THE INSTRUMENTATION OF MULTICS*

by

Jerome H. Saltzer

and

John W. Gintell

Summary: This paper reports an array of measuring tools devised to aid in the implementation of a prototype computer utility. These tools include special hardware clocks and data channels, general purpose programmed probing and recording tools, and specialized measurement facilities. Some particular measurements of interest in a system which combines demand paging with multiprogramming are described in detail. Where appropriate, insight into effectiveness (or lack thereof) of individual tools is provided.

This paper is to be presented at the Second ACM Symposium on Operating System Principles, Princeton, New Jersey, in October, 1969.

---

* Work reported herein was supported in part by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of defense, under Office of Naval Research Contract Number Nonr-4102(01). Reproduction in whole or in part is permitted for any purpose of the United States Government.

July 29, 1969

The Instrumentation of Multics[+]

Jerome H. Saltzer
Massachusetts Institute of Technology
Department of Electrical Engineering and Project MAC
Cambridge, Massachusetts

John W. Gintell
General Electric Company
Cambridge Information Systems Laboratory
Cambridge, Massachusetts

In the construction of a modern, complex computer operating system, sophisticated tools are needed to measure what is going on inside the system as it runs. The list of hardware and software tools and techniques used for the measurement of Multics is interesting both from the point of view of what has proved to be important to measure, and what has not. Multics is a project whose intent is to explore the implications of building a comprehensive computer utility. The specific goals of Multics are described in a series of papers written in 1965[1]; briefly, the objective is to create a computer operating system centered around the ability to share information in a controlled way, and permitting application to a wide variety of computational jobs. A spectrum of user services, including a hierarchical file organization, sharing of information in core memory, dynamic linking of subroutines and data, parallel processing and device-independent input/output facilities characterize the system, and contribute to a complexity that makes careful instrumentation mandatory.

[+] Work reported herein was supported (in part) by Project MAC, an M.I.T research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Nonr-4102(01). Reproduction of this report, in whole or in part, is permitted for any purpose of the United States Government.

Two implementation techniques used in Multics call for specialized measurements. The first of these is a multiprogrammed multiprocessor organization, chosen to facilitate continuous operation of the utility and for ease of system scaling. The second technique is exploitation of an ability to begin executing a program which is not completely loaded into primary memory: this technique, usually named demand paging, is intended to exploit and encourage a tendency of many programs to localize their references to primary memory in any given period of time. Since these two techniques when applied simultaneously require interacting multiparameter controlling algorithms, specialized measurements must be made to check on the resulting performance and to allow adjustment of the parameters.

Multics, as a research project, contains many new ideas and new combinations of old ones. As a result, in its design there have been a dismayingly large number of choices to be made: strategies, algorithms, parameter settings, and emphasis on importance of design and speed of individual modules. Since the presumption was made at the start that some wrong choices are inevitable, there has been an emphasis on integrated instrumentation from the earliest design of the system. The result has been an ability to move in rapidly to unclog bottlenecks. In particular, two effects have been observed:

- Frequently, the best guesses by system programmers as to the cause of some performance problem have been proven wrong by the detailed measurements. Each such surprise while possibly affecting the programmer's ego, has saved work redesigning or streamlining

a module which was not causing the trouble. (Of course, the record has not been perfect: some unimportant modules have been redesigned in spite of (or for lack of) instrumentation results.)

Many otherwise undetected performance problems have been discovered in exploring instrumentation output. Probably because of normal variations of response in interactive systems, a flaw which degrades average response time by 20% may not be recognized immediately as such by console users. It frequently takes a healthy factor of two before the user realizes something is wrong.

The measuring techniques described here have been directed primarily toward understanding what goes on inside the operating system, rather than on measuring "throughput", system capacity, or the characteristics of system load. This direction is partly because of the research nature of the Multics project and partly because when an operating system provides a large variety of user services, its capacity depends very sharply on what exactly the users choose to do, making any single measure of capacity highly suspect.

Many of the measurement techniques used on Multics are not new. They are mentioned anyway, because it is the array of techniques used together which has been valuable, and also the relative importance of various techniques, old and new, is probably different in the Multics environment than elsewhere.

One should not presume that all the measurement techniques described here were thought out in advance, though many were. Much of the experience in measuring Multics has been to discover what additional measuring facilities were needed; this process, of course, continues.

We begin by describing three hardware tools which aid in construction of measuring facilities. Then eight particularly useful, general, programmed measuring facilities are described. This is followed by a brief discussion of the built-in instrumentation used to monitor multiprogrammed demand paging. After a description of techniques for obtaining controlled measurements, a few observations about measurement of operating systems conclude the discussion.

Hardware Tools for Instrumentation

Before describing the programmed measurement techniques used in Multics, one must know of three specially designed hardware tools provided by the General Electric 645 computer on which Multics is currently implemented. These three are: a program readable calendar clock which provides a uniform and precise time base for all measurements, a memory cycle counter in each central processor, and a special input/output channel which permits another computer to monitor the contents of the GE-645 primary memory.

The calendar clock consists of a 52-bit register whose contents are increased by one, once each microsecond, under control of a quartz crystal oscillator. This rate is in the same order of magnitude as the instruction processing rate of the GE-645, so that timing of ten-instruction subroutines is meaningful. The register is wide enough that overflow requires several tens of years, so that it serves as a calendar.* The calendar clock fits into the GE-645 system structure like a bank of memory: it is directly readable as a double-precision integer by a single instruction from any central processor. A library subroutine is provided to read the clock and in principle a language translator can implement a built-in function for the purpose. This design encourages use of the clock reading as a "date and time stamp" to uniquely identify messages, files, and processes in the system.

---

* All Multics installations contain the same calendar clock setting: the number of microseconds since 0000 GMT, Jan. 1, 1901. The clock is set by a field engineer, who is supplied with a printed table of such settings keyed to local standard time.

There are three significant advantages to this hardware clock design compared, say, with a software clock simulated by a processor interval timer (a technique which requires no extra hardware in most present-day computers):

- The simplicity of usage of the hardware clock is very great, both for supervisor and for user procedures.
- In a multiprocessor system there is no question as to which processor is maintaining the simulated clock (and no problem of separately maintained and potentially unsynchronized clocks.)
- Confidence in the accuracy of the clock is easy to gain; one does not have to worry about accuracy of special code which compensates for interrupts, or loss of "ticks" during register reloads.

Associated with the calendar clock is a program loadable 52-bit time-match register, which is continuously compared by hardware with the calendar clock register. Whenever a time match occurs the clock generates a "time-match" interrupt. A supervisor procedure coordinates the use of this facility among the users of the system and the system's own metering facilities.

The second hardware tool is simply a modification of the ubiquitous processor interval timer; in the GE-645 this "timer" counts the number of memory references made by the central processor rather than the number of clock ticks. There are at least three reasons for interest in such a measurement:

- In a multiprocessor system which exhibits interference on access to primary memory, it permits a load-independent measure of cpu usage, for accounting purposes.
- Comparison with calendar clock readings permits measurement of memory interference.
- Comparison with instruction counts permits a check on the associative memory of the GE-645 central processor, to see to what extent it reduces memory accesses for page table words.

This tool has not yet been used in any significant way since gross system measurements suggest that scratchpad memory effectiveness is near its theoretical upper limit and memory interference is insignificant in the present configuration. Sustained production use of two processors will probably rekindle interest in this tool.

The third hardware tool is an input/output channel which can run in an endless loop, once initialized, without attention from the operating system. The particular endless loop programmed is a "read into the address part of the next command" followed by a "write out repeatedly the contents of the word whose address was just read". The channel is connected, by a 2400 Baud telephone line, to a Digital Equipment Corporation PDP-8/338 programmable display computer. With this channel, the PDP-8/338 program can monitor the contents of any Multics data bases for which it knows the location and format. The data rate involved -- less than 60 words per second -- presents a negligible I/O and memory cycle load to

the GE-645 system. Since no GE-645 processor code is executed in obtaining the data (as would be the case if one of the system's users probed periodically into a data base) one can be quite confident that the act of probing has not significantly affected the measurement. This slow data rate does make it difficult to monitor a rapidly changing data base.

General Software Tools for Instrumentation

A number of general programmed measurement tools have been implemented as part of Multics; eight of them are reported here. All of these tools are built into the system in such a way that the tool is always invokable. Thus, any would-be observer can make observations and perform experiments without making system modifications, and with minimum effect on the conditions of measurement. The performance degradation caused by these permanent installations has been both estimated and measured to be quite small.

The first, and most elaborate of these tools is a general metering package which records time spent doing selectable supervisor primitive operations while the system is running. For each selected primitive the metering package records the number of times the primitive is invoked and the total execution time accumulated within each of a number of ranges of execution time.

Four primitives in Multics associated with implementation of the Multics virtual memory[2], were intuitively felt to be potential system bottlenecks and thus were chosen for initial integration with the measurement package. The first of these is the dynamic linking procedure, which is invoked when a procedure makes a symbolic reference to another procedure or data. The second primitive is the missing-segment procedure which is invoked to set up the environment required for paging. The third primitive is the missing-page procedure which is invoked when a program refers to a

page not in primary memory. The fourth primitive is the wall crossing procedure which is invoked each time the process needs to switch from one protection ring (domain of access) to another ring. Such a switch occurs, for example, on each call from a user program to a supervisor procedure.

The measurement of time spent executing a primitive is complicated by two problems. The first problem is that the central processor is multiplexed among many processes; thus something more than reading the calendar clock at the beginning and end of execution of a primitive is required to compute time spent executing in a primitive. Time spent waiting for I/O operations on missing pages or for a lock to clear is not counted as part of the primitive whenever the situation permits multiprogramming during the wait. The second problem is that primitives may invoke other primitives (including themselves) during their operation and provision must be made for this situation. For example during the handling of a missing-segment both missing-pages and additional missing segments may occur, each of which must be handled in order to proceed with the original missing-segment handling. The rule is that time spent in a nested primitive is not charged to the nesting primitive if the nested primitive is also being metered. By this rule it is possible to perform a pair of experiments to learn the amount of time spent in a nested primitive as a result of handling the nesting primitive. For example, if one first meters both missing-segment handling and missing-page handling, and then later meters missing-segment handling only, the second experiment will show missing-segment handling time increased by just the amount of missing-page

handling which was triggered by missing segment handling.

A segmented system provides a simple way to detect how time spent in the system is distributed among the various components. The second tool, a segment usage metering facility, sets the calendar clock to interrupt periodically (typically every ten milliseconds). When the interrupt occurs, the segment number of the segment which was executing is used to index into an array of per segment counters and the appropriate counter is incremented by one. After the system has run for a while this table can be sorted and the resulting distribution of segment usage can be printed out, listing the most popular segments first. This facility is similar to the one described by Cantrell and Ellison[3].

A realted, third tool records on a per-segment basis the number of missing pages and segments encountered during execution in that segment. Both of these measurements can br coupled to the general metering package described earlier. If coupled, the arrays are updated only during handling of metered primitives or possibly only when outside the metered primitives but within a specified process. This latter option permits detailed analysis of any user program.

Two examples of the use of these three measurement facilities illustrate their utility. The first significant application of these packages was in the analysis of missing-page handling. By obtaining the time distribution function for missing-page handling and running segment usage metering during this time only, it was possible to compute how much time was spent in each module of the missing-page handler. A heavy imbalance of time spent in the core management module suggested a redesign of that module so as to not handle multiple page sizes.

As a second example, a user analyzing his own program can use the packages in several ways. By requesting the timing of all supervisor primitives, and then running segment usage metering only during time outside the metered primitives the user can deduce the central processor time expended in each of the procedures which are part of his program. The supervisor primitive timing permits him to see the cost of the specific types of primitives he is using. The per-segment missing-page counters allow him to see if one of his own segments encountered an unexpectedly large number of missing pages -- perhaps because it uses a data structure ineffectively.

A typical example of the (somewhat cryptic) output of the general metering package is shown in Figure 1. This output was obtained during execution of the standard "certification" script described below. The primitives measured are the fault and interrupt handling modules.

A fourth tool, very different in scope from the above described facility, counts the number of times procedures are called. A standard call-save-return sequence is used for all interprocedure reference in Multics. An "add-one-to-storage" instruction is included in this sequence which increments a counter each time a procedure is entered. This counter enables a programmer to determine later exactly how many times a procedure has been called and to relate that number to the number of calls to other procedures.

A fifth tool is a specially designed software package named the Graphical Display Monitor, the subsystem of PDP-8/338 programs that use the previously described synchronous data channel to interrogate locations of memory in the GE-645. Multics obliges this display by building, during system initialization, a table containing pointers to interesting data

bases. A set of display generating tools permit one to prepare a new display program in a few hours time. Some standard displays have been developed to observe the traffic controller's queues, the arrays of primitive handling time distributions, and the use of primary memory. Observations of these displays have been helpful in detecting bottlenecks in the system and on several occasions have exhibited the system passing through states previously thought to be impossible. A more complete description of this tool and examples of its output will be found in the paper by Grochow[4].

Perhaps the most useful software measurement tool of all is the sixth and simplest one: following the completion of each typed command the command language interpreter types a "ready message" consisting of three numbers. The first number is the time of day at the preparation for printout of this ready message. The second number is the cpu time used since the previous ready message to the nearest millisecond. The third number is the number of times the process had to wait for a page to be brought in. These pieces of information, which appear automatically and almost free of charge, give immediate feedback to the programmer as to the resource usage of the command just typed. This feedback is an invaluable aid to the programmer in seeing the influence of program changes upon performance. For cases where there are several ways to perform the same task the user is given guidance as to which is more economical. For example, there are two text editors currently available on Multics; the cheaper one is readily apparent and thus generally the chosen editor.

Perhaps the most significant drawback to providing powerful system facilities such as a large virtual memory and a full PL/I compiler is the ease with which even a sophisticated system programmer can unintentionally trigger unbelievably expensive operations. One of the principal Multics tools to fight back at misuse of virtual memory as though it were real memory is a missing-page tracing package. In this package, the missing page handler retains in a ring buffer the segment and page number and the time of day of the last 256 missing pages of the process under measurement. Printing out the contents of the ring buffer following execution of some library program is often very revealing, since it provides unimpeachable evidence of which were the different pages the program touched. This tracing package frequently reveals that a large working set is the result of unnecessary meandering in the path of control of a program. The list of pages touched gives a programmer precise ammunition on how to reorganize his program to improve its locality of reference.

Finally, a second tracing package monitors the effect of the system's multiprogramming effort on an individual user. The general strategy here is to write a user program which goes into a tight loop repeatedly reading the calendar clock. Normally, successive clock readings differ by the loop transit time. If a larger difference occurs, it is a result of control of the processor having been snatched away from the loop to handle an interrupt or run another process. These larger time differences, and the time they were noted, are added to the end of a growing table of interruptions, and the program returns to its loop. When the table is filled, the program

prints out the table, showing the time of occurrance of each interrupt, and the length of time it took to handle it. This table helps build confidence that the processor scheduling algorithm is working as predicted, and it occasionally discovers a misprogrammed data channel which is producing more frequent interrupts than necessary. It also provides an independent confirmation of the time required to handle each interrupt. This measurement is a good example of one for which a simulated software clock would barely suffice since the clock simulation itself is likely to interact with the scheduling algorithm and the interrupt handlers, whose functions are being measured. On the CTSS system, a predecessor of Multics for the IBM 7094, the lack of a calendar clock forced this type of measurement to be made using arrival of words from a magnetic tape as a kind of pseudo-clock.

Apart from the two techniques just described, Multics does not have built-in general event tracing packages, such as those reported by Campbell and Heffner[5]. This lack can probably be attributed to a suspicion that the volume of interesting traceable events in Multics would preclude intelligent analysis; nevertheless there have been times when it was thought that a built-in general trace would have been very handy.

Special Instrumentation for Multiprogrammed Demand Paging

Multiprogramming[6] has been in use for a long time in a variety of systems. In a few words, multiprogramming consists of keeping several programs in primary memory, so that when one program encounters an I/O roadblock control of the processor can be immediately switched to another.

The objective is to keep the central processor busy more of the time, and thereby increase the rate of job completions. This improved utilization of the processor comes about at the expense of extra primary memory required to hold programs which are ready to absorb a released processor.

If the primary memory can hold only a few programs, there will be times when all available programs are roadblocked simultaneously. The central processor then must idle, waiting for some program's I/O requirements to be satisfied. This idle time we will term "multiprogramming idle", to distinguish it from "true idle" time which occurs when, despite space in primary memory for another program, there is simply no customer waiting for the system's services.

We have thus far identified two measures of central processor utilization for which instrumentation must be provided. If multiprogramming idle time is a significant fraction of the total real time, it may indicate a shortage of primary memory or an unhelpful scheduler. If true idle time is large, the system is probably not fully loaded.

The fundamental complication introduced by the ability to run a program without all its pages in primary memory is that there is no longer a simple rule to determine whether or not one more program will fit. In fact, with limits which are too generous to be helpful, there is always room for one more program in primary memory. Unfortunately, addition of another program to this"eligible set" may either allow some otherwise idle processor time to be used, or it may cause the programs in the eligible set to fight over the available memory.

Some system designers[7] have partitioned primary memory among the eligible programs. If a program in one partition is not allowed to steal space for its pages from a program in another partition, the question of adding another program to the eligible set is just like that of multiprogramming without demand paging. Unfortunately, this strategem breaks down when many pages are (and any page may be) shared among several programs, as required by the objectives of Multics[8]. We have therfore explored the non-partitioned avenue, by controlling the size of the eligible set, at first rigidly, and in the future dynamically. For either rigid or dynamic control, the existence of control implies that there will still be multiprogramming idle time; the decision about adding another program to the eligible set turns on whether or not any additional paging activity thereby introduced either wipes out the recouped idle time, or causes unacceptable job delays.

A variety of special purpose meters are therefore included as an integral part of the Multics multiprogramming scheduler and the page-removal selection algorithm. Measures of paging activity include total processor time spent handling missing-pages, number of missing pages, average running time between missing pages, and average length of the grace period from the time a page goes idle until its space is actually reused.

As a rough measure of response time for a time-sharing console user, an exponential average of the number of users in the highest priority scheduling queue is continuously maintained. The exponential average is computed by a method borrowed from signal data processing[9].

An integrator, I, initially zero, is updated periodically by the formula

$$I \leftarrow I*m + N_q; \qquad 0.0 \leq m < 1.0$$

where N_q is the measured length of the scheduling queue at the instant of update, and m is an exponential damping constant which determines the average distance into the past over which the average is being maintained. In general, the sample which was taken

$$k = \frac{1.0}{1.0 - m}$$

samples in the past will have an effect on the value of the integrator 1/e as large as the current sample. The average queue length is approximately

$$\overline{N}_q = I/k$$

This averaging technique, which requires only a multiply and an add instruction slipped into a periodic path, is an economical way to maintain an average which does not "remember" conditions too far into the past.

If the recent average queue length is multiplied by the average run time in the first queue an estimate is obtained of the expected response time of the moment. This estimate has been used on CTSS to dynamically control the number of users who may log into the system. In Multics, this estimate, as mentioned above, is also a guide with which to measure effective uses of dynamic control of the size of the multiprogramming eligible set.

Control of Measurements: Script Driven Tests

A persistent problem in a complex operating system is evaluating the effect of a small change in a factor presumed to affect performance. If the system is observed under a normal load of usage before and after the change, fluctuations in the nature of the load may wipe out the effect

to be measured. To get around this problem a standard "benchmark", or series of programs, is often devised. This benchmark can then be run against a new system while taking measurements to compare with the old.

When the system under test is designed to be used interactively from time-sharing consoles, two difficulties are introduced:

- A load simulator must maintain a large number of simultaneous but low density input streams.
- Each individual input stream should be somehow representative of a human user conversing with a computer system. For example, inputs should be separated by pauses representing "think times".

Greenbaum[10] attacked this problem for Multics by developing a program for the PDP-8 computer which via telephone lines to the GE-645 simulates one to twelve simultaneous interacting human users, each of which is following a (possibly different) script of commands to be input to the system with interspersed "think" time intervals.

A number of scripts have been developed, but the one most frequently used is one which represents a user typing in and debugging a small FORTRAN program. This script goes as follows:

- type in program
- try to translate it, discovering errors
- edit the program to correct it
- translate the program, this time successfully
- rename the program
- run the program
- print the program on the user's typewriter
- list all files associated with the program
- delete the program

This script "maps into" the available command language of a variety of time-sharing systems, and can therefore be used as a basis for inter-system comparison of usage charges and response time. When used on CTSS, this script produces a measured load similar to that observed by Scherr[11] over a long period of actual use. Although the script simulates only a very specific class of user, the mix of system services invoked (note that actual running of the user's program is a small part of the script) is probably similar to that invoked by a wider class, so that if a system change improves the performance of the script, it can be expected to similarly improve the performance of the system under actual load.

Because of the sheer logistic problems of extending a telephone-line driven technique to more than a few simulated users, an internal driver program for Multics has also been developed. The driver can create as many processes as desired and have them each perform some script (the script described above is usually used) in competition with one another. Only a minimal change to the normal operating conditions is required, because the Multics I/O system provides the capability of attaching an input/output stream to a file rather than a typewriter. Even so, this technique has the limitation that it does not exactly simulate real users. The I/O path to a file is inescapably different from the path to a typewriter (especially as to number of different pages in the working set) and the driver program itself competes for resources at least at the beginning and end of the test. In addition, the current version of the internal user simulator does not insert "think times" between commands, although addition of this feature is contemplated. Despite these difficulties, the internal user simulator has proven very useful because of its simplicity of operation and repeat-

ability of the measurements taken while it operates. When new Multics systems are installed, they are first required to be "certified" by running the user simulator as a check on both performance and functional capability.

Observations

One conviction gained from experience with Multics, and earlier with CTSS, is that building permanent instrumentation into key supervisor modules is well worth the effort, since the cost of maintaining well-organized instrumentation is low and the payoff in being able to "look at the meters" any time a performance problem is suspected -- or even when one is not -- is very high. In a large system, a kind of inertia frequently impedes quick changes to a module such as installation of temporary meters in response to some suspected problem.

A second conviction, arising from a variety of experiences when an apparent performance bug turned out to be an instrumentation bug, is that the meter readings are always suspect. Whenever possible, an independent, perhaps gross, measurement which can confirm some aspect of a measurement in question is very worthwhile.

A third observations is that most system programmers are not by training or temperament scientists, and they often lack the patience to methodically set up an experiment which is precisely controlled. An alarming number of "non-experiments" are performed, with a total useful (?) result of (for example) "I brought the system up with a shorter scheduling quantum and response seemed a little better." Although much useful information and insight can be gained by on-line monitoring of uncontrolled live users, use of such measurements for performance comparison must always be suspect since the particular user population at any instant may be non-"average". The

rules that apply to all scientific measurements also apply to measuring computer systems:

- Controlled experiments require great care, and may be quite expensive.
- Uncontrolled experiments are uninteresting -- one must make only one change at a time if he is to honestly evaluate the change.
- Before embarking on an experimental change one should first make a prediction of what measurements should change, and then spend some time understanding why they didn't change exactly as predicted.
- One must always be on the watch for unintentional misinterpretation of a result by a system programmer (or his manager) who has a large personal stake in a hoped-for outcome. One is dealing with human beings, and the psychology of error is no different than in other situations.

Acknowledgements

Almost everyone who contributed to the design of Multics has contributed at least one suggestion toward its instrumentation. F. J. Corbató and E. L. Glaser offered helpful suggestions on almost all aspects. Contributions to the design of the calendar clock were made by Chester Jones, Joseph Ossanna, and George Futas. Victor Vyssotsky suggested the cpu memory cycle counter. Early work on the PDP-8/Multics I/O channel was done by Daniel Edwards and Thomas Skinner. The PDP-8/338 graphic/monitor display was designed and implemented by Jerrold Grochow. Contributions to the fault metering package came from Charles Clingen and David Vinograd. Robert Rappaport and Steve Webber contributed to the design of metering for dynamic paging and multiprocessor scheduling. The internal script driver was designed and implemented by David Stone and Richard Feiertag; the external (PDP-8) version, by Howard Greenbaum and Akira Sekino.

References

1. Corbató, F.J., et al, "A New Remote-Accessed Man-Machine System," AFIPS Conference Proceedings 27 (1965 FJCC), Spartan Books, Washington, D.C., 1965, pp 185-247.

2. Bensoussan, A., C.T. Clingen, and R.C. Daley, "The Multics Virtual Memory," Second ACM Symposium on Operating System Principles, Princeton, N.J., October, 1969.

3. Cantrell, H.N. and A.L. Ellison, "Multiprogramming System Performance Measurement and Analysis," AFIPS Conference Proceedings 32, (1968 SJCC), Thompson Book Co., Washington, D.C., 1968, pp. 213-221.

4. Grochow, J.M., "Real-Time Graphic Display of Time-Sharing System Operating characteristics," to be presented at 1969 Fall Joint Computer Conference, Nov., 1969.

5. Campbell, D.J., and W.J. Heffner, "Measurement and Analysis of Large Operating Systems During System Development," AFIPS Conference Proceedings 33, (1968 FJCC), Thompson Book Company, Washington D.C., 1968. pp. 903-914.

6. Critchlow, A.J., "Generalized Multiprocessing and Multiprogramming Systems," AFIPS Conference Proceedings 24 (1964 FJCC), Spartan Books, Baltimore, 1963, pp. 107-126.

7. Denning, P.J., "Resource Allocation in Multiprocess Computer Systems," Ph.D. Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering, May, 1968. (Available as M.I.T. Project MAC Technical Report TR-50).

8. Corbató, F.J., "A Paging Experiment with the Multics System," to be published in a festschrift for P.M. Morse, 1969.

9. Blackman, R.B., and J.W. Tukey, The Measurement of Power Spectra, Dover, New York, 1958 (Originally appeared in Bell System Technical Journal, January and March, 1958).

10. Greenbaum, H.J., "A Sumulator of Multiple Interactive Users to Drive a Time-Shared Computer System," S.M. Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering, October 1968. (Available as M.I.T. Project MAC Technical Report TR-58.).

11. Scherr, A.L., An Analysis of Time-Shared Computer Systems, M.I.T. Press, Cambridge, Mass., (1967).

```
DATE: 06/05/69  2104.7 EST Thu
RUN DURATION: 06:51.850000                    CASE: 0

**********SUMMARY OF RESULTS**********

AVG. FAULT TIMES:

PAGE: 00:00.007421  LINK: 00:00.023143
SEG.: 00:00.013856  WALL: 00:00.001418
INTR: 00:00.000939

TOTAL NUMBER OF FAULTS
PAGE:   6295        LINK:    732        INTR:  15301
SEG.:    140        WALL:   3236

FAULT PROCESSING DURATIONS

TIME IN METERED PROCESSES: 06:51.340000

PAGE: 00:46.720989  %=  11.35
SEG.: 00:01.941238  %=    .47
LINK: 00:16.941152  %=   4.11
WALL: 00:04.676431  %=   1.13
INTR: 00:14.379650  %=   3.49

**********FAULT TABLE**********

FAULT      TIME            COUNT          TOTAL TIME        MEAN
TYPE       GROUP         NUM     %         MS.     %        MS.

Intr    256-512us.      9523   62.23      3733   25.96      .39
Intr    .5-1ms.          406    2.65       261    1.81      .64
Intr    1-2ms.          3050   19.93      4189   29.13     1.37
Intr    2-4ms.          2317   15.14      6162   42.85     2.65
Intr    4-8ms.             5     .03        32     .22     6.58

Page    .5-1ms.           16     .25        14     .03      .88
Page    1-2ms.            34     .54        59     .12     1.74
Page    2-4ms.           796   12.64      2004    4.29     2.51
Page    4-8ms.          3916   62.20     24567   52.58     6.27
Page    8-16ms.         1265   20.09     13910   29.77    10.99
Page    16-32ms.         243    3.86      5131   10.98    21.11
Page    32-65ms.          24     .38       965    2.06    40.21
Page    65-131ms.          1     .01        68     .14    68.75
```

Figure 1--Sample result of fault metering.