Copyright © 1979, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

BENCHMARKING UNIX:

A COMPARATIVE STUDY

by

L. F. Cabrera

Memorandum No. UCB/ERL M79/77

15 November 1979

Benchmarking UNIX †
A Comparative Study

Luis Felipe Cabrera ‡

Computer Science Division
Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California
Berkeley, California 94720

ABSTRACT

A performance analysis of computer systems on which UNIX runs is made. Measurements done on three very different computer systems are presented. No one system presents itself as a clear best.

Four different approaches to characterizing the work load of a system are used. Their relative merits are discussed as well as decisions on the display of the data. A thorough analysis of the performance indices is presented in graphical form.

A fully transportable benchmark was used thus allowing for similar measurements to be easily obtained in any installation in which UNIX runs.

† UNIX is a trademark of Bell Laboratories.

‡ This research was supported in part by the National Science Foundation under grants MCS78-24618 and MCS78-07291.

Benchmarking UNIX †
A Comparative Study

Luis Felipe Cabrera ‡

Computer Science Division
Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California
Berkeley, California 94720

CONTENTS

Acknowledgements

I am very grateful to all my professors, colleages and staff of the Department who have helped me with this project in their various ways. The amount of support I received from them while working in different stages of this project far exceeded all my expectations.

In particular, I wish to thank Domenico Ferrari for encouraging me to work in this area, for his patience and thoroughness when revising earlier versions of this report and for his enthusiastic support for my work. I would like to thank Robert Fabry for the stimulating discussions we had which certainly made this a better report. To Robert Kridle for all the special conditions he set which allowed me to gather data in different systems. To the different members of the PROGRES Group for their helpful remarks about the project and to the EECS support staff which introduced me to the software used for plotting.

Finally, I want to specially thank my wife Marcelle for her continuous encouragement, patience and understanding for my work. Without her active support my present studies would certainly have never materialized. No words I know can really express my gratitude.

Benchmarking UNIX
A Comparative Study

1. INTRODUCTION

Since the first release by Bell Laboratories in 1974 of the Timesharing Operating System UNIX [1], it has run on a wide variety of hardwares. Primarily among them is the PDP-11 family manufactured by Digital Equipment Corporation. Moreover UNIX also runs on Digital's recently released VAX 11/780 even though its hardware presents drastic differences with the PDP-11 family. To mention just one, the VAX 11/780 uses 32-bit words while the PDP-11's use 16-bit words.

It has then become of interest to measure the effect that the underlying hardware has on the performance of the various computer systems on which UNIX runs. A complete study of this requires benchmarking UNIX in *all* distinct hardware configurations on which it runs. Such a formidable task is beyond our means, but a first step towards its accomplishment is presented here.

In what follows a study of three computer systems is presented. Each was deliberately chosen to be very different from the other two. In fact not only their underlying hardware was different but their work loads were quite different as well. The aim was to observe the effect on performance of leaps in technology and design rather than to observe the effect on performance of specific changes made to a configuration (i.e. we tried to avoid comparing systems which differed only in few aspects of their configurations). Nevertheless, each of the systems selected for study went through a change of configuration while the study was being done. This has also permitted us to observe the effect on performance of these changes.

Moreover the intrinsically different modifications made in each of the systems provide good examples of what may be achieved when one performs such a change in an installation. In one system a cache memory was temporarily installed, to the second one megabyte of main memory was added and to the third a new disk drive was added.

All the systems studied operate in the Electrical Engineering and Computer Science Department of the University of California at Berkeley. Measurements were taken during a span of four months (April through August 1979) beginning by the third week of the Spring Quarter of 1979. As much as was possible, they were taken every other week while the systems were in operation.

The rest of this paper is divided as follows: Chapter 2 contains the basic design decisions about the experiment. Chapter 3 discusses the measurement techniques and the reduction of the data. In Chapter 4 we present some graphs which compare the performance of the systems. A discussion of the the data presented in them is given. Chapter 5 shows the results of the distinct upgrading changes which the systems went through. In Chapter 6 we present some conclusions.

In the Appendices the bulk of the graphs which reduce the data gathered are presented. The only exception is **Appendix A** which contains the script used together with the files and programs required by it. **Appendix B** contains all percentiles and cumulative distribution curves for each event and system considered. **Appendix C** contains all graphs which show our performance indices plotted against each of our characterizations of load.

2. BENCHMARKING UNIX

2.1. Preliminaries

A great advantage in benchmarking an operating system which runs on several distinct computer systems (whose underlying hardware configurations may be very different) is that one can define a high-level language benchmark and use it unmodified in each of the systems. Thus *functional equivalence* of the benchmark is immediately obtained [4]. That this equivalence corresponds or not to a *resource consumption* equivalence will depend on the actual implementation of the operating system. Determining whether or not it yields a *response time* equivalence is the main purpose of our study. One would expect that evolving hardware should produce better response time for a given task under the same load conditions. But, as we shall see, it can happen that effects of hardware and software changes may result in the degradation of response time of a given task. Of course other advantages that these changes may bring with them might make them acceptable.

Choosing response time as the main observed performance index of our study is justified by our belief that in any timesharing system, from the user's point of view, what counts most is the responsiveness of the system to user supplied tasks. Nevertheless, using standard UNIX instrumentation we have also monitored system time and user time for each of our tasks. This has proven very helpful in analyzing possible causes for effects noticed while studying the performance of each system.

Characterizing the *work load* is a central problem in any benchmarking experiment. Questions such as: under what conditions should the experiment be done? or in our case of benchmarking an operating system, under what work load should one run the benchmark? have been given different answers.

Perhaps the best way to answer both of them is by setting an experiment where one runs a benchmark in each system on a stand alone basis and where complete control of the work load is achieved by loading the system with some kind of synthetic, internal or external, driver. This method has as main advantages its total reproducibility and absolute control of all the activity being done in the system.

The main difficulties it presents are related with the availability of tools to drive the system and with the design of the work load that those tools will implement. The (artificial) work load under which the system is to be studied must be such that results obtained from its usage should yield information about the system's performance under its natural work load.

Lack of such tools at the time when we had to make this decision lead us to the less sofisticated approach of monitoring the systems periodically under their *natural* work load with the aid of a *shell script.** Then, in order to analyze our results, we had to find satisfactory characterizations of load with respect to which our performance indices corresponding to a given task would be studied.

We have characterized work load by taking two different, yet related, points of view which in all yield four different characterizations. Two of them are based on an *outside* view of the system (how much work is being done on it) and the other two are based on an *inside* view (how much work is being demanded from the system).

The first point of view is carried out by counting the number of users logged in when a certain task (in our benchmark) is run while for the second one we count the number of processes present in the process table. Although these two

* In UNIX, the *shell* is a command language interpreter through which the user inputs its tasks to the system [1]. Moreover it is also a programming language [2] and thus shell programs *(shell scripts)* can be written with it.

characterizations of load prove to work reasonably well, it was felt that they were too rough. Thus two refinements, one for each characterization, were also implemented. We named them "Active Users" and "Real Processes". The former is straight forward to define: it is the number of users logged in which are executing some task when our benchmark begins to run. The latter attempts to count processes which are likely to run while one of the tasks in our benchmark will be running. Its precise definition is the following: a process in the process table is a "Real Process" if it is not a *login shell* or a *sleeping shell*.

The three systems we monitored had the C *shell* [2] running in addition to the ordinary *shell*. To standarize and make portable our experiment in our environment, we used C *shell* facilities in all three systems, although no essential new features of this shell were used. Thus a script for the C *shell* was written and run on the background in each of the systems, throughout what constituted our data gathering period. Its text is reproduced in **Appendix A** together with the text of all programs and files used by it. **Appendix A** also has documentation regarding usage of the script and changes needed to convert the script into one for the standard UNIX shell.

2.2. The Script Driver

Our strategy for monitoring the system's responsiveness was to run periodically a set of predefined benchmarks. This was achieved by using a shell script which contained these tasks together with commands which gathered statistics about the system and the time it took the tasks to complete. The *script*, after cycling through its instructions once, would go to sleep and then wake up to initiate the cycle again. This was achieved using the command *sleep 1200*, which would send the process to sleep for 1200 seconds. In this form we could obtain an interval of at least twenty minutes between any two runs of the script.

This data gathering method is not new. It can be categorized as a time-sampling tool ([3] Section 2.4 pg. 57) and in fact is very similar to Karush's [8] *terminal probe* method. The only difference with Karush's method arises from the units used in presenting the measurements. We use the output of our measurement tools while Karush goes through an intermediate step of determining a unit of responsiveness, based on running simultaneously different number of copies of the script and measuring the performance indices, and then using that unit to present the measurements.

In UNIX the natural way to implement a sampling tool of this nature is by using a shell script. In fact the way to implement any task that one wants performed every time a predetermined event occurs is through a shell script. UNIX itself has scripts embedded in it, to mention just one, every time the system comes up the file /etc/rc (which may contain several scripts) is executed.

Moreover, for some time now, Robert Fabry has been running here in Berkeley a script which is similar to ours in its general form. Our knowledge of his work helped us make design decisions when building ours. The main differences between the two scripts come from the fact that we gather statistics related with our internal characterization of load, that we have chosen a different set of tasks and in the length of completion time of each task.

Although running our script affects the load of the system, and thus its responsiveness, it was felt that this was irrelevant for a comparison study because all systems were going to be presented with the same script. In fact the main purpose of this experiment is precisely to observe how each system reacts to this stimulus.

Our commitment to use standard UNIX features, for portability reasons as well as for assuring the functional equivalence of the benchmark, made us decide upon the usage of the *time* command [6] as the measurement tool for our tasks. The *time* command returns, upon the completion of the command with which it is called, three measurements: the elapsed time during the command *(response time)*, the time spent in the operating system *(system time)*, and the time spent executing the command *(user time)*. These last two are accurate to one tenth of a second while the first is accurate to one second. The *time* command truncates, does not round off, and thus one always obtains lower bounds of the actual elapsed times.

This low resolution of *time* together with our desire that no individual measurement be off by more than 10% led us to consider tasks which would never take less than five seconds to complete. The 10% error bound could then be achieved by adding half a second to the response time given by the *time* command.

On the other hand we should not come up with a set of tasks that would overload the system every time they were run, if we were to run them periodically for an extended period of time. Our script was designed as a compromise between these requirements.

Four tasks were timed at each run: a C compilation, the execution of a CPU-bound job, the retrieval of the manual page of the on-line copy of the UNIX Programmer's manual, and a mix, which included the above three plus some IO bound tasks and two system commands; the mix was called "Script", even though it did not correspond to the actual script we were running.

Measurement results for each session were kept in files which had to be extensively edited before the data gathered in them was used. Not editing the data while it was being gathered was another consequence of our desire to keep the script as small as possible.

2.3. The Tasks Chosen

The goal of functional equivalence also forced several decisions on us. Our original intention was to measure a set of tasks which were representative of the user's tasks and that would stress distinct aspects of a configuration. Given that we certainly wanted to look at a CPU-bound job the obvious choice for a mixed task, given our university environment, was to do a Pascal compilation of a short CPU-bound job. At the time no Pascal Compiler or Interpreter was available for the VAX 11/780 and so this solution had to be disregarded. Second best to that, again due to our environment, was the use of the language C [7].

Our choice of writing a (trivial) CPU-bound program which had nine variables and two constants declared in it, and consisted of two nested **for** loops, versus that of writing a one-line program, was motivated by our desire to exercise somehow the compiler. We felt that with a simpler program most of the time of the compilation would be spent loading the compiler instead of running it. That may very well be true even with our program but, again, the phantom of overloading the system with the benchmark made us decide on this program. An obvious alternative would have been to run a precompiled CPU-bound program and compile a larger program each time.

Our CPU-bound program executes its inner sequence of instructions 100,000 times. In all it performs 1,200,000 integer arithmetic operations plus those needed for the **for** loops (100,000 increments and comparisons). It was decided against using floating point arithmetic because one of the measured systems, the 11/40, does it in software. This precluded any meaningful hardware oriented comparison.

As for our third task, rather than having a strictly IO bound task (such as copying a file from disk to disk) it was thought to be more interesting to present the system with a task which would use more features than just the speed of the disks and the efficiency of the IO subsystem. Thus our choice of the command *man man*, which retrieves the entry for the manual page out of the on-line copy of the UNIX Programmer's manual. This command was chosen because as the on-line copy is kept in compact form on disk (to save space), it is retrieved using the formatting program *nroff* which is a utility program widely used in text processing. In the script this task was given the name "IObound" although from a resource utilization point of view it is not a truly IO bound task.

The last task timed is what we called "Script". For technical reasons (in one of the three measured systems any given terminal at any given time can not have more than five processes associated with it) the *time* command could not be used in a nested way. Instead we used the *date* command (as a timestamp with other measurements) to determine total elapsed time. The error per measurement introduced by this method is far less than our 10% goal, but the drawback is that with it we could not obtain system time nor user time for this task.

Script, besides including the other three tasks, also includes six short IO bound tasks, appending the contents of the temporary user file YYY1 to six other user files, a couple of *date* commands and two commands which gather data about the system. In fact *ps - alx* produces a long listing of all processes in the process table, *who* gives data about users logged in, and *wc - l* counts the number of lines of a file given as input. In our case we used it to count the number of lines *who* gave which is exactly the number of people logged in the system at that moment. Script is thus a fairly balanced task.

2.4. The Systems Measured

Our main motivation when choosing which systems to monitor was diversity. We wanted to have systems which were as different as possible from each other. Our decision to monitor only three systems was mainly due to the large amount of personal care that had to be given to the script and files during the whole data gathering period. To mention one, the growth of the file PS is so big that it had to be transferred to tape every other day in order not to use up too much disk space.

Our choices were the following:

[1] PDP 11/40 with 200K bytes of main memory, one DIVA disk controller with three DIVA disk drives which have 50M bytes disk packs. This system has 23 ports and no floating point arithmetic unit.

[2] PDP 11/70 with 1.3M bytes of main memory, a 2K byte cache memory, one DIVA disk controller with four DIVA disk drives which have 50M bytes disk packs. This system has 81 ports.

[3] VAX 11/780 with 512K bytes of main memory, 8K byte cache memory, one Digital disk controller with one RP06 disk drive with 177M bytes disk packs. This system had 16 ports.

Throughout the rest of this paper we shall refer to them as the 11/40, the 11/70 and the VAX respectively.

The configurations described above correspond to the predominant configurations each system had during our data gathering period. As we mentioned earlier, each of the three systems underwent changes in its configuration during our data gathering period.

The 11/40 was equipped with a 2K byte cache memory for a period of six weeks. Measurements made before the cache was installed and after it was removed show no statistically significant difference. Thus the system is considered not to have been altered when considering sample points corresponding to the period before the cache was installed and after it was removed.

To the 11/70 1M byte of main memory was added two weeks after we had begun gathering data. It only had 300K bytes of main memory before that change. Disk configuration remained essentially unaltered.

During the summer the VAX was supplemented with a CDC 9400 disk controller and a CDC 9762 disk drive with 80M byte disk packs. Even though we had sufficient data for our report, it was felt that extending the monitoring period for this system would provide us with interesting information, as it did. Towards the end of the summer new ports were added to the VAX configuration but we had already stopped our data gathering operations.

The work loads of the three systems are as diverse from each other as their underlying hardwares. The 11/40 is mostly used for administrative matters. The 11/70 is mostly used by undergraduate students in coursework related activities (thus, a lot of program development and testing is done). The VAX is primarily used by advanced students in research related tasks.

Given this diversity of natural work loads, the problem of work load characterization in order to compare the responsiveness of the different systems becomes very important as well as more difficult.

Clearly the ideal situation for a perfect comparison is when one has total control of the work load in each system and is able to make it the same everywhere. But, as we said before, no tools to load the systems appropriately were available at the time, so we compared the systems from the points of view of our four characterizations of load described in section 2.1.

These are:

(i) number of users

(ii) number of active users

(iii) number of processes (in the process table)

(iv) number of real processes (in the process table).

Of course, when analyzing our results, considerations peculiar to individual system work loads and/or users inevitably permeated into the analysis.

3. REDUCING THE DATA

Our script provided us periodically with data points which were gathered in appropriate files. One of these files, PS, had to be written into tape every other day because of its great growth. This was caused by our desire to retain all the information obtained from the systems and thus be able to recover any desired measurement. After the data gathering period, with the help of timestamps, it was easy to correlate the distinct pieces of information with one another.

We were thus able to plot mean user-time, mean system-time and mean response-time versus "Users", "Active Users", "Processes" and "Real Processes" for each task in each system. The procedure is simple: for each performance variable of interest gather together all data points for a given value of users, active-users, processes, real-processes, and compute the mean, standard deviation and standard error of the sample. The standard error is obtained dividing the variance by the number of points in the sample and then taking the square root the result. It is used to determine how "stable" a sample is. A small standard error gives us confidence that our sample is large enough so as to be representative, or that the measurements are stable enough so that one requires a small sample to characterize the value.

Whenever we found that the standard error was too large (as normally happens when we are interested in response time) we plotted "percentile" curves. In this case, instead of plotting the mean of a given distribution we ploted the percentiles of a distribution (for example, the 80th percentile of a distribution is that value below which 80 percent of the observations fall). In percentile terms the median of a sample corresponds to the 50 percentile. We chose to plot the 75, 83 and 90-percentile curves to have a variety of views for comparison, even though our analysis was mostly based on 90-percentile curves in which the minimum size of a sample was carefully chosen so as to leave out the outliers.

It is regrettable that in [5] Y. Bard mentions all of these concepts but makes no use of most of them. We found that for the total elapsed time, which we have been refering to as response time and we will continue to do so, percentile curves provided the best way to analyze the data. Mean curves, i.e., when one plots the mean of a distribution, have the observed property of smoothing too much the results and, unless one makes statistical assumptions on the distribution of the samples, few things can be said about the likelihood of a measurement being in the neighborhood of a plotted point. This is not the case with percentile curves. Nevertheless, for system-time and user-time, which proved to be very stable performance indices under each of our four characterizations of load, one can (and should) obtain meaningful information from mean curves also.

When comparing the different systems and analyzing the effect of a change due to tunning efforts or upgrading changes, we chose to use 90-percentile curves. This choice was primarily motivated by the fact that 90-percentile curves provide a very good "worst case" analysis with the advantage of not considering most of the outliers. The latter is obtained by choosing appropriately the minimum size for a sample to be plotted.

Determining the minimum size a sample must have in order to be representative is a very important design decision that normally goes correlated with the data being collected. This problem appears in any sampling procedure but becomes specially delicate in our case because we want to make no assumptions on the probability distribution of the samples. Given that our main interest is response time and that ocasional outliers in a sample may influence tremendously the mean and standard deviation of the sample, thus distorting their meaning, we did not use mean curves for our analysis. They have the disadvantage of always considering the outliers that might occur in a sample. The negative effects caused by outliers are minimized when the size of the sample is large, say 100, and in this case they become quite irrelevant. In our case, though, such large samples are practically impossible to obtain for each value of the variable characterizing load in each of the systems.

An added advantage of 90-percentile curves is that once one has obtained one for a given task on a given system, the sole assumption that the work load of the system will remain constant enables us to assert that the task will perform in at most the time given by the curve with .9 probability.

The idea of arbitrarily *clustering samples*, i.e., of considering in one sample all the measurements which correspond to a predetermined number of values of the variable, allows one to obtain larger samples but has the possible drawback of rounding up too much. We prefer to plot means and standard errors of smaller samples and also percentile curves. The 90-percentile curves obtained this way certainly provide estimates that in an absolute sense may be pessimistic but, for the purpose of relative comparisons, are perfectly valid. If one clusters too many samples, what normally happens is that fine resolution is lost.

Perhaps the best experimental way to determine a sufficient sample size is to gather measurements until the standard deviation and the standard error are essentially stable. By this we mean that one should keep track of both quantities while new data-points increase the sample size and, when a monotonic nonincreasing behavior is observed, decide that the sample is large enough. In our case that would have meant to leave the length of the data gathering period open which was not reasonable to do. The moments of high load in a system are few during each day and so to require large numbers of data points per sample really implies a data gathering period of several months.

To determine the minimum size of a sample we opted for a mixed strategy. For each system we determined the largest value of the variable to be plotted for which we had a sample of at least nine measurements. This value was then taken as the upper bound of values to be plotted for that system. Then beginning from the smallest value to be plotted, we clustered samples (if necessary) until we had at least sixty points and only then, from the larger sample, the 90-percentile measurement was found. If the final sample had been obtained from clustering several samples the value of the x-coordinate was chosen to be the average of the first and last values of the x-coordinates considered in the clustering.

The size nine (9) was chosen because with high probability it would provide three distinct data points for the 75, 83 and 90-percentile curves. The minimum sample size of sixty (60) was used because it proved to effectively leave out global outliers for each measured task. By using the histogram of a task one can determine these outliers and then by observing the output obtained by plotting with different sample sizes one finds the desired minimum sample size.

Analyzing our histograms of sample size for the distinct characterizations of load we discovered that most of our samples had at least 20 points. We even had a 225 point sample for active users on the VAX. In retrospective, we feel we had sufficient data for the VAX with 1368 sample points, a fair amount for the 11/40 with 536 sample points, but due to the large amount of ports that the 11/70 has we did not have enough with the 1007 sample points we gathered for that system. If our present experience can be taken as a model, one should try to obtain at least fifty measurements per value of the variable that characterizes load. This would almost certainly guarantee that outliers will not be present in 90 percentile curves without having to cluster samples.

4. COMPARISON AND ANALYSIS

In the following two chapters as well as in Appendices B and C graphical presentation of our measurements is given. **Appendix** B contains all histograms and cumulative distributions for each task measured in each of the three systems. **Appendix** C contains graphs of our performance indices for each task versus our four characterizations of load. There we have included graphs of mean response time with the standard error plotted (at correct scale) as vertical lines whose midpoint is the mean, and also percentile curves for response time.

In one graph three percentile curves; 75, 83 and 90-percentile are given. It is in those graphs where one can appreciate the differences among these curves when considered as representing responsiveness. It also becomes apparent from them the need to determine a proper minimum size for a sample to be plotted. All percentile curves presented in **Appendix** C were plotted using the 9-point criterion mentioned in Chapter 3. As one can easily check from reading the corresponding histograms, outliers were plotted in several occasions.

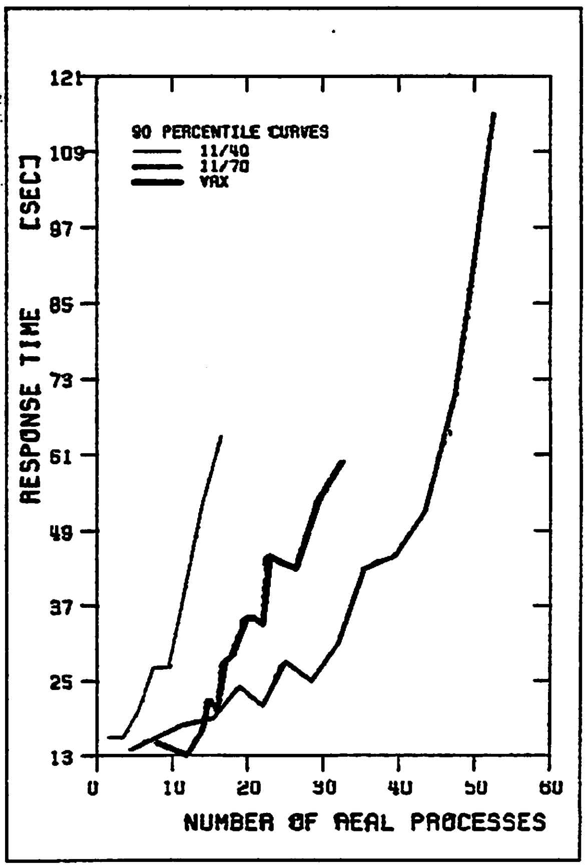
Of our four characterizations of load there is one which turned out not to be very useful for comparisons between systems. That one is the "Number of Processes". Its dependence on the number of ports a system has is so big that often makes two systems not readily comparable under it. Nevertheless, we could observe that on a given system and for a given task it was under this characterization of load that one obtained the greatest variations of response time as can be seen in the graphs presented in **Appendix** C. As for comparisons, the "Real Process" variable provided us with the greatest overlap between systems an thus gave the best overall view as to how they performed relative to each other. Of course if we had been monitoring systems which had similar number of ports, the variables "Active Users" and "Users" would also have provided excellent overlapping.

Before going any further a word of caution must be given once more: our characterizations of load do not even attempt to model the actual work load that a system has. Moreover, each system has its own peculiar work load and that might appear in our measurements in ways that may make it look comparatively inferior or superior. We must thus analyze all aspects of the data before asserting supremacy.

With the following four sets of three graphs each we are able to compare our four measured tasks from three points of view. All graphs presented in this chapter are 90-percentile curves. The exact number of sample points considered per task can be obtained from the histograms and cumulative distribution curves given in **Appendix B**. They are roughly as follows: 536 samples for the 11/40, 1007 samples for the 11/70 and 1368 samples for the VAX.

4.1. C Compilation

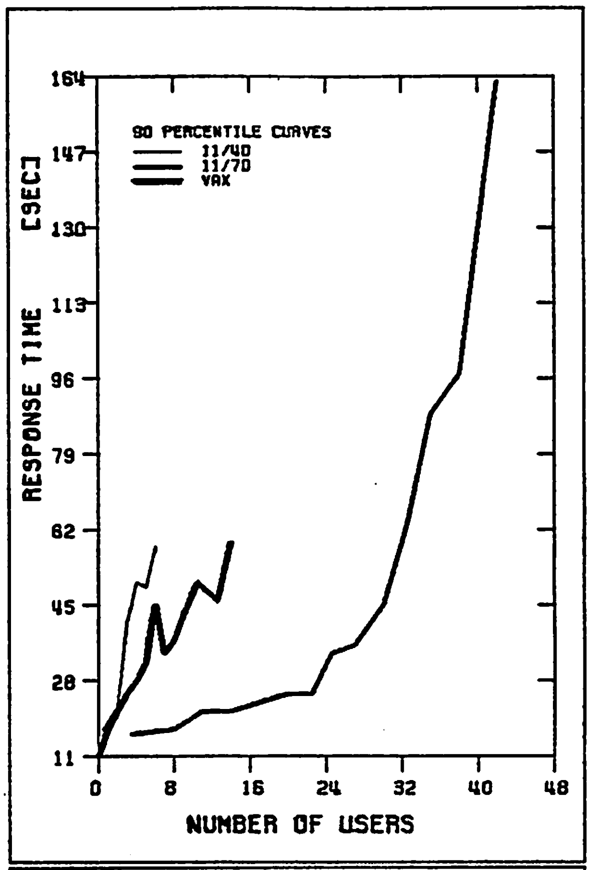
It can be said that the relative ranking for this task is the 11/70 first, the VAX second and the 11/40 last. Due to the configuration of the systems we know that with respect to the 11/70, the other two systems must begin swapping with less load. This may account for some of the differences in 'slope' of the response time curves.

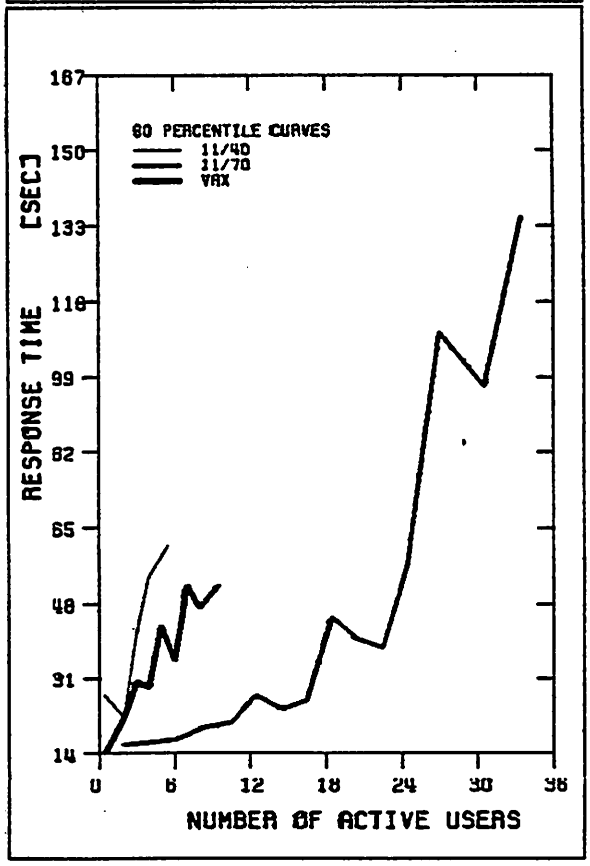


It is interesting to notice that under the characterization of load by

Real Processes the VAX shows better responsiveness than the 11/70. This is not the case when analyzing our other two characterizations of load. It seems clear that at the level of 43 real processes, 30 users and 24 active users the 11/70 becomes saturated. The slope observed in the saturated region of the 11/70 is larger than the one observed in the other two systems but close to being equal to the one observed in the 11/40.

Considering that the address space of the 11/40 is only 64K bytes and the fact that it only has 200K bytes of main memory, the amount of swapping in that system must be proportionally larger than in the other systems. So, we may assume that the 11/40 is working at the saturation level as soon as a minimal amount of load exists. However, given the size of the C compiler, we do not expect the slope to drop very much for this task in this system if more memory were added.





When comparing the 11/70 and the VAX, we can not fail to notice the linearity of the VAX's responsiveness under every characterization of load. This gives us some hope that more memory (as well as adding a second disk drive to be able to overlap IO activity) would improve the responsiveness of the VAX, and perhaps we might observe that in the higher range of load the VAX would perform as well as the 11/70.

We believe that the excellent behavior shown by the 11/70 before it reaches its saturation is due to the fact that the C compiler was almost *custom made* for the 11/70 while a more *portable* version of the compiler, which was partially generated using automatic tools, is used in the VAX, and also because the 11/70 configuration is much better balanced than the others.

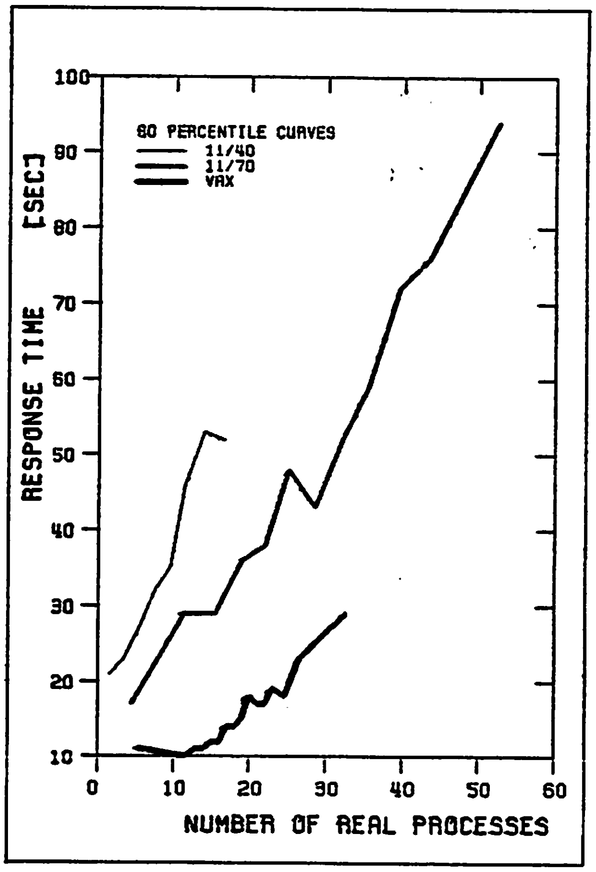
One must also have in mind that the program we compiled was small and thus we might not be measuring as much of the C compiler activity as we would ideally like and thus deficiencies in the IO subsystem of the VAX may be affecting its rating.

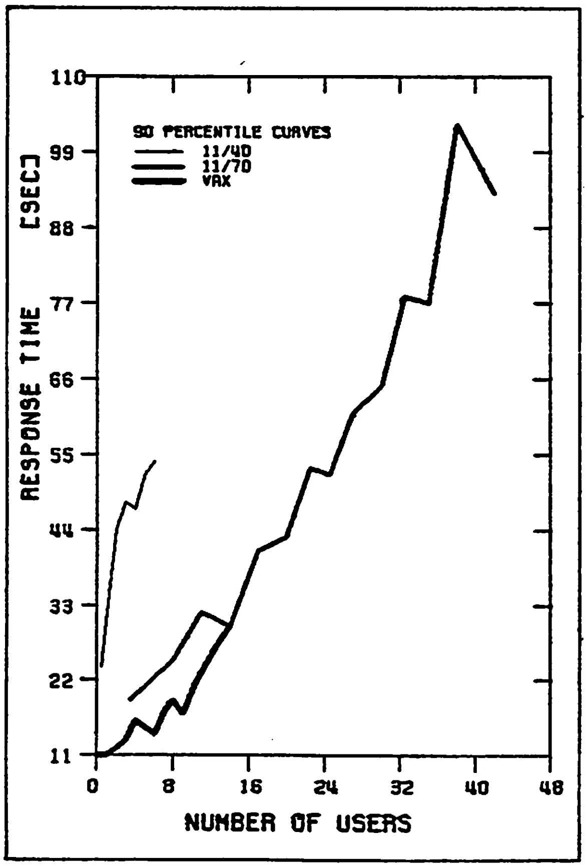
4.2. CPU-bound Job

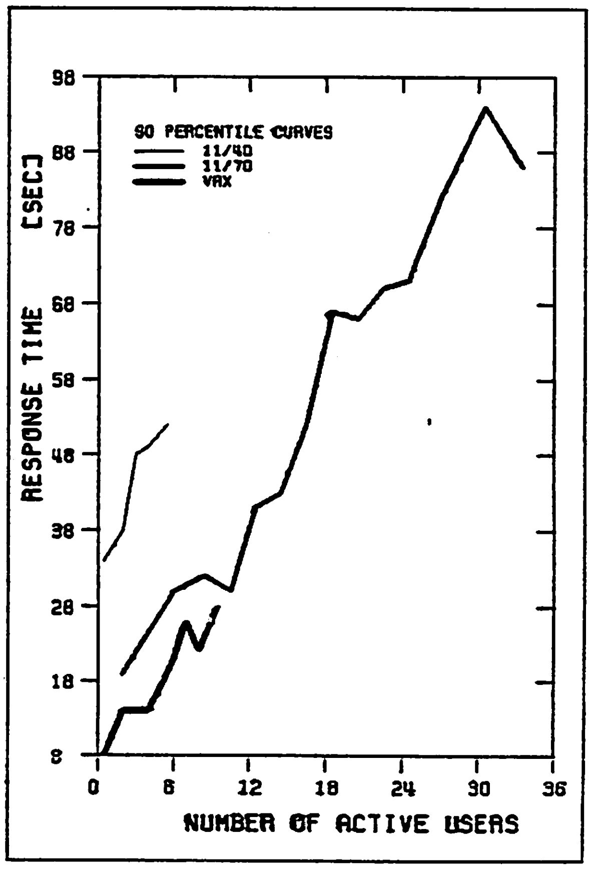
In this task, whose main purpose is to establish a rating according to speed of the CPU (we assume that scheduling is essentially the same in all systems), the VAX ranks first while the 11/40 ranks third. It is here that we can appreciate a marked difference between the three systems. At low load the VAX responds much faster that the other two systems while the 11/40 clearly shows to be a slower machine. Under our external characterizations of load we see that the rate of growth of response time of the VAX and the 11/70 are almost identical. In contrast to this a much better behavior of response time for the VAX is observed when plotted against Real Processes. We believe this change of behavior is due to the quality of users the VAX has ‡.

‡ The VAX 11/780 is the computer used for research computing in the Computer Science Division, EECS Department at the University of California, Berkeley. All the reduction of the data and the plots for this project were made on the VAX.

Real Processes seems to us to be the best yardstick by which to compare the systems, because, even though it does not differentiate among types of processes (say between editing and a lisp program), it appears to take better into account the amount of work demanded from the system. With this characterization of load one user that has several jobs running on the background makes the system's load jump to a higher state than what a user running one job does.







In the graphs per task included in **Appendix C** we can appreciate the great difference in the amount of *user time* that the cpu-bound job takes in each system. We have that at the *five* user level the 11/40 took on the average *21.0* seconds, the 11/70 *8.5* seconds end the VAX *5.66* seconds. At the *fourteen* user level the 11/70 took *8.61* seconds while the VAX took *5.81* seconds. We feel that *user time* for this task is a very good indicator of the responsiveness of the systems and the speed of computation of the machines. The VAX does show to be a faster machine. The difference between the 11/40 and the VAX is remarkable.

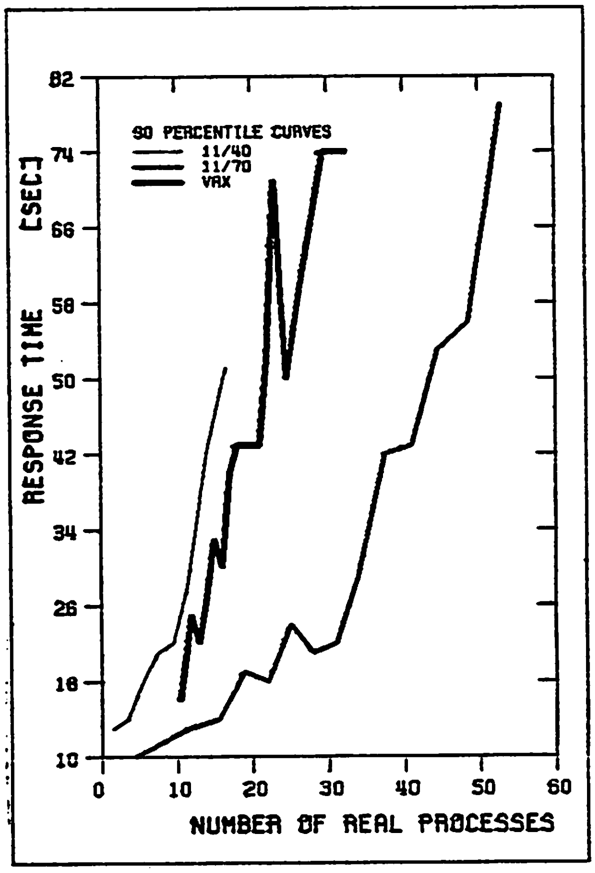
As it was expected for this task, the observed behavior of response time as a function of each characterization of load is almost linear. As no IO is done and the size of the program is small, the probability of being swapped out to disk while on the system is remote, and thus effects of IO bottlenecks and/or saturation are absent. Even in the 11/40, which has a rather small main memory, we do not perceive the effects of swapping at higher loads (which are normally observed in terms of a non linear behavior of the curve). Response time for the 11/40 does show to have a larger rate of growth than the other two systems.

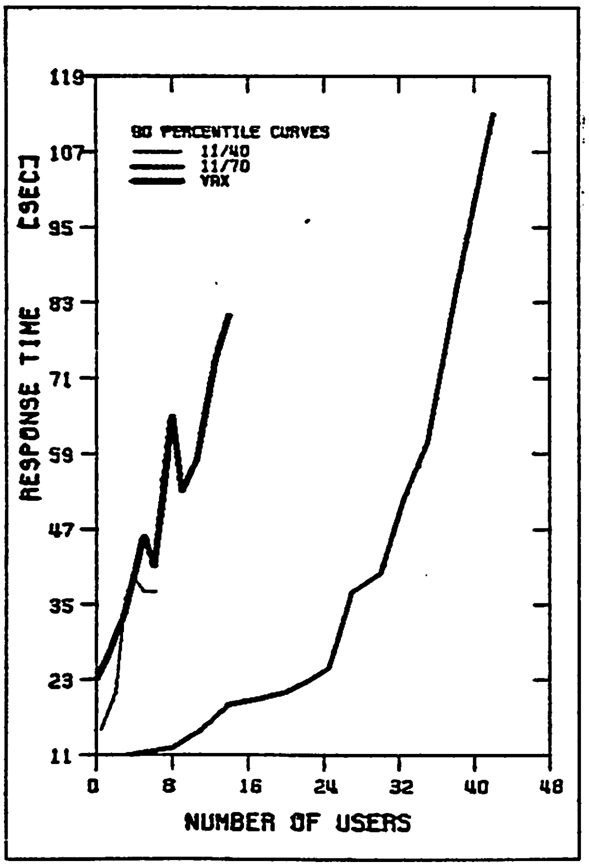
4.3. "Man man"

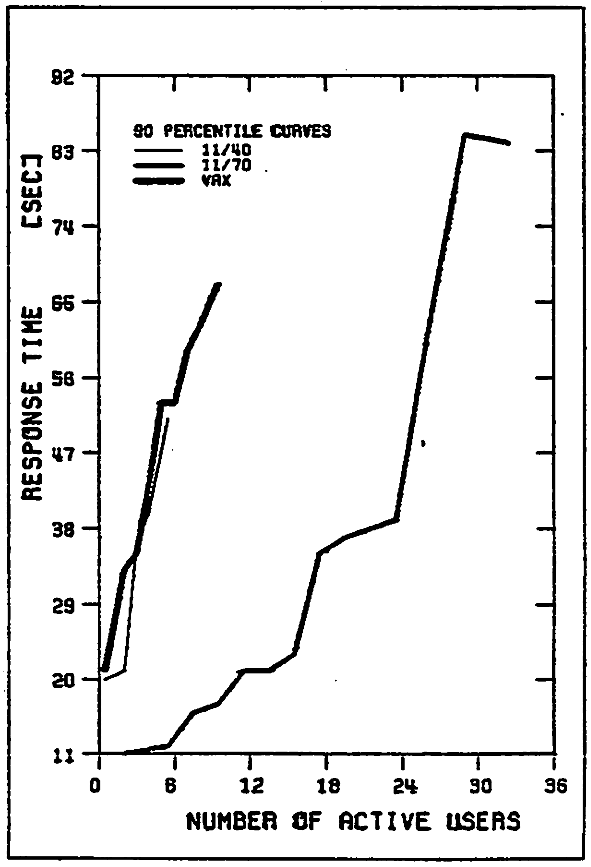
The execution and response time of the command *man man* came as a surprise. What is an obviously functionally equivalent task turned out to be a very different task from the points of view of resource consumption and response time. As it turned out, the three systems retrieved texts of different length and the way they were processed (using *nroff*) also depended on the system.

The file retrieved by the 11/40 has 744 characters in 136 words, the one retrieved by the 11/70 has 886 characters in 162 words and the one retrieved by the VAX has 1857 characters in 866 words.

Nroff, as a program, is also different in each system. On the VAX, for example, it knows and has to go through more macro libraries than what it does on the other two systems. Moreover, the number of entries in the VAX version of the manual is larger than the other two systems and so not only the VAX processes a longer text but also more time is spent searching through the proper directories to find the desired entries.







In each of our external characterizations of load the 11/40 and the VAX are pretty much in a tie, although the 11/40 appears to perform slightly better. When using Real Processes, however, the VAX shows better performance but here the nature and complexity of the jobs run on the VAX shows up making this curve very noisy even though samples of size sixty are used. This task, when run on the VAX, turns out to require a fair amount of resources; on the average *5.71* seconds of user time and *3.57* seconds of system time. Thus, its response time is very sensitive to all resource consuming demands the system may have, being they CPU demands or IO demands.

That the processing of this task is different in each system can also be appreciated by observing the ratios of mean user time over mean system time in each system. While in the 11/40 the ratio is *0.47* in the 11/70 it is *0.65* and in the VAX it is *1.60*. The reverse of the general trend in the VAX can be explained in terms of the larger amount of macros that are encountered in the (compacted) VAX version of the manual on disk.

Only in the curves corresponding to the 11/70 we can appreciate a non-linear behavior of response time. The saturation points are the same as for the C compilation. Specially the 24-active user level seems to point to where intense swapping begins.

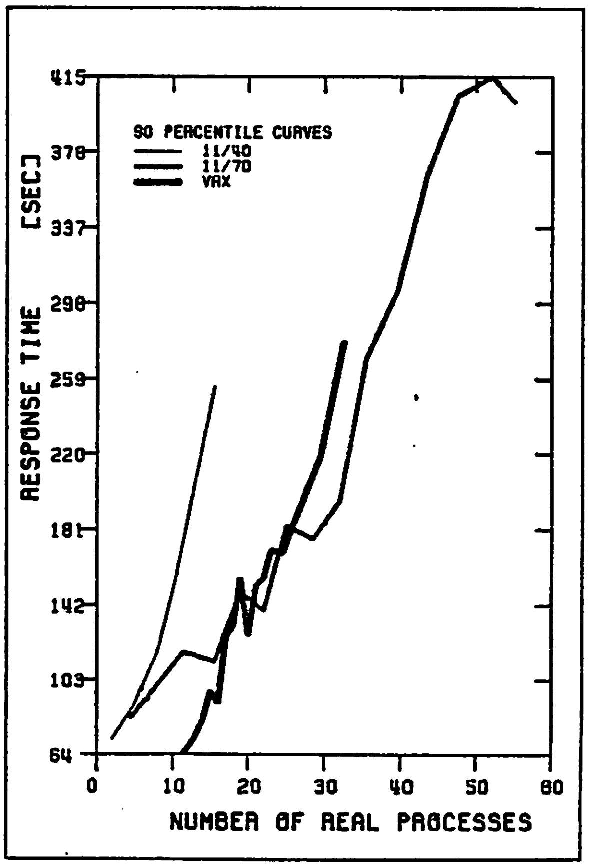
The large slope of the VAX curves are most probably due to the IO limitations of the system. We would expect the slopes to decrease with the addition of a new disk drive and appropriate re-configuration of the file system. A similar comment applies to the 11/40 but there, given the size of *nroff* and the small amount of main memory the system has, one would also need to add more main memory to decrease substantially the amount of swapping.

4.4. Script

Under each of our characterizations of load we see that the 11/40 takes a longer time to complete this task and moreover, the rate of growth of the 11/40 curve is the largest of all three curves. It is interesting to notice that under low load the VAX responds quicker than the other two systems. This suggests that with a better configuration we might get the VAX to outperform the 11/70. One would clearly need more main memory to increase the threshold of load that saturates the system and also more disk drives to increase parallelism in IO service.

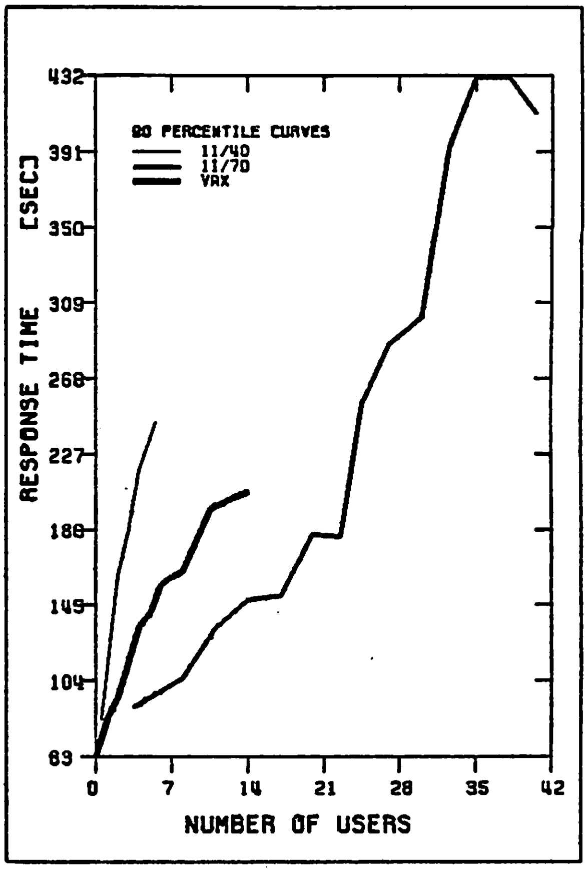
Our main hope that these upgrading changes would improve the VAX's rating come from the fact that when plotting against Real Processes the VAX's curve growth is comparable to the growth of the 11/70 curve when the 11/70 is saturated. In the Real Processes plot we can also appreciate that for almost half of the range the VAX performs better than the 11/70.

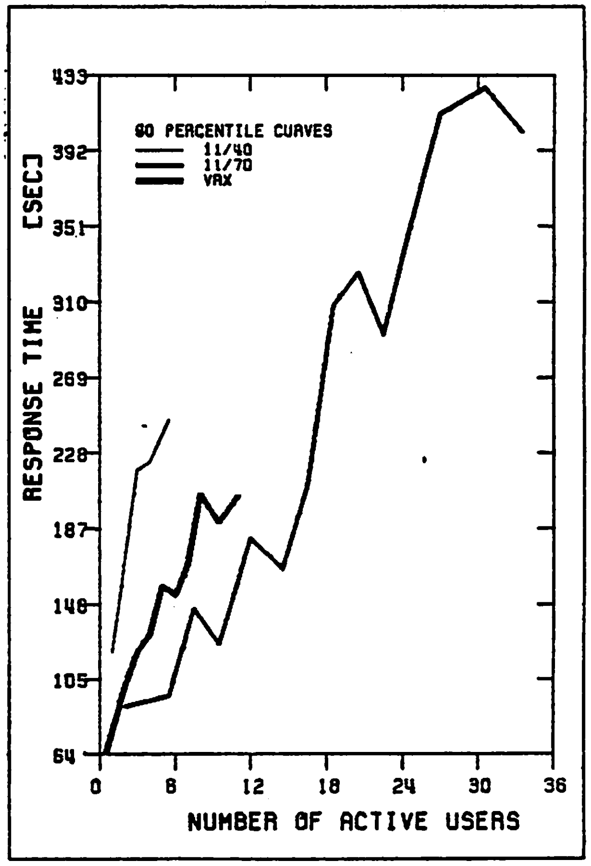
Given the balanced nature of this task and its over all exercising of the systems we can say that the passage from the 11/40 series to the 11/70 ones was very beneficial from the response time point of view. An added advantage to this change is the fact that the maximum size a process could have was doubled. This is a clear software improvement.



As for the VAX, we can see that under one characterization it behaves as the 11/70 but under the external ones its responsiveness is lower. This phenomenon has been constantly observed in each of the component tasks of Script and its justification, as pointed out before, lies in the nature of the VAX's users.

It is interesting to note the fairly smooth behavior of response time as a function of our characterizations of load. Only when plotted against "Active Users" we see that in one system, the 11/70, behaves somewhat different. It seems to us that the neighborhood of 20 active users is the border line where intense swapping activity begins in the 11/70. This explains the irregularities that can be seen in each curve corresponding to a task which somehow depends nontrivially on the IO subsystem. We believe that this phenomenon is not clearly observed in the 11/40 and the VAX due to the small amount of main memory that





these two systems have. In them swapping of large processes, like the **C** compiler or *nroff*, must occur at almost all levels of our characterizations of load.

5. EFFECTS OF UPGRADING CHANGES

As was mentioned in the Introduction, the three measured systems underwent configuration changes during our data-gathering period. In this chapter we shall see the effects of those changes. Unfortunately for us, the number of measurements we had for the 11/70 before more main memory was added, 175, is not large enough to provide significant information. Nevertheless we shall present them for the sake of completeness. In the case of the 11/40 our curves were based in 239 measurements without the cache memory and 303 measurements with the cache memory. As for the VAX, we used 934 measurements with one disk drive and 433 measurements with two disk drives.

We have chosen to present the effects of upgrading changes from the viewpoints of those load characterizations involving users, since they provide a more immediately understandable interpretation. From reading the graphs one can see the relative advantage (or disadvantage) of a change in terms of how many users (or active users) can now be accommodated at a given value of response time, our main performance index, compared to the number that could before the change was made.

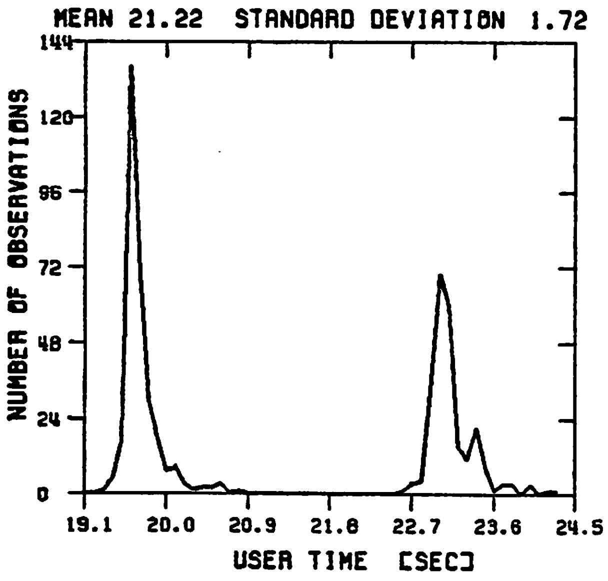
All graphs presented in this chapter are 90-percentile curves where the size of samples has been chosen, as before, so as to exclude global outliers from appearing in the graphs. In the cases of the 11/40 and the VAX, we used samples of size 60, but for the 11/70 we used samples of size thirty. We had to use smaller size samples for the 11/70 because with 175 sample points one can get at most three-point curves with samples of size sixty. Given the large range of the variable users in the 11/70 three-point curves are not acceptable.

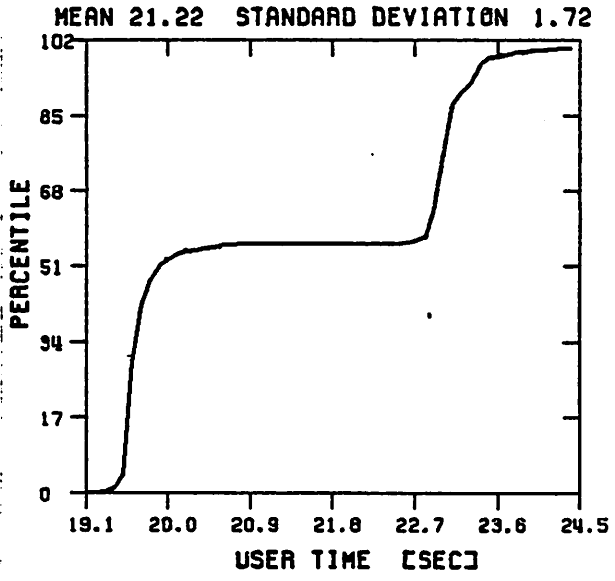
5.1. ADDING A CACHE MEMORY TO THE 11/40.

Perhaps the most spectacular use of a histogram of all measurements taken on a given system for a given task is that of witness to a basic configuration change. The phenomenon of **multiple peaks** in performance indices such as user and system time is almost certainly associated with a basic configuration change. This can be seen in all the histograms included in **Appendix B** corresponding to tasks measured in the 11/40. In effect the addition of the cache memory was responsible for *shifting* the peaks to the left in each of the three basic tasks.

Most remarkable is the case of the cpu bound job where the peak in user time went down from 23.0 seconds to 19.6 seconds, mean user time decreased from 23.13 seconds to 19.71 seconds and the standard deviation of the user time

11/40: Cpu-bound; user time.

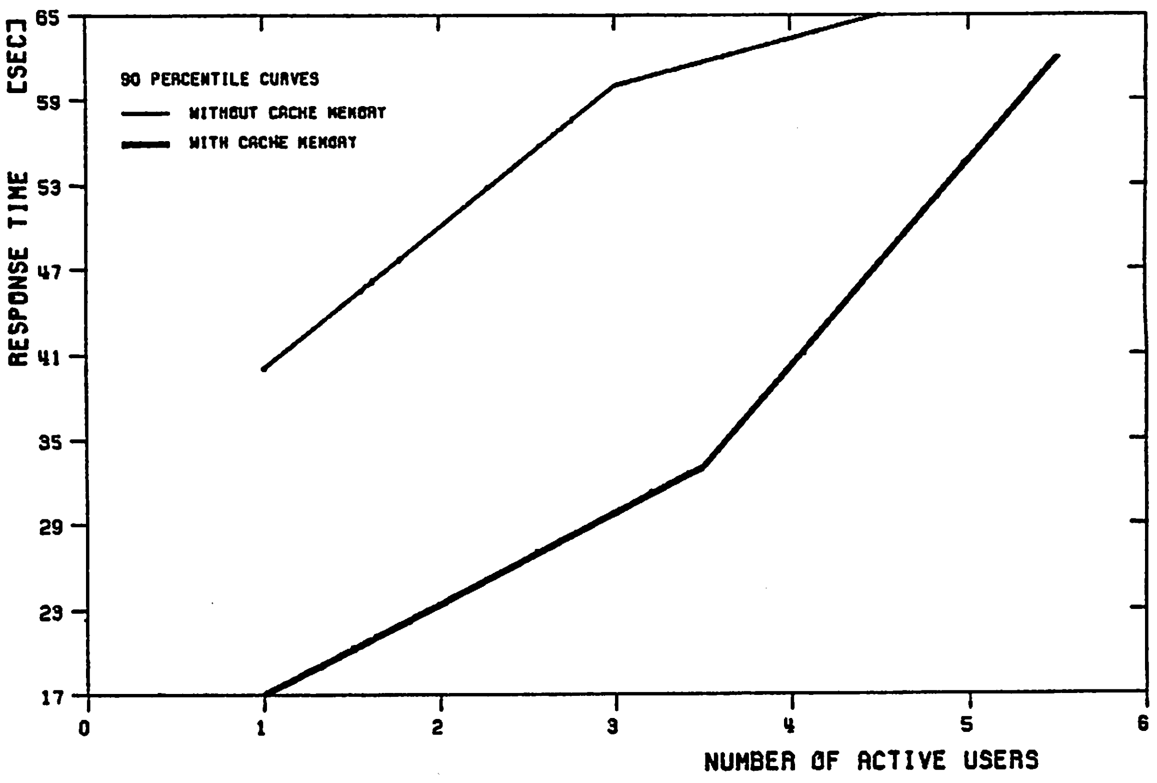


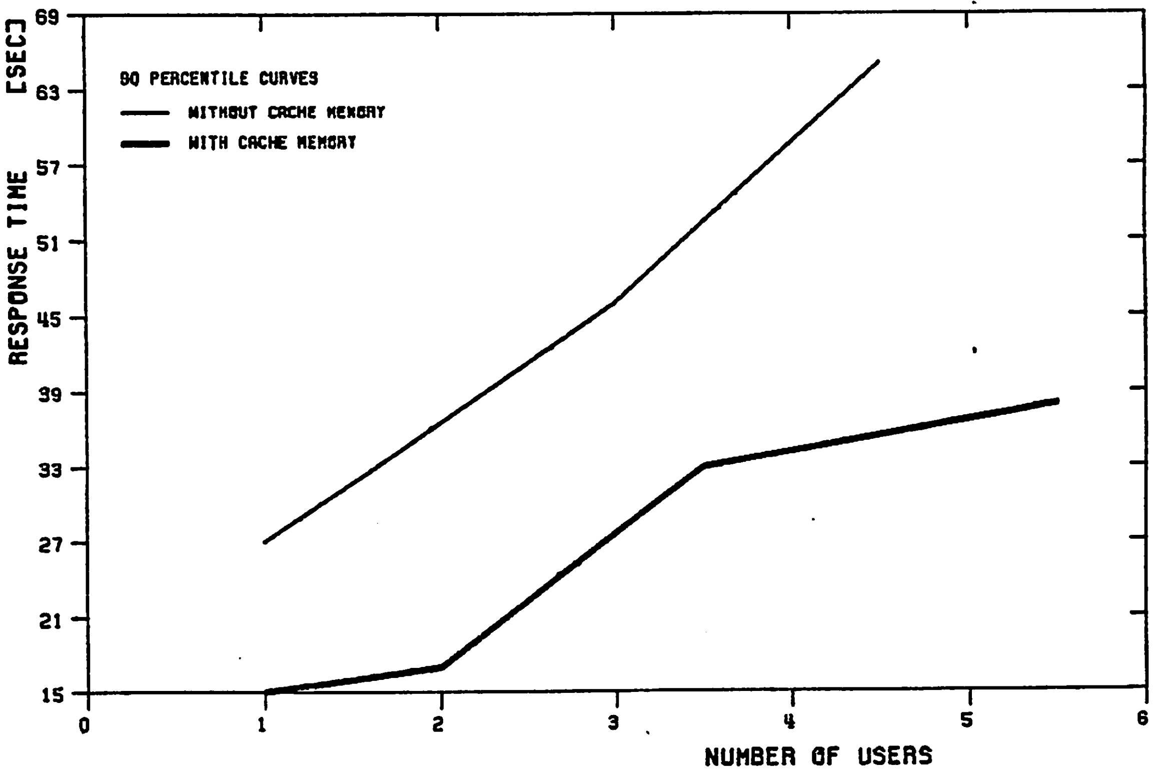


HISTOGRAM

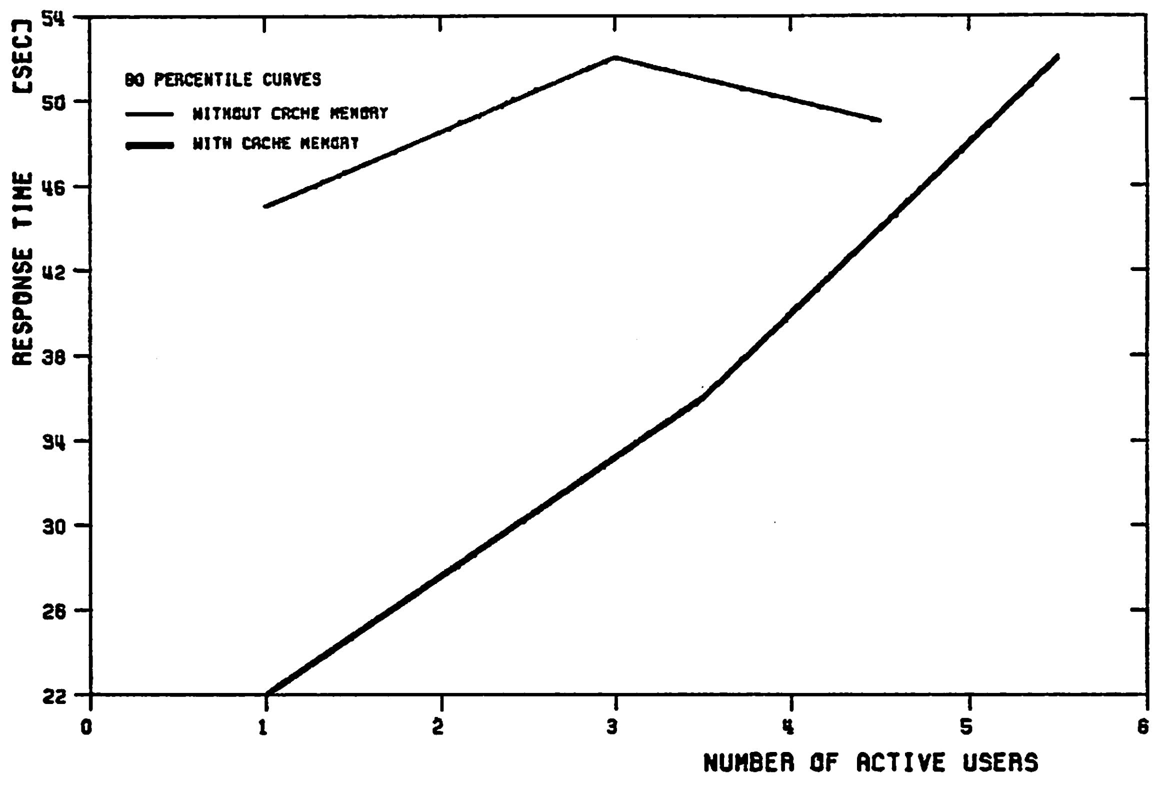
CUMULATIVE DISTRIBUTION

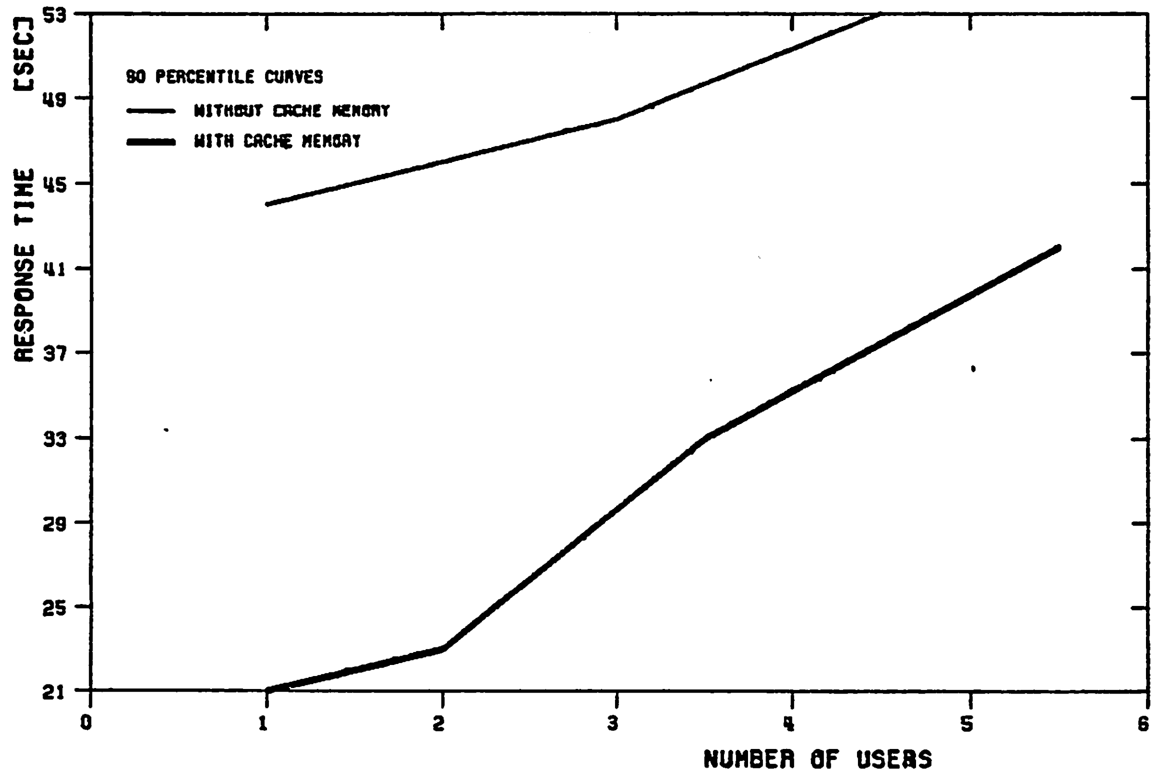
11/40: C compilation



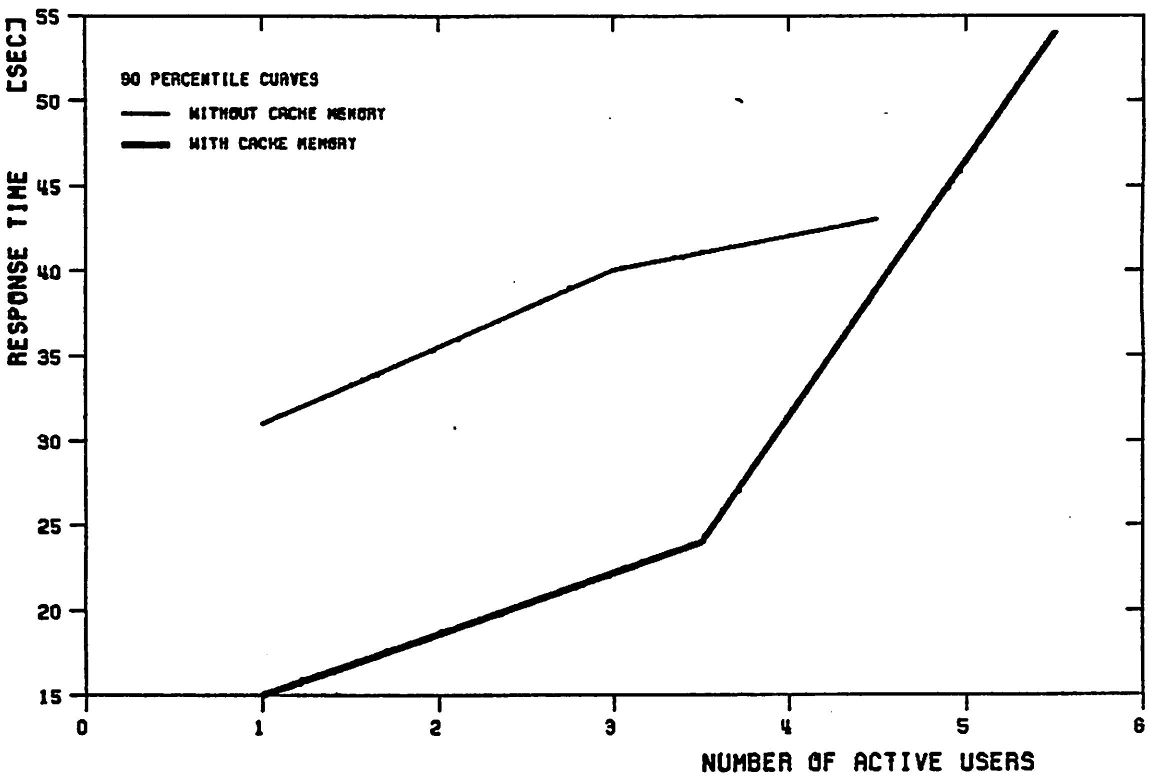


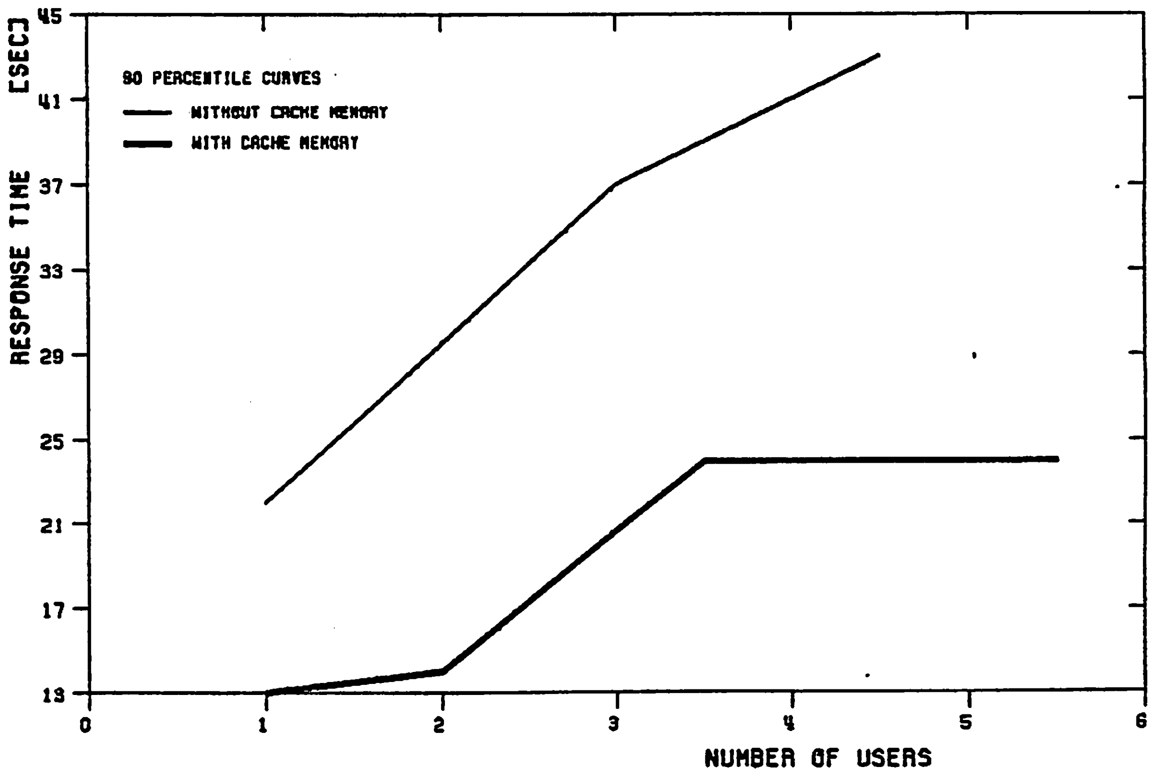
11/40: CPU-bound job



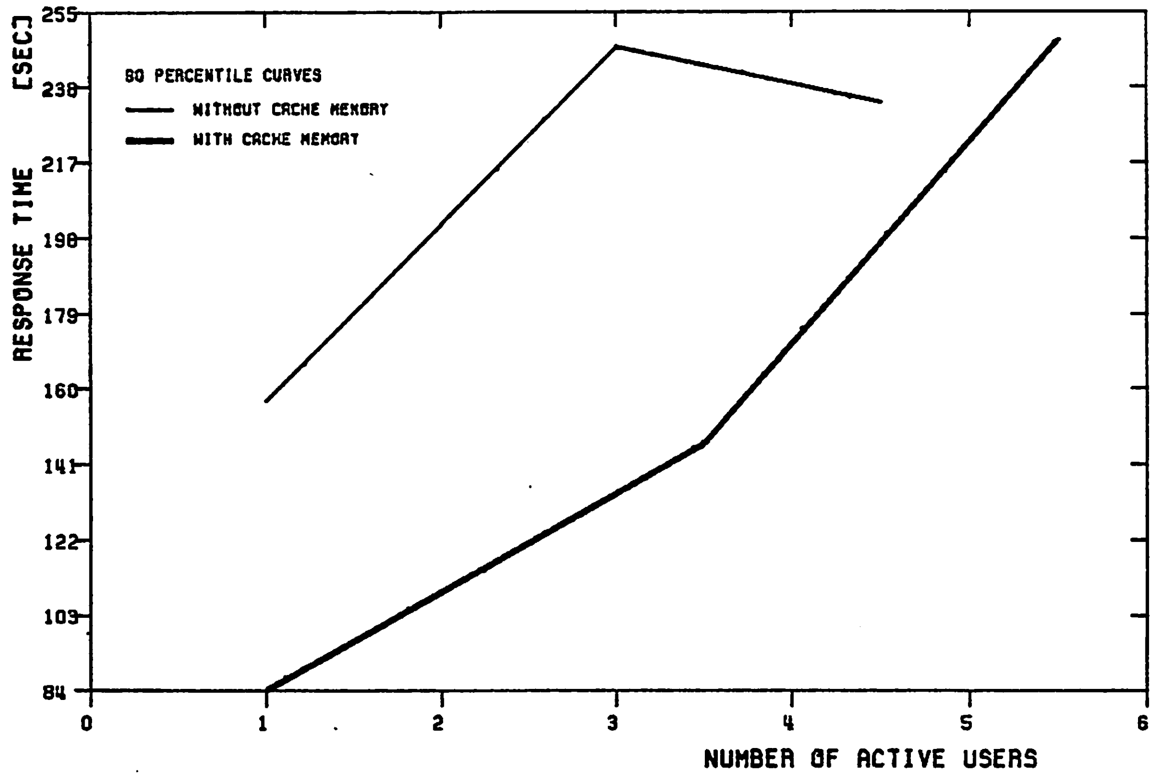


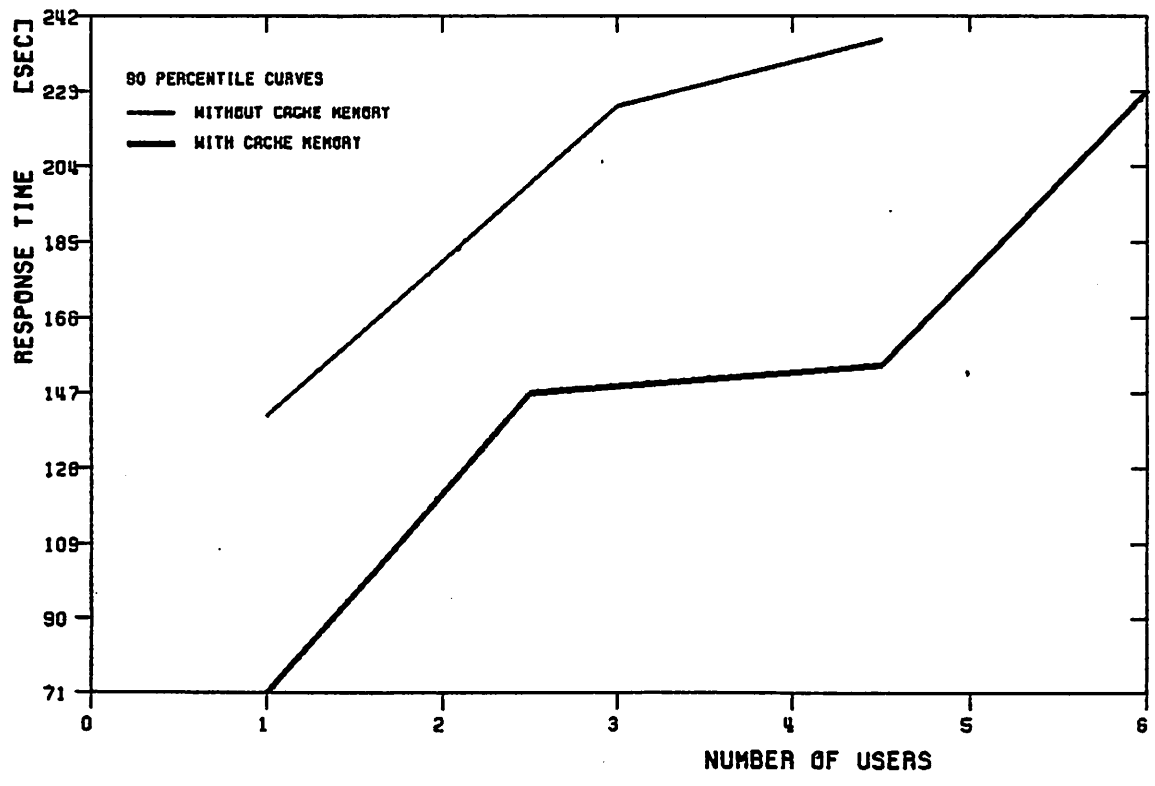
11/40: "man man"


RESPONSE TIME [SEC]
55
50
45
40
35
30
25
20
15
0
1
2
3
4
5
6
NUMBER OF ACTIVE USERS
90 PERCENTILE CURVES
WITHOUT CACHE MEMORY
WITH CACHE MEMORY


RESPONSE TIME [SEC]
45
41
37
33
29
25
21
17
13
0
1
2
3
4
5
6
NUMBER OF USERS
90 PERCENTILE CURVES
WITHOUT CACHE MEMORY
WITH CACHE MEMORY

11/40: Script


RESPONSE TIME [SEC]
255
238
217
198
179
160
141
122
103
84
80 PERCENTILE CURVES
WITHOUT CACHE MEMORY
WITH CACHE MEMORY
0
1
2
3
4
5
6
NUMBER OF ACTIVE USERS


RESPONSE TIME [SEC]
242
223
204
185
166
147
128
109
90
71
80 PERCENTILE CURVES
WITHOUT CACHE MEMORY
WITH CACHE MEMORY
0
1
2
3
4
5
6
NUMBER OF USERS

sample decreased from 0.25 to 0.20. System time also improved dramatically for this task, mean system decreased from 0.68 seconds to 0.33 seconds and the standard deviation of this sample decreased from 0.88 to 0.36.

Because of the amount of measurements we had, plotting with samples of size 60 made us have three and four point curves for each task in our comparison graphs. Given that the range of the free variable in these graphs is seven we do not think this is a drawback. Moreover in each graph the differences appreciated are so big that one can with no doubt assess the positive efect of the change. Most memarkable is the better responsiveness obtained at low levels of load, which, given the small amount of main memory that this system has, is the best indicator of how much improvement one could hope to obtain if we had an appropriate amount of main memory.

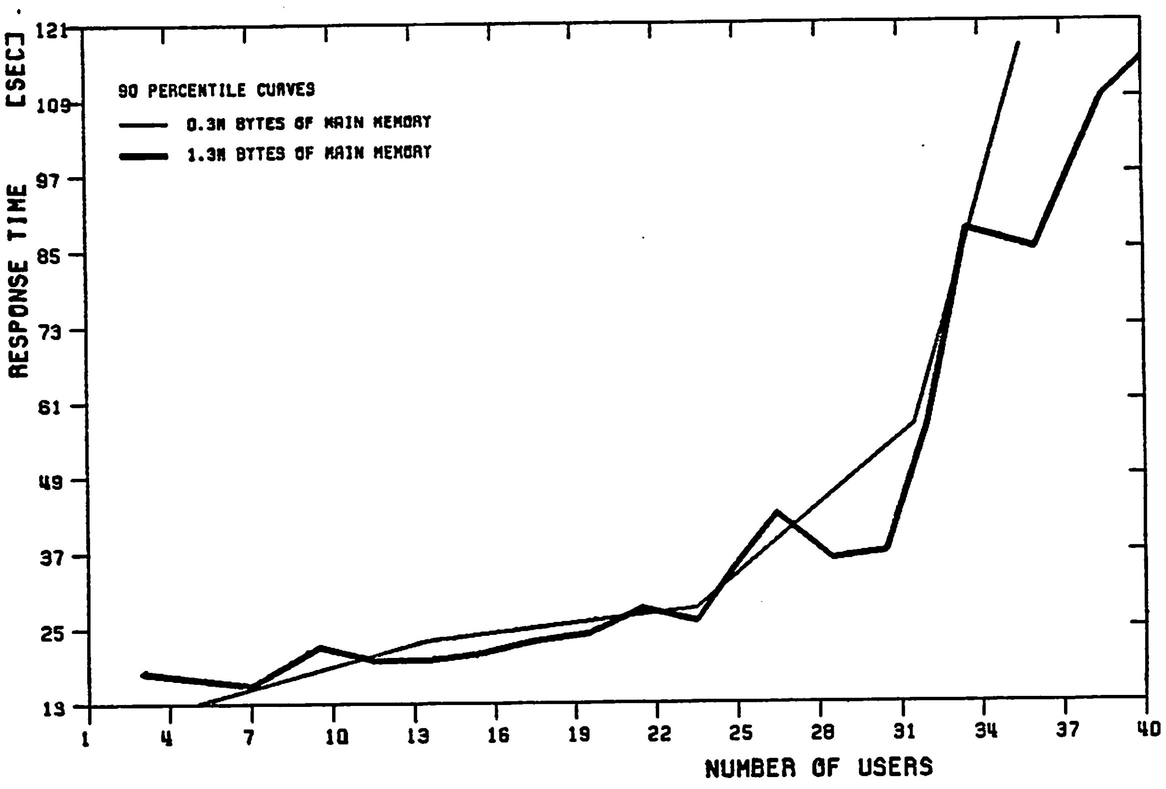
If in the previous chapter we had presented only those measurements obtained with the cache memory the curves for the 11/40 would have looked a lot smoother and the system better.

It can certainly be stated that both responsiveness and throughput are enhanced when a cache memory is added. It is then only a matter of cost-benefit analysis to determine its size and type.

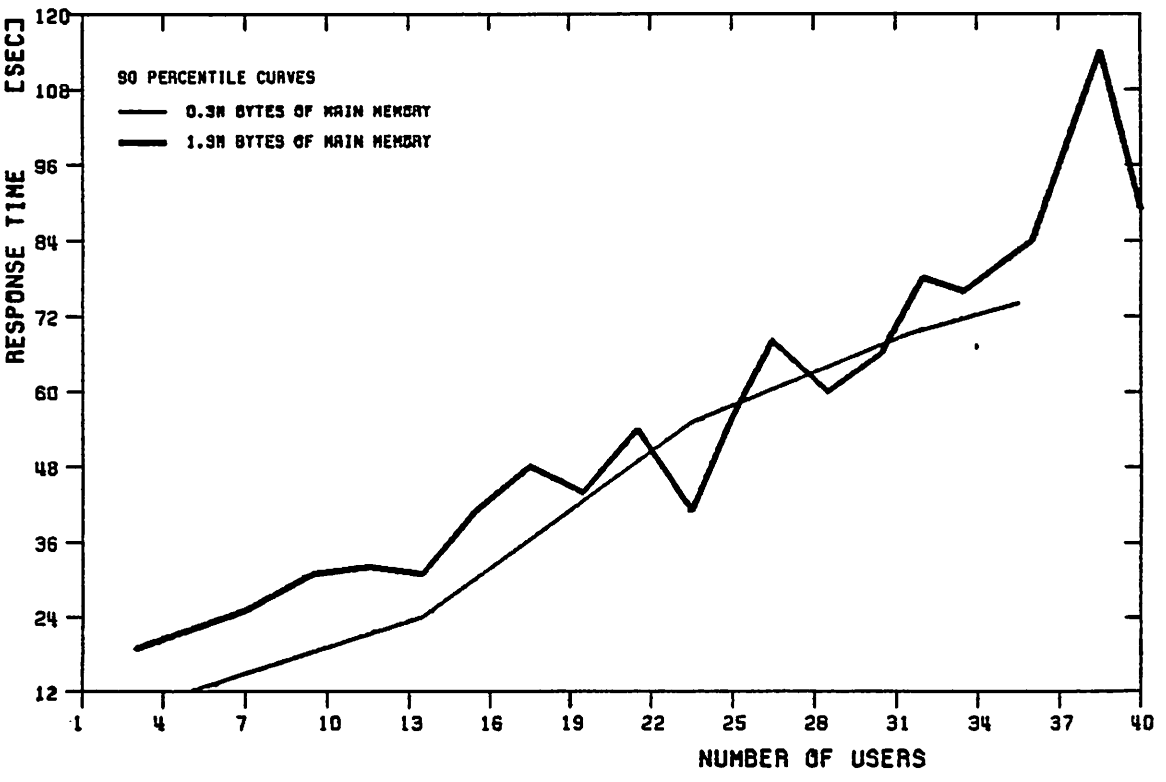
5.2. ADDING MAIN MEMORY TO THE 11/70

It is very unfortunate that we did not have enough samples before this change was made. The four graphs presented in this chapter with respect to this upgrading (or expansion) change almost succeed in proving that the system took longer in each task with the addition of more memory. We do not believe this to be so and here one must be warned about the usage of 90-percentile curves with an insufficient amount of data: they may present a biased behavior.

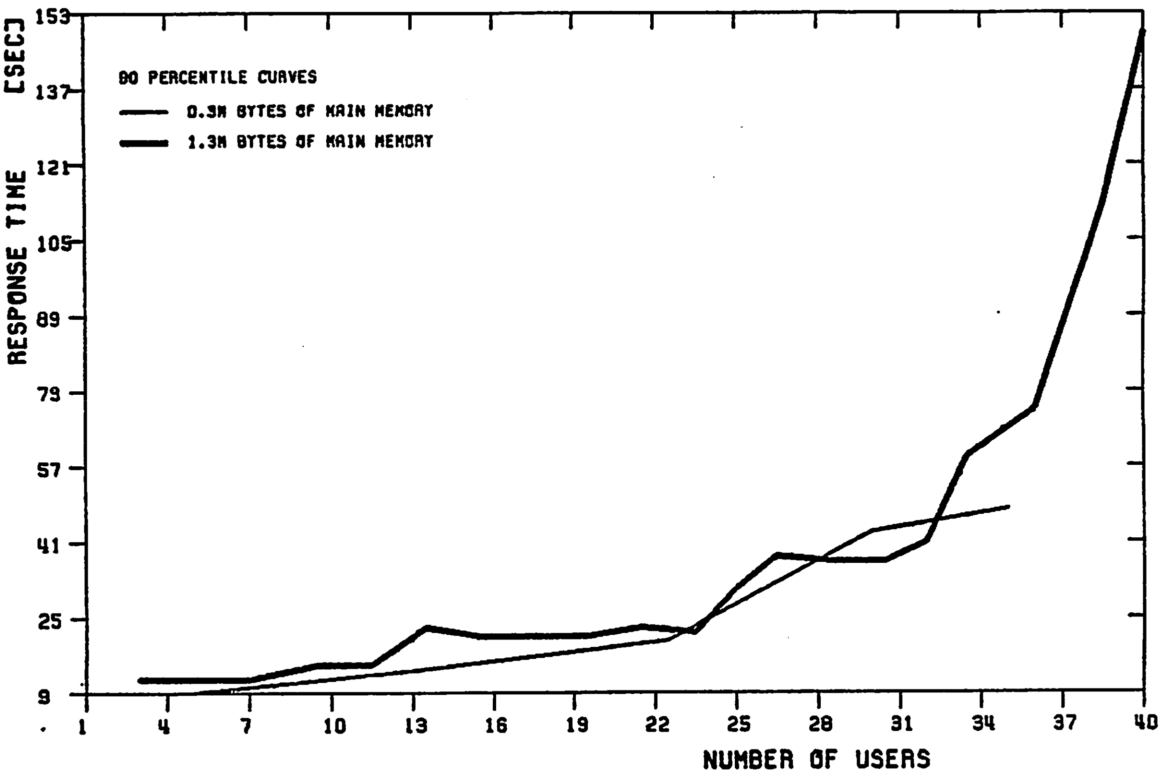
11/70: C compilation



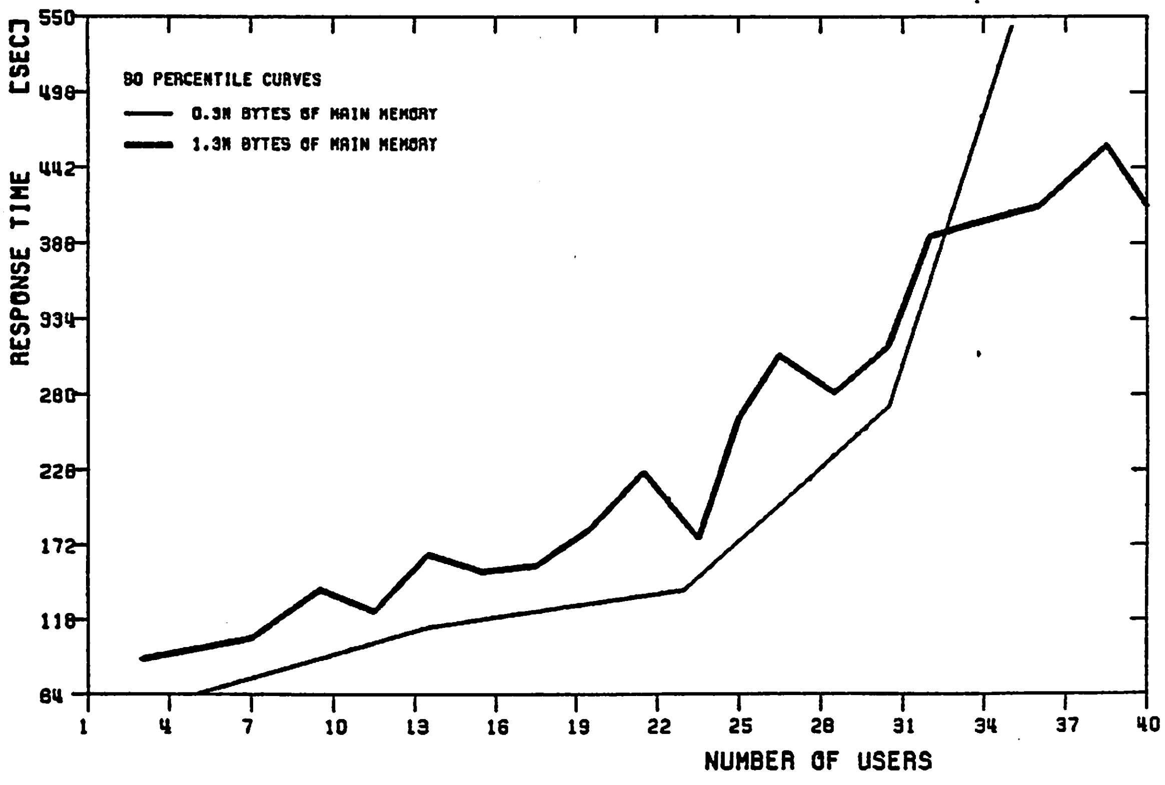
11/70: CPU-bound job



11/70: "man man"



11/70: Script



We had far too few measurements before the memory was added (175) and a fair amount after the change (830). Thus the curve for the case with 1.3M bytes of main memory is realistic as to what to expect from the system, while most points of the curve with 0.3M bytes of main memory were chosen from small samples and they might just be too optimistic. Moreover, the fact that all the points of the 0.3M byte curve were obtained from measurements made at the beginning of the academic quarter makes them likely to be the result of a not too loaded system. It is known that students push the system much harder towards the end of the quarter when their sofistication has improved.

Nevertheless we can appreciate some of the expected effects of such a change specially in the C compilation, where the curve with more memory shows a smaller slope at low levels of load and it takes longer for it to saturate. Another effect that can be appreciated is when we consider the task Script, where we can observe that the overall rate of growth of this task is smaller when the system has more memory. The 0.3M byte configuration shows saturation at a lower level of load than what the 1.3M byte configuration does. Given the balanced nature of this task, as well as the overall exercise of the system it produces, makes us believe in the positive effect that this change had in the 11/70. Given our insufficient data, what we can not assert is how much the system improved.

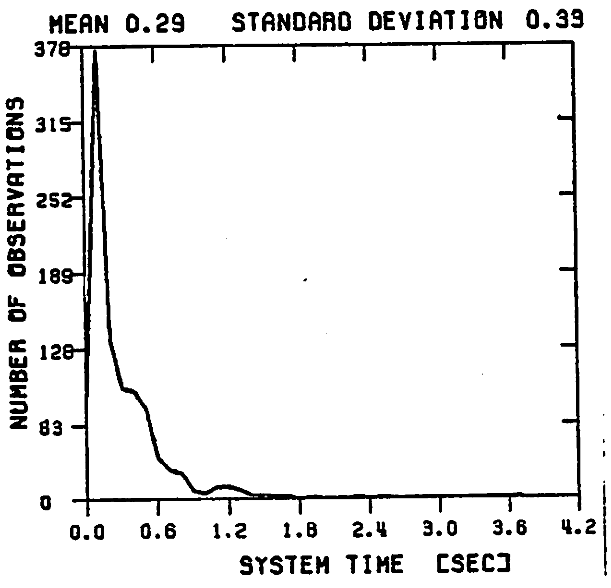
There is another aspect to this change worth mentioning; it certainly increases the amount of multiprogramming the system handles before saturation and that has a very positive effect in trivial tasks (say a *who* or a *date* command). That type of effect is not measured by our script and so we can not report on it directly, although it does play a role in the response time of our task Script.

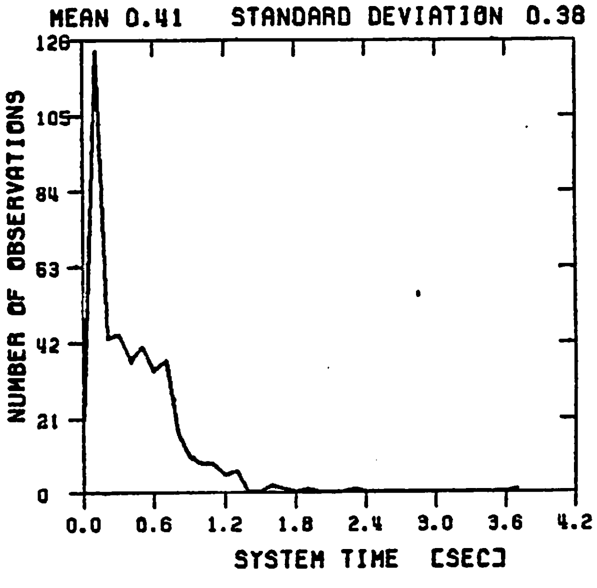
5.3. ADDING A SECOND DISK DRIVE TO THE VAX

The addition of a CDC 9762 disk drive with a CDC 9400 controller and 80 M-bytes disk packs to the VAX configuration was a mixed blessing. It proved to yield an overall improvement in the responsiveness of the system, as can be observed from the graphs with Script data, but for one type of task it proved to be disadvantageous.

Cpu-bound jobs took longer to finish with the two disk configuration. When analyzing the histograms of user time and system time for this task for the one-disk configuration and the two-disk configuration, we see that mean user time remained essentially unaltered; 5.66 seconds with one disk and 5.69 seconds with two disks. Moreover the standard deviation (0.21) of the user time sample did not change.

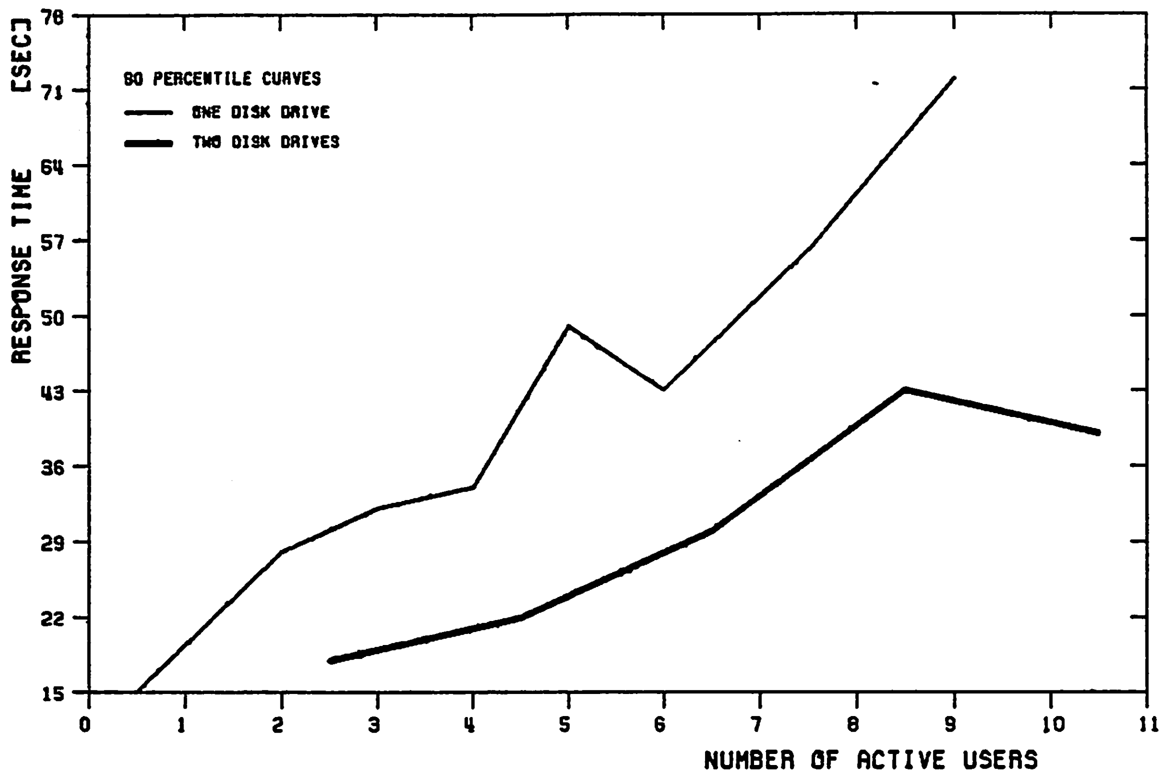
This is not the case for system time. Mean system time increased from 0.29 seconds in the one-disk configuration to 0.41 seconds in the two-disk

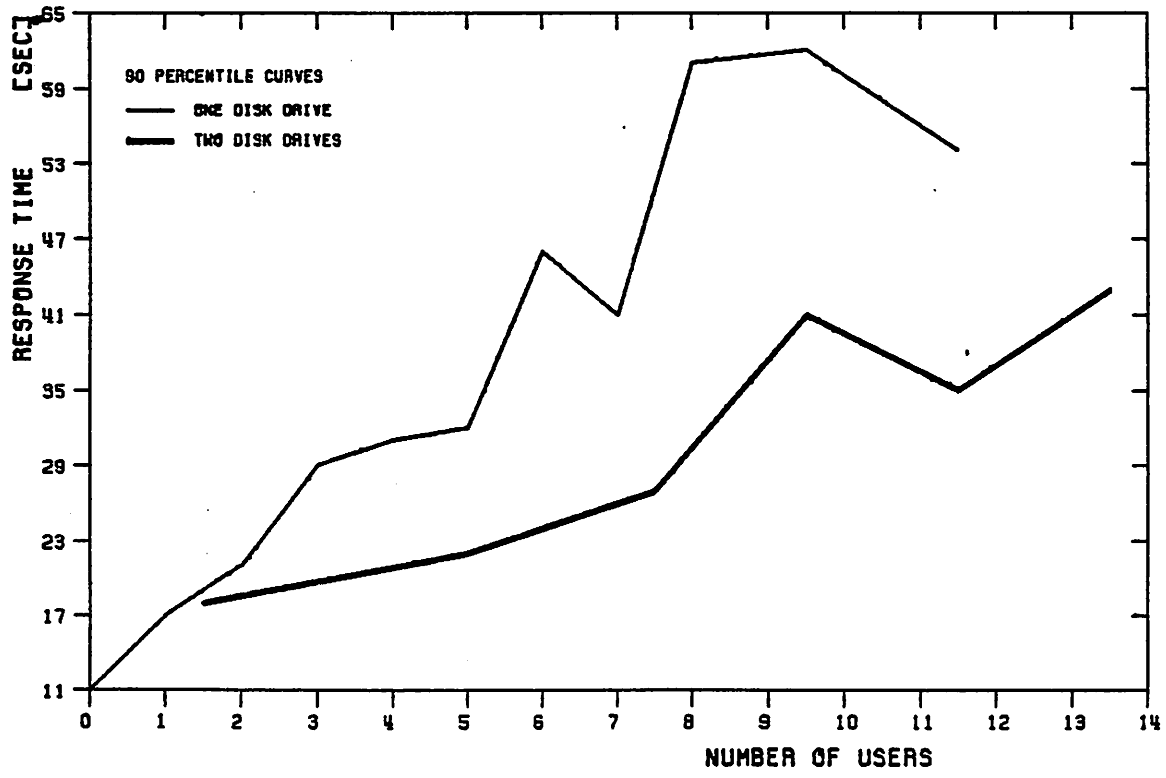




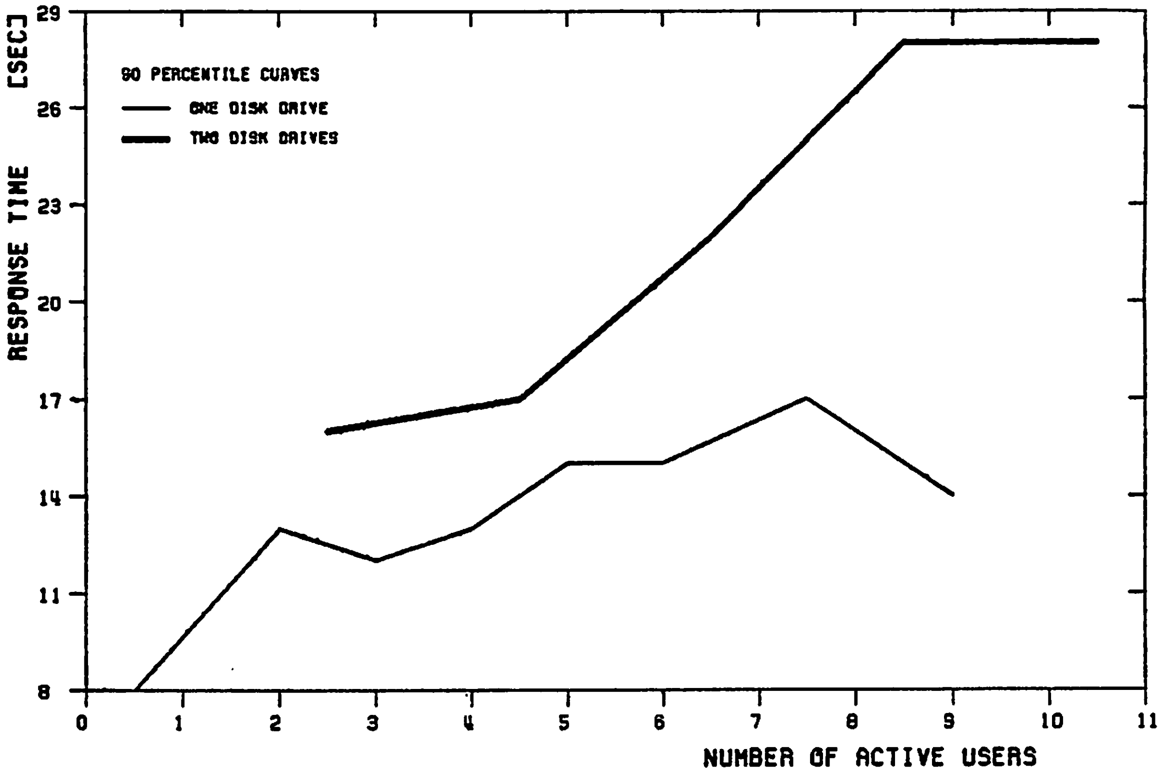
HISTOGRAMS: one-disk **two-disk configuration**

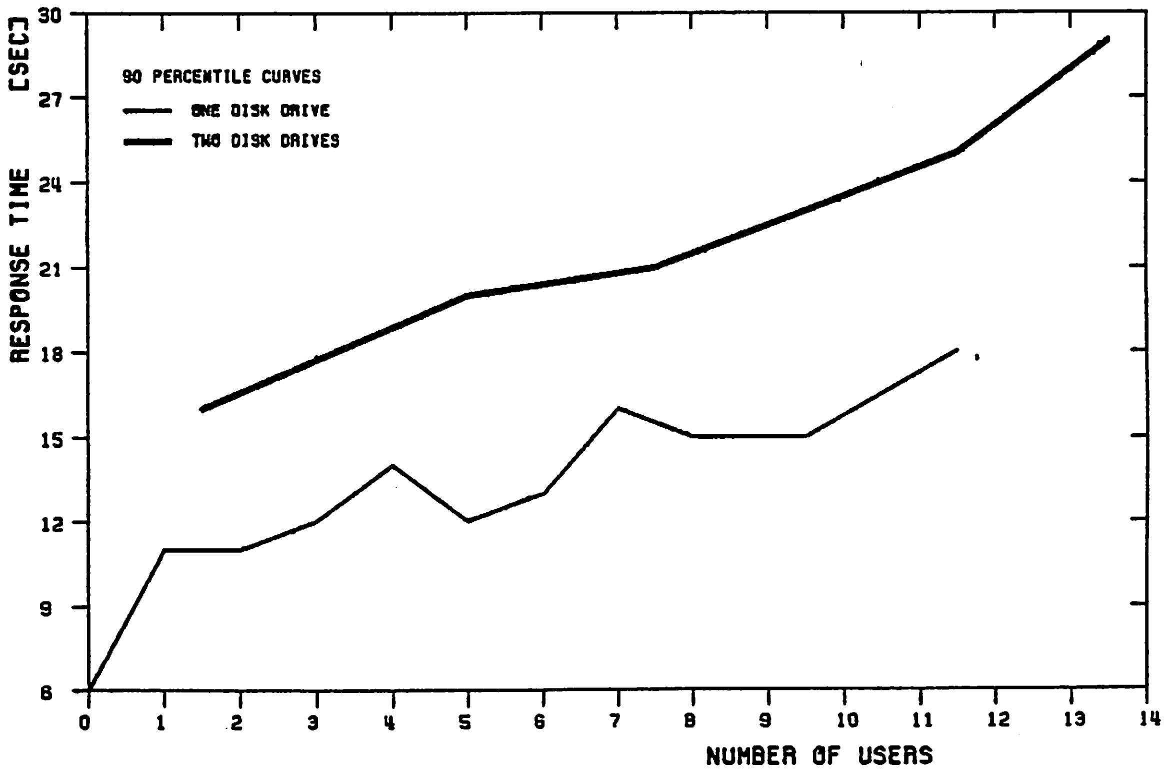
VAX: C compilation



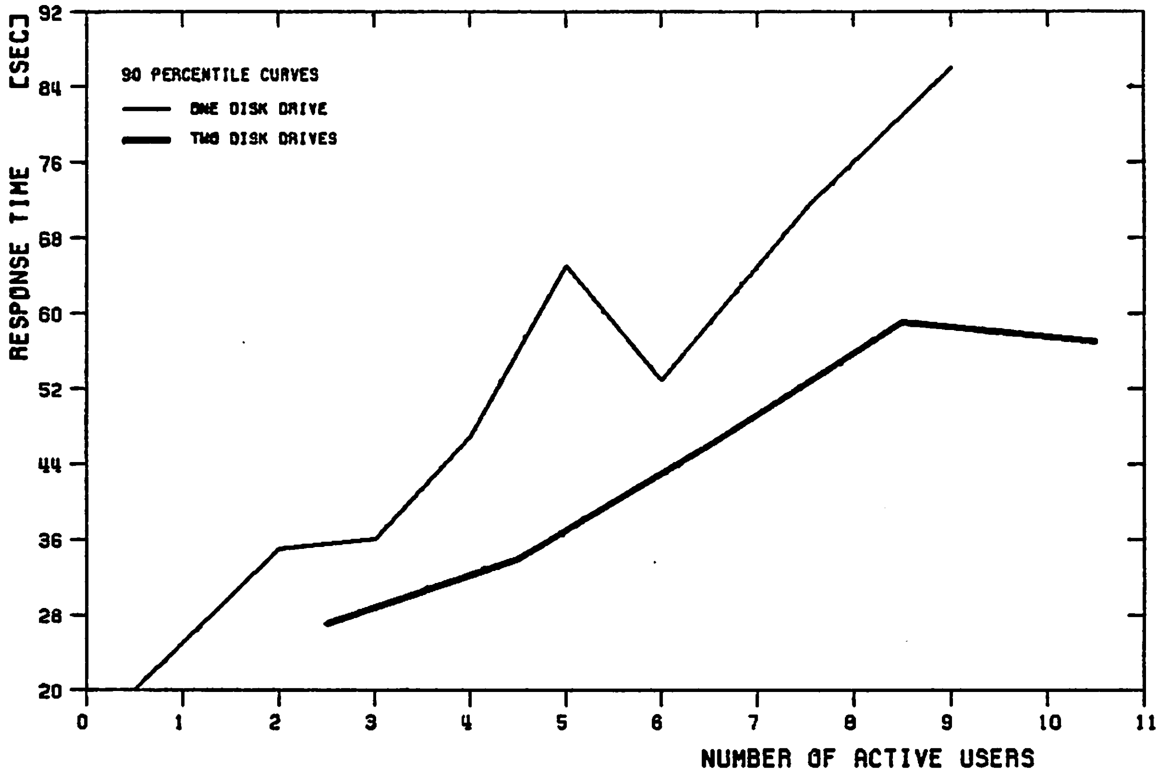


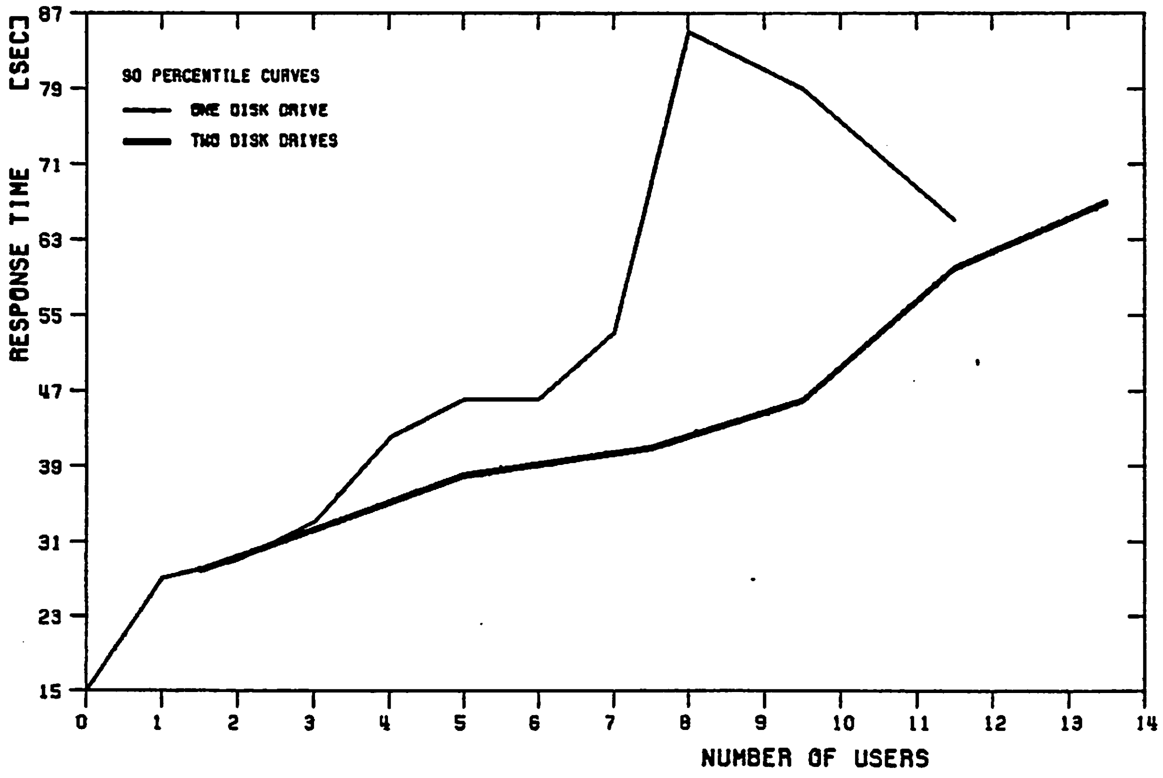
VAX: CPU-bound job



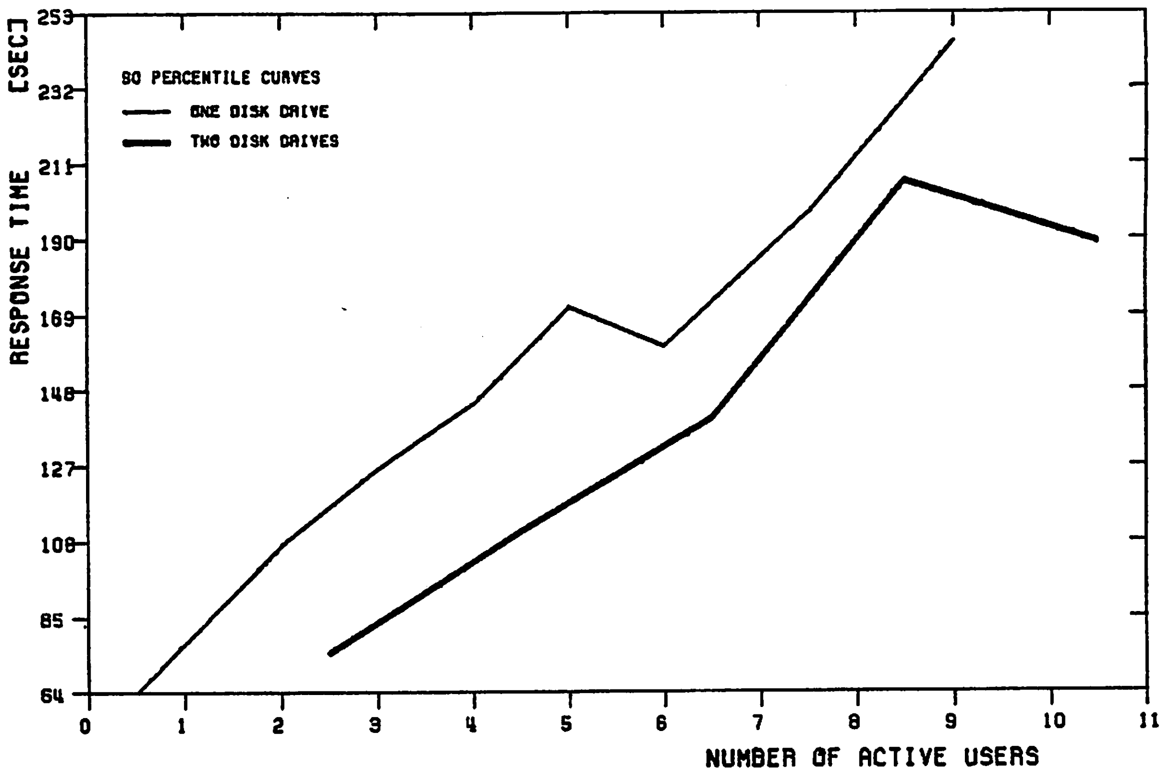


VAX: "man man"

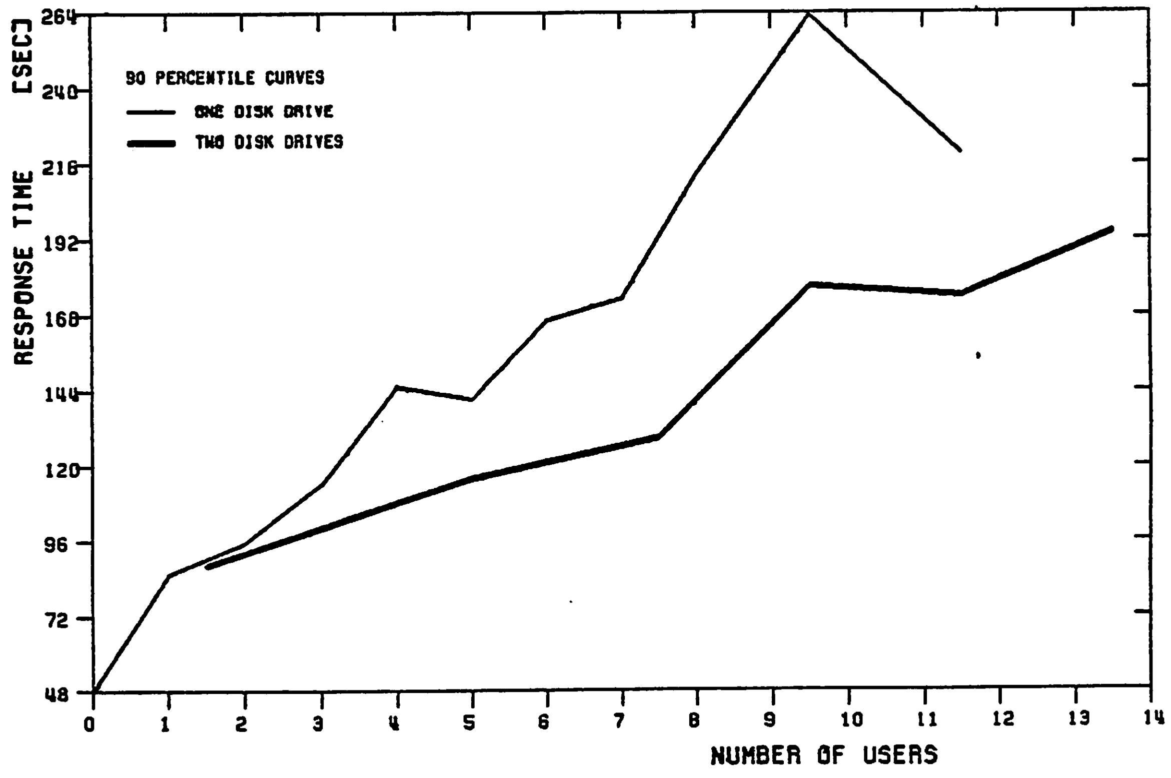

RESPONSE TIME [SEC]
90 PERCENTILE CURVES
ONE DISK DRIVE
TWO DISK DRIVES
NUMBER OF ACTIVE USERS
92
84
76
68
60
52
44
36
28
20
0
1
2
3
4
5
6
7
8
9
10
11


RESPONSE TIME [SEC]
90 PERCENTILE CURVES
ONE DISK DRIVE
TWO DISK DRIVES
NUMBER OF USERS
87
79
71
63
55
47
39
31
23
15
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

VAX: Script



RESPONSE TIME [SEC]
253
232
211
190
169
148
127
106
85
64
0
1
2
3
4
5
6
7
8
9
10
11
NUMBER OF ACTIVE USERS
90 PERCENTILE CURVES
ONE DISK DRIVE
TWO DISK DRIVES



RESPONSE TIME [SEC]
264
240
216
192
168
144
120
96
72
48
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
NUMBER OF USERS
90 PERCENTILE CURVES
ONE DISK DRIVE
TWO DISK DRIVES

configuration. The standard deviation of the sample increased from 0.33 seconds in the one-disk configuration to 0.38 seconds in the two-disk configuration. We believe this increase in system time to be directly related to the cause of the longer response time observed, because it probably is an indication that now jobs get interrupted more often due to completed IO requests and thus sent to *wait* queues more times than before. This of course degrades their response time.

On the other hand, as was expected, tasks with IO activity which can be overlapped with CPU activity, like C compilations, did run faster with the two disk configuration. This was achieved by reconfiguring the file system in such a way that temporary files, like those created by the compilers, the editor, and several other utility programs, were placed on a different disk than user files.

It can also be observed that the rate of growth of response time has decreased in each of the three basic tasks as well as in Script. This makes more plausible the possibility of augmenting the number of users in the system with less risk of reaching the point of saturation.

Another aspect to consider in this type of upgrading change is the effect of the change in the storage capacity of the system. It is undoubtly advantageous for most applications to have large disk storage capacity. From this point of view the addition of 80 M-bytes of disk storage capacity is a major leap forward.

6. CONCLUSION

The increasing amount of different hardwares on which the same timesharing operating system, UNIX, runs calls for a performance evaluation study oriented towards assessing the influence of the equipment on a system's global performance. In this report we have done that for three very different systems. Our aim has been to observe the effect that evolving hardware has on performance.

Lack of portable tools that would enable us to load systems in a controlled way made us decide for the "terminal probe method". We thus designed and run in the background on each system a script that periodically "probed" it. In this way we measured the response the systems had with their *natural* work load to our tasks. We are aware of the theoretical difficulties of this method, with respect to conclusions to be drawn from such a study, but from the results obtained we feel that a useful comparison can be made this way. We also believe the method not to be too expensive in terms of resources used. Roughly speaking it uses 3% of the cpu cycles. This amount does depend on the system.

The four characterizations of load used, "Users", "Active Users", "Processes" and "Real Processes" turned out to represent fairly well the load each system had. The substantial consistency in the results obtained made us believe in the final curves obtained from our measurements. Best of all we think that a great advantage of this method is its total portability. New systems can now be easily added to our study. We do believe, though, that one has to observe more than one of these characterizations of load at the time to draw solid conclusions from such a study. The instability of the *natural* work load and the rather rough characterizations of load we use require this.

For purposes of comparison, we feel that our notion of Real Process is the best. Its drawbacks (say of not distinguishing between different types of processes in the process table) are amply compensated by the overlap between systems it provides and by reflecting much better the amount of work demanded from the systems.

Our approach to determine the minimum size a sample must have for it to be representative, and thus plottable, seems to go a long way in solving the conflicting problems of insufficient number of data-points, presence of outliers in the samples, large variance of some samples and finite amount of resources. As discussed in Chapter 3, we determined the minimum size of a sample by the iterative method of plotting the curves with different size samples until we observed that the global outliers (for a given task) were not plotted. This method can be taken as a criteria to stop gathering data on a system when running our experiment. As soon as the curves are smooth for all values of the variable it means that we have enough points. In our case we had to cluster samples, when necessary, to obtain the size (60) we wanted. The clustering normally took place for the high values of the variables.

No one system presented itself as a clear best although it is obvious that a configuration involving a PDP 11/40 lags behind the other two. It is also clear from our study that it is vital for the performance of a system to have a *balanced* configuration. Moreover certain additions, such as a cache memory, prove to be almost indispensable.

Having observed that the change from PDP 11/40's to PDP 11/70's was very beneficial (faster cpu, better responsiveness, larger address space), the question was whether the same conclusion could be reached for the change from the PDP 11/70 to the VAX 11/780. The main obstacle against drawing strong conclusions from our study was the marked difference in each system's configuration. Unfortunately the VAX 11/780 did not have as much main memory as the PDP 11/70 and it only had one disk. Moreover, the size of UNIX had grown in the VAX implementation, thus leaving proportionally less core space for user processes.

Nevertheless, we could observe that the VAX does indeed have a substantially faster cpu. Responsiveness for cpu-bound jobs is clearly better. Better yet, when a second disk was added to the VAX configuration, it performed tasks such as C compilations faster and more stably than before, ranking now very close to the 11/70. The performance that such tasks now achieved make us believe that each system is at least as good as the other. At this point it has to be emphasized that the C compiler for the VAX is a fairly portable one, while the one that the PDP 11/70 has was essentially custom made. This is of course an advantage for the VAX.

Last but not least, the VAX 11/780 uses 32-bit words. Now this is a great leap forward. Only now the system has the potential address space to be converted into a virtual memory system. In fact, a paged virtual memory version of UNIX developed at the University of California, Berkeley, is soon to be released. Depending on the applications, this fact by itself might be worth the change of machines.

7. REFERENCES

[1] Ritchie, D.M. and Thompson, K., "The UNIX Time-Sharing System" *Communications of the ACM* 17(7) (July 1974), pp. 365-375.

[2] Joy, William, "An Introduction to the C Shell" *Computer Science Division, Department of EECS. University of California, Berkeley.* December 1978.

[3] Ferrari, Domenico, "Computer Systems Performance Evaluation", Prentice-Hall, 1978.

[4] Ferrari, Domenico, "Characterizing a Workload for the Comparison of Interactive Services", *Proceedings 1979 NCC*, pp. 789-796.

[5] Bard, Y., "Performance Analysis of Virtual Memory Time-sharing Systems" *IBM Systems Journal* No 4, 1975, 366-384.

[6] UNIX Programmer's Manual *Bell Telephone Laboratories Incorporated, Holmdel, New Jersey.* 7th Edition, December 1978.

[7] Kernigan, Brian and Ritchie, Dennis, "The C Programming Language" Prentice-Hall Software Series, 1977.

[8] Karush, A.D., "The Benchmarking Method Applied to Time-sharing Systems". *Rept. SP-3347, System's Development Corporation.* Santa Monica, CA. August 1969.

APPENDIX A:
Documentation for the Script

APPENDIX A: Documentation for the Script

USE OF THE SCRIPT

In our monitoring experiment we used a shell script to obtain periodic measurements in each system. It was our only tool for collecting data. In this **Appendix** we present it together with all the files and programs it requires. The script has been written in standard C shell syntax and so it should be portable to any installation which has the C shell installed within their UNIX system. The way to use it is simply to run it as a process in the background. We have included extensive comments in the text of the script so that each command executed is explained.

The files required to exist when one initiates the script are four: CC, CPU, MAN and test.c . They should be in the same directory as the script. CC, CPU and MAN contain commands for tasks to be executed by the system.

CC requests and times the C compilation of test.c. Test.c is a (trivial) CPU-bound C program. CPU runs and times the result of the compilation done by CC (which is then a CPU-bound task) and finally MAN requests and times the command *man man*. As we have discussed in Chapter 2, this command is not a truly IO-bound job. Its execution might require a fair amount of *user time* (up to 6 seconds).

To convert this script to a standard shell script, all one needs to do is to replace in the text occurences of the files CC, CPU and IO by commands which directly execute and time the events. For example, instead of having *csh CC* we would have *time cc test.c*.

Sep 27 Ø6:ØØ 1979 files_needed Page 1, Line 1

File CC:

```
# Will run and time a Ccompilation.

set time=Ø
cc test.c
```

End of file CC.

File CPU:

```
# Will run and time a CPU-bound job, assumed to be in a.out.

set time=Ø
a.out
```

End of file CPU.

File MAN:

```
# Will run and time the command "man man" .

set time=Ø
man man > /dev/null
```

End of file MAN.

```
#

loop:

# We begin by  requesting the date.  This will be used as a TIMESTAMP for
# indivial measurements  of each task  and to  determine the elapsed time
# of one task,  "Script".
date > YYY1

# The following  seven files will be  updated  each time the script runs.
# They contain the raw data.

# Ccompilation is the file that will contain the  measurements correspon-
# ding to the C compilation.  Here we timestamp the  comming measurement.
cat YYY1 >> Ccompilation

# CPUbound contains the ones  corresponding to the CPU-bound job. Here we
# timestamp the comming measurement.
cat YYY1 >> CPUbound

# manman the ones  corresponding to the command manman. Here we timestamp
# the comming measurement.
cat YYY1 >> manman

# Script  the ones  corresponding to a mix that will be clarified latter.
# Here we timestamp the comming measurement.
cat YYY1 >> SCRIPT

# This file grows  extremely quickly.  It will contain the output of the
# command 'ps -alx'.  Exists only to keep all information gathered.
cat YYY1 >> PS

# In this one we will keep information about "Users" and "Active Users".
cat YYY1 >> USERS

# In this one we will keep information about "Processes".
cat YYY1 >> PROCESSES

# This is the way we obtain most of the information from the system.  It
# is from the output of this command that we find the number of  "Active
# Users" and the number of processes.
ps -alx > YYY2

# This appends the number of users logged-in to the file USERS.
who | wc -l >> USERS

# Executes the file CC  and appends the output to the file Ccompilation.
csh CC >>& Ccompilation

# Executes the file CPU and appends the output to the file CPUbound.
csh CPU >>& CPUbound

# Executes the file MAN and appends the output to the file manman.
csh MAN >>& manman

# Here we finish  the task  "Script".  We use a  second date command to
# determine the elapsed time.
date >> SCRIPT


# Here we append to the file PS the contents of YYY2 before using it to
# decode the information we want.
cat YYY2 >> PS

# We delete the first line of YYY2 (which brings headings).
ex - YYY2 << 'EOF'
1d
w YYY2
q
'EOF'
# We  count the number of lines  YYY2  has and  append it to PROCESSES.
# they correspond to the number of processes in the process table.
wc -l YYY2 >> PROCESSES


# Now we will find a  number that will enable us to  find the number of
```

Sep 26 10:46 1979 script Page 2, Line 77

```
# "Active  Users"  when we  process  the  data.  This  number  will in
# general  exceed  by at least one the  true number of "Active  Users".
# The  reason is that we count all those  enries under the  TTY  column
# that appear  more than once.  Unfortunately in this  column  we will
# always have the   ?  symbol associated  with  processes owned by the
# system and  may have  symbols for  different nets.  A study of these
# factors has to  be done in  each system  and then  used when reducing
# the data.  In the same way, each system has to determine the position
# of the column where "TTY" begins.  What is here presented corresponds
# to our VAX 11/780.
ex - YYY2 << 'EOF'
1,$s/.......................................................//
1,$s/ .*//
w YYY3
q
'EOF'

# The following command does what has been described above.
sort YYY3 | uniq -d | wc -l > YYY4


# We append the number obtained to the file USERS.  Thus two numbers
# get appended to this file every time the script cycles.
cat YYY4 >> USERS


# We remove all temporary files created by the script.
rm YYY* a.out

# We go to sleep for 20 minutes.
sleep 1200

# We go to the beginning of the cycle again.
goto loop
```

Sep 25 12:09 1979 test.c Page 1, Line 1

```
/*
 * This program attempts to measure CPU activity.  It performs 100,000
 * times the inner sequence of instructions.   All of them are integer
 * arithmetics operations.   Their relative frequency is choice of the
 * author.
 */

main()  {

    int i,j,k,l,m,n,o,J,K ;

    J  = 16833333 ;
    K  = 56983122 ;


    for ( j = 0 ; j < 40  ; j++ )  {
        for ( i = 0 ; i < 2500 ; i++ )  {
            l = J * K ;
            m = J + K ;
            n = J / K ;
            l = J * J * K ;
            m = m + n ;
            m = J * K ;
            n = J / K ;
            l = m * n ;
            o = K * K * K * K ;

        }
       /*  End of the inner loop.  */
    }
   /*  End of the outer for loop.  */
}
/*  End of main.   */
```

APPENDICES

In the interest of paper conservation, appendices B and C have been omitted from this report. The complete set of appendices is available from the author, or they may be found in a M. S. Project by the same title filed in the E. R. L. Reference Library.