A SWAPPING IMPLEMENTATION FOR THE MINIX OPERATING SYSTEM

VICENTE FRESQUEZ, JR.

Department of Electrical Engineering

APPROVED:

Dr. David H. Williams, Chair

Dr. Darrell C. Schroder

Dr. Daniel E. Cooke

Dean of the Graduate School

A SWAPPING IMPLEMENTATION FOR THE MINIX OPERATING SYSTEM

by

VICENTE FRESQUEZ, JR., B.S.E.E.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Electrical Engineering

THE UNIVERSITY OF TEXAS AT EL PASO

December 1990

ACKNOWLEDGEMENTS

There are many who helped while I did this research, but I would like to thank my thesis committee first. I thank Dr. Schroder for leading me into engineering in the first place and encouraging me to continue with the Computer Engineering program. I thank Dr. Cooke for showing me the science in Computer Science which helped me to avoid some of the difficulties that previous researchers had encountered. I especially thank my principal mentor, Dr. Williams. He encouraged me to move forward when I needed it and kept me pointed in the right direction when I had lost my way. I have known no finer educator and I am most fortunate to have worked with him. I also thank Jim Coffman, my supervisor at AT&T Bell Laboratories during this research, for providing me with the equipment and materials to carry out this research. My gratitude goes out to my friends in the Computer Laboratory who helped me keep things in perspective. Finally, I thank my parents and family for their understanding and encouragement. Most of all, I thank my wife, Julie, for her unconditional patience, care, and love during the one year ordeal. Things are looking up!

This thesis was submitted December 22, 1989.

ABSTRACT

This thesis discusses the implementation of swapping for the MINIX operating system on the IBM PC/AT. MINIX is a UNIX®-like operating system that was originally implemented on the IBM PC. The architecture of MINIX was limited by the architecture of the platform of the original implementation. Some of these limitations were eased by moving the platform to the IBM PC/AT, but the PC/AT had it's own limitations which impacted this implementation. These issues are discussed in further detail. The modifications made to MINIX in order to support swapping are discussed and the difference and code listings of these changes are presented in the appendices.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

In late 1987, a platform for image processing and embedded control systems was needed for research in the Department of Electrical Engineering. An inexpensive platform did not exist at that time, so the following performance characteristics were specified. Relatively large address spaces were needed for acceptable image processing performance. An embedded control system would need to be designed for reliability since the system could be physically located in a location that makes access impractical. (A satellite is a good example.) A reliable design would need to include such features as memory protection against errant programs and other protection mechanisms. To make system prototyping and development easier, widely used or standard hardware and software architectures was desired. Both applications would require the ability to interface to various peripheral devices with ease.

With these requirements in mind, a hardware platform was selected first. There was experience in the department with a variety of processors, but most of the experience was with the DEC™ PDP-11™ [1], Int$_e$l®[2], and the Motorola family

[1] DEC and PDP-11 are trademarks of Digital Equipment Corporation.

[2] Int$_e$l is a registered trademark of Intel Corporation

of processors. The experience with the Motorola processors were with the smaller processors, and the address space was insufficient. The DEC PDP-11 family of processors appeared to be reaching the end of their lifetimes. In addition, the address space on most of the PDP-11 line was too small for image processing. For these reasons, the PDP-11 family was not selected. Experience with the Intel 8086 family of processors was more favorable. The higher end processors had a suitable address space for image processing applications as well as hardware mechanisms for memory protection. The Intel 8086 was used in one of the most widely available computers ever known: the IBM®[3] PC. The higher end Intel 80286 was also used in a platform that at that time was gaining wide acceptance: the IBM PC/AT™ [4]. (At the time, the Intel 80386 was not yet as well accepted as the 80286.) The PC/AT was almost as widely available as the PC, and had a well defined hardware interface, just as the PC did. Since it met the needed criteria, the IBM PC/AT was selected as the hardware platform for development.

With the hardware chosen, a software platform needed to be found. One of the most common operating systems used on the IBM PC/AT was MS-DOS® from

[3] IBM is a registered trademark of International Business Machines Corporation.

[4] PC/AT is a trademark of International Business Machines Corporation.

Microsoft®[5]. The problem with MS-DOS was that it would not allow direct access to more than 640k of memory. This would be unduly limiting to image processing applications. Although there existed standard methods for adding drivers for new hardware with MS-DOS, the source code for the operating system itself was unavailable. Clearly MS-DOS was an unacceptable choice.

The UNIX®[6] operating system and other similar systems were another popular operating system for the IBM PC/AT. The UNIX operating system had the added benefits of supporting multitasking and multiuser operation, which would be useful for many applications. The prospect of adding new drivers without source code, however, was unappealing to say the least. The source code for the UNIX operating system itself was too expensive to buy for this purpose. The XINU operating system was somewhat similar to the UNIX operating system, and source code was available for it, but it was not available for the IBM PC yet [1]. Another operating system like the UNIX operating system was MINIX [2]. MINIX also was available with source code, and was available for the IBM PC. Although MINIX did not meet all of the previously mentioned criteria, it could be readily modified to do so. So the choice for the software platform was MINIX, and a list of required changes was made. The required changes were to allow for large address spaces and to

[5] Microsoft and MS-DOS are registered trademarks of Microsoft Corporation.

[6] UNIX is a registered trademark of AT&T.

provide memory protection from errant and malicious processes.

A change that would take care of both of these problems would be the addition of virtual memory to MINIX. In this discussion, the following definition of virtual memory given in [3] will be used:

> The memory management scheme called *virtual memory* allows execution of processes even when only portions of their address spaces are resident in primary memory.

The classic paper in virtual memory is by Denning [4], which is definitely recommended reading. A more up to date paper, also by Denning, is [5]. The previous implementations of virtual memory were built upon paging architectures, such as the implementation described in [6], or a segmented-paging architecture, such as Multics [7]. Unfortunately, the Intel 80286 was strictly a segmented architecture [8]. This fact would make the implementation a fairly interesting prospect.

The remainder of this thesis discusses the attempt of a virtual memory implementation and the subsequent swapping implementation that followed. The next chapter discusses the architecture of the hardware platform and the software modifications needed to make use of the hardware support for the protection of memory. The following chapter discusses the software architecture of MINIX and the changes needed to implement swapping. The last chapter discusses the results and observations of this research, and gives some suggestions as to what steps may be taken to

provide a more satisfactory platform.

Chapter 2

HARDWARE ARCHITECTURE AND SOFTWARE MODIFICATIONS FOR PROTECTED MODE

This chapter discusses the architecture of the IBM PC and IBM PC/AT, and the implications of those architectures on the MINIX Operating System. The central processing units (CPU) are discussed first, followed by other pertinent details of the memory layout of the system. The ROM BIOS (Read Only Memory Basic Input/Output System) is also briefly discussed. To illustrate how the host architecture affects the operating system, a presentation of the way the original MINIX system handled context switching and interrupts is made. Finally, the changes made in order for MINIX to run in protected mode on the Intel 80286 are discussed.

2.1 A Discussion of the IBM PC Architecture

The IBM PC is one of the most widely spread microcomputer platforms available today. This is what made it attractive for MINIX to be developed to run on the IBM PC. Dr. Tanenbaum wanted students to be able to modify and, thus, learn about operating systems using readily available hardware [9]. Though most of the implementation is unencumbered by limits placed on it by the hardware, some compromises were made. Notably, memory made available to processes was somewhat small. (Please note that Dr. Tanenbaum did not consider this to be a

compromise.) This limited process space is what drove this project to find a suitable method for allowing processes to be larger than 64k text and 64k data. A discussion of the appropriate points of the architecture follows.

2.1.1 The Int$_e$l 8086/8088

The CPU for the IBM PC (and the PC/XT) is the Int$_e$l 8088, a close relative to the 8086 [10]. The Int$_e$l 8086 is a 16 bit microprocessor with a 20 bit address space. The Int$_e$l 8088 differs only slightly with the primary difference being that all bus transfers occur 8 bits at a time. In this discussion, 8086 will be used interchangeably with 8088. The 8086 has 4 16 bit registers that double as 8 bit register pairs, 5 16 bit pointer registers, a processor status word, and 4 segmentation registers. The interrupt facility allows for 256 maskable vectored interrupts and a nonmaskable interrupt. Each interrupt service routine is pointed to by a fixed location in low memory. Interrupts may also be software generated.

The 20 bit address space allows for 1 megabyte of accessible memory. The segmentation implementation allows for the use of 16 bit offsets into a given segment to locate code or data. The 20 bit address is generated by adding the contents of the appropriate segment register, left shifted by 4 bits, to the offset from the instruction. The segment register to be used is implied by the instruction, although it may be overridden by an instruction prefix. The use of segmentation allows for

relative position independence of programs and data that fit within a 16 bit (or 64k) address space. This relocation capability is valuable in operating system implementation. The shifting of the value of the segment register by 4 bits lends to a name for 16 byte quantities of memory: paragraphs or clicks.

Other than the relocation capability of the segment registers, the 8086 has no memory management capabilities. This implies that the operating system has to provide protection against damage to process memories. That is not a trivial task. In order for MINIX to provide total protection of process memory, each instruction would have to be inspected before execution to prevent errant changes to the segment registers or the use of segment-offset combinations that point to memory outside of the process space. MINIX instead opts to ignore most of dangers of errant processes. It does try to deal with the common problem of stack overflow by making sure that, at process startup, a gap exists between the data for a process and the stack for that process. The penalty of this is that the stack segment register must point to the same segment as the data segment register.

2.1.2 The Memory Map of the IBM PC

The IBM PC does not allow access to all of the 1 megabyte address space. Instead, only the lower 640 kilobytes is available and part of this is reserved for the interrupt vectors. Addresses above 640k or 0xA0000 (where 0xNNNNN is a hexa-

decimal number) are reserved for the BIOS ROMs, video adapter screen memory, and other ROMs, notably BASICA [2]. This leaves 10 non-overlapping segments for use by the operating system and the programs. This outlook for available memory space is what drove this investigation for another platform, namely the IBM PC/AT.

2.2 The Architecture of the IBM PC/AT

While not yet as widely available as the IBM PC or PC/XT, the IBM PC/AT is also very popular. The standard configuration comes with 1 megabyte of usable memory, which is more than what was available on the PC. Most PC/AT's have a fixed disk offering secondary memory with tolerable speed and reliability. The PC/AT uses an Intel 80286 as the CPU. This processor has built in memory management and protection facilities. As a more suitable platform for use with MINIX, the PC/AT appeared to be good choice. A further detailed discussion of the architecture follows.

2.2.1 The Intel 80286

The Intel 80286 is an upwards compatible processor of the 8086 family that includes memory management capabilities (see [8], [11], [12], and [13]). It provides access to a 24 bit, 16 megabyte physical address space. The virtual address space is 29 bits or 1 gigabyte. 512 megabytes of the virtual address space is shared

amongst all processes. The other half a gigabyte may be different for each process. The location of the interrupt vectors may be any place in the memory space. The access of memory is protected by the use of 4 different privilege levels and segment limit checking.

The 80286 is able to run in two different modes, real and protected. In the real mode, the 80286 acts like a fast 8086. In the protected mode, the way segment registers are dealt with is significantly different. Most of the above listed features are not available in real mode. In real mode, the segment register value is the starting click of the segment. In protected mode, the value in the segment register is a segment selector. This selector indexes into one of two tables to find where the segment is physically located. The selector contains 13 bits of an index (8k selectors), 1 table selecting bit, and 2 privilege level bits. The entry found at the index of the table selected by the segment selector is called the descriptor. All descriptors that are available to all processes, or global, are in the Global Descriptor Table (GDT). All descriptors that are available only to a particular process are in a particular Local Descriptor Table (LDT). Contained in a segment descriptor for code or data are:

- a 16 bit zero word for compatibility with the Intel 80386.
- an 8 bit byte defining the access rights for the segment.

- a 24 bit base address defining the physical location of the segment.
- a 16 bit limit defining the length of the segment in bytes.

The access byte defines whether or not a segment is present in memory, the privilege level of the segment, the type of segment, and whether the segment has been accessed. The types of segments used by protected mode MINIX in the GDT or LDT are, mostly, either code or data segments. Code segments may be read or executed. Data segment may be read or written. Other operations on these segments generate processor exceptions. Any offset that is larger than the limit in the descriptor of the segment in use will cause a processor exception. If a segment register is loaded with a selector that references a descriptor that is marked not present, a processor exception occurs. If a segment register is loaded with a selector that references a descriptor of a higher priority, typically, an exception is raised. Privilege transitions that are acceptable are discussed later.

The GDT and LDT must be placed somewhere in memory and the CPU notified of where they are. These need not be fixed locations as the interrupt vectors for the 8086. Instead, the 80286 provides special registers to designate where in physical memory these tables are. The GDT register (GDTR) is a five byte register with 3 bytes for the base address, and 2 bytes for the limit on the length of GDT. To load this register, the processor must be running at the highest privilege level and pass a pointer to six bytes (the last byte is ignored) to the LGDT instruction.

(After this is done, the programmer may start protected mode by setting a bit in the Machine Status Word (MSW). Once in protected mode, the processor must be reset to return to real mode.) The LDT register (LDTR) is a 2 byte user visible register with an associated 5 byte user invisible descriptor cache. The LDTR is loaded under the same conditions as the GDT except that a selector of the GDT must be passed to the LLDT instruction. The selector must index to a GDT descriptor that has the type field set to type LDT descriptor. In that descriptor, the base address and the table length limit is found. The IDT has an IDTR that is analogous to the GDTR. It is loaded using the LIDT instruction under the same conditions as the GDTR.

The IDT table may only contain three types of descriptors: interrupt gates, trap gates, and task gates. The vector used in real mode to index into low memory on the 8086 is now used to index into the IDT on the 80286. The action taken depends on the type of the descriptor found. Protected mode MINIX only uses interrupt gates, so that is what will be discussed. An interrupt gate contains an access byte, a 16 bit selector (14 bits used), a 16 bit offset into the selected segment, and various padding fields to pad to descriptor size. The privilege level in the access byte determines the minimum privilege level allowed to generate that interrupt using a software interrupt instruction.

At this point, a small discussion of privilege level transitions is appropriate. The 80286 will only allow a process access to segments with descriptor access bytes containing privilege levels (DPL) that are numerically greater than or equal to the current privilege level (CPL). The CPL is determined every time a privilege level transition occurs. The exceptions to the above rule are the cases of dealing with stack segments or when making a transition because of an interrupt or jump/call through a gate. The rule used for stack segments is that each of the four levels must have it's own stack segment descriptor and stack pointer. The rule for privilege transition is that the new privilege must be of the same or greater (numerically lesser) privilege level. When a transition occurs, the code segment register (CS), instruction pointer (IP), processor status word, previous stack segment register, and stack pointer are placed on the stack at the new privilege level. The new stack segment register and stack pointer are loaded from the Task State Segment (TSS, discussed later). The new CS and IP were loaded from the gate. Execution now begins at the new privilege level. At the end of the interrupt service routine, a return from interrupt instruction is executed. This will restore the process back to it's original stack and privilege level.

The TSS provides hardware support for context switching. There are entries in it for all parts of a process's hardware context. Hardware safeguards are used to prevent the storage of a process context into a TSS that is currently storing another

context. An entry for the LDTR is also in the TSS so that each process may have it's own LDT. The location of the current task's TSS is in the Task Register (TR) which contains a selector in the GDT with a type TSS descriptor.

2.2.2 The Memory Map of the IBM PC/AT

The IBM PC/AT also does not allow full access to all 16 megabytes of physical memory. Instead, just over 15.5 megabytes is accessible to the user. The remainder is assigned to ROMs and video display memory. In [14], the following memory map is given:

- from 0x000000 to 0x09FFFF is assigned to bank 0 of motherboard RAM.
- from 0x0A0000 to 0x0BFFFF is assigned to video display RAM.
- from 0x0C0000 to 0x0DFFFF is assigned to I/O expansion ROM.
- from 0x0E0000 to 0x0FFFFF is assigned to BIOS ROM and USER ROM.
- from 0x100000 to 0x15FFFF is assigned to bank 1 of motherboard RAM.
- from 0x160000 to 0xFDFFFF is assigned to I/O memory expansion.
- from 0xFE0000 to 0xFFFFFF is mapped back to BIOS and USER ROMs.

The initial 640k of bank 0 of the motherboard and 384k of bank 1 are typically filled on the standard configuration for the AT. Note that to access the 384k above 1 megabyte, the processor must be running in protected mode.