

ABSTRACT

WORTH, STEPHEN G. III. Benchmarking, Analytic Modelling, and Performance Analysis of a Data General Eclipse MV/8000. (Under the direction of William J. Stewart)

This study develops an analytic model for the prediction of response times on an Eclipse MV/8000 based upon input parameters related to user workload. This model was designed for the system owned by the Computer Science department at North Carolina State University. This system, at the time of the study, was used for teaching introductory courses. The model uses inputs related to execution times, I/O activity, and paging activity of student programs along with information about the operating system and disk activity. The model predicts the response time of student programs compiled and executed in the batch stream along with the response time necessary to edit an average file.

The model is based on a two level hierarchical closed queueing network using two customer classes, batch and interactive. The low level model represents the activity at the multiprogramming level of the system including operating system overhead. Service times for the disk activity at this level are estimated from a linear approximation based on the average number of tracks seeked per disk access. The high level model represents the activity of the system at the user's level. Service times at this level are based upon the throughputs from the low level model. The high level model uses separate batch and interactive servers to provide the response times for each class of work.

The model was validated with the aid of a benchmark. The benchmark was implemented using script files and was designed to simulate the user workload measured during a characterization study. The characterization study required the development of several procedures to build a user workload profile and to monitor data during benchmarking. The model, once validated, was used to evaluate the effect on response times by making a CPU upgrade from an MV/8000 to an MV/10000. The overall effect of the upgrade was an average improvement of 250% in response times.

Benchmarking, Analytic Modelling, and Performance
Analysis of a Data General Eclipse MV/8000

by
Stephen G. Worth III

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Studies Program

Raleigh

1985

Approved By:

Chairman of Advisory Committee

BIOGRAPHY

Stephen Gardner Worth III, was born in Fayetteville, North Carolina on July 8, 1959. In 1965 his family moved to Reidsville, North Carolina where he graduated from Wentworth High School in 1977. During his youth the author was active in the scouting program attaining the rank of Eagle Scout along with receiving the Vigil Honor in the Order of the Arrow.

The author entered North Carolina State University and graduated in 1981 with a Bachelor of Science degree in Chemistry. That fall he worked as a teaching assistant in the Computer Science department at North Carolina State University and took courses on a part-time basis. In January of 1982 the author enrolled in the Computer Studies Program and since then has been working as a teaching assistant while working towards his degree.

The author is married to the former Natalie Jeannine Sutton. Mrs. Worth works in the Financial Aid department at North Carolina State University and is finishing a Bachelor of Arts degree on a part-time basis.

ACKNOWLEDGEMENTS

I would like to thank the chairman of my advisory committee, Professor William J. Stewart, for his constant advise in all the aspects of this work. I also wish to thank Professor Harry G. Perros who was always willing to listen to the many problems encountered in model development and whose advice was helpful in resolving those problems. Special thanks also go to the manager of the Leazar Hall computing facility, Harry Kuhman, whose help was instrumental in understanding the architecture of the Eclipse MV/8000 and in setting up the benchmark. Thanks also go to Professor David McAllister who carefully proofread much of this work. To my friends and colleagues, Larry Hodges and Kenneth Clark, I extend much thanks for all the interesting and usefull discussions we had during the course of this work.

Finally, I wish to extend my deepest appreciation to my wife, Natalie, without whose support and constant encouragement this work would not have been completed.

TABLE OF CONTENTS

	Page
LIST OF TABLES	v
LIST OF FIGURES	vii
1. INTRODUCTION	1
1.1 Hardware Description	3
1.2 File Structure and Input/Output Processing	8
1.3 Protection and Memory Management	11
1.4 Process Scheduling	18
2. WORKLOAD CHARACTERIZATION	25
2.1 Background	25
2.2 Data Collection Methods and Problems	31
2.3 Characterization Results	41
3. ANALYTIC MODELLING AND BENCHMARKING	71
3.1 The Low Level Model	72
3.2 The High Level Model	80
3.3 Benchmarking the MV/8000	84
3.4 Validation of the Model	89
4. PERFORMANCE PROJECTIONS AND CONCLUSIONS	96
4.1 Performance Projections	96
4.2 Conclusions	103
5. LIST OF REFERENCES	107
6. APPENDIX	109

LIST OF TABLES

	Page
2.1 Enrollment Figures for Introductory Computer Science Courses	26
2.2 Summary of User Console Processes over Study Period from SYSLOG_REPORT	43
2.3 Summary of Process Types over Study Period from SYSLOG_REPORT	43
2.4 Summary of Process Statistics over Period from November 1982 through June 1983	56
2.5 Summary of Interactive Process Statistics over Period from November 1982 through June 1983	57
2.6 Summary of Batch Process Statistics over Period from November 1982 through June 1983	58
2.7 Summary of Process Statistics During Peak Hours (2 - 5 pm.) over the Period November 1982 to June 1983	59
2.8 Summary of Interactive process Statistics During Peak Hours (2 - 5 pm.) over the Period November 1982 to June 1983	60
2.9 Summary of Batch Process Statistics During Peak Hours (2 - 5 pm.) over the Period November 1982 to June 1983	61
2.10 Summary of Average User Work Session During Peak Hours (2 - 5 pm.) over the Study Period	63
2.11 Revised Summary of Interactive Process Statistics During Peak Hours (2 - 5 pm.) over the Period November 1982 to June 1983	66
2.12 Revised Summary of Batch Process Statistics During Peak Hours (2 - 5 pm.) over the Period November 1982 to June 1983	67
2.13 Revised Summary of Process Statistics During Peak Hours (2 - 5 pm.) over the Period November 1982 to June 1983	68
2.14 Revised Summary of Average User Work Session During Peak Hours (2 - 5 pm.) over the Study Period	69
3.1 Throughputs from the Low Level Model	90

3.2	Throughputs from the High Level Model	91
3.3	Response Times from the High Level Model	92
4.1	Input Values Used for CPU Upgrade Study	98
4.2	Results of CPU Upgrade Study	99
5.1	Data from the Thirty User Benchmark	110
5.2	Low Level Model Results for the Thirty User Benchmark	115
5.3	High Level Model Results for the Thirty User Benchmark	117

LIST OF FIGURES

	Page
1.1 Architectural Diagram of the MV/8000	4
1.2 Addressing Form for Two Level Page Address Translation	14
1.3 AOS/VS Paging Activities	16
1.4 Process States in AOS/VS	19
1.5 AOS/VS Scheduling Queue Structures	23
2.1 Sample Process Termination Dump	34
2.2 Hierarchy of Batch Jobs on the MV/8000	37
2.3 Average Users Logged on in November 1982	47
2.4 Average Users Logged on in December 1982	48
2.5 Average Users Logged on in January 1983	49
2.6 Average Users Logged on in February 1983	50
2.7 Average Users Logged on in March 1983	51
2.8 Average Users Logged on in April 1983	52
2.9 Average Users Logged on in June 1983	53
2.10 Average Users per Day Over Period from November 1982 - June 1983	54
3.1 Low Level Queueing Model of the MV/8000	73
3.2 High Level Queueing Model of the MV/8000	81
4.1 Response Time of Batch Jobs	100
4.2 Response Times of Interactive Processes	101

1. INTRODUCTION

This thesis presents the steps taken to develop and the results produced from an analytic performance model of the Data General Eclipse MV/8000 operated by the Computer Science Department at North Carolina State University. The study was motivated by the need for a performance prediction tool that could relate program response time against a number of users given a certain workload. At the start of this study the system was being used to teach introductory FORTRAN and Pascal programming courses supporting a user community of 1500 students. These users were typically editing and running small programs which could execute in under 30 seconds of execution time. It was planned that over time the machine would be used more for higher level undergraduate courses and less for introductory courses. The model could be used in evaluating system response times based on workload changes caused by the new courses.

The only prior performance study of an MV/8000 by Kinicki, et. al.[1] used a benchmark to evaluate changes in memory configuration along with operating system upgrades of a small system. The results of the study showed that upgrades in the operating system by the manufacturer lowered the processing times of programs through reducing the amount of paging. The study demonstrated that a memory upgrade sufficient to eliminate process swapping also lowered processing times. Finally the study identified the link step as the most resource intensive portion of compiling and running a program. Studies of other machines in the same size class have a bearing on the model developed

in this work. In particular, the studies of Lazowska[2] and Hodges[3] on the Digital Equipment Corporation's VAX 11/780 provided help in understanding the use of a hierarchical system model along with background on the internals of a swapping and paging system.

This thesis is organized as follows. Chapter 1 is devoted to a study of the MV/8000 architecture along with its associated operating system, AOS/VS. Close attention was paid to the areas of memory management, input/output processing, and process scheduling with an emphasis towards the development of a model that represented the activity of the machine. Chapter 2 is a characterization of the workload over an eight month period from November 1982 to June of 1983 using the limited information provided in the system accounting logs. This characterization led to several assumptions that aided the modelling process. Chapter 3 develops the two stage hierarchical model. Each stage of the model is a closed queueing network using two customer classes. The models are solved using the technique of Mean Value Analysis[4,5,6,7] as provided by QNAP2[8], a queueing network solver. The model's lowest stage uses a linear approximation for disk service times based upon measurements of the average seek distance per access. This model was validated using a benchmark composed of editor script files and FORTRAN programs organized to resemble a heavy user workload. In Chapter 4 the validated model is used to analyze the effect of a processor upgrade from an MV/8000 to an MV/10000 which, according to the manufacturer[9], is about three times faster.

1.1 HARDWARE DESCRIPTION

The Data General Eclipse MV/8000 is a 32 bit minicomputer with capabilities to function in both a minicomputer and mainframe environment. Until its recent redesign into the MV/8000II and before the introduction of the MV/10000, the MV/8000 was the most powerful member of the Eclipse family of computers. Typical specifications for the processor boasts a fixed point addition time of 220 nanoseconds and an input/output (I/O) to memory bandwidth of 18.2 Mbytes/second. An instruction set of over 200 instructions contains instructions for the manipulation of fixed point data, floating point data, commercial data, stack elements, and queue elements. This machine was originally designed to compete in the marketplace with Digital Equipment Corporation's VAX 11/780.

The hardware configuration[10, 11] of the system used at NCSU is presented in figure 1.1. The MV/8000 is equipped with 6 Mbytes of primary store logically arranged in 16 bit words. The memory is word addressable and is physically arranged as a series of twelve 512 Kbyte modules (printed circuit boards). Each module is split into four interleaved planes of 32K double words allowing 16 bytes of memory to be read on each memory cycle. A 16 byte block of memory can be read in 440 nanoseconds. The Bank Controller is used to refresh the semiconductor memory every 15 microseconds along with performing single error correction and double error detection on every word transferred into or out of memory.

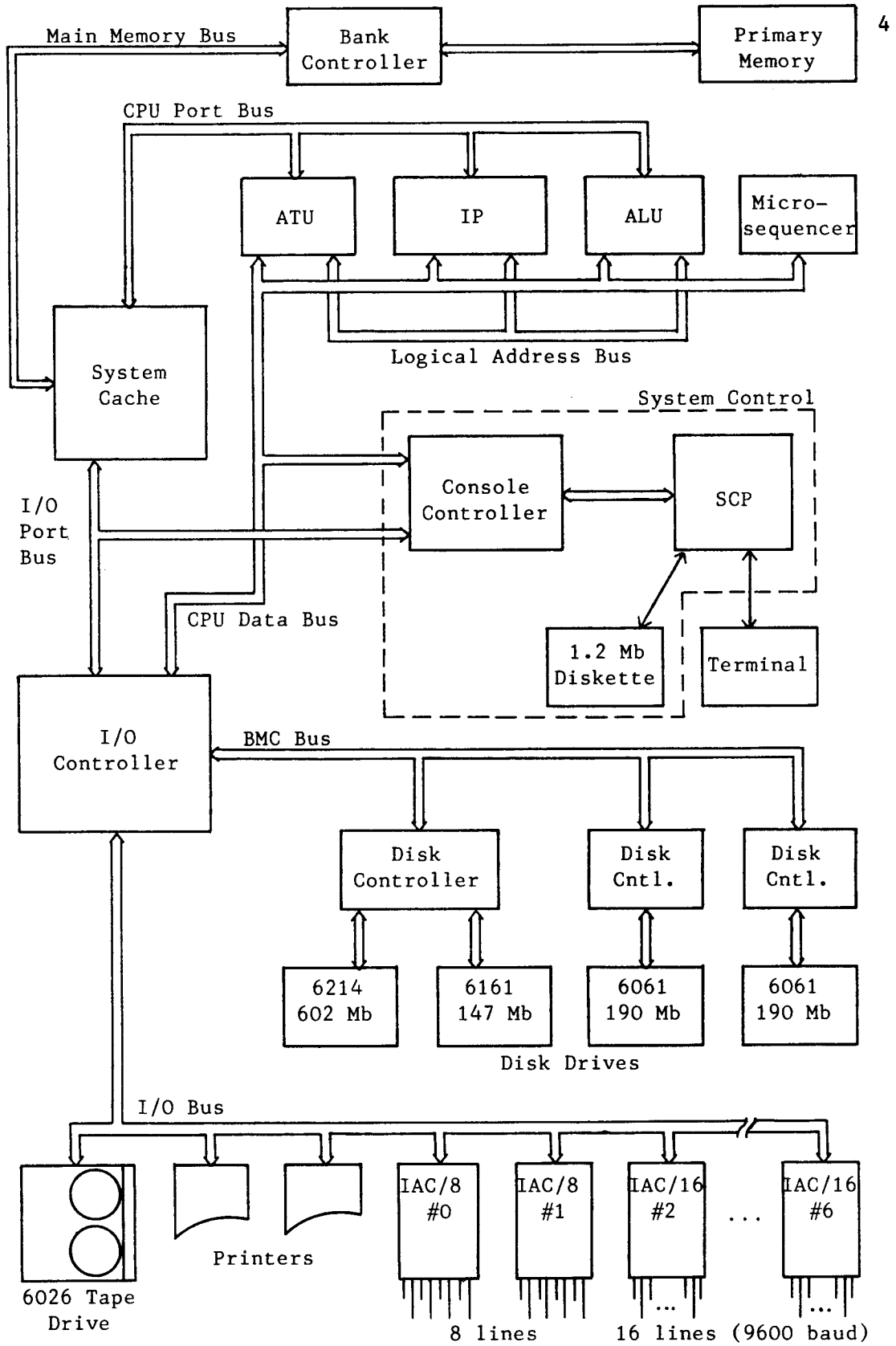


Figure 1.1
Architectural Diagram of the MV/8000

At the heart of the machine is the 16 Kbyte System Cache which speeds up processing by serving as a lookahead and lookbehind buffer. Primary storage is physically mapped onto the System Cache in the following manner. The cache is arranged as 1024 rows of 16 bytes numbered block 0 through block 1023. Due to its physical structure the primary store is mapped into 64 groups of the same dimensions as the cache. When a transfer is made in or out of primary store the 16 byte block of memory must be transferred into or out of the corresponding block number of the System Cache. The cache is maintained so that a block, if modified, will be written back to primary store before the next block is read. An unmodified block is simply overwritten by the new block. Instructions and data being used for I/O operations must pass through the System Cache. The cache has dual ports allowing an 18.2 Mbyte/second bandwidth for both the I/O and Central Processing Unit (CPU) bus.

Shown below the primary store in figure 1.1 are the processing boards which consist of the Address Translation Unit (ATU), the Instruction Processor (IP), the Arithmetic Logic Unit (ALU), and the Microsequencer. The Instruction Processor uses a four stage pipeline along with a 1 Kbyte cache that maps directly onto the System Cache to increase processing speed within program loops. The stages of the pipeline are to fetch the instruction from the instruction cache, decode the instruction opcode to obtain the starting address of the microcode instruction, read the microcode instruction from memory, and execute the microcode instruction. The pipeline works in conjunction with the Microsequencer which contains control logic for the execution

of microinstructions along with a control store for the microcode.

The Arithmetic Logic Unit contains four 32 bit fixed point accumulators, four 64 bit floating point accumulators, a 31 bit Program Counter, a fixed and floating point status register, and four 32 bit wide stack management registers. The wide stack holds parameters and return blocks for both procedure calls and processor interrupts. The Address Translation Unit is a combination of both processing logic and associative store. The unit takes a logical address from the Instruction Processor and converts it to the correct physical address for referencing memory. The Address Translation Unit accelerates the translation process by maintaining an associative store of the 256 most recently referenced pages in memory.

The System Control subsystem consists of the Console Controller, the System Control Processor (SCP) which is a Data General microNOVA computer, a 1.2 Mb diskette drive, and an operator terminal. The system control subsystem provides overall operator control for the MV/8000. This subsystem on power up loads and verifies microcode going into the control store. It can monitor and trace the operation of the system and provide a hardware error detection and logging facility to help field engineers in machine repair.

The I/O Controller is essentially an interrupt driven peripheral processor which provides support for three types of I/O. The Burst Multiplexor Channel (BMC) is used to allow the transfer of blocks of information to and from disk subsystems. Data can be written over the

Burst Multiplexor Channel at a rate of 14.54 Mb per second. Burst multiplexor I/O generally does not contend with processor communications. Some contention occurs when data must be transferred from memory into the System Cache while burst multiplexor I/O is happening. The I/O Bus is used to support both data channel I/O and programmed I/O. Data channel I/O is used to provide synchronous data transfer of multiple words with medium speed I/O devices such as the tape drive and the two printers shown in figure 1.1. The maximum output transfer rate for data channel I/O is 1.3 Mbytes/second. I/O transfer across the I/O bus does not cause blocking with the Burst Multiplexor Channel unless the I/O bandwidth of 18.2 Mbytes/second for the I/O bus is exceeded.

Programmed I/O, which is the transfer of individual bytes of information to and from slow devices, is performed across the I/O Bus by a series of Intelligent Asynchronous Controllers (IAC)[12]. Data General manufactures two types of Intelligent Asynchronous Controllers. An IAC/8 controls eight full duplex asynchronous lines providing modem controls for transmission rates from 50 to 19200 bits per second (bps). An IAC/16 controls 16 full duplex terminal lines without modem controls over the same data transfer ranges. The NCSU MV/8000 has two IAC/8s and five IAC/16s for the total hardware support of 96 terminals. In particular IAC #0 is used to control eight modem lines; 4 at 300 bps and 4 at 1200 bps. The remaining six Intelligent Asynchronous Controllers control terminals at 9600 bps. Each IAC uses a polling scheme to receive data from the user terminal. Each line has a byte of data collected automatically by a programmable

communications interface, inside the controller, which interrupts the IAC when a full byte is assembled. The IAC then transfers this byte to memory via the I/O Controller. If the IAC receives simultaneous interrupts from one or more programmable communications interfaces priority is given in the order of line 0 of the IAC to line 16.

The four disk drives attached to the Burst Multiplexor Channel through three disk controllers provide a total online storage of 1,129 Mbytes. The bulk of the online storage is contained in the 602 Mb and 147 Mbyte fixed media drives which are connected through the same controller. Both drives are Winchester technology disks that require a special controller of which there is only one on the system. The remaining two disks use removable media.

The operating system used on the system is Data General's Advanced Operating System/Virtual Storage (AOS/VS)[13,14]. This operating system provides interactive and batch processing support for up to 256 processes. Software used on the MV/8000 includes ANSI FORTRAN 77, AOS/VS Pascal, FORTH, XENIX (a UNIX interface with AOS/VS), AOS/VS C, SQL (a database management program), PRESENT (a database program for presentation of data), SLAM2 (a simulation language), and other Data General utilities to manipulate programs.

1.2 FILE STRUCTURE AND INPUT/OUTPUT PROCESSING

AOS/VS uses a hierarchical inverted tree structure for the maintenance of the file system. The base of this tree structure is considered the "root" directory. The leaves of this root directory

can be either additional directories or individual files. Any file in the system can be specified by giving the complete pathname of the file. The pathname is a list of the directories one would trace down the tree, from the root, in order to find the file. The maximum number of levels in a pathname is restricted to eight. The file structure is merged onto the four physical disks as follows. The root directory is the 147 Mbyte Winchester disk shown in figure 1.1. This directory contains a copy of the operating system to use when booting the system, swapping files for swapping user processes out of memory, paging files for holding single pages of a process while it is still memory resident, the system log file, and other utilities associated with operating system management.

The other three disks are subordinate to the root directory in the operating system file structure. The large 602 Mbyte Winchester disk is the directory :UDD, the first 190 Mbyte disk shown in figure 1.1 is the directory :UDD2, and the last 190 Mbyte disk (the rightmost one) is the directory :UDD3. The directory, :UDD holds user directories for users whose last names begin with the letters A - H. It also holds large files used for a student program submission system. Occasional backups of the 190 Mbyte disks are made onto :UDD because backups cannot be made on offline storage. The directory, :UDD2 holds user directories for the users I - N. The last disk, directory :UDD3, holds user directories for users O - Z along with a directory that contains all the compilers and editors for the languages supported on the system. This file organization was established to help distribute the load on the disk controllers in a

uniform fashion.

Disk files are constructed in multiples of a size known as a file element. On the MV/8000 a file element is four 512 Kbyte blocks, the same size as a physical memory page. A file is maintained using index levels. A file, when created, is given a single index level. An index level is a disk block that holds logical pointers to other file elements which contain the data of the file. When the first index level space is exhausted AOS/VS will provide a second index level. The pointers in the second index level point to first index level pointers. The operating system will provide up to three index levels as the file grows allowing a maximum file size of 2^{23} blocks. The MV/8000 is arranged such that several blocks can be transferred across the Burst Multiplexor Channel without program intervention. AOS/VS manipulates its file system in units of the file element size. Thus an access of the disk will result in the transfer of four blocks to or from the disk. The operating system maintains the file structure through the use of a bit map to indicate which blocks of the disk are open and which are occupied. To find a place for a file element on disk the operating system scans the bit map to find an open area of the file element size, marks that area as full, and transfers the file element to disk. A file is removed by deleting its directory entry and setting the bits for those blocks in the bit map to indicate free space.

1.3 PROTECTION & MEMORY MANAGEMENT

The MV/8000 supports a logical address of 4 gigabytes which is implemented in eight 512 Mbyte segments. These segments are associated with a MULTICS like ringed protection system. Each ring, corresponding to a segment, has associated with it a different level of protection. Ring 0, the innermost ring, has the highest level of protection and is used to hold the kernel of AOS/VS while ring 7, the outermost ring, has the lowest level of protection and is used to hold user programs. The protection system works as follows. While active a user process is executing within a segment known as the current segment. The program counter wraps around to ensure that the process stays within the segment. A program can access any location within the current segment along with any location in a higher numbered segment. Access is not allowed to any location within a lower numbered segment. This means that the kernel, in segment zero, has access to any location within the logical address space while the user program running in segment seven has access only to data within segment seven.

A program is allowed to call a procedure within a lower numbered segment by making a subroutine call through a gate array. A gate array is a list of information placed at the start of a segment which is used to validate what other segments may call routines within this segment. The gate array also specifies the displacement from the start of this segment to the beginning of the procedure being called. A call to an inner ring is validated by the processor with the information from the gate array. The current segment is then changed to the new

segment and execution continues with the start of the called procedure. The information in the gate array is established by the operating system and is incorporated into the first page of the segment, known as page 0. This protection system allows the user program and operating system procedures to exist within the same logical program space and yet provide protection of operating system routines.

In AOS/VS the segments are used in the following manner. Segments 0 through 3 hold operating system routines. Segment 0 contains the kernel of the operating system which includes memory management routines, interrupt handlers, schedulers, and most device drivers. Privileged instructions can only be executed within segment zero. Segment 1 holds information about the process currently running. This includes address translation information in the form of page tables, device maps for I/O, and resource status. Segment 3 holds routines which validate system calls from user programs before passing the call onto the kernel in segment 0. Segments 4 through 7 are reserved for use by user programs. Initially a user program is loaded into segment 7. The user can then request that other routines or data be loaded from disk into rings 4 through 6 by making appropriate system calls. This segmented memory system constrains user programs to a maximum size of 512 Mbytes.

A segment base register value is associated with each segment and is used in translating logical addresses into physical addresses. The segment base register value is stored in page 0 of the segment when it

is not the active segment. The segment base register value for the active segment is kept in a program register. Page 0 of each segment is used to keep general information for the segment. This includes such items as the address of the segment gate array, starting addresses of different interrupt handlers, and pointers for manipulating the wide stack. The physical memory of the MV/8000 is split into a series of 2 Kbyte page frames. Translating a logical address into a physical address requires looking up the physical address of the page frame from a page table entry (PTE) in the Address Translation Unit. A logical to physical translation can require either one or two page table lookups depending upon the size of the program in that segment. A program of up to 1 Mbyte in size needs only a one level page table translation. Larger files require a two level table.

The following steps are performed to convert a 32 bit logical address into a physical address (assuming a two level translation). Figure 1.2 shows how the information in an address is used in the translation. The first three bits of the logical address reference a segment base register. The segment base register contains the physical address of the first page table in memory. A page table is a page of memory which holds 512 page table entries. In the case of a two level translation the first page table has entries which point to 512 other page tables. The physical address from the segment base register is used to reference the first page table. The next nine bits from the logical address form an offset into the first page table to refer to a particular page table entry. This first page table entry

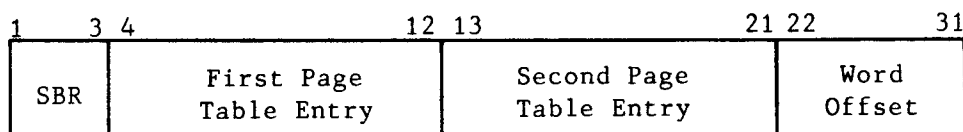


Figure 1.2
Addressing Form for Two Level Page Address Translation

contains the physical address of the second level page table needed next. This physical address points to the second page table and the second nine bits of the logical address form an offset into this second page table. The PTE referred to in this second page table holds the physical address of the page of memory needed. The remaining 10 bits of the logical address form an offset to the particular word of the page, the word to be addressed. The action of translating a logical to physical address is performed by the Address Translation Unit automatically with each memory reference.

The MV/8000 uses a demand paging strategy to bring the pages of an active process into memory and a least recently used (LRU) algorithm to determine which page to replace in the working set. A process, when it becomes active, is assigned, by the operating system, a working set size which is expressed in the form of a working set size and a working set minimum and maximum (WSS, WSSMIN, and WSSMAX respectively). In addition the process is assigned a page and swap file for use in paging out pages of the process or for swapping the process out. When the process initially begins, two pages are paged into memory. These pages are page zero of the current segment and the first page of the process known as the PC page. As the process executes each new page needed will generate a page fault that will

cause the operating system to read the new page from disk and add it to the working set. Once the working set size reaches WSSMIN the working set cannot be lowered below that value unless the process is being swapped out of memory. The working set size cannot surpass WSSMAX. Once the working set size of a process reaches WSSMAX the process will page against itself using the LRU algorithm. The LRU algorithm is implemented by the use of a referenced and modified bit attached to each page frame in memory. On every major clock interrupt the hardware sets the referenced bit of all pages to zero. When a page is referenced the referenced bit is set to one to indicate that it has been recently used. If a page is modified the modified bit is set to one to indicate that the page must be written to disk before the page frame can be reused for another page.

The paging activities of a typical process are shown in figure 1.3. When a process generates a page fault the fault will be resolved using the following general outline:

- Check to see if the working set size is at WSSMAX. If so a page must be released before the new page can be added to the working set. Determine which page to release from the working set using the LRU algorithm. If this released page has been modified first write it out to the paging file. Release this page by placing it on the LRU list.
- Scan the LRU list for the page to be added to the working set. If the page is on the LRU list add it to the working set and set the referenced bit. This action is known as a logical page fault for it does not result in a page being read from disk.

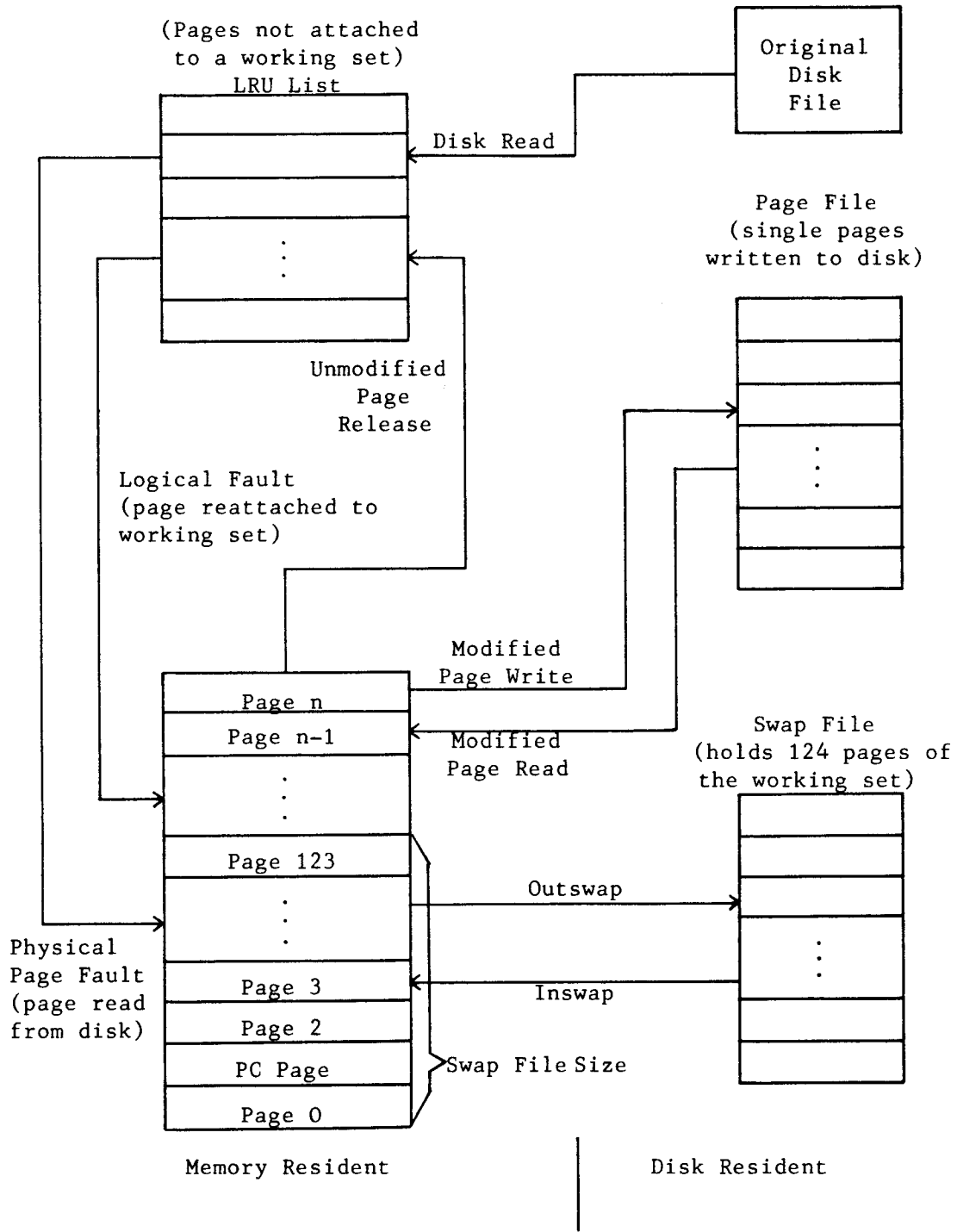


Figure 1.3
AOS/VS Paging Activities

- If the page is not on the LRU list then it must be read from disk. First check the paging file for the page. If the page is there then read it from the paging file. If the page is not in the paging file then read the page off disk from the original disk file. This action is known as a physical page fault because a page is read in from disk.

Before a page can be read in from disk there must be a free page frame on the LRU list to hold the page. There is always the possibility that a free page will not exist on the LRU list. In these circumstances the paging algorithm must reallocate pages from the working set of other processes for use with the current process needing the page. The decision of who to steal from (a technique known as page stealing) is performed as follows:

- Scan the working set of blocked processes and remove any unreferenced pages from their working sets and place the pages on the LRU list.
- If more pages are needed (and enough could not be obtained from above) then remove referenced pages from blocked processes, writing modified single pages into the paging file. Place the released pages onto the LRU list. Remember the working set cannot be lowered below WSSMIN without swapping the process out of memory.
- If more page frames are still needed then swap blocked processes out of memory. This is accomplished by first paging out single pages, if modified, into the paging file until the working set is reduced to 124 pages. At this point swap the remaining pages out into the swap file. All the released pages are placed on the LRU chain.
- If, after all of the above, more page frames are still needed then force the process needing the page frames to page against itself using the LRU algorithm.

Before a swapped out process can be swapped back in it must first get a reallocation of memory from the operating system based upon its process priority. The contents of the swap file are paged as a group

back into memory after obtaining the necessary number of free page frames. Any additional pages in the working set are demand paged back into memory.

The operating system will modify the working set size of a process every major clock cycle. If the process has generated too many faults over the last cycle then the working set size is increased. Unreferenced pages are removed from the working sets of processes which have few page faults to help keep a number of free page frames on the LRU list.

1.4 PROCESS SCHEDULING

The MV/8000 supports three types of processes and uses a priority structure to schedule processes for execution. The three classes of processes are resident, preemptible, and swappable. The three classes are used to determine the priority of a process for gaining a memory allocation in order to become a process eligible for execution. Figure 1.4 displays the different states of a process. To execute, a process must first become memory resident by receiving a memory allocation based on its process type. The process next gains the Central Processing Unit based upon the CPU priority of the process. Resident processes have the highest priority for memory. Once a resident process has gained memory it cannot be swapped out until the process has executed to completion. Preemptible processes have the next highest priority level. A preemptible process will gain a memory allocation before a swappable process but can be forced out of memory by the appearance of a resident process or a higher priority

preemptible process. Swappable processes have the lowest priority. They gain memory only when there are no other resident or preemptible processes to be run.

A process at its initial creation is disk resident and ready to be run once it gains memory and the CPU. A memory allocation is granted a process based upon its process type and CPU priority. In

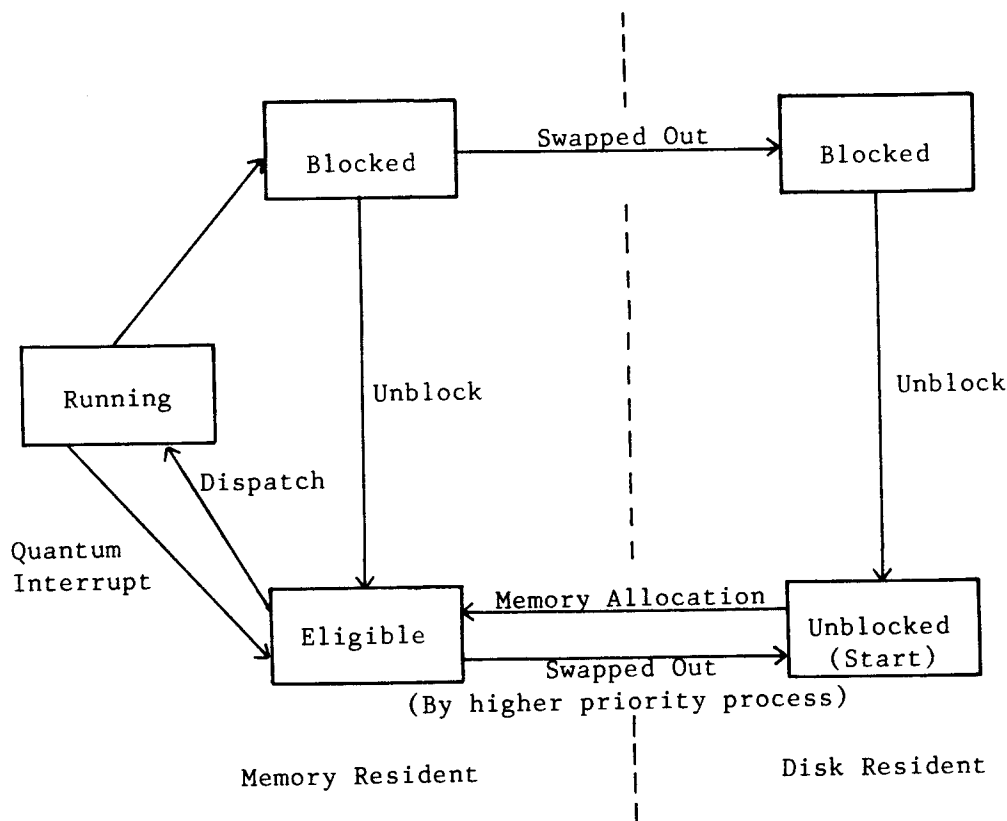


Figure 1.4
Process States in AOS/VS

deciding who gets a memory allocation the operating system will choose the highest priority resident process. Once resident processes have

been satisfied the operating system will move on to allocate memory for the preemptible processes and swappable processes in that order. Once a process gains memory it is considered an eligible process. It now contends for the CPU based upon its CPU priority. Priority factors for the different types of processes supported under AOS/VS are given below:

Process Type	Priority
Resident	(high) 1 - 255
Preemptible	(high) 1 - 255
Swappable	(high) 1 - 3

Process are dispatched to the CPU in the order of resident, preemptible, and swappable with the highest priority process of each group being first. When resident and preemptible processes gain the CPU they will execute to completion unless there are several processes of the same priority level. In this case the group of processes will execute in a round robin fashion with each receiving an equal time slice. The time slice is determined by dividing the major time cycle by the number of processes at this CPU priority level. The process time slice is reevaluated on every major clock (rescheduling) interrupt.

Swappable processes are handled differently. For several swappable processes of the same priority the operating system determines the time slice granted an individual process based upon the length of time the process has already been executing. New swappable processes are given a small time slice in the belief that this is a

simple request and will finished in one slice. If the process uses the whole time slice the time allotted to the process is increased on the next scheduling cycle. However, this increase in time slice will cause the scheduler to run the process less often. The length of a time slice for a swappable process is governed by the time slice exponent which can vary from 1 to 7. When the time slice exponent is one a process will receive the smallest possible time slice. When the time slice exponent becomes 7 the processes receives the longest time slice. The time slice exponent is incremented by one when the process uses its complete time slice. The length of a time slice will vary based upon the number of processes of this priority level on each rescheduling interrupt.

These different process types and priorities were provided to allow a system manager to configure the machine for the proper working environment, be it interactive program development or real time system control. Resident processes are generally used in real time applications where it is desirable for the process not to be swapped out if a higher priority process comes along. Swappable processes were designed for use in an interactive environment allowing short requests to finish quickly. Preemptible processes are a cross between the two. Preemptible processes compete for the CPU on the same basis as resident processes, but can be swapped out to memory if the space is needed by a higher priority process.

Once dispatched to the CPU a process executes until either it reaches completion, a time slice (quantum) interrupt occurs, an I/O

instruction is executed, or the process is explicitly blocked. When either an I/O operation or explicit block occurs the process leaves the CPU and enters a blocked, memory resident state. As soon as the I/O request finishes or the process is explicitly unblocked it moves from the blocked state to the eligible state. We should note that a resident process cannot be explicitly blocked since it cannot be swapped out of memory until completed. When a process enters the blocked state (preemptible and swappable only) it may become necessary for the operating system to reclaim the page frames being used by the blocked process. This reclamation may result in the process being swapped out to disk. When this happens the process remains in the blocked, disk resident state until it is unblocked. Once unblocked the process returns to the unblocked, disk resident state and must again compete for memory and the CPU based upon its process type and CPU priority.

The scheduling of memory allocations and CPU dispatching is performed through the use of two queues. The first of these is the core manager which is responsible for granting memory allocations and the second is the eligible process queue which is used by the scheduler to determine which process to dispatch next. The structure of the two queues is given in Figure 1.5. The core manager has an entry at the front of the eligible process queue. This is done so that when a process is created it will receive a memory allocation (if possible) before the next process is dispatched. An entry for the new process is placed into the core manager's request queue and a bit in the entry for the core manager on the eligible process queue is set.

Eligible Process Queue Structure:

Core Manager Process	Resident & Preemptible Processes in FIFO Order. Priority range 1 - 255.	Swappable Processes in FIFO Order. Priority Range 1 - 3.	B L U O S O Y P
----------------------------	--	--	--------------------------

Request Queue Structure for Core Manager:

Resident Processes in FIFO Order.	Preemptible Processes in FIFO Order.	Swappable Processes in FIFO Order.
---	--	--

Figure 1.5
AOS/VS Scheduling Queue Structures

When the operating system looks for the next process to dispatch it checks the bit in the core manager entry. If the bit is set then the core manager is run to allocate memory for the processes in its request queue. If there are no entries in the core manager request queue then the scheduler will dispatch the highest priority process on the eligible queue after the core manager entry.

The MV/8000 has the capabilities for executing both interactive and batch processes. To execute a program as a batch job the user submits the program to a batch input queue. The batch input queue is a single queue of all batch jobs arranged in a priority fashion where 0 is the lowest queue priority and 255 is the highest queue priority. The operating system will support the creation of anywhere from zero

to seven batch streams. A batch stream is essentially another user to the system where the user is being brought off the batch input queue. The batch streams have the ability to be selective about what priority jobs they accept from the batch input queue. A batch stream when generated is given an upper limit on the batch queue priority that it will accept. When the batch stream is ready for a new job it scans the batch input queue and selects the highest priority job that it is allowed to execute based on the queue priorities assigned to it. Jobs are removed from the queue in a first in, first out fashion. This mechanism allows the user to select which batch stream his batch process should run in through selecting the proper batch queue priority. Each batch stream can be a process running at a different CPU priority thus allowing the user to selectively choose different CPU priorities based upon the chosen batch stream.

2. WORKLOAD CHARACTERIZATION

2.1 BACKGROUND

The Eclipse MV/8000 system at NCSU was installed in Leazar Hall during September, 1982. The facility included the hardware described in Chapter 1 along with room for 100 user workstations, 2 operators, and 2 programming consultants. Initially 80 user terminals were installed. The majority of these terminals were Televideo Intelligent I terminals reconfigured to act like a Data General D200 terminal through a change of the internal EPROMs (erasable programmable read only memories). The remaining few were Data General D200 terminals. This facility was constructed to support the two undergraduate introductory programming courses, CSC 101 and CSC 111. At the time there was insufficient computing resources to support the needs of the research and teaching environments on campus. CSC 101 is an introductory course in Pascal programming required of all computer science majors. CSC 111 is a first course in FORTRAN 77 programming geared for students primarily in engineering and the physical sciences. CSC 111 and 101 carry two and three semester credit hours respectively.

Students in these two classes began using the machine during October 1982. Figures for class enrollments during the first four semesters of operation are presented in Table 2.1. These figures provide a feel for the resource demand that had to be met. To provide a proper perspective, the total students in the fall semester of 1982

Semester	CSC 101	CSC 111	Total
Fall 1982	746	769	1515
Spring 1983	361	411	772
Summer 1983	181	187	368
Fall 1983	740	738	1478

Table 2.1
Enrollment Figures for Introductory
Computer Science Courses

constituted over 40% of the computer science department's teaching load for that semester. Enrollment figures for the fall are typically higher than the spring due to large number of entering freshmen who are supposed to take one of these courses during their first semester. During the summer school sessions only one class of each course is taught during the ten week session. Both courses place a heavy emphasis on programming with students in CSC 101 typically writing five programming assignments per semester and students in CSC 111 writing four programming assignments. The assignments vary in length as the complexity grows during the semester. Random measurements of student programs indicated that the average program contains approximately 150 lines of code and 100 lines of documentation. Typical programming assignments would include checking account balancing, scatter plotting on a line printer, least squares analysis, telephone list sorting, and word counting or manipulation.

During October, just after the machine was first installed, the working environment was as follows. Each user was given 500 Kbytes of disk space for storage of materials. Files were edited using Data General's full screen, line oriented text editor known as SED (Screen Editor). Programs, once edited, were then compiled, linked, and

executed interactively. This heavy interactive use proved to be disastrous to response time. With approximately thirty students logged on, it was not uncommon to hear complaints that it took over one hour for SED to open an old file for editing or that it took 45 minutes to complete the compilation step in running a program.

A concentrated effort was put forth to resolve the problems. First it was discovered that all the new user accounts had been created on the large 604 Mbyte disk drive (directory :UDD) which had to share a disk controller with the other 147 Mbyte Winchester disk drive. This disk controller was doing well over 50 percent of the total disk traffic. The contention at the disk controller was helping to slow the system down. The solution was to distribute the disk traffic across the four disks arriving at the file distribution described in Chapter 1. Distributing the user directories across the disks changed the workload to where there was less than 1 percent of disk requests blocked on the Winchester disk controller. This controller was left handling about 35 to 40 percent of the disk traffic with no evidence of request backup. There was some question about having one third of the user accounts (on directory :UDD) and the operating system files (on the root directory) all going through the same disk controller due to the potential for heavy use. It was felt that the Winchester controller being slightly faster could handle the traffic and in addition would allow the operating system files to reside as physically close to the CPU as possible.

The next discovery was that the response to user requests

increased whenever the machine began swapping processes. It turned out from direct observation that the operating system AOS/VS worked in essentially two modes; one with the swapper active and one with it inactive. The operating system decides whether to do swapping based upon the number of page frames on the LRU page list. If the number of free page frames begins to dwindle the swapper engages and at regular intervals will steal unreferenced pages from other processes to attach to the LRU list. The overhead involved in page stealing is significant and the unfortunate part was that the swapper did not turn off automatically. The system had to be rebooted for the swapper to be disengaged. The reason for the large amount of swapping was the presence of so many interactive compilations and links, each of which used large working sets. To help with this problem three batch streams were set up to perform program compilation, linking, and execution. The students were forced to run programs through the batch stream thus reducing the number of interactive processes on the system. The three batch streams were set to higher CPU priority and had CPU time limits of 60 seconds total for a compile and link, and 30 seconds for program execution.

The results of the changes were improved user response times. The redistribution of the accounts across the disks kept disk I/O from being a bottleneck. The three batch streams helped to keep the swapping system from engaging needlessly, thus releasing more of the CPU time for the users. The use of batch streams required students wait some time for program turnaround. Usually the delay was much shorter than the turnaround time for the same interactive work.

Problems remained. Turnaround time for batch jobs was roughly 2 minutes on a lightly used system and went up from there to approximately 1 hour as the number of users and jobs in the batch queue increased. The editor was still taking anywhere from 5 to 25 minutes to open an old file for editing.

More study revealed that the continued poor response time could be blamed directly on the functioning of the operating system. AOS/VS required that user console (logon) processes be initiated as swappable. On the NCSU system all user processes were initially created as swappable processes with a CPU priority of 2. A feature of AOS/VS is that swappable processes have built into their scheduling characteristics a biasing function to give preference towards short CPU requests. This biasing works as follows. When a swappable processes is first created it is given a short time slice. If a process uses the complete time slice without blocking for I/O or without finishing the request its time slice is increased by the operating system. However, as the time slice of the swappable process increases it is scheduled for execution less often.

The problems presented by this scheduling discipline became obvious when trying to edit a file. The SED editor begins by reading the complete file to split it into 1023 line pages. Reading a file of any reasonable size requires several time slices to complete the task. The result is that as the editor begins, the time slice received by the process is quickly increased to the maximum and is scheduled less often. In addition the edit process must compete with

other batch and interactive edit processes in the system; both old and new. The end result is that the edit process executed so rarely that the process appears to be indefinitely postponed.

The solution for this problem was provided by the system manager. He modified the operating system code over the Christmas break to allow console processes to be initially created as preemptible. Under the scheduling discipline for AOS/VS, preemptible and resident processes run to completion when given the CPU unless there is more than one eligible process of the same CPU priority. With several processes of the same priority the scheduler divides the next scheduling cycle evenly among the eligible processes of that priority, with each receiving the same time slice. Note there is no biasing of preemptible processes and this simple round-robin scheduling requires less overhead than the previous biasing method. Using the modified operating system each user process was created as a preemptible process with a CPU priority of 50. This change provided a major improvement in user response times. The elimination of the biasing and its associated postponement improved performance to where initialization of the editor was typically under 20 seconds, even under heavy use.

This was roughly the environment of the MV/8000 system during the period of study from November of 1982 through July of 1983. We note that the batch method of program submission was continued to help prevent process swapping even after the change over to using preemptible processes. The problem of process swapping was rare after

the first set of changes made in October of 1982 and virtually disappeared with the modifications made over the Christmas break.

2.2 DATA COLLECTION METHODS & PROBLEMS

Before starting the process of data collection for workload characterization an attempt was made to choose the response measures of interest and the information needed to develop them. The response measures needed was the turnaround time of a batch job, given the current user load and the expected CPU time of the job, along with the response time of a request while in the editor. The vast bulk of the student work being done involved editing and running programs. It was necessary to determine the average load on the system to determine the peak hours. Additional information was required about the amount of interactive and batch work being done, along with parameters related to this work including mean service times, I/O blocks transferred, and pages transferred on a per process basis.

There were three utilities available that could be used for data collection. The first was the Process Environmental Display (known as PED)[15]. PED is an online monitor that will show the state of the system and all the processes within the system at regular intervals. The user can control the cycle time, the range of processes viewed on the screen, and the parameters to be viewed while the program is running. Monitered items can range from CPU time to physical page faults to the number of I/O blocks transferred. PED is capable of writing its output into a file. It writes a copy of the screen output

into the file at the end of each update cycle. The appearance in the file is that of system snapshots pasted one behind another.

The second utility is the Monitor Program (known as MOP). MOP displays online overall status of the machine rather than information on individual processes like PED. The monitor is arranged in screens which can be requested with the function keys. There are screens which show memory activity, CPU activity, paging activity, I/O activity, and scheduling activity amongst others. MOP updates the information on the screens at regular intervals. It allows for the viewing of some parameters measured over the current cycle as well as cumulative information since the system was last initialized. MOP will dump its information into a file in the form of snapshots at regular intervals. MOP can run some subsidiary programs two of which are of interest. WHERE is a program that provides information about cumulative and current CPU usage along with the percentages of CPU time used by each active process. This program could be very helpful in monitoring a benchmark during execution. The second program, DISCO, provides disk usage information on a per drive basis. The information presented includes the number of disk accesses, the number of blocks read and written, the percentage of usage, the length of the disk request queue, and the average seek distance per disk access. This program is useful in measuring the disk traffic during a benchmarking operation.

The third utility is the SYSLOG/REPORT system, otherwise known as the system accounting log which is provided as a part of the operating

system[15]. The MV/8000 automatically maintains an accounting log (known as the SYSLOG) which is updated with information about a process at its termination. Data General provides a utility, called REPORT, that can process a SYSLOG file. REPORT can provide information on a per user basis for accounting or on a per process basis if individual process activity is needed. The information kept in an accounting log is shown in figure 2.1. The figure is a sample of the output REPORT produces on a per process basis known as a process termination dump.

Moving across from left to right a line of figure 2.1 contains the following information. First is the date and time of process termination when this record was written into the log. Next is a short description of the process given as the username appended with additional identifying information. Next is the elapsed real time the process existed in hours, minutes, and seconds. Next to this is the CPU time presented in the same format as elapsed time. The last two items are the number of I/O blocks and the pagesecods used by the process.

The limitations of these three utilities presented a problem. The most useful information could be obtained from PED or MOP, but their use would impose limitations. First, is the use of the monitors themselves. Being online monitors they contribute directly to the overall load being experienced on the system thus affecting service times. Second, is the problem that the monitors could not be used to watch the system over long periods of time. Due to their online

PROCESS TERMINATION DUMP

	PROCESS NAME	ELAPSED TIME	CPU TIME	I/O BLOCKS	PAGE SECS
1-APR-83	9:48:03	*** SYSLOG STARTED ***			
1-APR-83	9:48:12	J_DEVER:042	0:01:10	0:00:01.941	40 124
1-APR-83	9:48:12	F_DEFFENBAUGH:0	0:00:15	0:00:01.245	55 148
1-APR-83	9:48:14	F_DEFFENBAUGH:0	0:00:14	0:00:00.079	0 5
1-APR-83	9:48:15	F_DEFFENBAUGH:S	0:00:23	0:00:00.356	43 22
1-APR-83	9:48:15	D_TAYLOR:014	0:00:52	0:00:58.318	1224 316
1-APR-83	9:48:17	D_TAYLOR:STREAM	0:01:18	0:00:00.390	53 25
1-APR-83	9:48:17	R_TAYLOR:CON56	1:04:33	0:00:21.121	1164 1478
1-APR-83	9:48:24	K_JOHNSON:CON50	0:37:24	0:00:04.961	315 325
1-APR-83	9:49:18	D_TAYLOR:030	0:02:16	0:00:01.021	25 65

Figure 2.1
Sample Process Termination Dump

nature, use of the monitors in this manner would necessitate building and monitoring benchmarking programs. To develop an analytical model of the system that represented reality required information about the actual workload of the system over long periods of time. Thus it was decided to process the system accounting logs to obtain the workload information.

The first step in processing the accounting logs was to put the information into a usable intermediate form for later processing. SYSLOG files are arranged into a series of dynamic records arranged to use as little storage as possible. Realizing that a need might exist to process the information through more than one program and that all the records contained within the log files were not needed, the information was converted into a readable intermediate. This form was the process termination log produced by REPORT. From this readable log a program called SYSLOG_REPORT would generate summaries on a

monthly basis. The method was to have REPORT build a process termination log using only records from the month of interest. The system log files usually covered several days at a time and often crossed over from one month to the next. The process termination dumps produced by REPORT were to be appended together to produce one monthly intermediate log file.

Using this method resulted in further problems. REPORT was an old Data General utility designed for use on the 32 Kbyte AOS Eclipse systems keeping all the summary information in memory during execution. As a result REPORT did not have arrays large enough to handle more than about 300 users. We had over 1000. Rather than create a new REPORT a program was written that cut the large accounting log files into smaller files of about 6000 records. REPORT would process these smaller files and the process termination dumps would be appended together to get a monthly accounting file. Besides just stripping out records, this program also threw out any invalid records. When the machine "crashes" without warning many files are left open on the disks and must be closed before the system can be restarted. The file closing utility, FIXUP, closes the system accounting file by continuing the last record out to the end of the sector (a maximum of 512 bytes), making an extremely long record. Such records could not be trusted for accuracy and were removed during the splitting process. This scheme worked successfully. It was implemented by having the splitting program run in a separate batch stream at low priority.

The first attempt at obtaining workload measures involved mainly trying to determine the number of users on the system, service times for batch and interactive processes, length of logon times, and ranges of CPU service times. The most difficult problem involved in analyzing a log file was determination of which processes were related to terminals, batch jobs, and edit sessions. To resolve this required an understanding of how the system at NCSU was managed and how system accounting logs are kept. The MV/8000 is geared around the hierarchical process tree discussed in chapter one. Thus when a person logs onto the system a process is created for that terminal. Such processes can be identified in the accounting log by the presence of the word "CON" after the username in the process name field (refer to figure 2.1). Such a process is known as a console process to distinguish it from an interactive process. The two digit number following the word "CON" is the terminal number on the system. When editing a file using SED, a second process is created in the system subordinate to the console process. An interactive processes can be identified in the accounting log by the three digit process identification number (PID number) following the username. An interactive process does not necessarily have to be a edit session. On the NCSU system, however, all program compilations and executions have been delegated to the batch stream. Thus the vast majority of interactive processes are edit sessions. In this analysis all interactive processes are treated as such. This leaves only the batch jobs.

A batch stream is essentially a console process running the Command Line Interpreter (CLI) except that its command lines come from

a file in the batch stream queue. When a job arrives in the batch queue the stream will take the job and then create three son processes in strict sequential order to compile the program, link the program, and execute the program. The idea of the structure is presented in figure 2.2.

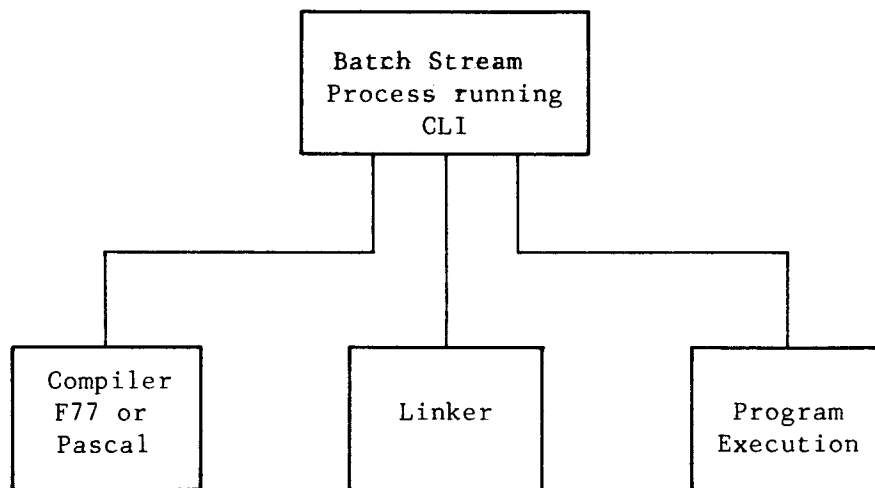


Figure 2.2
Hierarchy of Batch Jobs on the MV/8000

The father process, the batch stream process, is easily identifiable in the accounting log by the use of the keyword "STREAM" following the username. The three son processes show up in the log like other interactive processes with no indication that they belong to part of a batch job. A method was developed to pick out batch jobs from the logs. This method took advantage of the fact that there are never more than three subordinate processes in a batch job and that by being subordinate to the father process all must terminate first, thus

appearing in the accounting log prior to the stream process.

The method is as follows. Whenever a process appears in the log with a three digit PID number, that process is considered an interactive process by default. Also each termination record, when encountered, is placed at the front of a circular queue of the last fifteen processes. Whenever a batch stream termination appears it is counted as a batch job. One then scans back up the queue one process at a time looking for interactive processes under this username that really were batch processes. The batch stream termination record has encoded in it the elapsed time of that process. The sum of the elapsed times of the interactive processes created by this batch process must be less than or equal to the elapsed time of the stream process due to their sequential execution. Therefore in scanning backwards up the queue, we assume each process under the same username is associated with the batch job until either the elapsed times sum larger than the elapsed time of the stream process or a total of three interactive processes have been encountered. Totals kept on interactive and batch processes are then rearranged.

As an example of this method consider the three records for the user F_DEFFENBAUGH in figure 2.1. The arrangement of these three process terminations so close together suggests that they are part of the same batch job. Close inspection of the elapsed times indicates otherwise. Notice that the elapsed time of the batch stream process, indentified by the ":S" after the username is 23 seconds. The interactive process immediately above this one has an elapsed time of

14 seconds, indicating that it was a part of the batch job. The next process up has an elapsed time of 15 seconds and being interactive it appears to belong to this batch job. However, the combined elapsed times of the two interactive processes is 29 seconds. This is too long for both to be part of the same batch job since they have to be created and terminated in strict sequential order. Thus we can state that the first interactive process is an edit session and the next two processes are part of the batch job.

A final problem was to interpret the measure "pagesecods" due to conflicting definitions in Data General manuals. The AOS/VS Programmer's Manual[14] defined the measure as pages over CPU seconds giving it the appearance of a paging rate. Closer inspection of pagesecod values, using this definition, showed that the number of pages being transferred per process easily exceeded 1000 pages. Another definition in the AOS/VS Command Line Interpreter User's Guide[16] gave the measure as the number of pages multiplied by the CPU time of the process. This definition appeared to be more reasonable except that it does not state whether "pages" meant pages physically faulted, logically faulted, or both. PED was used to try and verify the pagesecod value in the following manner. With PED it is possible to monitor the logical page faults, physical page faults, working set size, cpu seconds, and pagesecods all at once. It seemed reasonable that by monitoring several programs one should be able to determine the number of pages transferred by a process from its pagesecod value and this number should agree with one of the page fault values or their sum.

The results of this monitoring process were disappointing. They demonstrated that for processes with large numbers of logical page faults the number of pages transferred (computed from the pagesecounds measure) did not agree well with any of the monitored paging measures. However, processes which did little logical page faulting had a close agreement with the working set size of the process. Initially it appeared that this correspondence with the working set size was useless since the measure could include both physically paged and logically paged pages. Further inspection and monitoring revealed that the pagesecound measure seemed to be a measure of the number of page frames used by a process multiplied by the CPU time; thus making it a measure of memory activity rather than paging activity. The result is that for a process which does lots of logical page faulting the working set size for the process will be increased by the operating system above the actual number of used pages. Processes which do little logical page faulting maintain a working set size close to the actual number of page frames used by the process. It turns out that on the NCSU system the only processes which do lots of logical page faulting are those associated with the operating system. The typical jobs that users run, due to their short execution times, tend to get into memory, fault in all the pages they need and finish before any of their page frames are stolen by the operating system for use with another process. This still leaves a problem with the relationship between physical page faults and the working set size. It was decided to proceed with data collection under the assumption that the number of physical page faults was proportional to the

working set size.

2.3 CHARACTERIZATION RESULTS

With the data interpretation problems resolved SYSLOG_REPORT was used to provide the first information about the system workload. Not having any prior data about the usage of the system it was arbitrarily chosen for SYSLOG_REPORT to provide the following information:

- number of processes, service time per process, arrival rate, number of I/O blocks transferred per process, and pagesecods per process given on a per hour basis averaged over the course of a month.
- number of batch, interactive, and console processes totaled by the hour over the month.
- number of console processes which fell in the elapsed time ranges: 0 - 30 minutes, 30 - 60 minutes, 60 - 120 minutes, over 120 minutes. This was totaled by the hour over the month.
- number of batch and interactive processes which had CPU times in the ranges: 0 - 1 second in 0.2 second intervals, 1 - 5 seconds in half second intervals, 5 - 12 seconds in one second intervals, and greater than 12 seconds.

The data analyzed were the accounting log files over the period from the start of November 1982 through July 1983. Days when the computing facility was closed (like over the Christmas holidays) were not included in the intermediate accounting file generated for processing by SYSLOG_REPORT. Specifically three periods of time are not represented in the accounting logs. The log covering the period from 11/20/82 through 11/30/82 was deleted before this project began. The log covering the period 12/22/82 through 1/1/83 is omitted due to the facility being shut down for the Christmas holidays. Finally the

accounting log covering the period from 4/12/83 through 4/30/83 was accidentally deleted during the processing of the data into an intermediate file.

The useful results from SYSLOG_REPORT follow. The number of user console logoffs per hour showed that the most consistent period of heavy use occurred during the hours of 3 - 5 pm. Usage information for console processes is presented in table 2.2. The batch and interactive processes provided the usage information given in Table 2.3.

The only noticeable trends from the data indicated that over the study period the CPU time of batch processes increased. Likewise there was an increase in the CPU times of interactive processes. Unfortunately, this was the bulk of the usable information that could be obtained from this analysis due to additional problems discovered in the collection of data.

The months of May and July did not contribute information that was "average" over the heaviest student use periods. May lies between the end of spring semester and the start of summer school. Exams are held during the first two weeks of May with a week and a half break before the start of summer school the last week of the month. Thus the only users over the period were a few people running large programs that did not represent the normal student workload under study. July presented a similar problem. Students used the machine during the month, but in addition there were several students working on research problems and the processing of the accounting logs was going on at the same time. This resulted in the mean service times

CPU Usage:

Percentage of Console Processes	CPU Time Used (seconds)
37.47%	0.0 - 1.0
39.37%	1.0 - 5.0
23.16%	over 5.0

Logon Time:

Percentage of Terminal Sessions	Elapsed Time Used (minutes)
57.58%	0 - 30
13.40%	30 - 60
14.30%	60 - 120
14.72%	over 120

Summary of User Console Processes Over
Study Period from SYSLOG REPORT.
Table 2.2

Batch Load:

Percentage of Batch Processes	CPU Time Used (seconds)
87.76%	0.0 - 0.6
17.24%	over 0.6

Interactive Load:

Percentage of Interactive Processes	CPU Time Used (seconds)
39.67%	0.0 - 1.0
40.38%	1.0 - 5.0
19.95%	over 5.0

Summary of Process Types Over Study
Period from SYSLOG REPORT.
Table 2.3

being very high as compared to the other months. It was decided to omit these two months from future analyses on the pretext that they were not representative of the workload being measured.

Other problems involved the proper measurement of time. A feature of SYSLOG_REPORT was that it attempted to account accurately for the available time on the machine by comparing dates and times recorded with termination records in the accounting log. The scheme was based upon the initial time and date being correctly entered into the system log when the computer was rebooted. There were several occurrences within the log files where the time and date were entered incorrectly causing the program to keep counters incorrectly, thus altering the results. A second problem had to do with measuring the number of users logged on. To save processing time the number of users logged on over an hour was measured by counting the number of console process terminations during the hour. Closer inspection revealed that this was not an accurate measure since a person could work on the system over the bulk of a previous hour and then be counted at logoff time in the current hour where the least amount of time was used.

The failures of SYSLOG_REPORT led to the development of two new accounting log reporting programs. The first, LOGSPY, was designed to replace SYSLOG_REPORT providing overall process information that would be helpful during the modelling phase. Some information from SYSLOG_REPORT was used in the construction of LOGSPY. In particular it was assumed that the peak hours of 2 - 5 pm. would hold true.

LOGSPY was designed to provide averages by the day of the week through the month and an average for the whole month. Specifically LOGSPY provided the following information:

- The average number of users logged on at each half hour interval was determined for each day of the week and for the whole month.
- The average number of interactive and batch processes active in the system (both blocked and running) at each half hour interval was determined for each day of the week and for the whole month.
- Average values for the number of processes, service times, arrival rates, I/O blocks transferred, pagesecods, and CPU utilization were determined individually for batch, interactive, and all processes combined (including console processes). This information was given in table form by the day of the week and for the whole month.
- A table of the same information mentioned immediately above was generated for process terminations during the hours of 2 - 5 pm.

LOGSPY did not attempt to correct for any down time the computer might experience. If the machine was up for any portion of a day the program assumed that the system was available for the complete day. Determining the intervals when a user was logged on required computing the starting and ending time of a console process (this includes wraparounds from one day to the next) and then updating the appropriate half hour counters.

Processing the accounting log files with LOGSPY provided much more accurate information about system workloads with the months of May and July omitted. Figures 2.3 through figure 2.9 show graphs of the user workload averaged by month over the study period. Figure

2.10 is a graph showing the user workload averaged over the seven months in study. The graphs verify that two peak times occur. One is the range from 2 to 5 pm and the other is from 8 to 10 pm. Figure 2.10 clearly shows that the larger peak is the 2 to 5 pm time slot previously selected. The graphs show the variations in student use during the study period.

The graphs for November and December show a heavier concentration throughout the day than the other graphs. This is because during these months interactive processes were still running as swappable and were taking large waiting times to open files for editing. Thus the students were having to work longer hours on the system to get their work finished. This in addition to the fact that the larger programming assignments come at the end of the course provide the reasons for the consistently heavier use throughout the day. January shows low usage because at the start of a semester it takes some time before the students begin receiving assignments that require working on the computer. The months of February, March, and April show a slow increase in usage representing the increase in complexity of programming assignments as the semester progresses. June shows a slight falloff from April because it is the start of the summer school session.

LOGSPY provided information about processes averaged for each day of the week and for the month. The averages for each day of the week had only slight variations indicative of the variations in daily student use. One interesting note is that Thursday and Sunday appeared

```

( 20) HOUR   .5:*****
( 16) HOUR   1.0:*****
( 11) HOUR   1.5:*****
( 11) HOUR   2.0:*****
( 11) HOUR   2.5:*****
(  9) HOUR   3.0:*****
(  7) HOUR   3.5:*****
(  7) HOUR   4.0:*****
(  7) HOUR   4.5:*****
(  6) HOUR   5.0:*****
(  5) HOUR   5.5:*****
(  4) HOUR   6.0:****
(  3) HOUR   6.5:***
(  3) HOUR   7.0:***
(  3) HOUR   7.5:***
(  4) HOUR   8.0:****
(  5) HOUR   8.5:*****
(  8) HOUR   9.0:*****
( 10) HOUR   9.5:*****
( 13) HOUR  10.0:*****
( 15) HOUR  10.5:*****
( 15) HOUR  11.0:*****
( 15) HOUR  11.5:*****
( 14) HOUR  12.0:*****
( 20) HOUR  12.5:*****
( 22) HOUR  13.0:*****
( 26) HOUR  13.5:*****
( 28) HOUR  14.0:*****
( 31) HOUR  14.5:*****
( 35) HOUR  15.0:*****
( 36) HOUR  15.5:*****
( 34) HOUR  16.0:*****
( 33) HOUR  16.5:*****
( 36) HOUR  17.0:*****
( 33) HOUR  17.5:*****
( 32) HOUR  18.0:*****
( 24) HOUR  18.5:*****
( 30) HOUR  19.0:*****
( 34) HOUR  19.5:*****
( 35) HOUR  20.0:*****
( 36) HOUR  20.5:*****
( 37) HOUR  21.0:*****
( 34) HOUR  21.5:*****
( 32) HOUR  22.0:*****
( 31) HOUR  22.5:*****
( 30) HOUR  23.0:*****
( 27) HOUR  23.5:*****
( 23) HOUR  24.0:*****

```

Average Users Logged on in November 1982
Figure 2.3

```

( 22) HOUR   .5:*****
( 24) HOUR   1.0:*****
( 20) HOUR   1.5:*****
( 21) HOUR   2.0:*****
( 20) HOUR   2.5:*****
( 19) HOUR   3.0:*****
( 18) HOUR   3.5:*****
( 14) HOUR   4.0:*****
( 15) HOUR   4.5:*****
( 14) HOUR   5.0:*****
( 14) HOUR   5.5:*****
( 16) HOUR   6.0:*****
( 15) HOUR   6.5:*****
( 15) HOUR   7.0:*****
( 14) HOUR   7.5:*****
( 15) HOUR   8.0:*****
( 12) HOUR   8.5:*****
( 15) HOUR   9.0:*****
( 16) HOUR   9.5:*****
( 19) HOUR  10.0:*****
( 18) HOUR  10.5:*****
( 18) HOUR  11.0:*****
( 19) HOUR  11.5:*****
( 19) HOUR  12.0:*****
( 19) HOUR  12.5:*****
( 19) HOUR  13.0:*****
( 22) HOUR  13.5:*****
( 24) HOUR  14.0:*****
( 26) HOUR  14.5:*****
( 26) HOUR  15.0:*****
( 27) HOUR  15.5:*****
( 27) HOUR  16.0:*****
( 27) HOUR  16.5:*****
( 25) HOUR  17.0:*****
( 24) HOUR  17.5:*****
( 23) HOUR  18.0:*****
( 19) HOUR  18.5:*****
( 23) HOUR  19.0:*****
( 20) HOUR  19.5:*****
( 23) HOUR  20.0:*****
( 24) HOUR  20.5:*****
( 23) HOUR  21.0:*****
( 22) HOUR  21.5:*****
( 23) HOUR  22.0:*****
( 24) HOUR  22.5:*****
( 23) HOUR  23.0:*****
( 21) HOUR  23.5:*****
( 21) HOUR  24.0:*****

```

Average Users Logged on in December 1982.
Figure 2.4

```

( 5) HOUR   .5:*****
( 4) HOUR   1.0:****
( 4) HOUR   1.5:****
( 4) HOUR   2.0:****
( 3) HOUR   2.5:***
( 3) HOUR   3.0:***
( 3) HOUR   3.5:***
( 3) HOUR   4.0:***
( 2) HOUR   4.5:**
( 2) HOUR   5.0:**
( 2) HOUR   5.5:**
( 2) HOUR   6.0:**
( 2) HOUR   6.5:**
( 2) HOUR   7.0:**
( 2) HOUR   7.5:**
( 3) HOUR   8.0:***
( 3) HOUR   8.5:***
( 4) HOUR   9.0:****
( 5) HOUR   9.5:*****
( 6) HOUR  10.0:*****
( 7) HOUR  10.5:*****
( 8) HOUR  11.0:*****
( 9) HOUR  11.5:*****
( 8) HOUR  12.0:*****
(10) HOUR  12.5:*****
( 9) HOUR  13.0:*****
(11) HOUR  13.5:*****
(11) HOUR  14.0:*****
(14) HOUR  14.5:*****
(14) HOUR  15.0:*****
(17) HOUR  15.5:*****
(15) HOUR  16.0:*****
(13) HOUR  16.5:*****
(11) HOUR  17.0:*****
(10) HOUR  17.5:*****
( 9) HOUR  18.0:*****
(11) HOUR  18.5:*****
(12) HOUR  19.0:*****
(12) HOUR  19.5:*****
(12) HOUR  20.0:*****
(11) HOUR  20.5:*****
(12) HOUR  21.0:*****
(12) HOUR  21.5:*****
(11) HOUR  22.0:*****
(10) HOUR  22.5:*****
( 8) HOUR  23.0:*****
( 7) HOUR  23.5:*****
( 6) HOUR  24.0:*****

```

Average Users Logged on in January 1983.
Figure 2.5

```

( 17) HOUR   .5:*****
( 15) HOUR   1.0:*****
( 14) HOUR   1.5:*****
( 12) HOUR   2.0:*****
( 10) HOUR   2.5:*****
(  9) HOUR   3.0:*****
(  8) HOUR   3.5:*****
(  7) HOUR   4.0:*****
(  7) HOUR   4.5:*****
(  6) HOUR   5.0:*****
(  6) HOUR   5.5:*****
(  7) HOUR   6.0:*****
(  6) HOUR   6.5:*****
(  5) HOUR   7.0:*****
(  6) HOUR   7.5:*****
(  7) HOUR   8.0:*****
(  8) HOUR   8.5:*****
(  9) HOUR   9.0:*****
( 11) HOUR   9.5:*****
( 13) HOUR  10.0:*****
( 15) HOUR  10.5:*****
( 17) HOUR  11.0:*****
( 18) HOUR  11.5:*****
( 18) HOUR  12.0:*****
( 19) HOUR  12.5:*****
( 23) HOUR  13.0:*****
( 22) HOUR  13.5:*****
( 25) HOUR  14.0:*****
( 31) HOUR  14.5:*****
( 30) HOUR  15.0:*****
( 32) HOUR  15.5:*****
( 32) HOUR  16.0:*****
( 30) HOUR  16.5:*****
( 28) HOUR  17.0:*****
( 27) HOUR  17.5:*****
( 25) HOUR  18.0:*****
( 24) HOUR  18.5:*****
( 24) HOUR  19.0:*****
( 25) HOUR  19.5:*****
( 26) HOUR  20.0:*****
( 26) HOUR  20.5:*****
( 25) HOUR  21.0:*****
( 28) HOUR  21.5:*****
( 28) HOUR  22.0:*****
( 25) HOUR  22.5:*****
( 23) HOUR  23.0:*****
( 21) HOUR  23.5:*****
( 18) HOUR  24.0:*****

```

Average Users Logged on in February 1983.
Figure 2.6

```

( 11) HOUR   .5:*****
( 10) HOUR   1.0:*****
(  9) HOUR   1.5:*****
(  7) HOUR   2.0:*****
(  6) HOUR   2.5:*****
(  5) HOUR   3.0:*****
(  5) HOUR   3.5:*****
(  4) HOUR   4.0:****
(  4) HOUR   4.5:****
(  4) HOUR   5.0:****
(  3) HOUR   5.5:***
(  3) HOUR   6.0:***
(  2) HOUR   6.5:**
(  2) HOUR   7.0:**
(  2) HOUR   7.5:**
(  3) HOUR   8.0:***
(  4) HOUR   8.5:****
(  6) HOUR   9.0:*****
(  8) HOUR   9.5:*****
( 10) HOUR  10.0:*****
( 14) HOUR  10.5:*****
( 16) HOUR  11.0:*****
( 17) HOUR  11.5:*****
( 18) HOUR  12.0:*****
( 20) HOUR  12.5:*****
( 20) HOUR  13.0:*****
( 20) HOUR  13.5:*****
( 19) HOUR  14.0:*****
( 23) HOUR  14.5:*****
( 21) HOUR  15.0:*****
( 24) HOUR  15.5:*****
( 24) HOUR  16.0:*****
( 23) HOUR  16.5:*****
( 21) HOUR  17.0:*****
( 20) HOUR  17.5:*****
( 18) HOUR  18.0:*****
( 18) HOUR  18.5:*****
( 18) HOUR  19.0:*****
( 20) HOUR  19.5:*****
( 22) HOUR  20.0:*****
( 22) HOUR  20.5:*****
( 23) HOUR  21.0:*****
( 24) HOUR  21.5:*****
( 23) HOUR  22.0:*****
( 21) HOUR  22.5:*****
( 19) HOUR  23.0:*****
( 17) HOUR  23.5:*****
( 14) HOUR  24.0:*****

```

Average Users Logged on in March 1983.
Figure 2.7

```

( 25) HOUR   .5:*****
( 23) HOUR   1.0:*****
( 21) HOUR   1.5:*****
( 20) HOUR   2.0:*****
( 19) HOUR   2.5:*****
( 18) HOUR   3.0:*****
( 16) HOUR   3.5:*****
( 14) HOUR   4.0:*****
( 12) HOUR   4.5:*****
( 11) HOUR   5.0:*****
( 10) HOUR   5.5:*****
(  9) HOUR   6.0:*****
(  8) HOUR   6.5:*****
(  7) HOUR   7.0:*****
(  7) HOUR   7.5:*****
(  7) HOUR   8.0:*****
(  9) HOUR   8.5:*****
( 11) HOUR   9.0:*****
( 15) HOUR   9.5:*****
( 19) HOUR  10.0:*****
( 27) HOUR  10.5:*****
( 30) HOUR  11.0:*****
( 34) HOUR  11.5:*****
( 33) HOUR  12.0:*****
( 36) HOUR  12.5:*****
( 37) HOUR  13.0:*****
( 39) HOUR  13.5:*****
( 38) HOUR  14.0:*****
( 42) HOUR  14.5:*****
( 48) HOUR  15.0:*****
( 49) HOUR  15.5:*****
( 49) HOUR  16.0:*****
( 46) HOUR  16.5:*****
( 42) HOUR  17.0:*****
( 40) HOUR  17.5:*****
( 40) HOUR  18.0:*****
( 38) HOUR  18.5:*****
( 41) HOUR  19.0:*****
( 43) HOUR  19.5:*****
( 43) HOUR  20.0:*****
( 45) HOUR  20.5:*****
( 47) HOUR  21.0:*****
( 46) HOUR  21.5:*****
( 42) HOUR  22.0:*****
( 39) HOUR  22.5:*****
( 36) HOUR  23.0:*****
( 33) HOUR  23.5:*****
( 29) HOUR  24.0:*****

```

Average Users Logged on in April 1983.
Figure 2.8

```

( 10) HOUR   .5:*****
(   8) HOUR   1.0:*****
(   6) HOUR   1.5:*****
(   6) HOUR   2.0:*****
(   5) HOUR   2.5:*****
(   5) HOUR   3.0:*****
(   5) HOUR   3.5:*****
(   5) HOUR   4.0:*****
(   4) HOUR   4.5:****
(   4) HOUR   5.0:****
(   3) HOUR   5.5:***
(   3) HOUR   6.0:***
(   4) HOUR   6.5:****
(   4) HOUR   7.0:****
(   4) HOUR   7.5:****
(   6) HOUR   8.0:*****
(   7) HOUR   8.5:*****
(  10) HOUR   9.0:*****
(  11) HOUR   9.5:*****
(  13) HOUR  10.0:*****
(  16) HOUR  10.5:*****
(  19) HOUR  11.0:*****
(  22) HOUR  11.5:*****
(  22) HOUR  12.0:*****
(  21) HOUR  12.5:*****
(  22) HOUR  13.0:*****
(  23) HOUR  13.5:*****
(  23) HOUR  14.0:*****
(  27) HOUR  14.5:*****
(  29) HOUR  15.0:*****
(  27) HOUR  15.5:*****
(  27) HOUR  16.0:*****
(  26) HOUR  16.5:*****
(  22) HOUR  17.0:*****
(  21) HOUR  17.5:*****
(  17) HOUR  18.0:*****
(  16) HOUR  18.5:*****
(  18) HOUR  19.0:*****
(  18) HOUR  19.5:*****
(  20) HOUR  20.0:*****
(  21) HOUR  20.5:*****
(  22) HOUR  21.0:*****
(  25) HOUR  21.5:*****
(  22) HOUR  22.0:*****
(  20) HOUR  22.5:*****
(  18) HOUR  23.0:*****
(  15) HOUR  23.5:*****
(  13) HOUR  24.0:*****

```

Average Users Logged on in June 1983.
Figure 2.9

```

( 16) HOUR   .5:*****
( 14) HOUR   1.0:*****
( 12) HOUR   1.5:*****
( 12) HOUR   2.0:*****
( 11) HOUR   2.5:*****
( 10) HOUR   3.0:*****
(  9) HOUR   3.5:*****
(  8) HOUR   4.0:*****
(  7) HOUR   4.5:*****
(  7) HOUR   5.0:*****
(  6) HOUR   5.5:*****
(  6) HOUR   6.0:*****
(  6) HOUR   6.5:*****
(  5) HOUR   7.0:*****
(  5) HOUR   7.5:*****
(  6) HOUR   8.0:*****
(  7) HOUR   8.5:*****
(  9) HOUR   9.0:*****
( 11) HOUR   9.5:*****
( 13) HOUR  10.0:*****
( 16) HOUR  10.5:*****
( 18) HOUR  11.0:*****
( 19) HOUR  11.5:*****
( 19) HOUR  12.0:*****
( 21) HOUR  12.5:*****
( 22) HOUR  13.0:*****
( 23) HOUR  13.5:*****
( 24) HOUR  14.0:*****
( 28) HOUR  14.5:*****
( 29) HOUR  15.0:*****
( 30) HOUR  15.5:*****
( 30) HOUR  16.0:*****
( 28) HOUR  16.5:*****
( 26) HOUR  17.0:*****
( 25) HOUR  17.5:*****
( 23) HOUR  18.0:*****
( 21) HOUR  18.5:*****
( 24) HOUR  19.0:*****
( 25) HOUR  19.5:*****
( 26) HOUR  20.0:*****
( 26) HOUR  20.5:*****
( 27) HOUR  21.0:*****
( 27) HOUR  21.5:*****
( 26) HOUR  22.0:*****
( 24) HOUR  22.5:*****
( 22) HOUR  23.0:*****
( 20) HOUR  23.5:*****
( 18) HOUR  24.0:*****

```

Average Users per Day Over Period from November 1982 - June 1983
Figure 2.10

to be days of heaviest use. It is likely that these trends came from the students working on Thursday so they could leave campus for the weekend. Sunday was the day many people used to catch up or get ahead on work for the coming week. A review of assignment due dates also showed that most were due on a Friday or Monday which also contributed to the heavy use on these days. Summaries of the monthly statistics related to processes are presented in Tables 2.4 through 2.6. These statistics are averaged over a complete 24 hour day. Table 2.4 is a summary of batch, interactive, and console processes combined while Table 2.5 is a summary of just the interactive processes and Table 2.6 is a summary of the batch processes.

The information presented in table 2.5 shows a trend towards longer interactive service times over the period. This was unexpected because the work the students perform varies during the semester but is fairly consistent from semester to semester. The service times of the batch processes in table 2.6 demonstrate the trend one would expect to see. Notice that the service times increase through the duration of a semester, but fall off from one semester to the next. The increase in interactive service times was likely the result of having several long interactive processes during the month which might increase the service times. The same process information was collected during the peak hours from 2 - 5 pm. The results are presented in tables 2.7 through 2.9. Table 2.7 is a summary of all process types combined while Table 2.8 and 2.9 present summaries of the batch and interactive processes respectively.

Date	11/82	12/82	1/83	2/83	3/83	4/83	6/83	Average
Average No. of Processes	8240.8	6413.4	2920.7	7158.6	5407.2	8573.8	5466.2	6311.5
Avg. Service Time per Process (Seconds/Process)	3.025	2.688	2.053	2.480	3.167	3.073	3.782	2.888
Avg. Process Arrival Rate (Processes/Second)	0.0954	0.0742	0.0338	0.0829	0.0626	0.0992	0.0633	0.0731
Avg. I/O Blocks Transferred per Process (Blocks/Process)	177.4	202.9	188.7	206.4	221.8	242.9	193.4	204.8
Avg. Page-seconds per Process	252.3	217.5	189.5	228.9	286.1	236.3	439.2	264.3
Avg. Pages Transferred per Process	83.4	80.9	92.3	92.3	90.3	76.9	117.8	90.6
Utilization of CPU	28.85	19.95	6.94	20.55	19.82	30.49	23.58	21.45
Avg. Users Logged On	20	20	8	18	14	29	15	18

Summary of Process Statistics over Period
from November 1982 through June 1983.
Table 2.4

Date	11/82	12/82	1/83	2/83	3/83	4/83	6/83	Average
Average No. of Processes	2792.1	1695.7	819.2	1960.3	1454.7	2618.9	2219.5	1937.2
Avg. Service Time per Process (Seconds/Process)	4.728	3.621	1.862	2.463	3.242	3.252	2.678	3.121
Avg. Process Arrival Rate (Processes/Second)	0.0323	0.0196	0.0095	0.0227	0.0168	0.0303	0.0257	0.0224
Avg. I/O Blocks Transferred per Process (Blocks/Process)	51.2	64.4	47.7	44.5	59.3	61.2	55.6	54.8
Avg. Page-seconds per Process	271.9	231.7	167.7	229.2	297.3	248.8	344.0	255.8
Avg. Pages Transferred per Process	57.5	64.0	91.8	93.1	91.7	76.5	128.5	82.0

Summary of Interactive Process Statistics over Period
from November 1982 through June 1983.
Table 2.5

Date	11/82	12/82	1/83	2/83	3/83	4/83	6/83	Average
Average No. of Processes	4923.8	4422.0	1829.3	4555.5	3488.9	5231.1	2437.9	3826.9
Avg. Service Time per Process (Seconds/Process)	2.099	2.277	1.717	1.747	1.916	2.296	2.153	2.022
Avg. Process Arrival Rate (Processes/Second)	0.0570	0.0512	0.0200	0.0527	0.0404	0.0605	0.0282	0.0443
Avg. I/O Blocks Transferred per Process (Blocks/Process)	241.1	244.7	228.1	230.3	238.9	277.9	297.5	251.2
Avg. Page-seconds per Process	249.4	211.5	178.5	182.5	196.7	180.0	234.2	204.7
Avg. Pages Transferred per Process	118.8	92.9	104.0	104.5	102.7	78.4	108.8	101.2

Summary of Batch Process Statistics over Period
from November 1982 through June 1983.
Table 2.6

Date	11/82	12/82	1/83	2/83	3/83	4/83	6/83	Average
Average No. of Processes	1542.1	856.7	733.5	1407.0	1030.3	1368.0	1202.6	1162.9
Avg. Service Time per Process (Seconds/Process)	2.302	2.599	2.274	2.488	3.074	3.331	2.942	2.710
Avg. Process Arrival Rate (Processes/Second)	0.1428	0.0793	0.0679	0.1303	0.0954	0.1267	0.1114	0.1077
Avg. I/O Blocks Transferred per Process (Blocks/Process)	171.3	200.3	201.9	193.2	194.6	213.2	191.0	195.1
Avg. Page-seconds per Process	219.6	209.5	206.9	224.5	289.3	295.1	344.5	255.6
Avg. Pages Transferred per Process	95.4	80.6	91.0	91.7	94.1	88.6	117.1	94.1
Utilization of CPU	32.87	20.62	15.45	31.89	29.33	42.20	32.76	29.35
Avg. Users Logged On	34	26	14	31	23	46	26	29

Summary of Process Statistics During Peak Hours
(2 - 5 pm.) over the Period November 1982 to June 1983.
Table 2.7

Date	11/82	12/82	1/83	2/83	3/83	4/83	6/83	Average
Average No. of Processes	562.8	229.8	191.5	441.2	324.0	493.1	492.2	385.5
Avg. Service Time per Process (Seconds/Process)	2.763	3.355	3.423	3.551	5.011	4.834	3.519	3.779
Avg. Process Arrival Rate (Processes/Second)	0.0488	0.0213	0.0177	0.0409	0.0300	0.0457	0.0456	0.0357
Avg. I/O Blocks Transferred per Process (Blocks/Process)	44.9	53.0	132.1	87.2	88.0	60.7	63.7	75.5
Avg. Page-seconds per Process	216.0	218.5	285.0	305.7	448.9	469.6	432.5	339.5
Avg. Pages Transferred per Process	78.2	65.1	83.8	86.1	89.6	97.1	122.9	89.0

Summary of Interactive Process Statistics During Peak Hours
(2 - 5 pm.) over the Period November 1982 to June 1983.
Table 2.8

Date	11/82	12/82	1/83	2/83	3/83	4/83	6/83	Average
Average No. of Processes	903.8	582.2	453.2	873.5	640.9	777.6	603.4	690.7
Avg. Service Time per Process (Seconds/Process)	2.038	2.254	1.768	1.727	1.983	2.219	2.286	2.039
Avg. Process Arrival Rate (Processes/Second)	0.0837	0.0539	0.0420	0.0809	0.0593	0.0720	0.0559	0.0640
Avg. I/O Blocks Transferred per Process (Blocks/Process)	244.4	247.7	230.9	226.7	236.1	268.8	283.6	248.3
Avg. Page-seconds per Process	230.4	205.8	183.5	178.9	209.8	182.8	263.6	207.8
Avg. Pages Transferred per Process	113.1	91.3	103.8	103.6	105.8	82.4	115.3	102.2

Summary of Batch Process Statistics During Peak Hours
(2 - 5 pm.) over the Period November 1982 to June 1983.
Table 2.9

Notice that the interactive service times in table 2.8 increase over the period. The service times in these summaries are down slightly from the summaries over the whole day as presented in tables 2.4 through 2.6. This is odd since one would expect the service times between the peak hours and the whole day to be rather close if the type of work being done was the same. It turns out, however, that the off peak hours were being used for disk backups and to run off class programs from a student program submission system. Both of these actions require extra processing time thus altering the service times over a whole day.

A profile of an average user session is presented in Table 2.10. The table shows that students use almost equal amounts of CPU time for both interactive and batch processes. This must be tempered with the knowledge that a complete batch job consists of either two, three, or four processes depending upon the number of steps completed in the compile, link, and execute chain (see figure 2.2). Assuming that the average batch job consists of three processes means that the user will execute the editor almost five times during a session and will submit about 3 batch jobs to the batch stream. Another item to note is the low amount of I/O work done with an interactive process as compared to a batch process. This seems slightly out of place until one realizes that the steps involved with compiling a program require reading one file and then generating two output files along with some temporary intermediate files.

The measurements taken on the system by LOGSPY were all based on

No. of Processes created	12.41	
No. of Interactive Processes	4.45	
No. of Batch Processes	7.97	
Ratio of Batch to Interactive	1.79:1	
Service Time per Process	2.710	Seconds
I/O Blocks Transferred	195.1	Blocks/Process
Pageseconds per Process	255.6	Page-seconds
Pages Transferred	94.3	Pages/Process
Service Time per Interactive Process	3.779	Seconds
I/O Blocks Transferred (Interactive)	75.5	Blocks/Process
Pageseconds per Interactive Process	339.5	Page-seconds
Pages Transferred (Interactive)	89.8	Pages/Process
Service Time per Batch Process	2.039	Seconds
I/O Blocks Transferred (Batch)	248.3	Blocks/Process
Pageseconds per Batch Process	207.8	Page-seconds
Pages Transferred (Batch)	101.9	Pages/Process
CPU Time for Console Process	3.302	Seconds
CPU Time for Interactive Processes	16.817	Seconds
CPU Time for Batch Processes	16.251	Seconds
Total CPU Time for Work Session	36.369	Seconds/Session
Total I/O Blocks (Interactive)	336.0	Blocks
Total I/O Blocks (Batch)	1979.0	Blocks
Total I/O Blocks for Session	2315.0	Blocks/Session
Total Pages Transferred (Interactive)	399.6	Pages
Total Pages Transferred (Batch)	812.1	Pages
Total Pages Transferred for Session	1211.7	Pages/Session

Summary of Average User Work Session
 During Peak Hours (2 - 5 pm.) over the Study Period.
 Table 2.10

the assumption that all the users performed equivalent types of work and that no one consumed large amounts of CPU time. In order to verify this assumption the second new program, LOGACCT, was written which would process the intermediate log files and provide statistics on a per user basis. The information cumulated for each user included the number of total processes, the number of batch and interactive processes, the total CPU time, total I/O blocks transferred, total pagesecods, and the percentage of total CPU time normalized over all the users in the intermediate file. The results were surprising.

The CPU time used by the operating system is impossible to accurately measure because it appears that some portion of the time is not attributed to any process. Several experiments were attempted to account for the operating system CPU time. None of the attempts were successful. The major places where the time is accounted is with the processes IPMGR, EXEC, and OP. These processes all perform some operating system function. It turns out that there were records in the accounting logs for these processes and that the amount of CPU time attributable to these three processes was significant. The average student user rarely used over 0.2% of the CPU time during a month while each of these three operating system processes used between 1 to 10 percent of the CPU time. It should be noted that these seemingly low percentages for system overhead can be explained by pointing out that if the system "crashed" without a proper shutdown there would be no records for these processes entered into the accounting log. Beyond these three processes we discovered that about six other users were using anywhere between 1 to 10 percent of the CPU time.

Discussion with the individuals indicated that during the study period each was working on one or more projects that involved the design and execution of large programs. Most of them ran interactively. This group of people consisted mainly of the system manager and some of the senior operators.

It was felt that the workload of these extreme users did not agree with the bulk of the user workload being modelled. Realizing that the main concern was about the normal user CPU workload it was decided to reprocess the accounting logs removing any processes attributed to these heavy users. This modification was made to the program LOGSPY and the results are presented in tables 2.11 through 2.14. Table 2.11 is a revised summary of the combined (console, interactive, and batch) processes, Table 2.12 is a summary of the interactive processes, table 2.13 is a summary of the batch processes, and table 2.14 is a user session profile based on the modified input data.

Note there is not much change in the figures of the summary tables except that fewer processes are observed and that the slight increase in interactive service times has disappeared. Apparently some of these heavy users were running large interactive programs during the peak hours. The only other major difference is in the user session profile. The preference in service times is given to the batch processing by about 3 seconds per session. Also note that the number of interactive I/O blocks transferred has dropped significantly. Both of these are indications of the alteration the

Date	11/82	12/82	1/83	2/83	3/83	4/83	6/83	Average
Average No. of Processes	526.8	229.8	180.8	410.0	303.1	470.4	495.3	373.7
Avg. Service Time per Process (Seconds/Process)	2.763	3.355	2.074	2.482	3.425	3.247	2.531	2.840
Avg. Process Arrival Rate (Processes/Second)	0.0488	0.0213	0.0167	0.0380	0.0281	0.0436	0.0459	0.0346
Avg. I/O Blocks Transferred per Process (Blocks/Process)	44.9	53.0	41.7	39.6	61.4	56.7	57.5	50.7
Avg. Page-seconds per Process	216.0	218.5	181.2	230.7	330.4	260.5	326.9	252.0
Avg. Pages Transferred per Process	78.2	65.1	87.4	92.9	96.5	80.2	129.2	88.7

Revised Summary of Interactive Process Statistics During Peak Hours
(2 - 5 pm.) over the Period November 1982 to June 1983.
Table 2.11

Date	11/82	12/82	1/83	2/83	3/83	4/83	6/83	Average
Average No. of Processes	903.8	582.2	452.4	842.7	635.0	760.8	522.5	671.3
Avg. Service Time per Process (Seconds/Process)	2.038	2.254	1.765	1.711	1.983	2.209	2.176	2.019
Avg. Process Arrival Rate (Processes/Second)	0.0837	0.0539	0.0419	0.0503	0.0588	0.0704	0.0484	0.0622
Avg. I/O Blocks Transferred per Process (Blocks/Process)	244.4	247.7	231.4	225.1	236.7	266.4	296.7	254.1
Avg. Page-seconds per Process	230.4	205.8	182.1	178.0	209.5	186.0	235.6	203.9
Avg. Pages Transferred per Process	113.	91.3	103.2	104.0	105.6	84.2	108.3	101.0

Revised Summary of Batch Process Statistics During Peak Hours
(2 - 5 pm.) over the Period November 1982 to June 1983.
Table 2.12

Date	11/82	12/82	1/83	2/83	3/83	4/83	6/83	Average
Average No. of Processes	1542.1	856.7	719.1	1337.8	998.1	1324.5	1116.9	1126.5
Avg. Service Time per Process (Seconds/Process)	2.302	2.599	1.883	2.082	2.554	2.735	2.457	2.373
Avg. Process Arrival Rate (Processes/Second)	0.1428	0.0793	0.0666	0.1239	0.0924	0.1226	0.1034	0.1043
Avg. I/O Blocks Transferred per Process (Blocks/Process)	171.3	200.3	178.3	174.6	188.1	210.1	180.9	186.2
Avg. Page-seconds per Process	219.6	209.5	176.7	199.1	250.5	220.3	286.6	223.2
Avg. Pages Transferred per Process	95.4	80.6	93.8	95.6	98.1	80.5	116.6	94.1
Utilization of CPU	32.87	20.62	12.54	25.79	23.60	33.54	25.41	24.91
Avg. Users Logged On	34	26	13	29	21	44	24	27

Revised Summary of Process Statistics During Peak Hours
(2 - 5 pm.) over the Period November 1982 to June 1983.
Table 2.13

No. of Processes created	13.82	
No. of Interactive Processes	4.59	
No. of Batch Processes	8.24	
Ratio of Batch to Interactive	1.79:1	
Service Time per Process	2.373	Seconds
I/O Blocks Transferred	186.2	Blocks/Process
Pageseconds per Process	223.2	Page-seconds
Pages Transferred	94.1	Pages/Process
Service Time per Interactive Process	2.840	Seconds
I/O Blocks Transferred (Interactive)	50.7	Blocks/Process
Pageseconds per Interactive Process	252.0	Page-seconds
Pages Transferred (Interactive)	88.7	Pages/Process
Service Time per Batch Process	2.019	Seconds
I/O Blocks Transferred (Batch)	254.1	Blocks/Process
Pageseconds per Batch Process	203.9	Page-seconds
Pages Transferred (Batch)	101.0	Pages/Process
CPU Time for Console Process	3.148	Seconds
CPU Time for Interactive Processes	13.036	Seconds
CPU Time for Batch Processes	16.637	Seconds
Total CPU Time for Work Session	32.821	Seconds/Session
Total I/O Blocks (Interactive)	232.7	Blocks
Total I/O Blocks (Batch)	2093.8	Blocks
Total I/O Blocks for Session	2326.5	Blocks/Session
Total Pages Transferred (Interactive)	407.3	Pages
Total Pages Transferred (Batch)	832.2	Pages
Total Pages Transferred for Session	1239.5	Pages/Session

Revised Summary of Average User Work Session
During Peak Hours (2 - 5 pm.) over the Study Period.
Table 2.14

heavy users had upon the characterization of the normal user workload. An obvious objection to the manner by which the heavy users were discarded would be that this form of work is significant and should be a factor in modelling. LOGACCT demonstrated that in any given month the CPU time attributed to heavy users (including the operating system processes) was less than 15% of the total CPU time and was typically 7 to 9%.

3. ANALYTIC MODELLING AND BENCHMARKING

To develop an analytic model for the MV/8000 the procedure followed was to define a model that represented the physical characteristics of the system, to develop a solution method based upon the data available, and to develop a benchmark that would allow validation of the model. A typical model used to analyze computer systems similar to the MV/8000 is a variation of the machine repairman model. A terminal server is used to represent users that submit requests to the computer. These requests are queued outside the CPU until room exists to allow them into the multiprogramming mix. Once in the multiprogramming mix a request will cycle between execution at the CPU and I/O service at the disk subsystem until the request is completed. When finished the results from the request are sent back to the user's terminal. The user will then spend some amount of time thinking before submitting a new request into the system.

From this model one is usually interested in obtaining the utilization of the CPU and the response time of the requests. The response time is defined as the period between when the user presses his return key and the time the computer gives back a response. This response time is known as the response time per "interaction" where the term interaction is defined, for measurement purposes, based upon the type of work being monitored and the data one has about the system. Often, for an interactive system, an interaction is the time between the pressing of the return key and the receiving of the results from the computer system. In a batch environment the response

time is usually the turnaround time of the job.

The type of model mentioned above was unsuitable for work on the MV/8000 for two reasons. First is that for the model one needs the routing probabilities of all paths in the system. It was impossible to determine the routing probability of leaving the CPU and returning back to the user terminal. Secondly, to obtain the response measured^s needed two definitions of the term "interaction" had to be adopted depending upon the type of work being done. The goal for the model was to determine the response time of batch jobs and interactive edit sessions. For a batch job an interaction was the act of sequentially executing a sequence of 2 to 4 processes that constituted a batch job. For interactive processes the term interaction was the act of executing a single editor session.

To determine the response times of the interactions just mentioned it was helpful to use a two stage hierarchical queueing model. Closed network queueing models were used for they allowed easy solution using the technique of Mean Value Analysis[4, 5, 6, 7]. The two stage model consisted of one stage to represent the multiprogramming level of the machine while the second stage represented the user level. The model is presented in detail along with the assumptions made in order to provide a solution.

3.1 THE LOW LEVEL MODEL

A diagram of the low level model is presented in figure 3.1. It consists of a CPU server and four disk servers, one for each physical

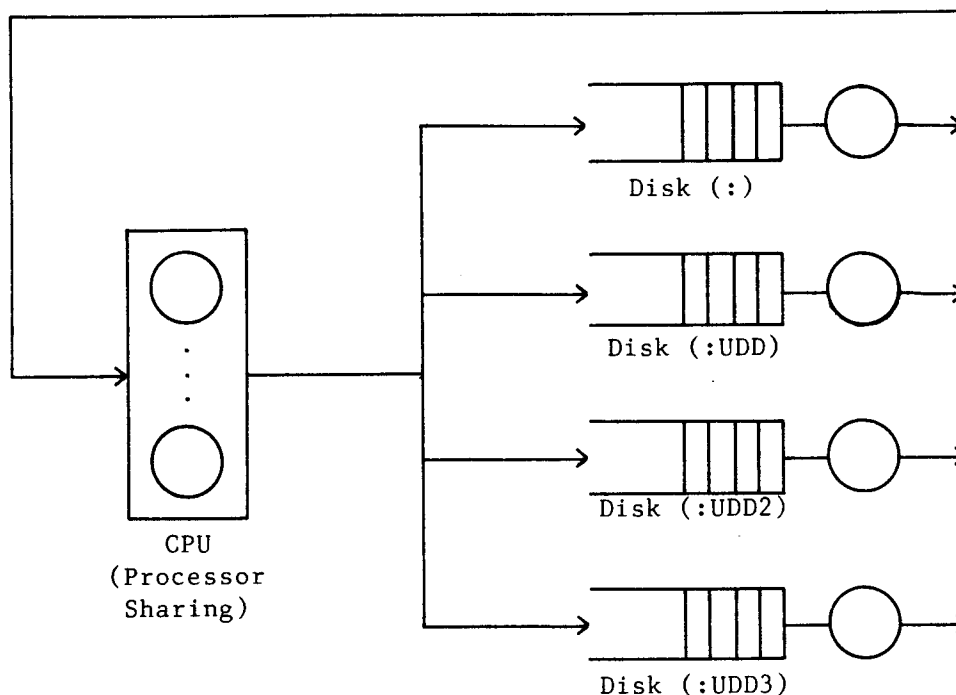


Figure 3.1
Low Level Queuing Model of the MV/8000

- disk on the system. This model represents the activity of the MV/8000 system at its multiprogramming level. The CPU is modelled as a processor sharing service center to account for time slicing among processes. Each disk server is a single server using a FIFO queuing strategy and represents the delay caused by I/O activity and page faulting. Two classes of customers are used in the model, interactive and batch.

This closed queuing model assumes that as soon as one process leaves the system another enters to take its place thus keeping the number of active users constant. Additionally, assumptions were made

that there is no logical page faulting within the system, no swapping of programs out of the multiprogramming mix, and no blocking at the disk controllers. From the workload characterization it is known that during peak usage times, which is the area of interest for modelling purposes, the number of users stays rather constant (the flow balance assumption holds). Likewise it is known that the amount of logical page faulting is small, and that swapping of processes rarely occurs on the system. Thus it was elected to ignore any possible effects of logical page faulting or process swapping on the model. Measurements with the online monitor DISCO showed that the amount of blocking at the disk controller shared by the 604 Mbyte and 147 Mbyte Winchester disks was less than 2%. It is noted that the majority of this blocking occurs when the operating system is initialized from the 147 Mbyte disk.

Since this model represents the multiprogramming mix of the machine the number of processes in the model can never be higher than the multiprogramming ratio. The multiprogramming ratio on the MV/8000 is governed by the number of processes the operating system can track. This maximum value is 255, meaning that up to 255 processes can be active on the system simultaneously. In practice this rarely happens. Most users have only one process active in the system at a time. This is not to say that they cannot have more than one process, but that when one creates a subordinate process the father process is usually blocked leaving only one active process in the system. On the NCSU system such is the case. Users are not allowed to create processes without blocking the father process and are allowed a maximum of three

processes. The initial logon CLI process is one of the three, leaving the user with only two son processes that he can create. Observation has demonstrated that the first son process is used for execution of edit sessions. The second son process is rarely, if ever, used.

In this study the multiprogramming ratio was fixed by the number of active users on the system. For the interactive class of customers this value was the number of users on the system. For the batch class this value is the number of batch stream servers active on the system. Remember that batch jobs are queued until a batch server becomes free to process the job. The number of batch servers is set in the operating configuration of the system. At the time of the workload characterization there were three batch stream servers. Note that this assumes enough batch jobs are in the system to keep the batch servers constantly busy. Observations during the workload characterization demonstrated that such was the case at peak times.

The results from this model are the throughputs of the batch and interactive classes at the CPU server. These throughputs are converted into service rates to use in the high level model. The inputs necessary for this model are the average execution time, the number of page faults, and the number of I/O blocks used by each class of processes. A complete description of the solution of the model using an example is provided in the Appendix. It is important to note that for this model everything is measured in a "per burst" fashion. At the multiprogramming level a process receives a burst of execution at the CPU before moving off to a disk to receive service before

returning back to the CPU.

A final point of concern is the omission of a quantum interrupt from the model. At the multiprogramming level of a machine it is common for a process to use up its complete quantum without blocking for I/O service. Data collected during the workload characterization study indicated that at the peak times of interest the average process would block for I/O service before its quantum expired. The scheduling discipline for preemptible processes used on the MV/8000 splits the time between now and the next scheduling interrupt equally among the processes in the ready queue. Measurements demonstrated that there were never over 60 users on the system simultaneously. This would give a total of about 70 active processes including the operating system and printer spooling processes. Assuming that 70 processes are in the ready queue the time slice for each would be approximately 29 milliseconds. Observations demonstrated that during peak loads the average time that a process would execute before blocking for I/O never exceeded 28 milliseconds. Thus it was assumed that on the average quantum interrupts did not occur. This was important from the standpoint of obtaining a solution since we were unable to determine the probability with which quantum interrupts occurred.

The actual burst service times for both the batch and interactive processes used in this model had to be modified to account for the execution time used by the console logon processes along with the overhead required by the operating system. This modification was

— accomplished as follows. The average CPU time of a console processes was determined by the program LOGSPY from evaluating the system accounting logs. Of this time some portion was attributable to running the necessary macros to submit a batch job. The time required to execute these macros was measured with a stopwatch on an unloaded system. The execution time required for the submission of a batch job was distributed into the burst service time of a batch process. The remaining console process CPU time was added into the burst service time for an interactive process. A correction for the operating system overhead was made by dividing the burst service time of each class by the percentage of CPU time spent executing user programs and then multiplying this result by the combined percentages of user and operating system time used at the CPU. These final burst service times were used in the low level model.

Another matter deserving discussion at this point is the determination of mean disk service times per disk access. Typically one might determine this service time by measuring the action of the disk system, using an online measurement tool, to provide the completion rate of each disk or by using the mean disk service time measurement provided in the manufacturer's specifications for the disk. The online measurement tool, DISCO, provided the number of accesses made to a disk since the machine was last initialized. This measurement, however, was useless for measuring the service time per access unless the disk subsystem was saturated. In this study such was not the case. Thus the specifications provided by the manufacturer were used. Experience has shown that unless a disk is accessed in a

truly random pattern the average disk service time given by the manufacturer is an overestimate of the actual service time. This can be attributed to the fact that in real use most accesses to a disk exhibit the principle of locality. This locality eliminates the notion of truly random accesses.

The specifications on disks provided by Data General[17, 18, 19] include the average disk access time, measurements on the time necessary to perform a seek across the complete disk surface along with a single track seek. Additionally they provide the rotation speed of the disk and the transfer time between it and the Burst Multiplexor Channel. This coupled with the fact that the online monitor, DISCO, provided the average number of tracks seeked by each disk per access suggested the idea of using a linear approximation to more accurately estimate the service time of each disk. The model was based on the following idea. The time necessary to perform a single track seek can be split into three time periods. The disk requires some amount of time to start the movement of the heads, some time to mechanically move the heads one track, and finally a portion of time to allow the heads to settle before sensing for sector boundaries. A seek across a complete disk consist of similar time periods: a time to start moving the heads, the time necessary to move across all the tracks of the disk, and the time necessary to settle the heads. It does not matter that the time necessary to start up the movment of the heads and the settling times are different, only that the total of those two time periods is the same regardless of the number of tracks moved.

Working with the above idea, track movement rates could be determined by solving the following two simultaneous equations:

$$X + 1 * Y = \text{single track seek time} \quad (3.1)$$

$$X + N * Y = \text{full disk seek time} \quad (3.2)$$

where X is the time necessary to start up the movement and settle the movement of the disk head^s, Y is the movement rate per track once the heads are in motion, and N is the number of tracks traversed on a seek across the full surface of a disk. With the values of X and Y in hand the average service time per disk access can be estimated as:

$$\text{DISK} = X + \text{SEEK} * Y + 8.333 + \text{BYTES} * \text{TRANS} \quad (3.3)$$

where DISK is the mean service time per disk access, SEEK is the average number of tracks seeked per disk access, BYTES is the average number of bytes transferred per disk access (SEEK and BYTES can both be determined with the information provided by DISCO), TRANS is the transfer rate of the disk controller, 8.333 is a constant representing the time for a rotational delay of half a revolution, and X and Y have the meaning given above.

To verify this model an experiment was devised where a program was executed that would read and write ten thousand, 512 byte blocks from a disk on a system with only one user. Under these controlled conditions one can accurately measure the service time of each disk and compare this with an estimate from the linear approximation model. The test was run twice for each disk. The online monitor,

DISCO, was used to provide the statistics on the number of accesses made to each disk. When finished the measured average disk service time was compared with the estimate provided by the linear approximation model. The averaged relative percent difference was 3.6% with a minimum of 1.3% and a maximum of 7.5%. Based on this result it was decided to use the linear approximation model for the estimation of disk service times during the validation phase of the modelling process.

3.2 THE HIGH LEVEL MODEL

The high level model, whose diagram is presented in figure 3.2, consists of a terminal server and two process servers. This model represents the activity of the system at the users level. The terminal server represents the delay caused by the user pausing to think before submission of the next process into the system. This terminal server is modelled as an infinite server so that it imposes only a simple delay. The interactive server, at the bottom of the figure, represents the service time of an interactive process. That is to say its service rate represents the time necessary to execute a complete edit session assuming that there is no think time between editor commands. This server is modelled as processor sharing since each of these interactive processes can be active in the system simultaneously. The batch stream server is a single server using a FIFO queueing strategy. This server represents the queueing of batch jobs waiting for a batch stream server along with the service requirement necessary to execute a batch job.

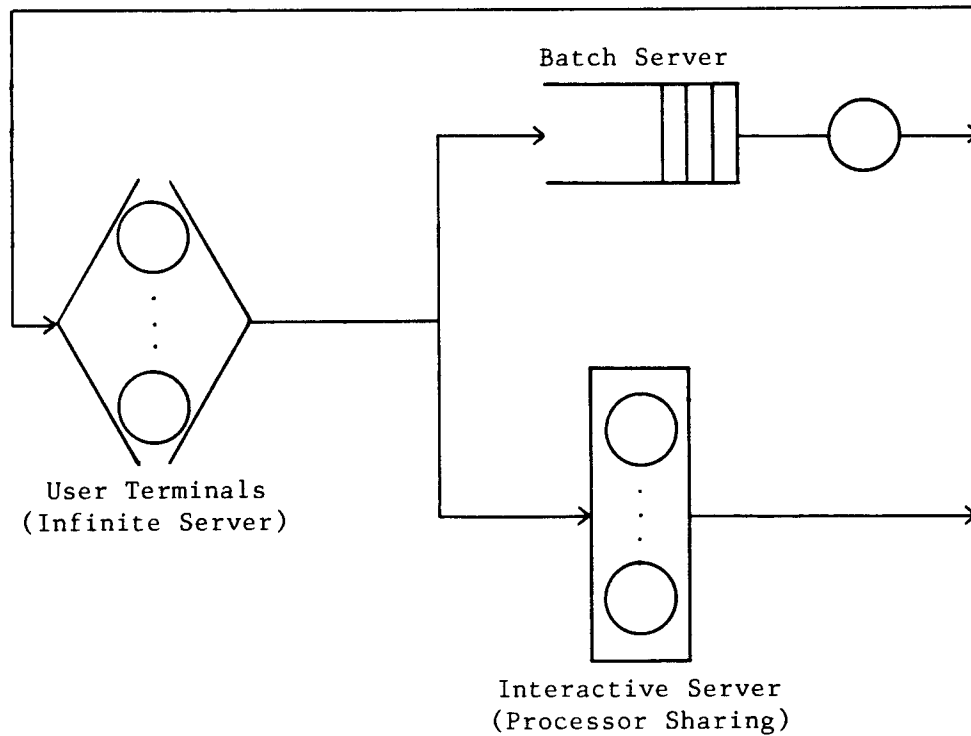


Figure 3.2
High Level Queueing Model of the MV/8000

This model also uses two classes of customers. The batch class of customers wait some delay time at the terminal server before entering the batch stream queue where they wait until receiving service at a batch stream server. Interactive processes wait some delay time at the terminal server and then enter the interactive server to receive their service requirement. The service times to be used in this model are obtained from the low level model. The service time used at the interactive server is the time necessary to process a complete edit session while at the batch server it is the time

- necessary to execute a complete batch job. Two difficulties with these
- service time measures had to be overcome. One was to determine how to
- relate the response time of an interactive process to a per request
- time, and how to determine the service time of a batch job when a batch
- job can consist of several processes and the low level model provided a "per process" service time.

The interactive processes are treated first. In building a computer system model one of the hardest parameters to estimate is the think time imposed by a user between entering requests into the system. Typically one might make an educated guess as to what the think time is or estimate it by taking measurements on a set of users as they work on the system. It was difficult to obtain think time with any reliability due to the inability of being able to get an average over a user population of more than 1000 users. Additionally it was impossible to encode delays into editor script files without creating additional, unrealistic CLI processes to perform the delays or by using personal computers to send the commands to the MV/8000 after waiting the proper delay times.

It was decided to extract all delay times from interactive processes and model them on that basis. This was done because the benchmark used to verify the model could not allow delays between editor commands. Note that determining the response time of an interactive process with no wait delays between requests is essentially a worst case situation. When there are no delays between requests the system sees its maximum workload because each process in

the multiprogramming mix is constantly demanding attention from the system. Thus the response time of the complete interactive process will be a maximum. To get the per request response time simply divide the response time of the interactive process by the average number of requests in an edit session.

As discussed earlier a batch job can consist of from two to four processes. This list includes the batch stream server (a Command Line Interpreter) which oversees the three steps of compilation, linking, and execution of the program. The service time used for the batch server in figure 3.2 must be the average service time for a batch job. The problem is to determine the average service time of a batch job. Observations have shown that of the four processes involved in the running of a batch job the Command Line Interpreter (CLI) takes the least amount of time. Typically it uses 1.5 seconds of CPU time if all three steps run. The longest two steps are the compilation and linking steps (typically 2.5 to 3.5 seconds per process). The average CPU time of a batch process was obtained from the program LOGSPY. Unfortunately, the program LOGSPY as it was used in the workload characterization phase provided the average of all batch processes including the CLI process which lowered the average significantly.

To resolve this problem LOGSPY was modified to give the average CPU times of compilation, link, and execution steps separate of the batch stream server process. Another task was to determine the number of processes in an average batch job. Again observations demonstrated that most batch jobs either failed on the compilation step or made it

to the execution step. LOGSPY was modified again and run on a couple of the accounting logs used during the workload characterization to give an average of 2.1 processes per batch job. Thus the batch job service time was determined as being the average service time of batch process multiplied by two batch processes per job plus an additional time for the CLI process based on the ratio of the average CLI process time to the average time for a batch stream process.

3.3 BENCHMARKING THE MV/8000

The model was validated using a benchmark. This was necessary because it was impossible to obtain disk performance characteristics from the information available for the workload characterization. Previous measurement work on a MV/8000 [1] has demonstrated that the best method for designing a benchmark is to write Command Line Interpreter macros that will submit user requests along with starting editor sessions and submitting batch jobs. Editor sessions are driven from script files. Practice quickly showed that the running of a benchmark could be completely automated from the time of user logon. This is done by making the benchmark driving macro be the startup macro run for the user run automatically by the system at logon time. A logoff command could be imbedded in the file to logoff the user when the benchmark was completed.

To help ensure that the benchmark accurately represented the user workload measured during the workload characterization, three FORTRAN programs were gathered that were written by students taking the FORTRAN programming class at the time. Of the three programs two

compiled and executed completely. Of these two, one was representative of the the first programming assignment given in the course while the second was representative of the final programming assignment. The remaining program was also representative of the final programming assignment. It performed only the compilation stage failing with three compilation errors.

Determining the proper form for an edit session proved difficult because no information was available from the workload characterization about the frequency of commands issued by students. Several beginning students using the machine for a numerical analysis course were visually monitored and lists made of the commands they issued during edit sessions. From these lists it was determined that an average student would issue approximately 25 commands during an edit session with most of them being the simple variety of position, list, delete, append, and insert rather than the more complicated find, spell, or replace commands.

Two problems with edit sessions had to be resolved. One was that it was impossible to have a script file either append or insert text directly from the script file into the edited file. The solution had the script append material from another file and to duplicate portions of the current file before deleting them. The other problem was that from a script file it was impossible to encode wait delays without creating additional Command Line Interpreter processes to encode the delays. It was deemed that such action would create an undue load on the system altering the performance of the benchmark from emulating

the original workload. Thus no delays were encoded into the benchmark between commands.

Lastly attention was turned to the task of encoding the appropriate commands to effect normal user requests to the Command Line Interpreter. The lists compiled about beginning students mentioned earlier demonstrated that over a short terminal session a user is likely to execute a couple of batch jobs, four edit sessions, and various other commands to delete files, type files, ask for help, and list directories. The final benchmark consisted of a script which would submit a batch job, edit a file, copy a file, list a directory, delete a file, and ask for help twice. This macro would repeat five times before terminating. It is important to remember that on the MV/8000 system at NCSU only one batch job per user is allowed in the batch stream at a time. An attempt to enter a second batch job into the queue before the first is finished will cause an error message to be printed on the user's screen and the request aborted. It is worth noting that there were no delays encoded between CLI commands to the system so that no undue load would be placed on the interprocess timing portion of the operating system.

This benchmark was implemented by creating 33 new user accounts with the usernames ATEST0 - ATEST10, MTEST0 - MTEST10, and ZTEST0 - ZTEST10. This naming convention forced the new users to be spread evenly across the three user disks. Each account was set up with a benchmark macro that would run automatically when the user was logged on. The three FORTRAN programs were spread evenly across the

different users so that the I/O rates across the disks would be even. The accounts were fixed so that when logged on the benchmark would repeat the sequence of batch job submission, edit session, and other CLI commands five times and then logoff the system.

To start the benchmark someone had to move around the computing facility logging users on. It is possible to log each user onto the system, block the console process of each user, and then release the group at one time. This was not done because it was felt that the operating system overhead of unblocking all the processes at once might skew the results. In addition, the fact that it is impossible to block all the console processes at exactly the same point eliminated the advantage of being able to start the benchmark at one time. There was no indication that the couple of minutes needed to logon users would violate the flow balance assumption.

To gather the input information necessary for the analytic model, three online monitors had to be run during the benchmark. PED was run using a 60 second snapshot delay with the output being written to a file so that the ratio of page faults to allocated working set sizes could be determined. This was needed to compute the number of page faults made by each process from the working set size information recorded in the system accounting logs. DISCO was run on a terminal to provide usage information on the disks at the start of the benchmark and at completion. This program was started slightly before the beginning of the benchmark and blocked manually until the end of the benchmark where it was restarted to update the disk statistics.

This information helped measure the disk service times per disk access. The final monitor, WHERE, provided information about the percentages of CPU time used by the users, the operating system, and the idle process. The program collects statistics about execution every ten seconds but updates the terminal screen only when requested. This information helped provide a correction for operating system overhead in the service times for the low level model.

The steps involved in running the benchmark are as explained below. To begin, the system was rebooted to reset the internal counters that keep the raw numbers on disk usage and to allow the resetting of the system accounting log to a new name. This ensured that the information measured by the monitor DISCO would not be skewed by previous usage of the machine. The MV/8000 was then brought back up with all modem lines and terminal lines external to the computing facility disabled. This prevented anyone from accidentally logging onto the system during the test and affecting the results. Next two terminals were logged on to run the monitoring programs. PED was started as a background process since all of its output went to disk to be processed later. DISCO and WHERE were started on the two terminals. With this accomplished the screen on the terminal running DISCO was frozen so that the starting disk information could be written down. Then as quickly as possible the appropriate number of users for this benchmark test were logged on making sure to have them evenly spread across the different users. These users were logged on around the different terminals in the building to ensure that no one Intelligent Achynchronous Controller was overloaded.

It was important to gather the beginning disk statistics because with this information any effects on the disk created by bringing the system up could be eliminated. The benchmark then ran under its own control until the last terminal was logged off. Shortly before the last terminal was to logoff the frozen monitor processes were released so that, as the last terminal logged off, the final statistics from both DISCO and WHERE could be collected. With this done the PED process was terminated. Finally several programs were used to process the information from PED and the system accounting logs to obtain the model input information. An example of this information and how it is used with the two models is presented in the Appendix. It is important to note that the use of the three monitors contributed enough to the system load so that they were included in the performance measurements collected on the system.

3.4 VALIDATION OF THE MODEL

To validate this analytic model the benchmark described in the last section was executed for 20, 25, 30, and 35 users. The throughputs and response times from the model along with the corresponding measured quantities were compared to determine the effectiveness of the model. The results of the benchmark runs are presented in tables 3.1 through 3.3. The throughputs in table 3.1 show that the estimated throughputs from the model agree rather well with the measured values. All of the relative differences are within fifteen percent and most are within ten percent. Two points can be made about this table. First is that the low level model constantly

20 USERS:			
	Model	Measured	Relative Difference
Interactive	0.01938	0.01793	8.1%
Batch	0.005813	0.000567	2.5%
Total	0.02519	0.02359	6.8%
25 USERS:			
	Model	Measured	Relative Difference
Interactive	0.01715	0.01512	13.4%
Batch	0.004179	0.000397	5.3%
Total	0.02133	0.01908	11.8%
30 USERS:			
	Model	Measured	Relative Difference
Interactive	0.01770	0.01556	13.7%
Batch	0.003455	0.00347	-0.4%
Total	0.02115	0.01903	11.1%
35 USERS:			
	Model	Measured	Relative Difference
Interactive	0.01672	0.01525	9.6%
Batch	0.002028	0.00214	-5.2%
Total	0.01875	0.01739	7.8%

(throughputs measured in processes(jobs) / per second)

Table 3.1
Throughputs from the Low Level Model

20 USERS:			
	Model	Measured	Relative Difference
Interactive	0.09929	0.09430	5.3%
Batch	0.007068	0.00654	8.1%
25 USERS:			
	Model	Measured	Relative Difference
Interactive	0.10679	0.09549	11.7%
Batch	0.005838	0.00564	3.5%
30 USERS:			
	Model	Measured	Relative Difference
Interactive	0.1093	0.09747	12.1%
Batch	0.004064	0.00432	-5.9%
35 USERS:			
	Model	Measured	Relative Difference
Interactive	0.1043	0.09612	8.5%
Batch	0.003913	0.00478	-18.0%

(throughputs measured in processes(jobs) / per second)

Table 3.2
Throughputs from the High Level Model

20 USERS:			
	Model	Measured	Relative Difference
Interactive	151.4	171.39	-11.6%
Batch	2497	2705.6	-7.7%
25 USERS:			
	Model	Measured	Relative Difference
Interactive	184.3	196.04	-5.9%
Batch	3389	4259	-8.7%
30 USERS:			
	Model	Measured	Relative Difference
Interactive	224.2	226.65	-1.1%
Batch	6840	6845	-0.1%
35 USERS:			
	Model	Measured	Relative Difference
Interactive	285.7	262.83	8.7%
Batch	8384	9456	-11.0%

(response times measured in seconds)

Table 3.3
Response Times from the High Level Model

overestimated the throughput of interactive processes. This caused the high level service times for interactive processes to be lower than they should be, forcing the response times per interactive process to be lower than the measured values. This effect can be observed in tables 3.2 and 3.3.

Secondly, notice the change in the batch process throughputs from overestimating to underestimating the measured throughput. This trend initially suggested that the modifications used to include the overhead of console processes and the operating system were not correct. Consideration of the method and the data indicated that the problem was in the service times used with the model, but not in the way that they were determined. The true problem appears to lie in the construction of the benchmark.

The benchmark was designed to cycle five times executing an edit session and attempting to submit a batch process on each iteration. The real time required to run the benchmark ranged from slightly over 1000 seconds to over 1600 seconds depending upon the number of users. On the surface it appears that this is a long enough period to accurately measure the activity of the system. Unfortunately, it only allowed enough time for the completion of at most six batch jobs.

This apparently is not enough jobs to obtain good averages due to the different execution times of the three programs used. Thus it is was felt that by running the benchmark for longer periods of time the averages for the execution of batch processes would have led to more accuracy in the batch throughputs. Also note that even during the 35

user benchmark the idle time of the CPU reported by WHERE was 3%.

- This seems a bit suspicious and suggests that possibly the counters for the percentages of usage are not being updated correctly.

Table 3.2 shows the throughputs of the two classes from the high level model. Most of the relative differences fall into the 15% range with the exception of the batch throughput for 35 users which was 18%. The service rates used in the high level model are determined from the throughputs of the low level model. Any error in the throughputs from the low level model will be carried into the high level model and magnified as the throughput from the low level model is converted from processes per millisecond to processes per second. This accounts for a portion of the differences between values. The remaining portion of the differences can probably be attributed to the technique used for altering the service time per batch process into a service time per batch job.

The last table, table 3.3, presents the response times given by the model as compared with the measured response times. The relative differences between the estimates and the measured values were within 11.6% indicating that the modelling technique predicts fairly well the response times of the system. Note, however, that almost all of the response times are underestimates of the measured response times. For the interactive processes this is mostly attributable to the overestimation of throughputs made in the low level model.

The underestimation with batch jobs probably has to do with the fact that with only six jobs completed per benchmark it is difficult

to accurately measure the queueing and service times of a batch job. A second cause may be directly related to the way by which the measured response times are determined. Information kept in the system accounting log provides only the elapsed time of the batch job while it is in execution. Thus the queueing time cannot be directly measured. It was indirectly determined by multiplying the elapsed time of a batch job by half of the customers in the queue (as given by the high level model) plus one half the elapsed time to account for the job in service.

The benchmark used issued 23 editor commands during each edit session. From this the average response time per editor command is estimated as:

# of users	response time (sec.)
20	7.45
25	8.52
30	9.85
35	11.43

The response times above are rather long to provide adequate response under normal working conditions. Usually a system is considered to be inadequate if the response time for a simple editor command is over four to five seconds. In defense of the measures is should be remembered that the benchmark used was simulating a given number of users who were working without any pauses between commands. Under a normal situation there would be pauses ~~in~~ between commands that would lower the response times because of the actual reduced load.

4. PERFORMANCE PROJECTIONS AND CONCLUSIONS

4.1 PERFORMANCE PROJECTIONS

With an analytic model now complete the last part of this study was to use the model to make performance projections of system changes. The workload characterization study showed that the most significant bottleneck of the system was a lack of CPU cycles to support the large user workload. For this portion of the study we decided to look at the effect of a CPU upgrade from the MV/8000 to the new Eclipse MV/10000. The specifications released in the MV/10000 product summary[9] indicate that the machine has an instruction cycle time which is roughly half of the MV/8000 cycle time. Also the system cache has been improved so that if the hit rate in the cache is around 95% the time necessary to load or store an item is approximately one third the time required by the MV/8000. According to the manufacturer benchmark tests of the MV/10000 hardware demonstrated that on the average it was about three times as fast as the original MV/8000. Based on the results of these benchmark tests it was decided to assume that the MV/10000 was three times as fast as the MV/8000 and to look at the effects of this upgrade assuming that all other parameters are held constant.

The input data used for the low level model was based upon the process statistics collected during the workload characterization for the student workload. This characterization provided only a small portion of the input data needed. The rest had to be estimated based

upon some isolated measurements taken with WHERE and DISCO during the characterization study along with data gathered during the benchmarking process. The input data used is presented in table 4.1 and contains both the measured and estimated data.

The values for the average number of blocks transferred per disk access, the disk routing probabilities, and the disk service times were estimated from isolated observations made with DISCO during the workload characterization. The disk service times were estimated using the linear approximation discussed earlier assuming that the average seek distance was one quarter of the disk surface. The values for the ratio of page faults to working set size were taken from the benchmark tests because there was no such data available from the period of the workload characterization. The percentages for the usage of the CPU by user and operating system processes were taken from WHERE during the characterization phase. The values presented in table 4.1 were used to establish the input value for the low level model as discussed in the ^Aappendix.

The model was solved using the described workload for user values ranging from 20 to 50 incrementing by 5 for each analysis. Following this the service times used in the low level model were reduced by a third to account for the faster CPU and the model was solved again over the same value range. In converting from the throughputs of the low level model to the service times of the high level model it was necessary to convert the batch process service time into a batch job service time. For this study it was assumed

	Batch	Interactive
* Avg. CPU time / process:	2.039	3.779 sec.
* Avg. I/O blocks transferred / process:	248.3	75.5 blocks
* Avg. working set size / process:	102.2	89.0 pages
Ratio of page faults / working set size:	0.640	0.245

Percent of CPU time spent in USER mode: 45%

Percent of CPU time spent in O/S mode: 52%

Avg. blocks transferred per disk access: 1.25 blocks

Disk routing probabilities:

directory	probability
:	0.350
:UDD	0.250
:UDD2	0.150
:UDD3	0.250

Disk service times per access:

directory	service time (msec.)
:	30.071
:UDD	28.905
:UDD2	28.577
:UDD3	28.577

* - indicates a measured value, all others are estimates.

Input Values Used for CPU Upgrade Study
Table 4.1

that the average batch job consisted of two processes and that the overhead of the batch stream CLI process constituted one half of the time necessary to execute a single batch process. There were three batch servers used in the low level model. Additionally, it was assumed in the high level model that the think time between the

completion of a process or job and the submission of the next one was one minute. The results of this analysis are presented in tabular form in table 4.2 and graphically in figures 4.1 and 4.2. It is necessary to point out that this study assumes no think time between requests made by interactive processes and assumes that each user keeps one interactive processes and batch job in the system constantly.

Batch Job Response Times: (seconds)		
	Original CPU	Faster CPU
20 users	1777	756
25 users	2686	1105
30 users	3773	1510
35 users	5040	1978
40 users	6486	2504
45 users	8111	3089
50 users	9915	3735

Interactive Process Response Times: (seconds)		
	Original CPU	Faster CPU
20 users	111.4	9.256
25 users	149.7	14.52
30 users	187.8	23.03
35 users	225.9	34.34
40 users	263.9	46.87
45 users	301.8	59.61
50 users	339.8	72.35

Results of the CPU Upgrade Study
Table 4.2

The response times of batch jobs, shown in figure 4.1, have curves that are exponential in nature as we would expect. Notice that

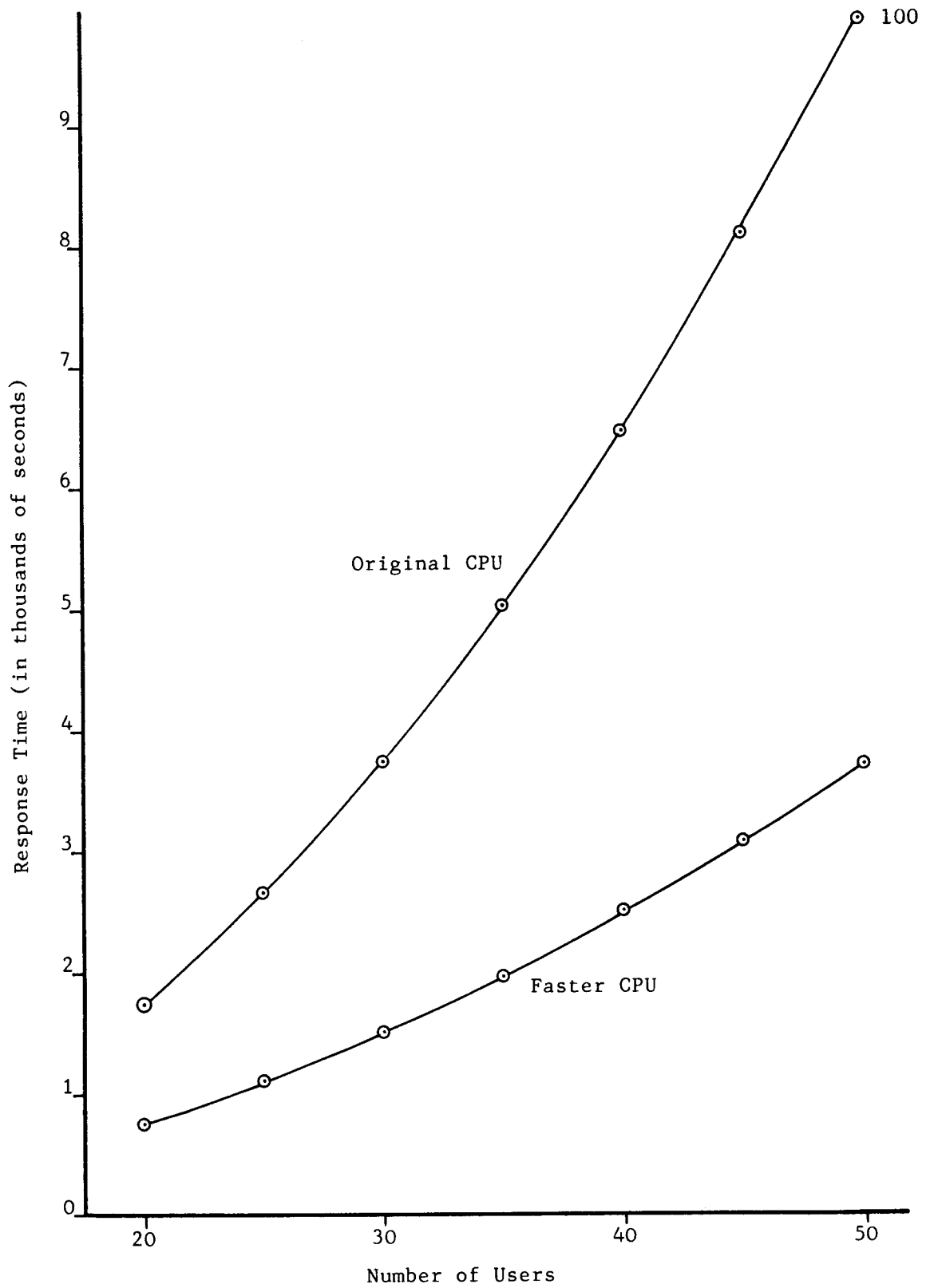


Figure 4.1
Response Time of Batch Jobs

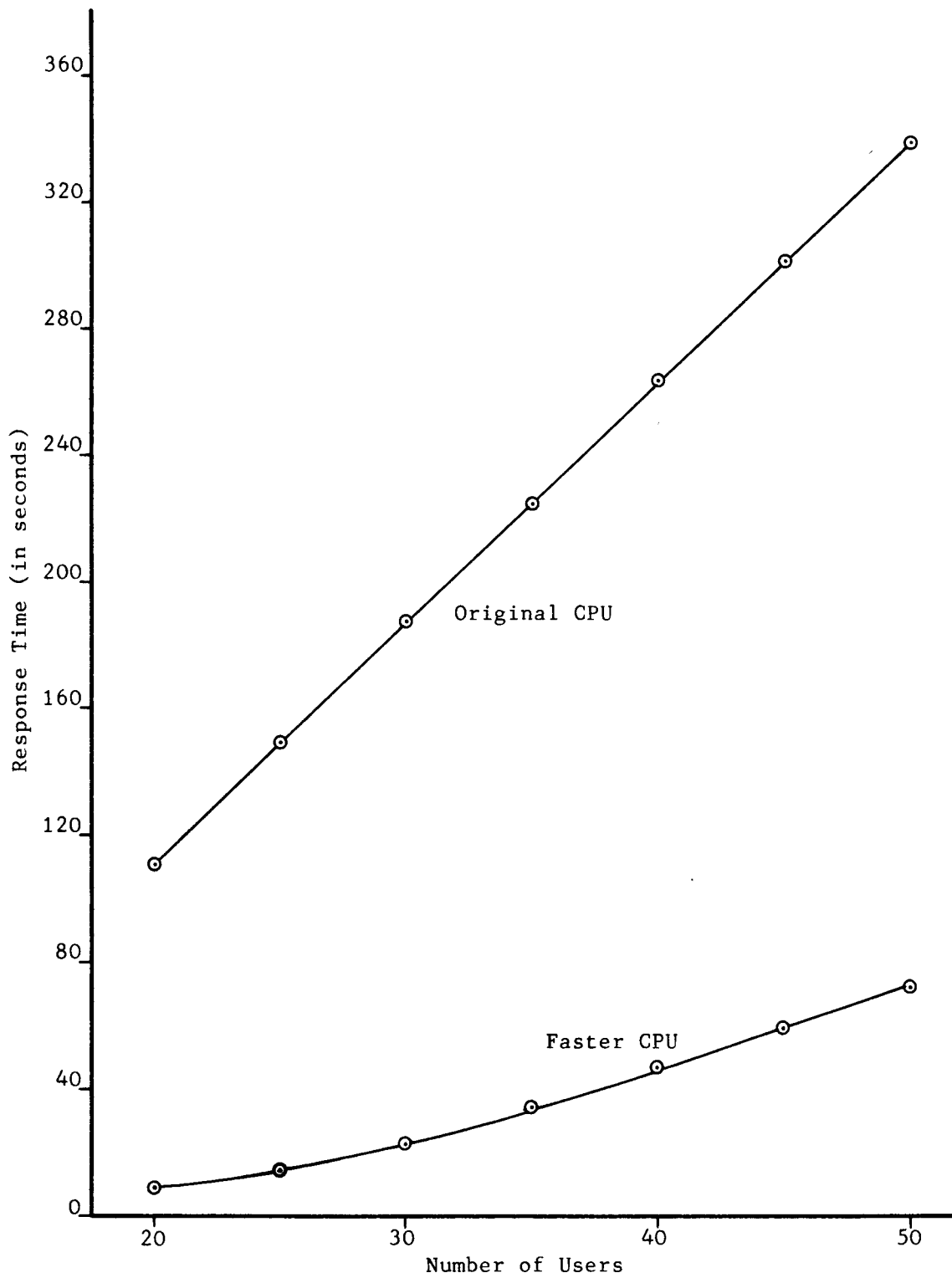


Figure 4.2
Response Times of Interactive Processes

the curve for the original CPU curves upward quickly indicating that the original CPU has probably reached its saturation point somewhere around 25 to 30 users. At 30 users the response time of a batch job, including both queueing and service, is slightly over an hour, while at 35 users the time increases to almost one and one half hours. The faster CPU gives a much improved curve. Notice that its rise is slow throughout the whole of the graph. This indicates that even at 50 users the graph is just beginning to change over into the portion of the curve that increases quickly. At 50 users the batch job response time is slightly over one hour. The evidence in this graph would suggest that with respect to batch jobs the CPU will saturate slightly beyond 50 active users. Comparison of the data for this graph shows that on the average the faster CPU will reduce the response times of batch jobs by a factor of 2.5.

The graph of the response times of interactive processes presented in figure 4.2 again show a large improvement for the faster CPU. The response times for the original CPU range from approximately two minutes to six minutes while the faster CPU ranges from about nine seconds to slightly over one minute. On the average this gives an average improvement in response times by a factor of 7.5. One point of interest is the fact that the graph of the response times for the original CPU is linear. Usually graphs of response times verses increasing numbers of users are exponential in form. This linear graph is probably a portion of the full curve where the graph is linear in this section and curves exponentially on each end. This idea is further supported by noticing that the graph of the faster

CPU begins with a slight exponential curve but the last five points form a linear line. This indicates that a portion of the full graph is probably linear. A last point is to notice that even at 50 users the graph for the faster CPU has not taken the sharp upward turn common to exponential curves. This indicated that more interactive users could easily be supported on the faster CPU.

A conservative estimate would be to say that an upgrade from a MV/8000 to a MV/10000 would reduce response times by at least a factor of 2.5 on the average. Taking into account the fact that this study was performed assuming that each user has both interactive processes and batch jobs in the system simultaneously it is probable that the actual reduction in response times will be higher. This is because in an actual environment a user will not always have a batch job in the system and will have some think time between editor commands. This slight reduction in the actual workload will create better system performance. It is necessary to point out that this prediction is based on each user having at most two active processes in the system along with the other assumptions imposed by the model constraints. The maximum number of users to allow on the system simultaneously can roughly be determined from figures 4.1 and 4.2 after one has decided what the maximum acceptable response times are.

4.2 CONCLUSIONS

The analytic model presented in this work is a simple hierarchical model that reasonably predicts the performance of the

MV/8000. This model has the virtues that it is simple in both design and solution, conforms closely with the physical activities observed in the machine, and seems to provide response measures that are relatively stable compared with variances in the input data. Unfortunately, the number of assumptions made in order to provide a solution force the model to be rather restrictive in its useful range. This model requires the machine to be in a steady state process flow, disk service times must be exponential, each user can have at most only one interactive process and one batch job in the system, there can be no disk blocking, there is negligible logical page faulting, no process swapping, quantum interrupts are negligible, and all users are scheduled at the same priority level. Modifications would be required to use this model with a machine that did not fit these restrictions. Additionally, the use of the hierarchical model helps to propagate errors. In the low level model the service time of a burst is measured in milliseconds and is determined from several input parameters. The throughput measured from the model is then converted into service time measured in seconds for use in the high level model. Any error introduced into the low level model will be magnified by a factor of one thousand before use in the high level model.

Most of the modifications needed to expand the range of usability of this hierarchical model could be easily made provided one obtains the necessary input values. For example quantum interrupts could be installed in the low level model if the routing probability could be determined. Swapping could be handled with a swapping server once the

necessary overhead time in both operating system CPU time and disk service time are determined. One would also need the probability of this happening. It is possible that with carefully designed experiments using benchmarking programs one could determine some of these parameters. The best solution, however, would be for the manufacturer to make improvements in their system accounting functions to record the data necessary for better performance predictions. The following improvements in the accounting system and monitors are suggested.

In the process termination log, SYSLOG, information related to physical page faulting, logical page faulting, and disk swapping should be added. This could be in the form of an actual page fault count for the process or expressed as a rate. Information on the amount of CPU overhead for the process should be provided. The process identification area should be increased from 15 to 20 characters to allow enough room so that processes can be completely identified by username, process identification number, and program name. The consistent use of program names and mnemonic typing of such names would allow each process to be uniquely identified as being interactive, batch, edit, program execution, compilation, or link so that workload characteristics could be easily determined. System managers should be carefully instructed on the importance of providing unique names for entry into the accounting log. Otherwise the manufacturer should create such names and have them automatically inserted into the log by the operating system.

The manufacturer should either add another log or include as part of the current system log disk statistics which are periodically updated by the operating system. This information should include the number of access to each disk, the number of blocks read and written, mean queue length, number of blocked requests, average seek distance, percentage of usage, and the percentage of the elapsed time that the disk was busy. These records should be written into the log at periodic intervals so they can be recovered for study at a later time.

The monitor DISCO needs modification to allow the output to be written to disk either together with or separately from the screen output. The user should be allowed to alter the screen update time. The screen editor, SED, needs alteration so that pauses can be encoded in script files without the need of creating additional CLI processes to handle the pause. Currently, to enter a pause, one must create an additional CLI process, pause the CLI for the specified amount of time, terminate the CLI, and then return to SED for the next script file command. Having the ability to pause the editor will aid future performance analysts by allowing them to use more representative benchmarking programs.

8. LIST OF REFERENCES

1. R. Kinicki, M. O'Donnell, S. Bishop, and R. Sacilotto, "Performance evaluation of a Data General MV8000 in an educational environment," CMG XIV Conference Proceedings, 1983, pp. 165 - 172.
2. Edward D. Lazowska, "The benchmarking, tuning and analytic modelling of VAX/VMS," Technical Report No. 79-04-01, Department of Computer Science, University of Washington, 1979.
3. Larry F. Hodges, "Benchmarking, analytic modelling, and performance evaluation in a research environment," Master's Thesis, Computer Studies Program, North Carolina State University, 1982.
4. M. Reiser and S. S. Lavenberg, "Mean-value analysis of closed multichain queueing networks," Journal of the ACM, Vol. 27 #2, April 1980, pp 313 - 322.
5. Steven C. Bruell and Gianfranco Balbo. COMPUTATIONAL ALGORITHMS FOR CLOSED QUEUEING NETWORKS, North Holland Publishers, New York, 1980.
6. Edward D. Lazowska, John Zahorjan, G. Scott Graham, Kenneth C. Sevick. QUANTITATIVE SYSTEM PERFORMANCE: COMPUTER SYSTEM ANALYSIS USING QUEUEING NETWORK MODELS, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1984.
7. Hisashi Kobayashi. MODELING AND ANALYSIS: AN INTRODUCTION TO SYSTEM PERFORMANCE EVALUATION METHODOLOGY, Addison-Wesley Publishing Co., Reading, Massachusetts, 1981.
8. M. Badel, D. Chandesris, J. Guillemaud, D. Potier, P. Saintoyant, and M. Veran. QNAP2 REFERENCE MANUAL, Cii Honeywell Bull, Paris, 1983.
9. Data General Corporation, ECLIPSE MV/10000 PRODUCT SUMMARY, Westboro, Massachusetts: DGC, 1983, Order No. 014-000737-00.
10. Data General Corporation, ECLIPSE MV/8000 PRODUCT SUMMARY, Westboro, Massachusetts: DGC, 1980, Order NO. 014-000650-00.
11. Data General Corporation, ECLIPSE MV/8000 PRINCIPLES OF OPERATION, Westboro, Massachusetts: DGC, 1980, Order No. 014-000648-00.
12. Data General Corporation, INTELLIGENT ASHCHRONOCUS CONTROLLER PROGRAMMER'S REFERENCE MANUAL, Westboro, Massachusetts: DGC, 1982, Order No. 014-000703-01.

13. Data General Corporation, LEARNING TO USE YOUR AOS/VIS SYSTEM, Westboro, Massachusetts: DGC, 1981, Order No. 069-000031-00.
14. Data General Corporation, AOS/VIS PROGRAMMER'S MANUAL, Westboro, Massachusetts: DGC, 1980, Order No. 093-000241-00.
15. Data General Corporation, HOW TO GENERATE AND RUN AOS/VIS ON YOUR ECLIPSE MV/FAMILY COMPUTER, Westboro, Massachusetts, 1981, Order No. 093-000243-01. DGC
16. Data General Corporation, AOS/VIS COMMAND LINE INTERPRETER USER'S GUIDE, Westboro, Massachusetts: DGC, 1982, Order No. 093-000122-05.
17. Data General Corporation, MODEL 6160/6161 DG/DISC STORAGE SUBSYSTEM, Westboro, Massachusetts: DGC, 1981, Order No. 014-000654-00.
18. Data General Corporation, MODEL 6060, 6061, 6067 DG/DISK STORAGE SUBSYSTEMS PRODUCT BRIEF, Westboro, Massachusetts: DGC, 1979, Order No. 012-336-4.
19. J. Oliphant, personal communication.

6. APPENDIX

The sample calculation presented below uses data collected in the thirty user benchmark run made during the validation portion of this study. The raw data obtained from the benchmark run is presented as it was used in the determination of values for the model. The calculations are presented in the order performed along with any explanation. We begin with the low level model. To solve the low level model information about the performance characteristics of the programs run on the machine along with service times and routing probabilities of the four disks were required. Program execution information was obtained from the system accounting logs by the program LOGSPY. The ratio of page faults to working set size was obtained from the information collected by PED. Disk information was collected by the program DISCO. For the thirty user benchmark the raw data is presented in table 6.1.

Table 6.1 gives that each batch and interactive process had on the average 61.8 and 33.4 page faults respectively. The first step was to determine the mean service time per burst for each type of process. This was accomplished by solving the following four equations. The first three equations (6.1 - 6.3) were solved for each type of process while the fourth equation (6.4) was solved only once. The mean service time per CPU burst is given as

$$\text{MU} = \text{AVCPU} / \text{DISK} \quad (6.1)$$

where MU is the mean service time per burst, AVCPU is the average

	Batch	Interactive	
Average CPU time / process:	3.143	4.173	sec.
Avg. I/O blocks / process:	301.1	83.9	blocks
Avg. working set size / process:	89.23	134.12	pages
Ratio of working set size to page faults:	0.693	0.249	

Table 6.1
Data from the Thirty User Benchmark

CPU time per process, and DISK is the average number of disk accesses per process. DISK is given by

$$\text{DISK} = \text{BLKSPR} / \text{BLKSAC} \quad (6.2)$$

where BLKSPR is the average number of blocks transferred per process and BLKSAC is the average number of blocks transferred per disk access. BLKSPR is determined by

$$\text{BLKSPR} = \text{I/O} + (4 * \text{PAGE}) \quad (6.3)$$

where I/O is the average number of I/O blocks per process, PAGE is the average number of page faults per process, and the constant of 4 represents the fact that each page on the MV/8000 consists of four disk blocks. BLKSAC is determined as

$$\text{BLKSAC} = \frac{\sum_{i=1}^4 [\# \text{ of blocks read}(i) + \# \text{ of blocks written}(i)]}{\# \text{ of disk accesses}(i)} \quad (6.4)$$

4

where the number of blocks read, number of blocks written, and

number of disk accesses are determined from DISCO by taking the values recorded at the end of the benchmark and subtracting them from the values taken at the start of the benchmark.

For this benchmark run the results from equations 6.1 through 6.4 were:

	Batch	Interactive	
Mean service timer / burst:	7.183	26.777	msec.
Avg. disk accesses / process:	402.3	159.6	
Avg. blocks transferred / process:	217.5	548.3	
Avg. blocks transferred / disk access:	1.363		

The raw data used to compute the average number of blocks transferred per disk access was omitted due to the large amount of data used for this calculation. Note that the term blocks used above designates 512 byte disk blocks. The user overhead incurred from execution of console logon (CLI) processes was determined as follows. From the program LOGSPY the overhead was 5.556 seconds per console process. During the benchmark each terminal logged on submitted one batch job and performed five edit sessions. Measurements conducted with no system load demonstrated that the submission of a batch job requires 1.162 seconds of CPU time. Splitting the remaining 4.394 seconds of the original 5.556 seconds across the five interactive processes gives 0.8788 seconds of console process overhead per interactive process. Converting these overhead times into additional service times per burst using equation 6.1 gave :

batch process console overhead = 2.888 msec / burst
 interactive process console overhead = 5.507 msec / burst

The addition of these additional service times to the original estimates provided new batch and interactive service rates of 10.701 and 32.284 milliseconds per burst. These new service rates were corrected into final service rates by accounting for the operating system overhead using the equation

$$FMU = MMU / USERP * UCP \quad (6.5)$$

where FMU is the final mean service time per burst, MMU is the modified mean service time determined in the previous step, USERP is the percentage of time the CPU spent executing user programs, and UCP is the percentage of time the CPU spent executing both the operating system and user programs. The final mean service times were 17.592 and 53.077 milliseconds per burst for batch and interactive processes respectively. These final values were determined using the information provided by the monitor WHERE. The monitor gave the percentages of time spent executing user and operating system programs as 59% and 38% respectively. These final mean service times were used in the low level model.

The next step was to determine the routing probabilities of the four disks. This information was provided directly by the monitor DISCO as:

directory	percentage
: (root)	0.317
:UDD	0.261
:UDD2	0.163
:UDD3	0.262

The service times of the four disks were determined using equation 3.3. For demonstration purposes we present the equation used for the directory :UDD3, which is a Data General model 6061 disk subsystem. The essential specifications of the drive are:

Tracks per surface:	815
Half revolution rotational delay:	8.333 msec
Data transfer rate:	806,000 bytes / second
Track to Track seek rate:	6 msec
Maximum seek rate:	60 msec

From this information and equation 3.3 we can derive the following equation for determining the mean service time of the disk per access:

$$UDD3SR = 14.26366 + (SEEK * 0.06634) + (BYTES / 806) \quad (6.6)$$

where UDD3SR is the mean service time of the disk drive for directory UDD3, SEEK is the average number of tracks seeked per access on the disk, and BYTES is the average number of bytes transferred per disk access. For this drive during the benchmark the value of SEEK was 91.3 and BYTES, determined from equation 6.4, was 697.86 bytes. The service times for each of the four disks are presented below:

directory	mean service time (msec)
: (root)	22.717
:UDD	21.914
:UDD2	20.437
:UDD3	21.186

With this information and the knowledge that there were thirty interactive processes and two batch processes in the system, the low level model was evaluated using mean value analysis as provided in the queueing network analysis package, QNAP2. The model gave the throughputs shown in table 6.2 which are compared with the measured quantities.

The throughputs in table 6.2 are used as state dependent inputs to the high level model. In the high level model some number of users will be held at the terminal server leaving a reduced total number of users active at the interactive server. This value is determined by first assuming a certain value, solving the high level model, and then verifying that the number assumed to be held at the terminal server is correct. Experience suggested that we assume the number of users held to be five. Measurements made on the benchmark with no load on the system indicated that interactive processes were submitted approximately 50 seconds after completion of the previous interactive process. This value was used as the service time of the terminal server. It is necessary to remember that this waiting time just mentioned as a service time is actually the time required to perform other CLI commands, but the overhead incurred by these commands was included in the low level model.

	Model	Measured	Relative Difference
Batch	0.003455	0.0347	-0.4%
Interactive	0.01770	0.01556	13.7%
Total	0.02115	0.01903	11.1%

(throughputs measured in burst / msec)

Table 6.2
Low Level Model Results for the Thirty User Benchmark

Assuming that only 25 of the 30 users are active at the interactive server, the service times for servers within the high level model were based upon the following set of equations. For an interactive process the equation is:

$$\text{INTSER} = (1.0 / \text{ITRUPUT} * \text{DISK}) / 1000.0 \quad (6.7)$$

where INTSER is the mean service time of the interactive server, ITRUPUT is the throughput of the interactive class of customers from the low level model, DISK is the number of disk accesses per process for an interactive process, and the constant 1000.0 is a conversion from milliseconds to seconds. For a batch job the equations are:

$$\text{BAT} = (1.0 / \text{BTRUPUT} * \text{DISK}) / 1000.0 \quad (6.8)$$

$$\text{BATSR} = 2.0 * \text{BAT} + \text{BAT} / \text{BPROC} * \text{CLIPROC} \quad (6.9)$$

where BATSR is the mean service time for a batch job, BAT is the service rate of an individual batch process, BTRUPUT is the low level throughput of a batch process, DISK is the number of disk accesses per batch process, BPROC is the average service time of a batch process (non-CLI processes), and CLIPROC is the average service time of a

- batch stream CLI process which controls a batch job. Equation 6.9 is a conversion from the service time of a single batch process to the service time of a batch job. The first term determines the service time of an average of two processes in a batch job while the second term adds on the percentage of time required by the batch CLI process relative to the other batch processes. The values of BPROC and CLIPROC were given by the program LOGSPY.

From the low level model the values of INTSER and BAT were given as 9.148 and 197.322 seconds respectively with 25 active users in the low level model. From this the batch job service time (BATSr) was determined as 246.072 seconds given that BPROC and CLIPROC were 3.143 and 1.553 seconds respectively. These values and the knowledge that there were 28 batch jobs active in the system along with 30 interactive processes provided the final information needed to solve the high level model using QNAP2. The results demonstrated that 5.3 users were held at the terminal server suggesting that the assumption for users held at the terminal server was correct. The results from the high level model are compared below with the measured throughputs and response times in table 6.3.

The throughputs along with the response measures were used to validate the analytic model. These response measures were the performance metrics desired for the evaluation of the system under different user workloads.

Throughputs:	Model	Measured	Relative Difference
Interactive	0.1093	0.09747	12.1%
Batch	0.004064	0.00432	-5.9%

(throughputs are measured in processes/jobs per second)

Response Times:	Model	Measured	Relative Difference
Interactive	224.2	226.65	-1.1%
Batch	6840	6845	-0.1%

(response times are measured in seconds)

Table 6.3
High Level Model Results for the Thirty User Benchmark