HIMICS: A VIRTUAL MEMORY ENVIRONMENT FOR MINI-COMPUTERS AND A DESCRIPTION OF ITS LEVEL 2 PROCESSOR

by

ARLAN E. BENTZ

B.S., Kansas State University, 1968

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1975

Approved by:

Virg Wallentine

Major Professor

THIS BOOK CONTAINS NUMEROUS PAGES WITH THE ORIGINAL PRINTING BEING SKEWED DIFFERANTLY FROM THE TOP OF THE PAGE TO THE BOTTOM.

THIS IS AS RECEIVED FROM THE CUSTOMER.

TABLE OF CONTENTS

APPENDICES

ILLUSTRATIONS

The HIMICS system is a hierarchical virtual memory system for a hierarchy of interconnected mini-computers. This paper describes the design of the software system. The software system design in this paper is a hierarchical design with two major processor levels. An overall description of both processors is given and then a detailed description of its level 2 processor is presented. The detailed description includes the algorithms, written in a dialect of PL/1, along with a written description of them. The HIMICS system will provide a virtual memory system for a network of mini-computers and also allow the emulation of high level languages. The implementation of this system should result in an increase of processor efficiency and system throughput for the mini-computers involved in the network. The paper is concluded with a dialectic comparison of a single processor system versus a multi-processor system.

CHAPTER ONE

1.1 INTRODUCTION

In this paper we propose a design for a hierarchical minicomputer system called HIMICS (Hierarchical multI-tasking MinI-computer Computer System). The system is designed with five major objectives in mind. These objectives are:

(1) To provide virtual memory capability.

(2) To provide support for emulators.

(3) To provide inter-emulator communication.

(4) To provide manageable software.

(5) To provide sufficient instrumentation and monitor capabilities in order to encourage meaningful system evaluations and comparisons.

A general discussion of each of these objectives will be given before we present the actual design of the system.

1.2 TECHNIQUES FOR RECURRENT USE OF MEMORY

There are several techniques that are commonly used in modern day computers to execute programs which have a larger address space than the primary memory available to them. Two of the more commonly used techniques are overlay structures (1,6,12) and virtual memory (2,12).

1.2.1 OVERLAY STRUCTURES

With overlay structures, segments of the program are kept on secondary storage and brought into main memory in an hierarchical sequence as they are needed. Pre-specified segments

may be overwritten by incoming segments. This is illustrated in Figure 1-1. Here the user has a 520K program to be run in a 320K address space. Segments A, B and C are first loaded and execution begins. As soon as segment B is no longer needed, segments D and E can be overwritten in B's address space. The same process happens when C and E are no longer needed. They can be overwritten by F.

From this illustration it is apparent that the user must have a knowledge as to what segments are to be overwritten. It is the user's responsibility to issue orders for the overlay to occur. This is the major disadvantage of the overlay technique. The second technique, virtual memory, does not require the user to have this additional knowledge.

1.2.2 VIRTUAL MEMORY

The key to virtual memory relies on the fact that, for an instruction in a program to be executed, only the instruction and the data that it operates on need be in primary memory. From the instruction's point of view the rest of the program may be located on any level of memory. This removes the requirement that the job's entire address space be in physical memory at once. Because this physical restraint is removed, the operating program has the illusion that it has an extremely large memory, thus the term "virtual memory". Since a job's entire address space need not all be in primary memory at once,

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

Overlay Structure

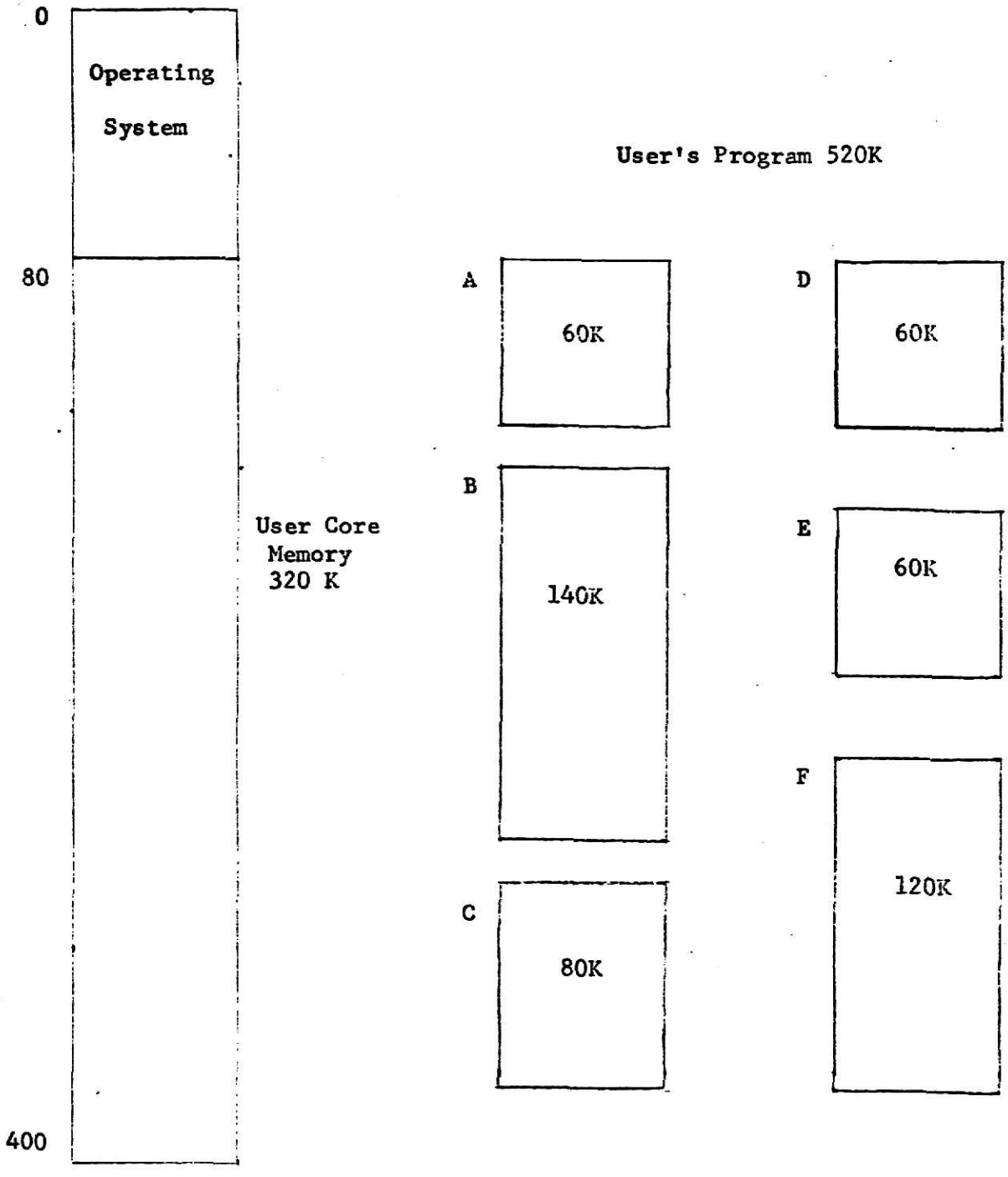

Figure 1-1

the sum of the address spaces of the jobs being multiprogrammed is permitted to exceed the physical size of main memory. This is illustrated in Figure 1-2. All three jobs are being executed in a physical memory space of 280K. The total sum of all three jobs is 460K.

The major constraint as to the size of the virtual space is limited by the hardware configuration. The hardware limits the number of addressable cells. The limiting factor is the number of bits used for an address. For example, if the hardware allows 8 bits for an address, then there are 256 addressable cells. The addresses would range from 0 to 255. This limits the virtual memory to this same size of space. The virtual space is usually considerably larger than the available primary memory of the machine.

1.2.2.1 VIRTUAL MEMORY TECHNIQUES

Two major virtual memory techniques are, demand-paged memory management (1,7) and segmented memory management (1). As was stated previously, virtual memory requires that the instruction and data to be operated on be located in primary memory. If this were to be done one instruction at a time it would be too time consuming. Instead they are retrieved in sections upon demand. If the sections are all equally divided into the same length, they are individually referred to as a page. This is the orgin of the term "demand-paged" memory.

Virtual Memory Structure

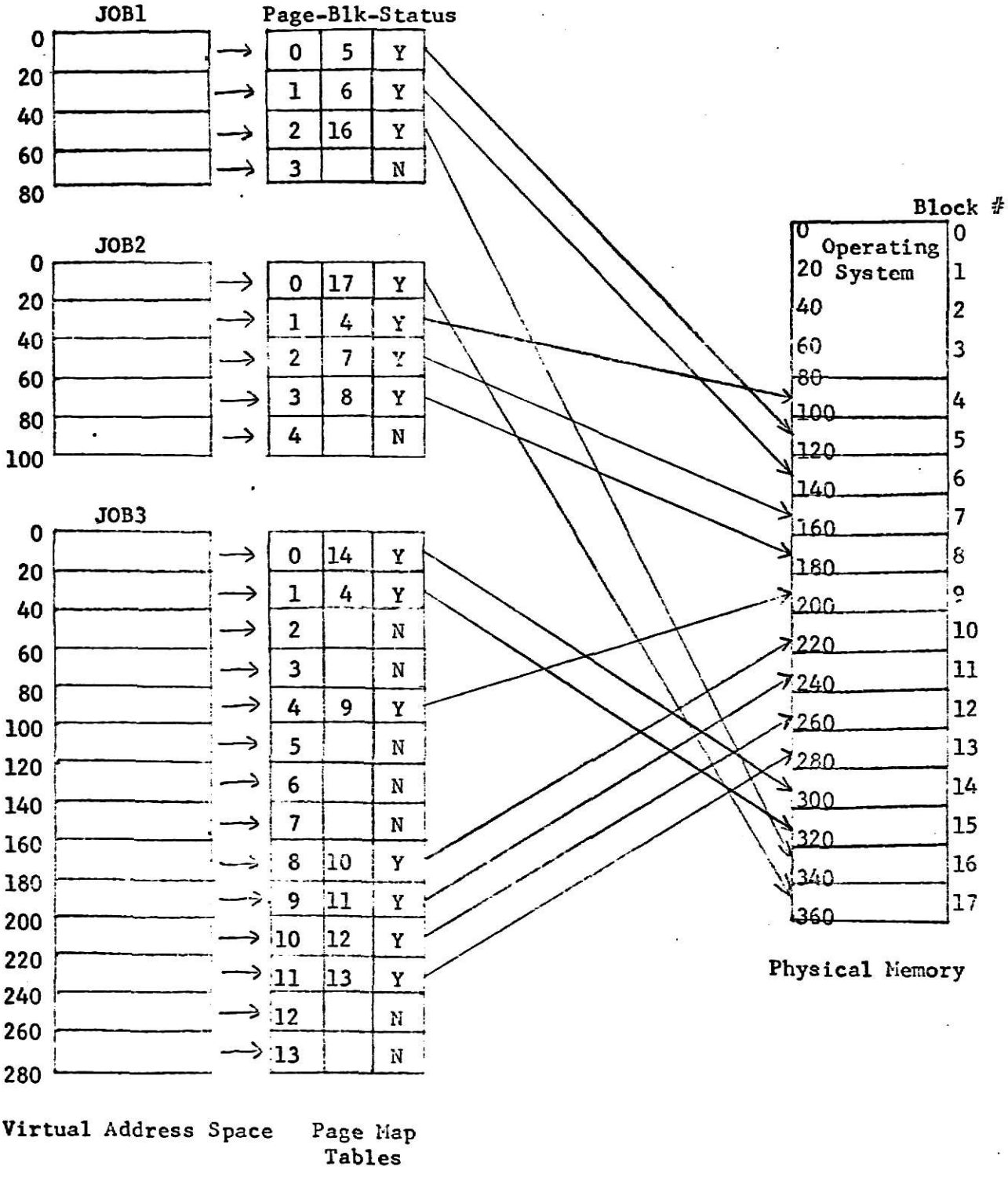

Figure 1-2

If the sections are unequal in length they are called segments, hence "segmented" memory. This paper will only be concerned with the former, demand-paged memory.

At the start of execution of a user's program, the first page is brought into primary memory. This is done by the virtual memory module, which will be explained in sextion 1.7.2. As each instruction is executed, the virtual memory module checks to make sure that all the address space referenced is in primary memory. If the address space is not in primary memory an interrupt, called a page fault (2), is generated. The operating system then processes this interrupt. This is done by loading the required page into primary memory. The process is then restarted from the point of the interrupt. Each additional required page is brought into primary memory upon request.

This can obviously lead to the point where primary memory is full when a new page is being requested by the virtual memory module to be brought into primary memory. To alleviate this situation page replacement (1) is necessary. This consists of removing from primary memory a page that does not have a high probability of being referenced in the near future. This page is then placed in secondary memory while the newly requested page is moved into primary memory.

1.2.2.2 ADVANTAGE OF VIRTUAL MEMORY

Virtual memory is commonly used today by many large computers. There are many advantages to be gained by using virtual memory. Some of these advantages include:

(1) Increase in the number of programs that can be multiprogrammed in a system.

(2) Capability of running a program whose address space exceeds the primary memory space currently available, if less than the maximum addressable memory.

(3) Makes programs more portable from large machines to small machines.

(4) Helps eliminate fragmentation of dynamic storage allocation.

These are especially appealing to mini-computers since mini-computers by nature have a smaller primary memory space.

1.3 SUPPORT FOR EMULATORS

For use in this paper, we will define an emulator to be a firmware interpreter. This interpreter will convert a user program from the original language, instruction by instruction, into the desired computer actions. An emulator written for this system must be aware of the manner in which to access the virtual addressing system of the host processor. An emulator will not create a machine language program. Instead, it will in effect execute a small microprogram for each instruction of the original language. The result of this activity will be the execution of the original instruction.

The host machine's user assembler language itself will be emulated by HIMICS to allow virtual memory capabilities. HIMICS will

also allow "high-level" Languages to be emulated. Languages such as PL/1, APL, and COBOL are likely candidates for emulation. These languages will require a large amount of space for their emulators, but due to the virtual storage capabilities of the system, this space requirement does not present a problem.

1.4 INTER-EMULATOR COMMUNICATION

In many programming situations, it is desirable to use different languages for different modules of the program. For example, one might want the processing portion of his algorithm to be coded in an assembler language, and the input/output sections written in a high-level language. Such process linkage will be allowed in this system. Interprocess communication will also be allowed in this system due to the capabilities of the host machine's operating system. A task or emulated process can start another task executing. After starting another task, the calling task may wait until the named task terminates. The calling task on the other hand may also continue processing and test for the called task's completion when necessary. A task can also cancel another task which is executing. This kind of communication is solely dependent upon the functions of the host operating system.

1.5 MANAGEABLE SOFTWARE

The key to manageable software is to keep it simple. This can be accomplished by using a structured design (3). In this

approach a complex system is divided into small independent modules. This allows one to comprehend each module without keeping the details of the entire system in mind. Furthermore, modifications to the system are simplified since a module can be changed or added without affecting other modules.

A part of this structured design is provided by the operating systems of the host processors. Even if the operating systems have to be modified to run in a virtual memory paging environment, it is worth the trade-off. These modules are not only structured, they will be almost error free from the start, and thus more manageable. Obviously it would require a great deal of time to produce the equivalent modules from scratch.

1.6 INSTRUMENTATION

1.6.1 RECORDED COUNTS

In order to evaluate a system's efficiency it is necessary to employ techniques to record the specific actions taken by the system. An obvious method of instrumentation is to keep a count of how many times a pre-specified event occurs. By knowing the system input, and the actions caused, it is possible to evaluate the system.

There are two counts which must be taken for every instruction. The first is a count for each unique operation code. When evaluating the system, the frequency of execution of each kind of instruction is essential. The second count records the number of times each

page is referenced. This will be used to evaluate the performance of the system under different paging options.

1.6.2 WORKING SET OPTIONS

Built into the system at system generation time is the ability to perform paging under one of three options. These three options have been choosen to yield different working set (1,8,9) sizes in order to evaluate the system under various work loads for maximum efficiency. In this paper, working set size refers to the number of pages contained in Level 1 and Level 2 memory. This composes a collection of the program's most recently used pages (11). The terms Level 1 memory and Level 2 memory will be explained in detail in section 1.8.2 of this chapter. The first option operates under a non-competitive fixed partitioned working set size. The second option will allow the system to operate with a competitive variable working set size based on a local paging rate. The third option, which operates on a competitive variable working set size also, is based on a global scale and allocates secondary memory using the LRU stack (1,10) principle.

Under option one, the total amount of secondary memory will be divided by the total number of jobs allowed to be multi-programmed. This will be set at system generation time. Each joo will then have a fixed working set the same size as any

other job. For example, if there are 180 page frames and 3 jobs, each job will have 60 page frames for its use. (See Figure 1-3A). Under option two the size of the working set for a job will be adjusted to approach maximum efficiency as best as can be determined. This will be based on a competitive paging ratio computed on a per-job paging rate over a given time period. Jobs having high paging rates tend to increase their working set size while low paging jobs decrease their working set size.

Option three will take the entire working set available and let all jobs have the space needed on a first come first use basis. This treats the Level 2 memory on a global basis, whereas under option one it was treated on a per-job basis or local level. This will allow, for example, two jobs, job 1 needing 20 page frames and job 2 needing 120 page frames, to be completely contained in Level 2 memory at once. (See Figure 1-3B). Under option one above, the fixed size per job, job 2 could only have 60 of its 120 pages in Level 2 memory at once.

Obviously in order to tell which of the above three methods is the best, monitoring of the different options is necessary. This will include a count of the number of page faults occurring out of each memory level for each job under the given option. There will also be an option to turn monitoring on or off.

1.7 OVERVIEW OF THE SYSTEM

The HIMICS system may be viewed as a hierarchical structure (4,11). A structure of this nature consists of modules located

Fixed Working Set Options

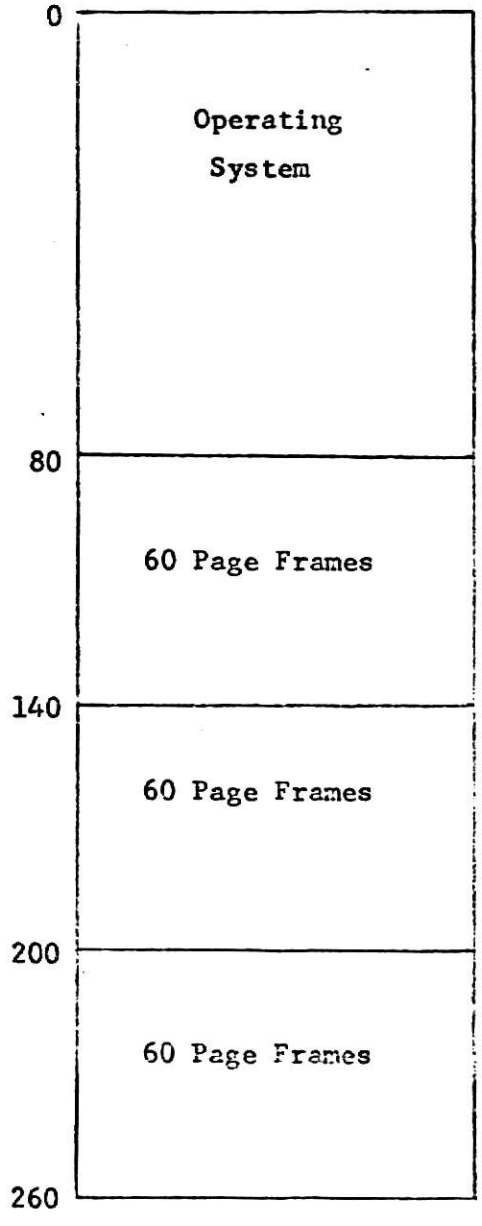

Figure 1-3A

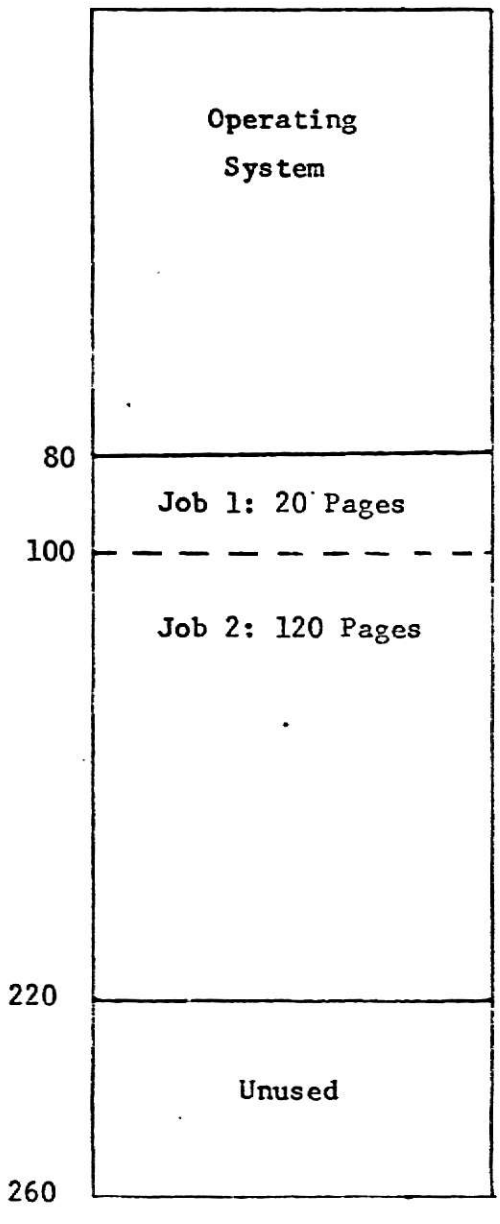

Figure 1-3B

Figure 1-3

on differert hierarchical levels (See Figure 1-4). This type of structure is called "layered insensitivity" (4,11). The levels are insensitive because each level is allowed to call upon the services of levels immediately above or below it in the structure, but not those levels farther than one level away. This means that each level is not concerned about how or where things are done in the levels above it or subordinate to it, and treats them all as one level. Each level may be referenced by the level above or below it in the hierarchy, but no level maybe dependent on a level which is not a logically sequential level in the hierarchy structure. For example, level 4 of the HIMICS system will interface with the file management system level 5, and virtual storage management level 3, but level 4 may not call upon levels 1, 2, or 6.

1.7.1 ADVANTAGES OF SYSTEM DESIGN

There are several advantages to this type of design.

(1) The system is easier to understand.

(2) Each module is easier to implement.

(3) The verification of the entire system is accomplished by verifying each individual level in a bottom-up fashion.

(4) Modification of the system is simplified.

(5) The software system is relatively portable (i.e. interfacing with different hardware requires only the lowest level of the system to be compatible, and the upper levels do not require modification).

Hierarchical Structure

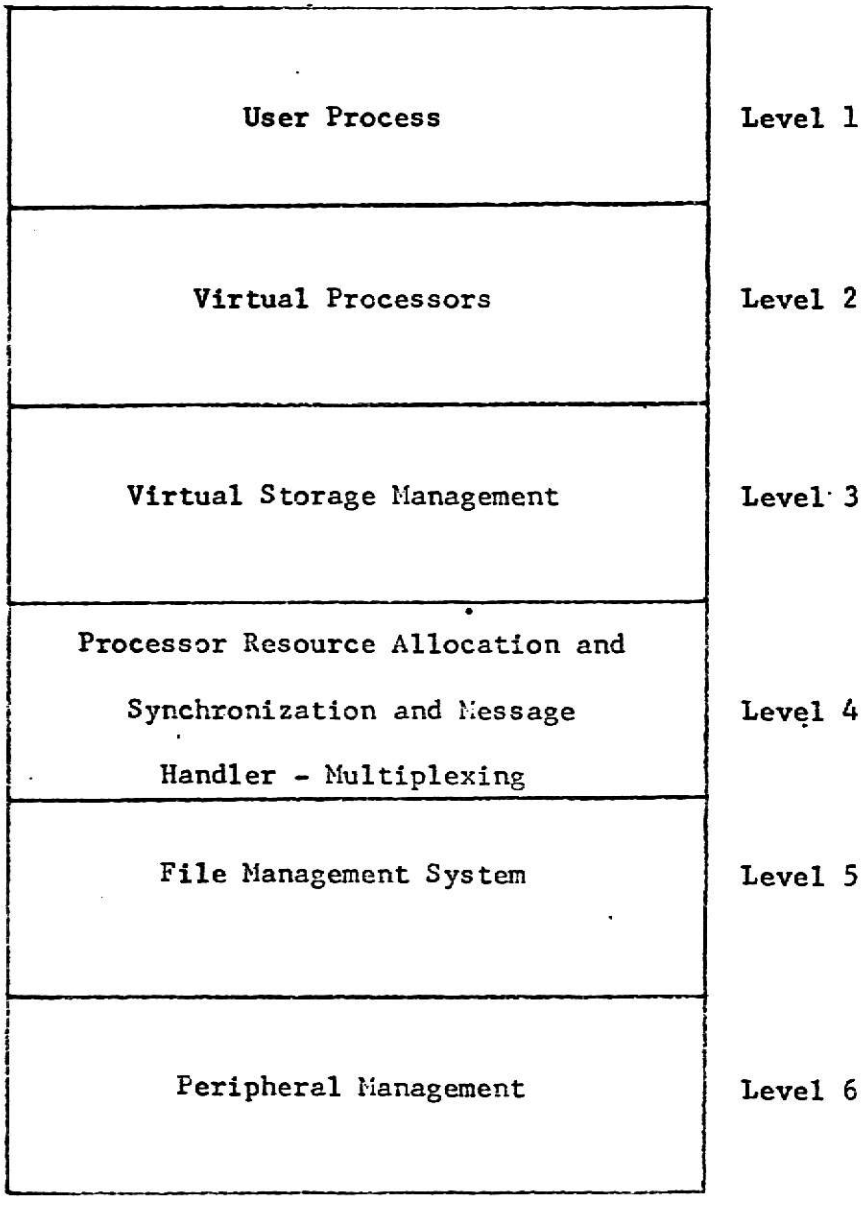

Figure 1-4

1.7.2 EXPLANATION OF LEVELS

A short explanation of what is contained in each module follows. Level 1 contains the user processes which are interpreted and executed in the primary memory of the host machine. These user programs may be written in any language supported by an emulator on the HIMICS system.

Contained in Level 2 are the virtual processors. This is the system of emulators which execute one instruction of the user's program at a time. Before each instruction is executed the current real address of the virtually addressed operands must be retrieved. Therefore all memory references must be detected and sent to level 3 to be converted before the emulation of the instruction may occur.

Level 3 contains the virtual storage management system. This system must detect any page faults which are generated from the virtual addresses of the instruction's operands. The virtual addresses of the operands must be converted to real machine addresses. This of course requires the page to be located in primary memory. This management system must interact with the file management system through the message handler in level 4 to retrieve pages to primary memory.

The message handler and resource allocation systems in level 4 are intermeshed deeply with the operating system of its host machine. The message handler is responsible for the generation

and control of all information transfer from the file management system. This communication will consist of I/O messages to and from level 6, page requests from secondary memory in level 5, and the current state of the system (i.e. "request page", "page being transferred"). This message exchange coordinates the activities of levels 5 and 6 with the upper 4 levels of the system.

The file management system will reside at level 5. It will handle all page requests between secondary and primary storage. All input and output messages will be processed by the file management system upon request. Included in the file management module will be tables which contain the current location of each job's pages. These tables are initialized when the job is started, and are updated as pages are moved from one level of memory to another. Level 4 will send page requests and I/O messages through the message handler requesting pages to satisfy page faults and I/O requests when needed.

Level 6 is the peripheral management module. This module will handle the interface with all peripheral devices connected to the system. This may include printers, card readers, teletypes, display terminals, or whatever hardware is available to interface with the system.

1.8 IMPLEMENTATION

We have just presented a machine independent description of the virtual addressing system for a mini-computer system. Now a

more detailed description of the implementation of the system at Kansas State University will be presented.

1.8.1 HARDWARE ALLOCATION

The layered insensitivity graph in Figure 1-5 makes the allocation of duties to actual hardware transparent. As can be seen, the upper four modules will be located in an Interdata 85 computer. The lower two levels will be located in a Nova computer.

The upper four levels are located in the Interdata machine because of its greater processing speeds. The extensive software required by the HIMICS system for each user instruction requires a fast processor. The Interdata cycle time of 270 nanoseconds meets these general speed requirements. The Nova machine is used as a peripheral processor. This processor will have more time to perform its duties, and yet removes a great processing overhead from the Interdata. This setup lets each machine do what it does best, and allows a more efficient and faster system. Using two CPU's in effect allows parallel processing. Real I/O may be supervised by the Nova while the Interdata is processing a user's program. Another view of the system design is given in Figure 1-6. The system shown is based on a multiprogramming environment of three users.

1.8.2 MEMORY LEVELS

It is apparent from Figure 1-6 that the use of two separate CPU's primary memory, and disk memory creates three levels of

Hardware View of Hierarchical Structure

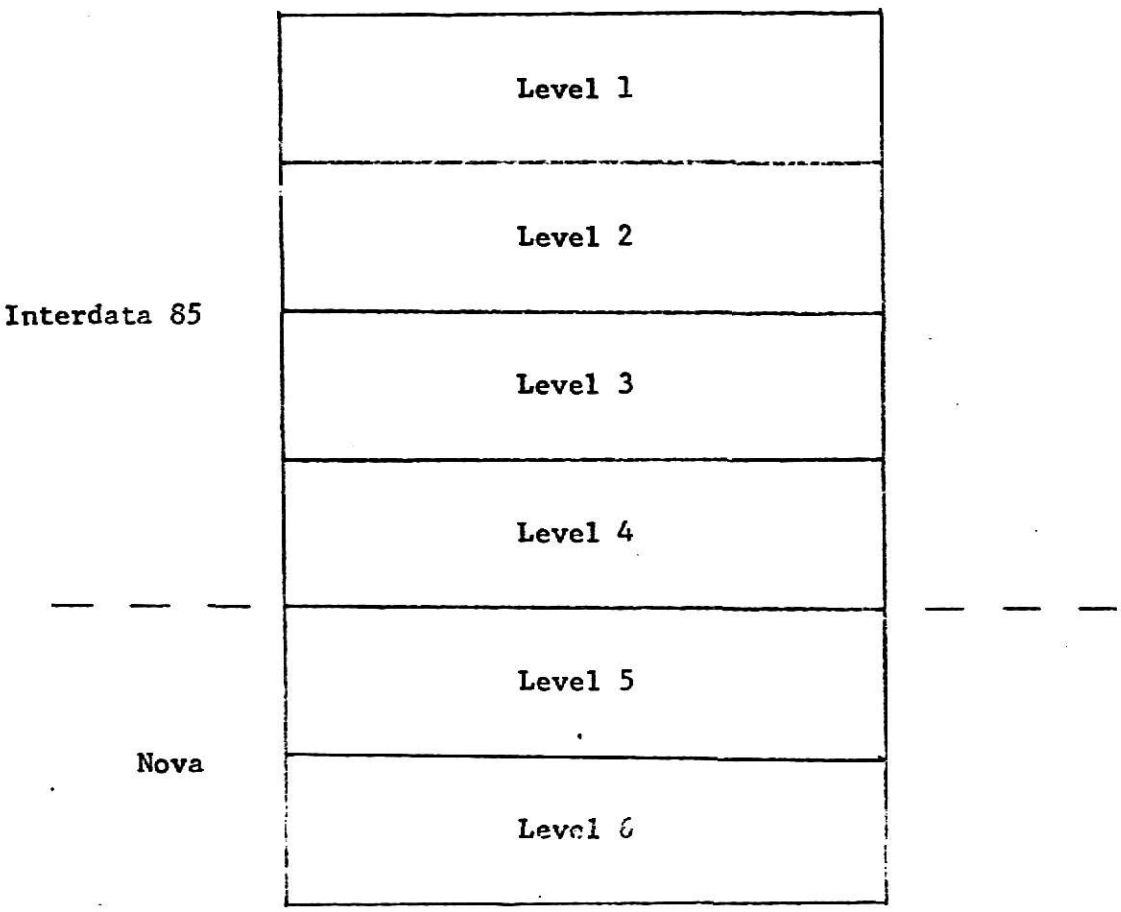

Figure 1-5

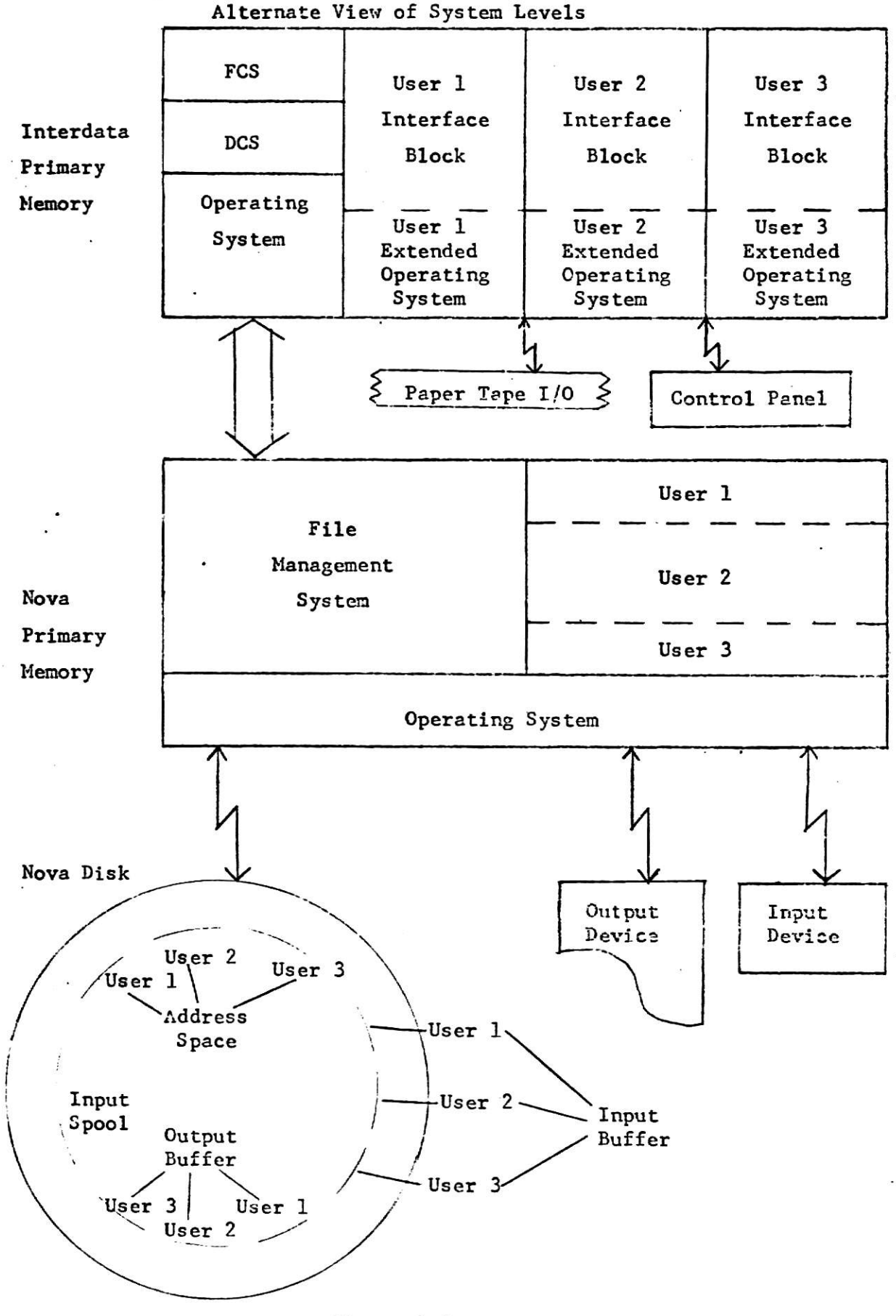
Alternate View of System Levels
Interdata Primary Memory
FCS
DCS
Operating System
User 1 Interface Block
User 2 Interface Block
User 3 Interface Block
User 1 Extended Operating System
User 2 Extended Operating System
User 3 Extended Operating System
Paper Tape I/O
Control Panel
Nova Primary Memory
File Management System
User 1
User 2
User 3
Operating System
Nova Disk
Output Device
Input Device
User 2
User 1
User 3
Address Space
Input Spool
Output Buffer
User 3
User 1
User 2
User 1
User 2
User 3
Input Buffer

Figure 1-6