r**2 value	Equations (forming M1)
12	.92	RT = 642 - 19.0 * PCTRMM
186	.92	RT = 625 - 9.5 * PCTRMM
354	.91	RT = -241 - 4.5 * PCTRMM + 204*PN

The r**2 values indicate that either there is a high experimental error (see section 5.5 for possible sources) or there is some other small, still hidden influence.

An examination of the duplicate runs in Tables III and IV

of (12,800,6,470) and (186,800,6,470) indicate higher than usual errors. These two runs were members of the trend mentioned in section 5.1.2 and thus tend to indicate that the trend is largely due to some form of error rather than an undiscovered factor. Thus we removed the observations which made up the trend and again applied regression techniques. The resulting model shall be called model M2 and is contained in Table VI. The higher values of r**2 are encouraging. Both sets of results shall be kept for validation.

Table VI
Results of Second Regression.

PC level	r**2 value	Equations (forming M2)
12	.96	RT = 588 - 16.9 * PCTRMM
186	.96	RT = 595 - 8.9 * PCTRMM
354	.91	RT = -241 - 4.5 * PCTRMM + 204*PN

From either Table V or VI, we see that PCTRMM is strongly correlated with RT. For PC=354, PN is also found to positively influence RT. This could be due to the interference on swapping caused by file accesses, but is unlikely since it should then also appear for PC=12 and 186. A more reasonable explanation seems to be that it is due to the longer cpu requests of this set, which, as the number of processes are increased, would tend to produce longer reaction times. This explanation also supports the hypothesis in section 5.1.5 that the longer cpu requests are increasing the length of the cpu queue and values

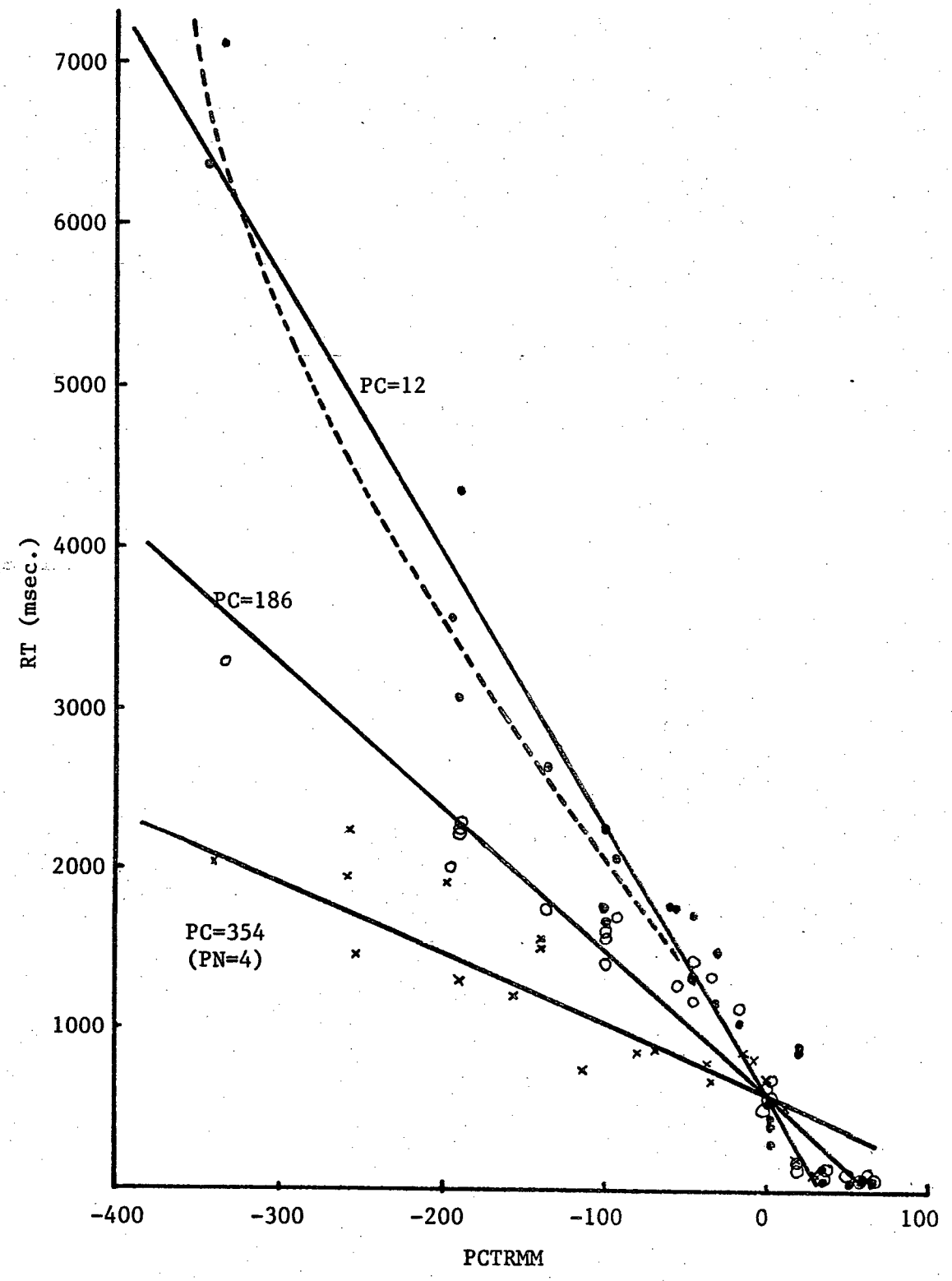
of RT are becoming more affected by it and less affected by the file access rate.

Using model M2 a graph was plotted of RT versus PCTRMM (figure 5.2). The solid lines indicate the regression lines. The data follows the lines well, but with substantial residuals. It is interesting to note that as the values of PC are increased, the RT decreases. This confirms the earlier indication (section 5.1.5) that generally in the context of this experiment, the PC factor is dominated by the file system influence rather than by the cpu requirements. The dotted line for PC=12 indicates that for large negative values of PCTRMM, RT is probably becoming non-linear, and if continued, would result in total thrashing.

As the value of PCTRMM increases, the three sets of data eventually converge and then drop in unison to very low values of RT. This occurs for positive values of PCTRMM, that is, runs where all the processes can fit into main memory. This explains why the three lines drop together, since swapping no longer takes place. This drop occurs at slightly positive values of PCTRMM rather than when it exactly equals zero because it must be remembered that there are a few background system tasks which occasionally occupy a small amount of main memory but which aren't considered when calculating PCTRMM.

Since PCTRMM is a composite quantity, graphs were produced

Figure 5.2
Graph of RT versus PCTRMM

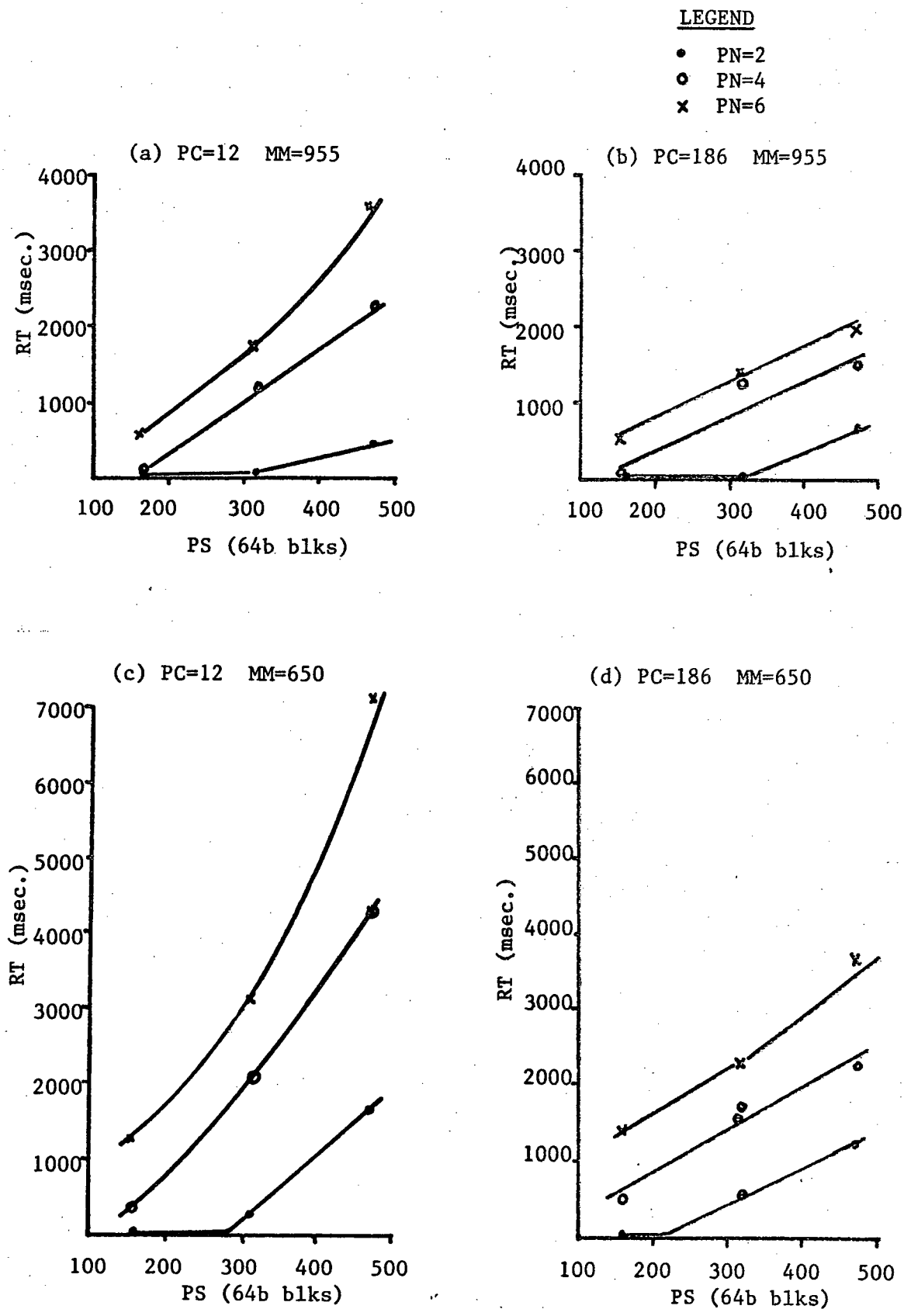


to help develop a sense of the individual effects of the four factors. These are found in figure 5.3 for PC=12 and 186 and MM=955 and 800. Because there are so few points, the lines are given merely as reasonable trends, not as exact relationships. As expected, when going from MM=955 to 650, the slope of the lines increase. When going from PC=12 to 186, the effects are lessened which again indicates the dominance of the file access rate on the PC factor. In figures 5.3(a), (c), and (d) the lines for high values of PN become non-linear, illustrating the increasing overhead due to swapping, which of course will eventually cause almost total thrashing. For PN=2 the lines start out horizontal and then make a fairly abrupt rise. This rise occurs when the processes can no longer all fit into main memory and thus swapping is initiated.

The next step in checking the results was to take a brief look at the residuals, which are the difference between the observed and predicted values. The purpose for doing this was to see if it would uncover any outliers or aberrant observations. A simple scatter plot of the residuals against the predicted and observed values revealed no such oddities. For a more formal analysis of residuls, the reader is referred to [AN63].

Finally, a brief description shall be given of the equation that resulted by trying to include the PC factor. It is given here mainly for interest, though for situations which do not

Figure 5.3
Graph of RT versus PS



conform well to the discrete PC levels of models M1 and M2, the equation may be of help. It must be remembered that its r**2 value was only around 0.9. The equation that resulted was:

$$RT = 703 - 15 * PCTRMM + 2.3 * PC - .004 * PC*PN*PS$$

This seems to support the idea of the dual nature of the PC factor, with the positive contribution of the PC term and the negative contirbution of the PC*PN*PS term. The latter quantity will have a more influential contribution when PN or PS is large, specifically when PN*PS > 575.

5.3 VALIDATION

The validation of a model is essential before much confidence can be placed in it. We know that there is a fairly large error in our models, and so validation simply consisted of three extra runs not used to calculate the regression equation, plus an examination of five observations from within the set of data. The five internal checks are derived from the five runs which were duplicated (the aberrant duplicate discovered in section 5.2 is omitted). The duplicates are used as they give us an indication of their experimental error, which in all cases is within the expected bounds. We use the means of the duplicates in each case to calculate the error. The results of the validation are given in Tables VII and VIII for both models M1 and M2.

Model M2 appears to be the more favourable one as there is less variation in the error. In all the external observations the predicted values form an upper bounds on the observed values, which is desirable.

Table VII
Validation Using Internal Observations.

Run	observed RT (2)	mean	model M1 pred. RT	model M1 err.	model M2 pred. RT	model M2 err
(12,955,6,314)	1755 1683	1719	2491	+45%	2236	+30%
(12,800,4,314)	1742 1749	1746	1725	-1%	1553	-11%
(12,650,2,314)	294 304	299	577	+90%	530	+75%
(186,955,2,157)	6.65 6.63	6.64	-10	-	-4.0	-
(186,650,4,314)	1672 1582	1627	1507	-7%	1426	-12%

Table VIII
Validation Using External Observations.

Run	observed RT	model M1 pred. RT	model M1 err.	model M2 pred. RT	model M2 err.
(12,650,6,314)	3037	4249	+40%	3803	+25%
(186,800,6,220)	1080	1240	+15%	1174	+9%
(12,800,6,220)	1019	1877	+84%	1689	+66%

5.4 SOME REALITY CHECKS.

It was decided to try and establish some indication of how well the model approximated the case where not all active processes are of the same size. And so two runs were performed, corresponding to (12,800,5,220) and (186,800,5,220), where instead of five processes of size 220 blks (7k words), two processes of size 314 blks (10k words) and three processes of size 157 blks (5k words) were used.

Table IX
Comparison with Non-uniform Runs.

Run	Uniform Run (RT)	Non-unif. Run (RT)	Difference (%)
(12,800,5,314)	1019	1212	15.9
(186,800,5,314)	1080	1207	10.5

See Table IX for the results, which are quite encouraging. Within the 5% experimental bounds placed around them, the differences between the PC=186 runs almost disappears, and for PC=12, it is lessened substantially. It should be noted though, that this close correspondence would be expected to drop if drastic differences in size were used. As well, a similar test needs to be conducted with PN being varied, and also for PN and PS combined.

5.5 SOURCES OF ERROR.

It was previously mentioned that a given workload with the 20 minute duration length could give around a 5% error in the RT value. As well, there was also an effect due to the everchanging nature of the system. During the entire stretch of the experimental runs, development of the workloads, validation, etc., certain system changes took place. Some, such as the addition of hardware, had no direct effect on the experiment but did require either the altering of the operating system or of the analysis routines. This increases the possibility of programming error. More likely error sources, however, were in

two other events. One was a change in the hardware interrupt priority of our pseudo-millisecond clock part way through the sets of runs. An after-the-fact attempt was made to measure any effect and while the influence seemed to be negligible, it is hard to be certain that this was true. The other effect came from the magtape which sometimes gave write errors during some of the intial runs but which was later cleared up. These were detected by the analysis routines and while the discontinuities were smoothed over as much as possible, there may have remained a very small though non-negligible error.

5.6 THE INTERNAL SYSTEM PARAMETERS.

A brief attempt was made to relate the internal system parameters to reaction time. It was found that a higher r**2 value could be obtained by using the natural log of RT (LRT) as the dependent variable. However such an equation still failed to produce an r**2 > .85, which is much too low for the equation to be of any value. So simple correlations of RT and LRT with the internal system parameters are given in Table X.

TABLE X
STATISTICS OF INTERNAL SYSTEM PARAMETERS.

VARIABLE	CORRELATIONS RT	CORRELATIONS LRT	MIN.	MAX.	MEAN	STD DEV
SWP RATE	.58	.85	0.0	138	69.2	42.9
SWP WAIT	.38	.62	0.0	611	303	127
DSK Q	-.17	-.09	0.4	5.1	1.4	.96
DSK WAIT	.27	.47	95.4	342	196	61.8
CPU INT.	-.30	-.17	13.0	157	54.2	33.8

N= 75 DF= 73 R@ .0500= .2272 R@ .0100= .2957

The table also includes the means, etc., of the parameters to give a better feel for the nature of the variables.

From the table it can be seen that the swap rate has the strongest influence on both RT and LRT. As well, both swap and disk waits have a positive influence on the dependent variables. It should be noted that the correlation values for the disk queue are not strong enough to be significant. Finally, it is interesting to see that while the uninterrupted cpu levels have a significant influence on RT, the quantity is not correlated with LRT. The fact that it negatively correlates with RT again supports the notion of the dominance in most of the runs in this experiment of the file access rate in the PC Factor.

5.7 SAMPLE APPLICATIONS.

The following hypothetical situations are given to help indicate areas where the information of this thesis may find application. In these examples it shall be assumed that the workload is moderately uniform and consists mainly of a short interaction type environment, such as editing, compiling, and executing of short or highly interactive programs. Thus the regression equation of model M2 with PC factor equal to 12 shall be used.

Suppose there was a UNIX installation which had 700 blocks of main memory available for users and that they were concerned about their heavy demand periods when there are an average of

eight user processes with an average size of 200 blocks each. To find their current reaction time we substitute into equation 5.1:

$$RT = 588 - 16.9 * \frac{(700 - 6*200)}{700} * 100$$

This gives a reaction time(*) of around 2.8 seconds which is, indeed a sluggish system.

(1) One situation might be that they had the option of buying 8K words (256 blocks) of main memory but were unsure of the effect it would have. Using equation 5.1, it could easily be determined that it would reduce reaction time to a little over 1.7 sec. While it's a definite improvement, the responsivenss is still not that good.

(2) Suppose instead that they wanted to know how much more main memory they would need to reduce the reaction time to 1 sec. To find this, let X represent the amount of additional memory needed, then it becomes a simple matter of solving the following equation:

$$1000 = 588 - 1690 * \frac{((700+X) - 1600)}{(700 + X)}$$

This yields a value for X of 587 blocks or a little over 18K words.

(*) It must be remembered that the reaction time values calculated from the model are simply rough estimates. As well, they will generally form upper bounds since the synthetic workload did not take advantage of the UNIX concept of 'text' which would reduce the swapping load somewhat in real systems And from validation it was seen that the model tends to overestimate the actual values.

(3) Let us assume that the installation was able to purchase the extra 587 blocks of memory and thus reduce their reaction time to under one second. According to the natural laws of increased capacity, let us presume that soon after the purchase, management wished to add more terminal lines into the system and wanted to know what effect it would have. If the computing centre staff could estimate on average how many extra processes it would introduce into the system, say for example three with the same average size of 200 blocks, then it can easily be calculated that the reaction time would increase to 1.8 sec.

The results of this thesis could also be used in system software development. For example, if an installation wanted to guarantee its users a certain response level, then a valuable and easy step would be to change the routine which logs users onto the system ("/etc/init") such that it first examines the current responsiveness of the system. If it exceeds a given threshold, then the user would be denied access. The responsiveness of the system should be calculated over a sufficiently long period of time probably from within the operating system as part of one of its numerous process table look-ups. Currently UNIX has no load control mechanism.

6. CONCLUSIONS.

6.1 THE BASIC GOAL & RESULTANT MODEL.

The goal of identifying the major influences of the system was met. The number of processes, size of a process, and size of main memory were, as intuitively felt, recognized as the most influential in affecting the responsiveness of the system. The strongest statement of their influence is expressed in the quantity pctrmm, the percentage of remaining main memory (equation 5.1).

The exact effect of the cpu requirements of a process and the file system access rate (to the disk) were not studied as they are related quantities and cannot be varied individually in the context of this experimental set-up. This is partly due to the design of the experiment which, to keep the size of the experiment reasonable, had fixed the ratio of the two quantities. Generally it is felt that for small PC values (e.g. Those less than the mean disk I/O time), the file access rate was the dominant influence over the cpu requirements.

Through regression analysis, a model for the data was developed. It follows the data well but requires sizeable error bounds, which was confirmed through some elementary validation. Thus its use lies mainly in the trends it demonstrates and the rough approximations it gives to predictions, rather than in an ability to give exact results. Some sample applications were described. An important characteristic of the model is that the

responsiveness of the system can be easily determined on any UNIX installation. Since the size of main memory is constant, it is only necessary to go through one system table to calculate the average number of processes and the average size of a process.

6.2 THE TOOLS AND METHOD.

The data extraction tools for this experiment proved very satisfactory for the high volume nature of the information being collected. The constantly changing nature of the system , and probably of most systems, underscores the need to keep the measurement experiments concise so that the implementation does not span any major system changes. This may have contributed slightly to the error in our results. With this in mind, the internal system parameters should have been left for a separate experiment.

The factorial design of the experiment, while allowing us to see clearly the relationships of three of the factors, proved inadequate for the pc factor. Keeping in mind the above recommendation to keep the size of experiments small, the solution seems to lie in making a separate study to uncover the influences of cpu requirements and file system requirements of a process.

6.3 DOMAIN OF APPLICABILITY.

It is the purpose of this section to indicate the domain of

this work by summarizing the relevant points. Probably the most obvious constraint was the use of identical processes in the synthetic workload. While this does not reflect reality, an experiment where the sizes were non-uniform gave results which corresponded quite closely to the model which used processes of uniform size. More work needs to be done on this, but it is felt that except for situations where there is a wide variance in either the sizes or number of processes, the results of this work can be used successfully to estimate responsiveness. In doing so, it must be remembered that this project did not take advantage of the UNIX concept of 'text' or pure code, which tends to reduce the swapping load. Thus the model derived from this work is likely to give an upper bounds on responsiveness when used on natural workloads.

Further with regards to workload, the domain is oriented to highly interactive environments where cpu requests are all generally under one second, that is, where the workload consists mainly of editing, various sizes of compilations, and execution of programs which are either short or interact with the terminal. As well, the results are geared to systems where swapping occurs and will be of little value in cases where all processes fit into main memory. In such a situation, the size of a process and the size of main memory have no effect on responsiveness, and PCTRMM tells us little of the expected reaction time. This is not considered a serious restriction since those most interested in improving the performance of

their system will almost always be suffering from swapping.

Another departure from reality is that there is no terminal input. This was not considered too crucial as a good approximation using terminal output was developed. However it did require the creation of a slighlty altered version of reaction time which was quite sensitive to the changes in performance. This must be kept in mind when making a comparison with other evaluation results which have been conducted elsewhere.

6.4 FURTHER RESEARCH.

This study could be used as a springboard for the following endeavours:

(a) an experiment to see the exact influences of cpu and file system access requirements,

(b) a study to determine the divergence from the model that would be caused by varying the number and sizes of processes in a workload,

(c) an attempt could be made to alter the existing swapping and cpu scheduling algorithms such that they use the easily determined value of PCTRMM to adapt to changing workloads,

(d) a study to see the relationship between our priority based reaction time and response time,

(e) to investigate the relationship between this and a paging system,

(f) to determine the improvement on performance that a faster

disk configuration would have. This could include a cost-benefit analysis with main memory.

BIBLIOGRAPHY

[AN63] Anscombe, F.J. & Tukey, T.W., The Examination & Analysis of Residuals, Technometrics, 5,2, 1963.

[BO76] Boi, MM.L., et al, A Performance Evaluation of the LII SIRIS 8 Operating System: Methodology, Tools, & First Results, Modelling and Performance Evaluation of Computer Systems, North Holland Publ. Co., 1976.

[BU76] Buzen, J.P., Fundamental Laws of Computer System Performance, Proceedings of International Symposium on Computer Performance Modelling, Measurement & Evaluation, 1976.

[CH75] Chanson, S.T. & Ferrari, D., A Deterministic Analytic Model of a Multiprogrammed Interactive System, National Computer Conference, 1975.

[CH76] Chu, Y., Guest Editor, IEEE Transactions on Computers, Vol. C-25, no. 10, Oct. 1976.

[CH77] Chandy, K.M. & Reiser, M., Editors, Computer Performance, North-Holland Publ. Co., New York, 1977.

[DE72] Denning, P., The developing Theory of Operating Systems, Infotech Report #14 (Operating Systems), 1972.

[DE78] Denning, P. & Buzen, J., The Operational Analysis of Queueing Network Models, Computing Surveys, Sept. 1978.

[DR66] Draper, N.R. & Smith, H., Applied Regression Analysis, Wiley & Sons Inc., New York, 1966.

[FE72] Ferrari, D., Workload Characterization & Selection in Computer Performance Measurement, Computer, July/August 1972.

[FE78] Ferrari, D., Computer System Performance Evaluation, Prentice-Hall Inc., New Jersey, 1978.

[FO76] Fox, D.J. & Guire, K.E., Documentation For MIDAS, Statistical Research Lab., U. of Michigan, 1976.

[FO78] Fox, M.C., Machine Architecture & the Programming Language BCPL, Master's Thesis, U.B.C., 1978.

[GR78] Graham, G.S., Guest Editor, Computing Surveys, Sept. 1978.

[HA70] Hansen, P.B., The Nucleus of a a Multiprogramming System, C.A.C.M., Apr. 1970.

[HA76] Haberman, A.N., et al, Modularization & Hierarchy in a Family of Operating Systems, C.A.C.M., May 1976.

[HO74] Hoare, C.A., Monitors: An Operating Sytem Structuring Concept, C.A.C.M., Oct. 1974.

[KI77] Kienzle, M.G., Measurements of Computer Systems for Queueing Network Models, Technical Report CSRG-86, U of T, Oct. 1977.

[KO78] Kobayashi, J., Modeling and Analysis: An Introduction to System Performance Evaluation Methodology, Addison-Wesley, 1978.

[LE71] Lewis, P.A. & Yue, P., Statistical Analysis of Series of Events in Computer Systems, Proceedings of Conference on Statistical Methods for Evaluation of Computer System Performance, 1971.

[LI72] Liskov, B.H., The Design of the Venus Operating System, C.A.C.M., Mar. 1972.

[LI77] Lions, J., A Commentary of the UNIX Operating System, U. of New South Wales, 1977.

[MA??] Maydell, U., Computer Peformance Studies & Statistics, U. of Alberta.

[MU78] Muntz, R.R., Queueing Networks: A Critique of the State of the Art & Directions for the Future, Computing Surveys, Sept. 1978.

[RI78] Ritchie, D.M. & Thompson, K., The UNIX Time-sharing System, Bell System Technical Journal, Part 2, July/August 1978.

[SH72] Shemer, J.E. & Robertson, J.B., Instrumentation of Time Shared Systems, Computer, July/August, 1972.

[SM66] Smillie, K.W., Introduction to Regression & Correlation, Ryerson Press, Toronto, 1966.

[SR74] Sreenivasan, K., & Kleiman, A.J., On the Construction of a Representative Synthetic Workload, C.A.C.M., Mar.1974.

[SV76] Svobodova, L., Computer Performance & Evaluation Methods: Analysis & Applications, Am. Elsevier Publ. Co., 1976.

[TH78] Thompson, K., UNIX Implementation, Bell System

Technical Journal, Part 2, July/August 1978.

[TS72] Tsao, R.F. & Margolin, B.H., A Multi-factor Paging Experiment: II. Statistical Methodology, Statistical Computer Performance Evaluation, Academic Press, 1972.

[WA72] Walpole, R.E. & Myers, R.H., Probability & Statistics for Engineers & Scientists, MacMillan Co., New York, 1972.

APPENDIX A. Swapping and CPU Policies.

A. SWAPPING.

1. Routine and variables used:

sched - routine called to swap in all processes that it can from dsk.

runout - a global flag which is set and slept on by sched when there are no more READY processes out on disk. Thus other routines can test runout and if appropriate, wake-up the scheduler.

runin - a global flag which is set and slept on by sched when it was unable to swap in all the READY processes. As well as being accessible to other routines, runin is tested every second by the clock routine.

2. Swapping-in policy.

i) when: sched is only called when a READY process is out on disk, and thus wants into main memory. This can occur due to two situations:

1. In the previous execution of sched, it was unable to load all the READY processes into main memory. Thus it set and slept on runin. In this case, sched will be awakened every sec. by the clock routine until there are no more READY processes left on disk.

2. In the previous execution of sched, it did load

all the READY processes into core (and thus set & slept on runout). Later, a process (which was not READY, e.g. sleeping due to I/O wait) out on disk became READY (i.e. was awakened). In this case, Sched will be executed whenever such a situation arises.

ii) who: The policy is based solely on the length of time a process has been out on disk. Sched starts with the READY process out the longest and tries to load all of them into core. If it fills all of main memory and there are still READY processes on disk, it will still try and load them in as long as (1) there are processes in core which are not READY (e.g. sleeping on low priority), - they will be successively swapped out to make room; or failing that, (2) the READY process(es)on disk has been there > 2 sec. And there is an in-core process (which is READY or sleeping on high priority) which has been in core > 1 sec.

3. Swapping out policy.

i) when: only when necessary as determined by the swap-in algorithm, i.e. not all the READY's out on disk will fit into core, etc.

ii) who: any process sleeping on low priority or being traced, and failing that, if the process on disk has been out there > 2 sec., then any process (READY or sleeping on high priority) that has been in-core > 1 sec. (starting with the one who's been in the longest, i.e.

based solely on elapsed time in core).

B. CPU SCHEDULING.

1. Prioritites for processes: -values vary from -127 to 127 (the higher the numeric value the lower the priority).
-there are set priorities for waiting for various events, e.g. -100 for swapping I/O waits.
-for user processes, values vary from 100 to 127, - it's based solely on the amount of CPU time (system & user) since its last sleep. This is regardless of context switches and swaps.
Specifically, when a process runs, it's priority gets lowered by 1 for each of the 1st 5 sec, and every 15 sec. thereafter.

2.Method.

The routine 'swtch' is responsible for doing context switches. When called, it simply goes through the process table and selects the highest priority READY process.

There are 2 ways of invoking 'swtch'. One is simply by doing a sleep. The other method is by using the flag 'runrun'. This flag is incremented whenever it had been

determined that there is a READY process with a higher priority than the current process (e.g. when a wakeup occurs, the newly re-activated process may have a higher priority). Runrun is tested (in the assembly code of UNIX) whenever a trap or interrupt occurs (which, due to the clock, is at least every 1/60 sec.). If it has been set, then after the interrupt has been proceesed, a call is made to swtch.

Summary of CPU policy.

-based solely on CPU time intervals(i.e. Intervals between its I/O's), exclusive of context switches & swaps.

-appears to be basically a loose, round-robin policy.

-there is no time quantum as such, though by the priority calculation, it will generally be at most 1 second (where it & the other processes are compute bound) and will be reduced as the amount of I/O increases (since the priority of a process after doing I/O, is reset to its highest possible value).

APPENDIX B: Selection of Reaction Time.

This appendix discusses the suitability of three slightly different definitions of reaction time. While a universal definition of reaction time doesn't seem to exist, it is usually defined as the time from the input of a command (usually via a carriage return) until the cpu starts to act on that command. This definition is inappropriate for our purposes since there is no terminal input. And so, we set up an experiment whereby we tested the sensitivity of the following three definitions of reaction time:

a) The above, usual definition is used. This will be called the "tty" method.

B) Record the times for all processes from when they wake-up (they are put to sleep whenever they must wait for an event such as I/O) until they are selected as the process to run next. This shall be refered to as the "all" method.

C) The same definition as in (b) except only for a select number of processes. This method shall be called "pri".

Several artificial workloads were constructed. While their make-up was based on commonly used programs such as compiling, editing, and system commands there was no formal attempt to make these workloads representative since we were only interested in the relative sensitivity of the three methods to workload changes. For the editing portion of the

workloads, a fixed sequence of commands were typed in at the terminal. These commands were present in each run since the "tty" method only picks up such reaction times.

Basically, the experiment consisted of running the edit commands with and without each workload for each of the reaction time methods. This gave a good indication of the sensitivity of the methods to changing workloads.

After running the experiment, some definite statements could be made about the three selection methods. The "all" method is simply too insensitive. The "tty" method is much better ,but suffers from the following: it requires someone at the console to feed in commands, it filters out the lower priority processes which tend to fluctuate most with changing environments, and it has slightly higher overhead. It does have the advantage of conforming to the original definition of reaction time. The "pri" method is the most sensitive to workload changes.

On these findings, we chose the "pri" method for our performance parameter. To avoid confusion with the generally accepted definition of reaction time, we shall label this as the "priority based reaction time".

APPENDIX C: Statistics of the Production Workload.

Statistics gathered on the UNIX system.
Length of observation: 151.8 minutes.
Default unit of time: millisec.

A. DSK INFORMATION.
# of async. requests: 27227 (32.5 % of the total)
# of seek overlaps: 10417 (12.4 % of the total)

dsk wait times (incl. transfer):

%	msec.	
0.0	-	
6.5	0	***
17.5	25	********
9.7	50	****
9.7	75	****
9.2	100	****
6.2	125	***
5.6	150	**
4.7	175	**
4.2	200	**
4.8	225	**
3.3	250	*
3.2	275	*
2.9	300	*
2.1	325	*
1.9	350	
1.6	375	
3.2	400	*
0.7	475	
0.6	500	
0.4	525	
0.5	550	
0.3	575	
1.2	600	

total: 83848

MEAN: 161.38 STD DEV.: 134.68

```
histo. of qlengths for dsk:

     %    #
  31.5 ! -      ***************
  27.6 ! 1      ************
  19.6 ! 2      *********
  11.7 ! 3      *****
   5.9 ! 4      **
   2.3 ! 5      *
   0.9 ! 6
   0.3 ! 7
   0.1 ! 8
   0.1 ! 9
   0.1 ! 10

  total: 83848

  MEAN:   1.47   STD DEV.:     1.48


B. SWAPPER INFORMATION.
    swap rate:  77.1 swaps/min. ( 1.28 swaps/sec.)


size of swaps:

     %   words
   0.0 ! -
  28.2 ! 0      **************
  48.9 ! 2048   ************************
   7.8 ! 4096   ***
   2.6 ! 6144   *
   5.8 ! 8192   **
   2.5 ! 10240  *
   1.6 ! 12288
   0.7 ! 14336
   2.0 ! 16384
   0.0 ! 18432
   0.0 ! 20480

  total: 11699

  MEAN: 3198.22   STD DEV.:  3900.82

size in 512-byte blocks:
  MEAN:    14.31  STD DEV.:    13.32
```

wait times (in swap Q only) for SWAPPER

%	msec.	
48.6	-	************************
47.7	1	************************
0.2	2	
0.1	4	
0.0	8	
0.0	16	
3.5	32	*

total: 11699

MEAN: 29.42

wait times (dsk wait) for SWAPPER

%	msec.	
0.0	-	
1.2	0	
7.1	25	***
6.6	50	***
8.0	75	***
8.2	100	****
6.3	125	***
5.1	150	**
4.8	175	**
5.4	200	**
7.5	225	***
5.4	250	**
5.4	275	**
5.0	300	**
4.0	325	**
3.3	350	*
3.0	375	*
6.1	400	***
1.4	475	
1.2	500	
0.9	525	
1.0	550	
0.7	575	
2.4	600	*

total: 15810

MEAN: 231.55 STD DEV.: 147.24

C. CPU INFORMATION.

uninterrupted CPU intervals

%	msec.	
0.0	-	
2.7	0	*
19.3	1	*********
0.0	2	
22.0	3	**********
14.9	4	*******
0.0	5	
7.2	6	***
4.4	7	**
0.0	8	
3.0	9	*
2.2	10	*
0.0	11	
3.4	12	*
2.2	15	*
3.6	20	*
4.4	30	**
2.6	45	*
3.0	60	*
2.1	90	*
1.0	120	
0.5	150	
0.8	180	
0.8	240	

total: 122685

MEAN: 19.33 ST. DEV.: 37.21

D. REACTION TIME INFORMATION.

reaction times:

%	msec.	
0.3	-	
20.0	1	**********
0.0	2	
19.9	3	**********
5.3	4	**
0.0	5	
7.5	6	***
3.0	8	*
7.1	10	***
2.4	15	*
0.8	25	
1.5	50	
3.9	100	*
6.0	200	***
7.5	400	***
2.4	800	*
6.9	1000	***
3.0	2000	*
1.3	3000	
0.9	4000	
0.4	6000	

total: 8198

MEAN: 376.20 STD DEV.: 864.09