BrokenThorn Entertainment

Operating Systems Development Series

Operating Systems Development - Basic Theory

by Mike, 2009

This series is intended to demonstrate and teach operating system development from the ground up.

Introduction

Welcome to the wonderful and crazy world of operating systems!

In the previous tutorial, we have defined what an operating system is. An operating system provides the basic interface between the user and the computer system. It provides the basic look and feel for the system.

We have also taken a look at alot of tools that will help us. Assemblers, Compliers, Linkers, PartCopy, MagicISO, and Bochs.

For anyone who has no programming experence who is reading this (I know there is some) shame on you :) Please go back and reread the "Prerequisites" section from the first tutorial. Why are you still reading me? Go go go!

In this article, we are going to look at operating systems in a different way. We will first go Back in Time(tm) to look at the history of operating systems. You will find there are many similarities between these operating systems. These similarities will then be classified into the basic things operating systems have in common--and the building blocks for your own.

Blast from the Past

Most of todays operating systems are graphical. These graphical user interfaces (GUI), however, provide a large abstraction layer to what is really going on in an OS.

Alot of the concepts about operating systems date back sinse programs were on tape. Alot of these concepts still remain active today.

Prehistory - The Need for Operating Systems

Prior to the 1950s, all programs were on punch cards. These punch cards represented a form of instructions, which would control every faucet of the computer hardware. Each peice of software would have full control of the system. Most of the time, the software would be completely different with each other. Even the versions of a program.

The problem was that each program was completely different. They had to be simply because they had to be always rewritten from scratch. There was no common support for the software, so the software had to communicate directly with the hardware. This also made portability and compatibility impossible.

During the realm of Mainframe computers, creating code libraries became more feasable. While it did fix some problems, such as two versions of software being completely different, each software still had full control of hardware.

If new hardware came out, the software will not work. If the software crashed, it would need to be debugged using light switches from a control panel.

The idea of an interface between hardware and programs came during the Mainframe era. By having an abstraction layer to the hardware, programs will no longer need to have full control, but instead they all would use a single common interface to the hardware.

What is this ultra cool interface? Why, its that sweet cuddley (sometimes nasty) thing we call an Operating System! :)

1950s - Yes there were OSs then

The first real operating system recorded, according to Wikipedia, is the GM-NAA I/O. The SHARE Operating System was a successor of the GM-NAA I/O. SHARE provided sharing programs, managed buffers, and was the first OS to allow the execution of programs written in Assembly Language. SHARE became the standard OS for IBM computers in the late 1950s.

The SHARE Operating System (SOS) was the first OS to manage buffers, provide program sharing, and allow execution of assembly language programs.

"Managing Buffers" relate to a form of "Managing Memory". "Program Sharing" relates to using libraries from different programs.

The two important things to note here are that,since the beginning of time (Not really :)), **Operating Systems are responsible for**

Memory Management and Program Execution/Management

Because this isnt the history of the world (nor computers) that I am describing, lets jump now ahead to good old DOS.

1964 - DOS/360 and OS/360

DOS/360 (or just "DOS") was a Disk Operating System was originally announced by IBM to be released on the last day of 1964. Do to some problems, IBM was forced to ship DOS/360 with 3 versions, each released June 1966.

The versions were:

- BOS/360 - 8KB Configuation.
- DOS/360 - 16KB Configuation with disk.
- TOS/360 - 16KB Configuation with tape.

A couple of important things to note is that DOS/360 offered no **Multitasking**, and no **Memory Protection**. The OS/360 was being developed about the same time by IBM. The OS/360 used "OS/MFT" (Multiple Fixed Transactions) to support multiple programs, with a **Fixed Base Address**. With OS/MVT (Multiple Varable Transaction), it can support varies program sizes.

Now we have a few more interesting words--**Multitasking, Memory Protection,** and **Fixed Base Address**. Adding to before, we also have **Program execution** and **Memory Management**.

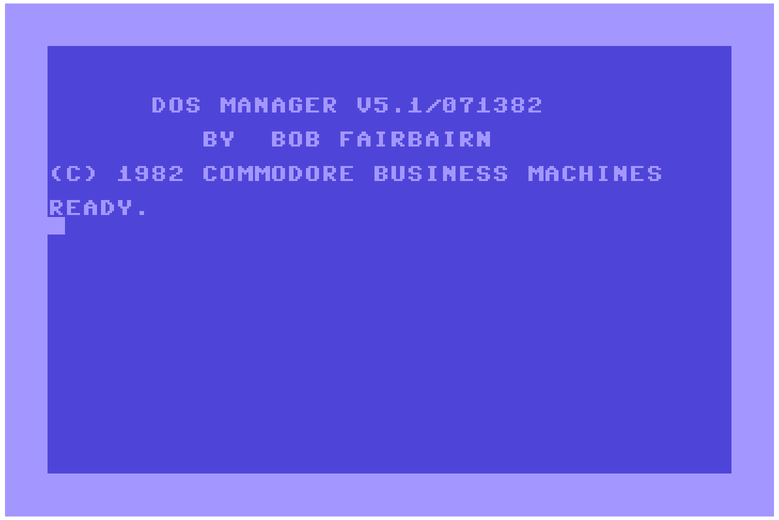
1969 - Its Unix!

The Unix Operating System was originally written in C. Both C and Unix were originally created by AT&T. Unix and C were freely distributed to government and acedimic institutions, causing it to be ported to a wider varity of machine families then any other OS.

Unix is a **multiuser**, **Multitasking** Operating System.

Unix includes a **Kernel**, **File System** and a **Command Shell**. There are alot of **Graphical User Interfaces (GUI)** that uses the **Command Shell** to interact with the OS, and provide a much friendler and nicer look.

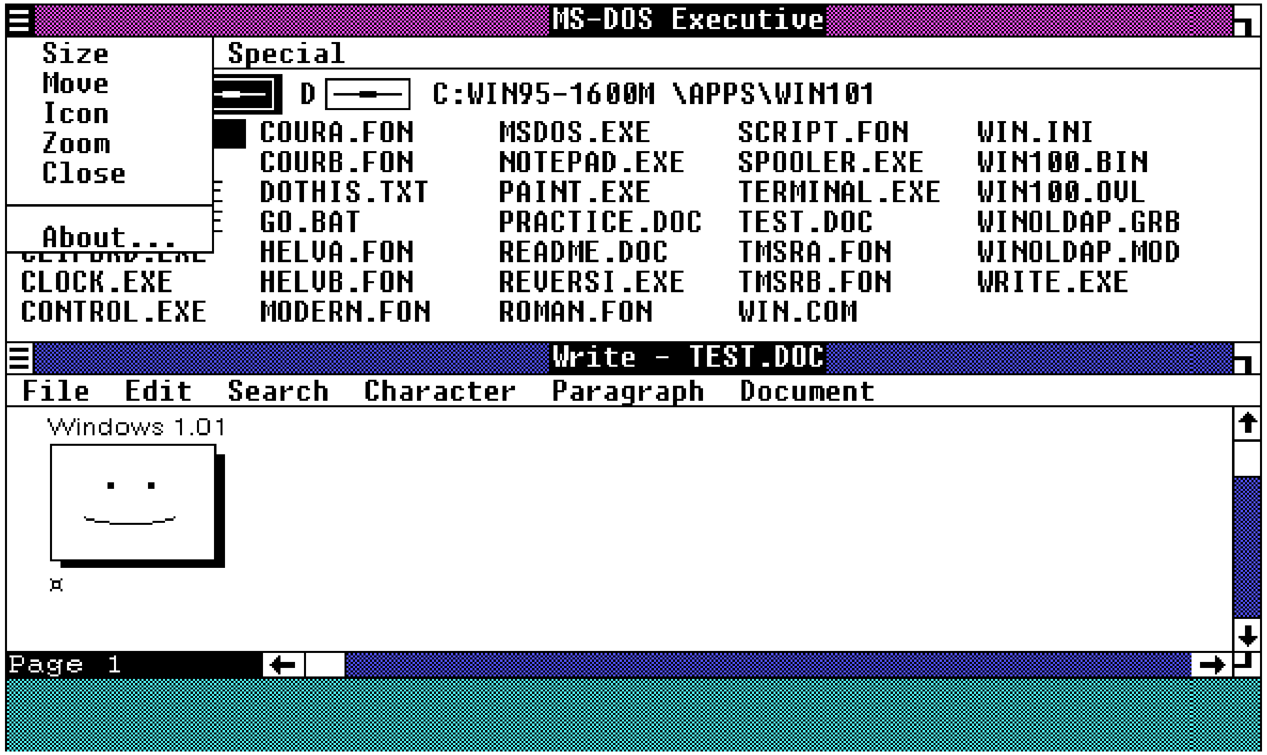
1982 - Commodore DOS



```
      DOS MANAGER V5.1/071382
         BY  BOB FAIRBAIRN
(C) 1982 COMMODORE BUSINESS MACHINES
READY.
```

Commodore DOS (CBM DOS) was used with Commodore's 8 bit computers. Unlike the other computers before or since-which booted from disk into the systems memory at startup, CBM DOS executed internally within the drive-internal ROM chips, and was executed by an MOS 6502 CPU.

1985 - Microsoft Windows 1.0

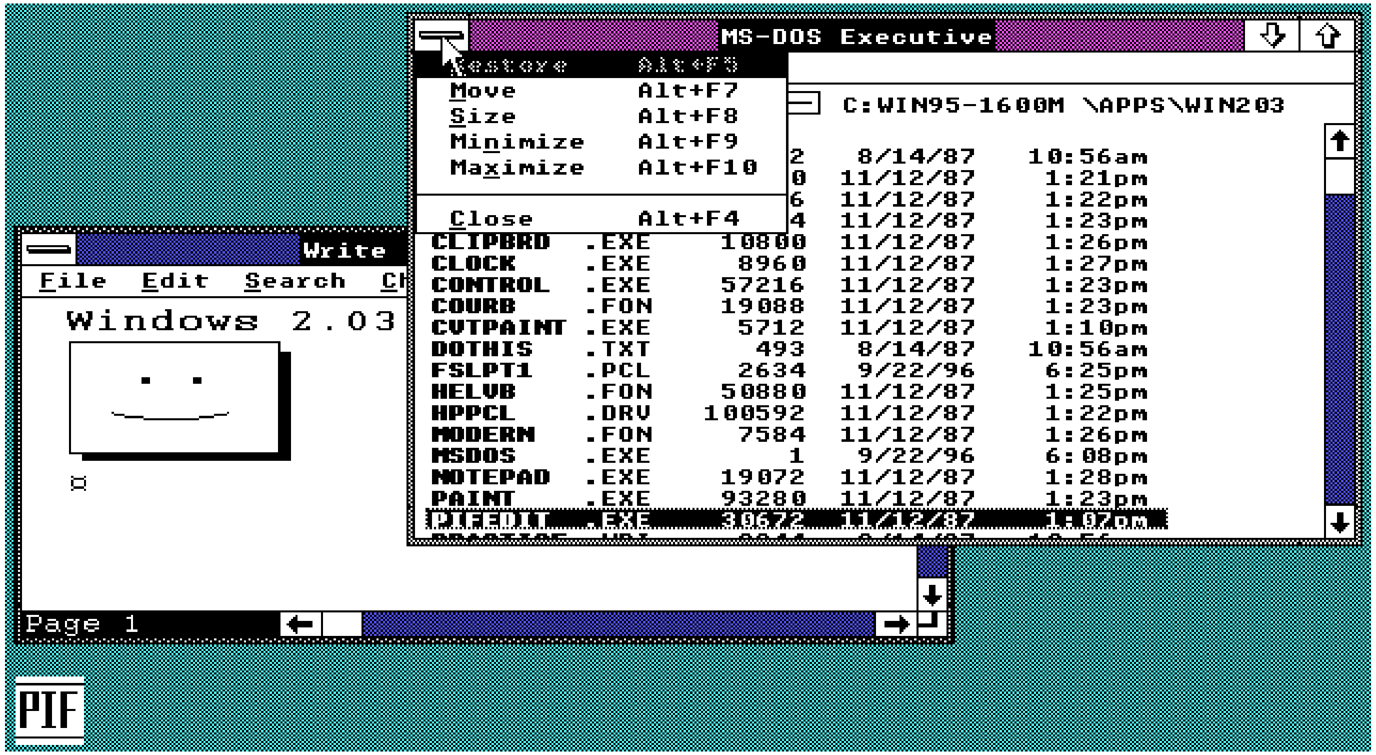


```
D  C:WIN95-1600M \APPS\WIN101

               COURA.FON     MSDOS.EXE     SCRIPT.FON    WIN.INI
               COURB.FON     NOTEPAD.EXE   SPOOLER.EXE   WIN100.BIN
               DOTHIS.TXT    PAINT.EXE     TERMINAL.EXE  WIN100.OVL
               GO.BAT        PRACTICE.DOC  TEST.DOC      WINOLDAP.GRB
               HELVA.FON     README.DOC    TMSRA.FON     WINOLDAP.MOD
CLOCK.EXE      HELVB.FON     REVERSI.EXE   TMSRB.FON     WRITE.EXE
CONTROL.EXE    MODERN.FON    ROMAN.FON     WIN.COM
```

The first Windows was a DOS application. Its "MSDOS Executive" program allows the running of a program. None of the "Windows" could overlap, however, so each "window" was displayed side to side. It was not very popular.

1987 - Microsoft Windows 2.0



```
C:WIN95-1600M \APPS\WIN203
          2   8/14/87   10:56am
          0  11/12/87    1:21pm
          6  11/12/87    1:22pm
          4  11/12/87    1:23pm
CLIPBRD  .EXE   10800  11/12/87    1:26pm
CLOCK    .EXE    8960  11/12/87    1:27pm
CONTROL  .EXE   57216  11/12/87    1:23pm
COURB    .FON   19088  11/12/87    1:23pm
CVTPAINT .EXE    5712  11/12/87    1:10pm
DOTHIS   .TXT     493   8/14/87   10:56am
FSLPT1   .PCL    2634   9/22/96    6:25pm
HELVB    .FON   50880  11/12/87    1:25pm
HPPCL    .DRV  100592  11/12/87    1:22pm
MODERN   .FON    7584  11/12/87    1:26pm
MSDOS    .EXE       1   9/22/96    6:08pm
NOTEPAD  .EXE   19072  11/12/87    1:28pm
PAINT    .EXE   93280  11/12/87    1:23pm
PIFEDIT  .EXE   30672  11/12/87    1:07pm
```

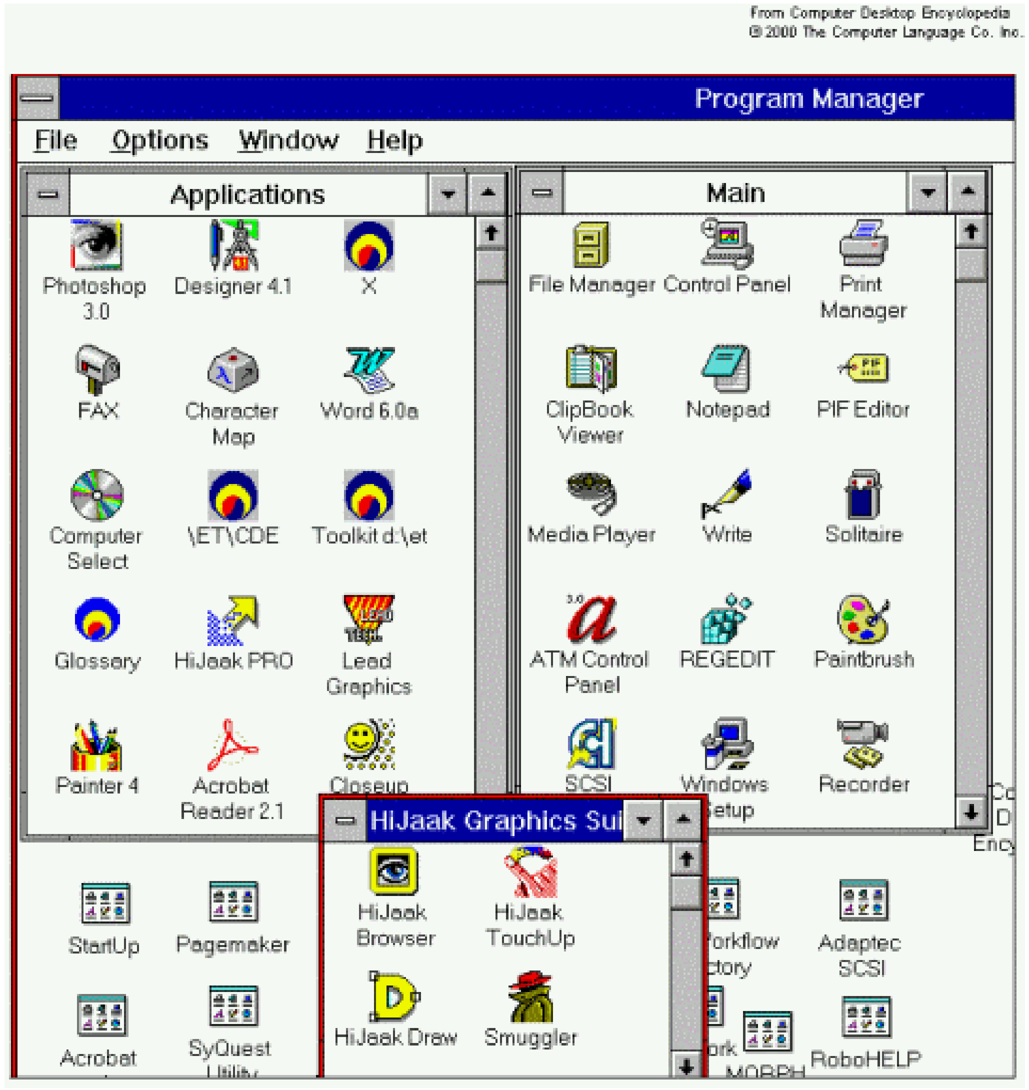
The second version of Windows was still a DOS **Graphical Shell**, but supported overlapping windows, and more colors. However, do to the limitation of DOS, it was not widley used.

Note: DOS is a 16 bit operating system. During this timeframe, DOS had to refrence memory through Linear Addressing, and disks through LBA (Linear Block Addressing). Because the x86 platform is backward compatible, When the PC boots it is in 16 bit mode (Real Mode), and still has LBA. More on this later.

Do to 16 bit mode limitations, DOS could not access more then 1 MB of memory. This is solved, today, by enabling the 20th address line through the Keyboard Controller. We will go over this later.

Because of this 1 MB limitation, Windows was far to slow, which was one primary reason of it being unpopular.

1987 - Microsoft Windows 3.0



Windows 2.0 was completely redesigned. Windows 3.0 was still a DOS Graphical Shell, however it included A "DOS Extender" to allow access to 16 MB of memory, over the 1 MB limit of DOS. It is supports **multitasking** with DOS programs.

This is the Windows that made Microsoft big. It supportes resizable windows, and movable windows.

Windows in relation to OS Developers

I have seen quite a few beginning OS developers want to develope the next Windows. While it is possible, it is extremily difficault, and is impossible with a one person team. Take a look at the above picture again. Remember that it is a **graphical shell** over a **command shell**, being executed by a **Kernel**. Also, remember that even Windows had to start here. The Command Shell was DOS. the Graphical Shell was "Windows".

Basic Concepts

Looking back though our small trip to memory lane brings some important new terms with it. So far, we only gave "operating system" a small definition. The previous section should help us in defining a better, more descriptive definition of what an operating system is.

To help define a better definition, lets put the above bolded terms into a list:

- Memory Management
- Program Management
- Multitasking
- Memory Protection
- Fixed Base Address
- Multiuser
- Kernel
- File System
- Command Shell
- Graphical User Interface (GUI)
- Graphical Shell
- Linear Block Addressing (LBA)
- Bootloader (From the previous tutorial)

Thats alot to think about, huh? And yet--The above list is techincally still an abstraction layer itself.

Lets take a look closer, shall we?

Memory Management

Memory Management refers to:

- Dynamically giving and using memory to and from programs that request it.
- Implimenting a form of **Paging**, or even **Virtual Memory**.
- Insuring the OS Kernel does not read or write to unkown or invalid memory.
- Watching and handling **Memory Fragmentation**.

Program Management

This relates cosely with Memory Management. Program Management is responsible for:

- Insuring the program doesnt write over another program.
- Insuring the program does not currupt system data.
- Handle requests from the program to complete a task (such as allocate or deallocate memory).

Multitasking

Multitasking refers to:

- Switching and giving multiple programs a certain timeframe to execute.
- Providing a **Task Manager** to allow switching (Such as Windows Task Manager).
- **TSS (Task State Segment) switching. Another new term!**
- Executing multiple programs simulataniously.

Memory Protection

This referrs to:

- Accessing an invalid descriptor in protected mode (Or an invalid segment address)
- Overwriting the program itself.
- Overwriting a part or parts of another file in memory.

Fixed Base Address

A "Base Address" is the location where a program is loaded in memory. In normal applications programming, you wouldnt normally need this. In OS Development, however, you do.

A "Fixed" Base Address simply means that the program will always have the same base address each time it is loaded in memory. Two example programs are the BIOS and the Bootloader.

Multiuser

This refers to:

- Login and Security Protection.
- Ability of multiple users to work on the computer.
- Switching between users without loss or curruption of data.

Kernel

The Kernel is the heart of the Operating System. It provides the basic foundation, memory management, file systems, program execution, etc. We will take a closer look at the kernel very soon, dont worry :)

File System

In OS Development, there is no such thing as a "file". Everything could be pure binary code (from the bootsector); from the start.

A File System is simply a specification that describes information reguarding files. In most cases, this referrs to Clusters, Segments, segment address, root directories, etc. the OS has to find the exact starting address of the file in order to load it.

File Systems also describe file names. There are **external** and **internal** file names. For example, the FAT12 specification states a filename can only be 11 characters. No more, no less. Seriously. This means the filename "KRNL.sys", for example, will have the internal file name

```
"KRNL    SYS"
```

We will be using FAT12 and be discussing it in detail later.

Command Shell

A Command Shell sits ontop the Kernel as a seperate program. The Command Shell provides basic input and output through the use of typing commands. The Command Shell uses the Kernel to help with this, and complete low level tasks.

Graphical User Interface (GUI)

The Graphical User Interface (GUI) simply refers to the graphical interface and interactions between the Graphical Shell and the user.

Graphical Shell

The Graphical Shell provides video routines and low level graphical abilities. It normally will be executed by the Command Shell. (As in Windows 1.0,2.0, and 3.0). Useually this is automatic these days, however.

Linear Block Addressing (LBA)

Operating Systems have control over **every single little byte in memory**. Linear Addressing refers to directly accessing linear memory. For example:

```
mov ax, [09000h]          ; There is no such thing as Access Violations in OS Development
```

This is a good thing, but is also a very bad thing. For example:

```
		mov bx, [07bffh]		; or some other address less then 7c00h
		mov cx, 10
	.loop1:
		mov [bx], 0h		; clear bx
		inc	bx			; go to next address
		loop .loop1			; loop until cx=0
```

The above code seems harmless. However, if the above code was found in a bootloader, the above code will overwrite itself by 10 bytes. Ouch. The reason is that bootloaders are loaded with a Fixed address of 0x7c00:0, and the above code starts writing from 07bffh: One byte before 07c00h.

Bootloader

The bootloader. We seen this term from the previous tutorial. From the previous tutorial, we know the bootloader is loaded by the BIOS, and is the very first program to execute for your operaing system.

The bootloader is loaded by the BIOS at absolute address 0x7c00:0. After loading, CS:IP is set to your bootloader, and the bootloader takes full control.

A Floppy Sector is only 512 bytes in size. Remember that the bootloader has to fit in a single bootsector. What does this mean? The bootloader is very limited in size, and cannot exceed 512 bytes.

Most of the time, the bootloader will either just load and execute the kernel, or a **Second Stage Bootloader**.

We will take a closer look at the booting process very soon. That is when we will look at bootloaders.

Conclusion

We have taken a look into the past, and learned a few more terms to our list. After the history lesson, we took the terms and built a more broader perspective on how everything works. We even got to see some code. A small amount, though.

After all of this, one should be able to develope a more concise definition of what we are doing.

"An interactive envirement that provides an interface for the user and supplied programs, providing a stable and safe envirement, an interface layer to System services and computer hardware."

Yep. Thats my new definition of "Operating System", what is yours?

In the next tutorial, we are going to take a look on the booting process in descrete detail. Afterwords, we will take a look at building and assembling of a real bootloader.

Until next time,

~Mike
BrokenThorn Entertainment. Currently developing DoE and the Neptune Operating System

Questions or comments? Feel free to Contact me.

Would you like to contribute and help improve the articles? If so, please let me know!

Chapter 1 Home Chapter 3