

Chapter 1

History and Overview

1.1. Reasons for Studying Operating System

A modern computer system consists of one or more processors, some main memory, disks, printers, a keyboard, a display, network interfaces, and other input/output devices. All in all, a complex system. Writing programs that keep track of all these components and use them correctly, let alone optimally, is an extremely difficult job. For this reason, computers are equipped with a layer of software called the **operating system**, whose job is to manage all these devices and provide user programs with a simpler interface to the hardware. An operating system is similar to a **government**. Like a government, it performs no useful function by itself. It simply provides an environment within which other programs can do useful work

At the bottom is the hardware, which, in many cases, is itself, composed of two or more levels (or layers). The lowest level contains physical devices, consisting of integrated circuit chips, wires, power supplies, cathode ray tubes, and similar physical devices.

Next comes the **microarchitecture level**, in which the physical devices are grouped together to form functional units. Typically, this level contains some registers internal to the CPU (Central Processing Unit) and a data path containing an arithmetic logic unit. In each clock cycle, one or two operands are fetched from the registers and combined in the arithmetic logic unit (for example, by addition or Boolean AND). The result is stored in

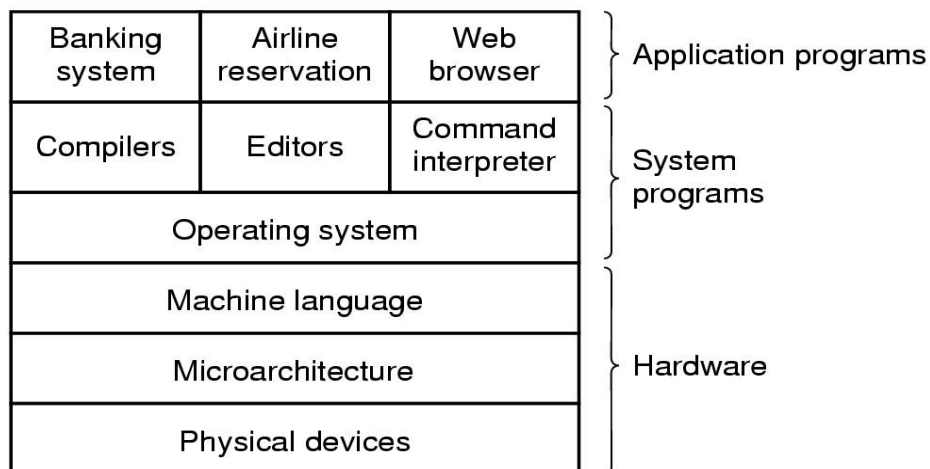


Figure 1.1 A computer system consists of hardware, system programs, and application programs

one or more registers. On some machines, the operation of the data path is controlled by software, called the **microprogram**.

The hardware and instructions visible to an assembly language programmer form the **ISA (Instruction Set Architecture) level**. This

level is often called **machine language**. The machine language typically has between 50 and 300 instructions, mostly for moving data around the machine, doing arithmetic, and comparing values. In this level, the input/output devices are controlled by loading values into special **device registers**.

Furthermore, for many I/O (Input/Output) devices, timing plays an important role in the programming. To hide this complexity, an operating system is provided. It consists of a layer of software that (partially) hides the hardware and gives the programmer a more convenient set of instructions to work with.

On top of the operating system is the rest of the system software. Here we find the command interpreter (shell), window systems, compilers, editors, and similar application-independent programs. It is important to realize that these programs are definitely not part of the operating system; the operating system is (usually) that portion of the software that runs in **kernel mode** or **supervisor mode**.

1.2. Important Areas

The area where most of the priority level the concurrency, memory management and file management are used in the field of ticket booking (Railways, airlines) weather forecasting, Banking and health and science.

1.2.1. The Operating System as an Extended Machine

The **architecture** (instruction set, memory organization, I/O, and bus structure) of most computers at the machine language level is primitive and awkward to program, especially for input/output. The function of the operating system is to present the user with the equivalent of an extended machine or virtual machine that is easier to program than the underlying hardware.

The computer system is quite complex to work with. It would be extremely difficult to program computers at the lowest level. For example, reading and writing to a disk include loading the address of the disk block in a special register; identify the type of disk involved, rotating the disk, moving the read/write arm to the appropriate spot on the disk, and managing inter sector gaps, among others. Programming would have been a frightening if a programmer had to face this complexity. Fortunately, the operating system is there to cover the complexity of the hardware and other low-level features and present a nice and simple to use computer system.

The operating system presents an extended machine or a virtual machine that is simple to use and easier to program. It provides windows through which users can access the hardware in the form of **various**

services. Programmers use these services by issuing *special instructions* called **system calls**.

1.2.2. *The Operating System as a Resource Manager*

Modern computers consist of a variety of devices, each with different characteristics. As a resource manager, the operating system provides an orderly and controlled allocation of resources among the various programs competing for them. In this view, the operating system's primary task is to keep track of who is using which resource, to grant resource requests, to account for usage, and to mediate conflicting requests from different programs and users.

Imagine what would happen if three programs running on some computer all tried to print their output simultaneously on the same printer. The first few lines of printout might be from program 1, the next few from program 2, then some from program 3, and so forth. The result would be chaos.

Resource management includes multiplexing (sharing) resources in two ways: ***in time*** and ***in space***. When a resource is time multiplexed, different programs or users take turns using it. First one of them gets to use the resource, then another, and so on. For example, with only one CPU and multiple programs that want to run on it, the operating system first allocates the CPU to one program, then after it has run long enough, another one gets to use the CPU, then another, and then eventually the first one again.

The other kind of multiplexing is space multiplexing, instead of the customers taking turns, each one gets part of the resource. For example, main memory is normally divided up among several running programs, so each one can be resident at the same time (for example, in order to take turns using the CPU).

1.3. Purpose of an operating System

What is the big deal about operating systems in the first place? What do they actually do? An **operating system** is the level of programming that lets you do things with your computer. The operating system interacts with a computer's hardware on a basic level, transmitting your commands into language the hardware can interpret. The OS acts as a platform for all other applications on your machine. Without it, your computer would just be a paperweight.

At its heart, a computer is a number-crunching device. It takes input in the form of zeros and ones -- **bits** -- and channels them through various circuits and processors. The hardware behaves according to strict rules. We define these rules using things like **logic gates**, which take input and

produce an output in a predictable way. Some simple computers have no need of an operating system because they only perform a specific task. However, personal computers need to be more versatile. The operating system allows complex programs to access the capabilities of the hardware to get results. Only the hardware's physical properties and our own imaginations can limit what programs can do.

You could design an operating system by physically programming it into a computer's circuits. This would require building electrical pathways using millions of logic gates. However, such an operating system would be inflexible. That is why operating systems like Mac OS X and Windows are software. Software is more malleable than hardware -- you can make changes through software patches and version updates. To do the same with hardware would mean switching out physical chips and circuit boards.

Operating systems are like the manager for a computer. It is the job of the OS to monitor what software needs and what the hardware can provide. As you run applications on your computer, the OS allocates the resources necessary to complete the task. That can include processing power, memory allocation and computer storage access, among other things. Ideally, the OS will make sure that your computer's hardware is never overtaxed.

The OS also allows programs to run on a computer. Without an OS, a programmer would have to design an application to run on the hardware directly. This is not very efficient. An operating system acts as an application interface to the hardware. The OS does this through an application program interface (API). Program developers build applications for the API. Assuming the programmer has done a good job at building an application without any serious bugs, it should run just fine on the operating system.

One important part of the Mac computer is the firmware. **Firmware** is a level of programming that exists directly on top of a hardware layer. It is not part of the operating system itself. The Mac firmware is the first stored program that executes when you turn on a Mac computer. Its job is to check the computer's CPU, memory, disk drives and ports for errors. The PC equivalent to the Mac firmware is called BIOS, which stands for basic input-output systems. A second program called a bootloader loads the Mac OS X, assuming there are no errors reported by the firmware.

1.4. Functions of operating system

An operating system has **three** major functions.

I. Controlling operations (control program)

- ✓ Coordinates, or supervises the activity of the computer system.
- ✓ Decides where programs and data should be stored in the computer memory.

- ✓ Handles communications among the computer components, applications software and the user.
- ✓ Controls the saving and retrieving of files to and from disks in the disk drive.
- ✓ It performs all its controlling tasks without the involvement or awareness of the **user**.

II. Input/output Management

The Operating System acts as the manager of resources such as **CPU time, memory space, file storage, I/O devices**. Since there may be many conflicting requests, Operating System allocates resources in an optimal manner. That is, Operating System allocates resources in such a manner so as to achieve the maximum best possible result.

- ✓ The I/O manager coordinates the computers communication with outside world, flow of data to the display screen and other output devices (printers/ plotters) and from the key board or other input devices.
- ✓ Handles the flow of data to and from the disk drives (file management).
- ✓ Handles the process of preparing a disk for use, the copying, renaming, erasing task of a file.

III. Command Processing (command Interpreter)

- ✓ It interprets the commands or what you enter using the keyboard or other input devices.
- ✓ If you write an internal command it carries out the function of that command, if it is external command or other executable file it searches for the corresponding file in the default (current disk) or the user specified disk, loads the file into memory and transfers control to that program. Once this program is terminated, control returns to command.com and the program for that command or file is discarded from memory.

1.5. History of Operating Systems

Operating systems evolve through time. For the sake of discussion, and as operating systems are highly related to the structure of the underlying hardware, the evolution is presented in **generations** of computers.

1.5.1. First Generation (1945-1955)

On the first generation, vacuum tubes were used as main circuitry component of the computer. The general features of all of them are large size, slow processing power, high heat, very high price tag, and very complex to use. There were **NO OPERATING SYSTEMS**. Those machines were designed, built, programmed, operated, and maintained by a small group of people. All programming was done in **absolute machine code**; by plugging and unplugging wires on **plug boards**. For their primitive nature, those machines were used for limited simple numerical applications.

1.5.2. Second Generation (1955-1965)

The replacement of **vacuum tubes** by **transistors** changed the characteristics of the hardware dramatically. There was even separation of designers, builders, programmers, operators, and maintenance personnel. **Punched cards** were used for input. Programmers used to write programs on paper and encode it by punching into cards. The cards were delivered to operators who put them into the card reader of the computer to run the encoded program. Other cards, such as cards for compilers and data, which are essential to run the program, are also loaded. The whole card set forms a job.

Much time was wasted while operators walk around the machine room bringing in new jobs and delivering the output of a completed job. To reduce the impact of this problem, a new method of running jobs, called **batch system**, was adopted. In a batch system, cards of several jobs were read into a tape drive and the tape is delivered to the main processing machine. The operator rewinds and mounts the tape containing the batch of jobs on the tape drive and loads a special program that reads the first job, run it, and continues to the next job automatically. The special program that reads and runs individual jobs from the tape was a very rudimentary operating system. The output of jobs was written to a second tape. When the batch of jobs completed, the operator replaces both the input and output tapes by the next input tape and another output tape respectively. The removed output tape is mounted on another machine to print the output on paper.

Third Generation (1965-1980)

ICs replaced **transistors** and much better computers were produced. A new feature of this generation was production of family of computers. Computers of a family vary in computational power and price but share the same architecture and support similar software. The line of computers satisfies the needs of many customers of varying needs and purchasing ability.

A major breakthrough of the third generation was the introduction of **multiprogramming**. The main memory is partitioned into several pieces and each partition holds a different job, as depicted in Figure 1.2. While one job was waiting for I/O to complete, another job could be allowed to use the CPU. If enough jobs could be held in main memory at once, the CPU could be, at least theoretically, kept busy nearly 100 percent of the time. Third generation computers were equipped with the hardware required to keep multiple jobs safely in memory without having each job messing with others.

Another major feature of the third generation was **spooling**. (Simultaneous Peripheral Operation online). In spooling jobs are read from cards onto the disk as soon as they were brought to the computer room. The operating system loads a new job from disk when the running job is terminated. Spooling removes the need for separate machines to read jobs on cards to tapes and the time wasted by operators carrying tapes around the computer room.

Multiprogramming is essentially a **batch system**. As such it lacks fast response. The desire for quick response time gave way to a variant of multiprogramming known as **timesharing**. In **timesharing systems**, multiple users log on to a computer through terminals and the computer serve them all. The response time can be varied according to the interactivity of the application. Interactive applications that require quick response can be used alongside batch systems which typically do not need quick response. The first timesharing system appeared at the end of the

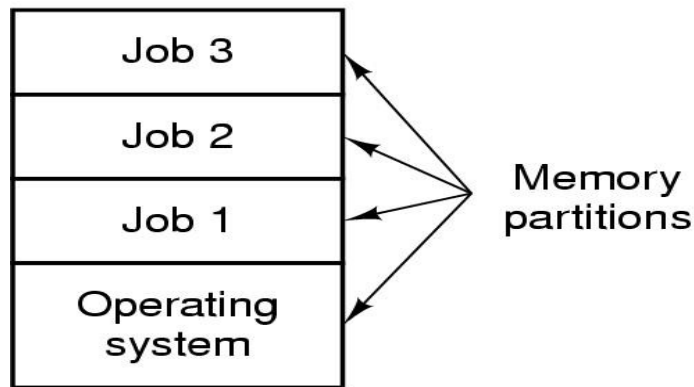


Figure 1.2 A multiprogramming system with three jobs in memory

second generation but was not popular until the necessary protection hardware became widely available during the third generation.

Fourth Generation (since 1980)

The development of **large scale integration (LSI)** gave birth to personal computers (also known as **microcomputers**). The fourth generation is dominated by personal computers. Different operating systems were developed for the newly developed microcomputers. Early operating systems of this era were **command based**. **CP/M**, **MS-DOS** and **UNIX** were popular command based operating systems of early fourth generation. User friendly GUI operating systems (Mac OS, Microsoft Windows, Windows XP, Linux, etc.) were developed later on. Starting from the mid-1980s, **network operating systems** and **distributed operating systems** are used on personal computers.

Types of operating system

1. Mainframe Operating Systems

Mainframes are used mainly in areas where there is a need for large data processing capacity. The operating systems of mainframes are required to process many tasks at once, most of which require large amount of I/O. They typically offer three types of services: **batch**, **transaction processing**, and **timesharing**. *Batch systems process routine jobs with less user interaction. Transaction processing systems handle large number of small-sized works. Timesharing systems allow large number of interactive users to run jobs simultaneously.*

2. Server Operating Systems

The next high end operating systems are server operating systems, which run on servers. They serve multiple users at once over a network and allow them to share hardware and software resources.

3. Multiprocessor Operating Systems

One way of getting more computing power is connecting multiple CPUs on a single system. The flavors are parallel computer, multicomputer, and multiprocessor systems. They need operating systems that are similar to server operating systems.

4. Personal Computer Operating Systems

These are the most popular operating systems. Their main job is to provide a good interface to a single user. They are typically used on single processor personal computers.

5. Real-Time Operating Systems

Real-time systems are characterized by their time sensitive nature. That means they are expected to meet deadlines. Real-time systems can be either hard real-time systems (in which deadlines must be met) or soft real-time systems (in which occasional failure of meeting deadlines is acceptable).

6. Embedded Operating Systems

These operating systems are used on systems with limited resources (memory, processing power, size and power). These systems are generally specific purpose computers. They include palmtop computers (PDAs, personal digital assistance), mobile telephones, TV sets, microwave ovens, etc.

7. Smart Card Operating Systems

The smallest operating systems run on smart card. Smart cards are small cards containing a CPU chip. These systems have very severe processing power and memory constraints. Some of them can handle only a single function while others can handle multiple functions.

1.6. The Meaning of an interrupt

An **interrupt** is the automatic transfer of software execution in response to a hardware event that is asynchronous with the current software execution. This hardware event is called a **trigger**. The hardware event can either be a busy to ready transition in an external I/O device (like the UART input/output) or an internal event (like bus fault, memory fault, or a periodic timer). When the hardware needs service, signified by a busy to ready state transition, it will request an interrupt by setting its trigger flag. A **thread** is defined as the path of action of software as it executes. The execution of the interrupt service routine is called a background thread. This thread is created by the hardware interrupt request and is killed when the interrupt service routine returns from interrupt (e.g., by executing a **BX LR**). A new thread is created for each interrupt request.

It is important to consider each individual request as a separate thread because local variables and registers used in the interrupt service routine are unique and separate from one interrupt event to the next interrupt. In a **multi-threaded** system, we consider the threads as cooperating to perform an overall task. Consequently we will develop ways for the threads to communicate (e.g., FIFO) and to synchronize with each other. Most embedded systems have a single common overall goal. On the other hand, general-purpose computers can have multiple unrelated functions to perform. A **process** is also defined as the action of software as it executes. Processes do not necessarily cooperate towards a common shared goal. Threads share access to I/O devices, system resources, and global variables, while processes have separate global variables and system resources. Processes do not share I/O devices.

There are no standard definitions for the terms mask, enable, and arm in the professional, Computer Science, or Computer Engineering communities. Nevertheless, in this class we will adhere to the following specific meanings. To **arm** a device means to allow the hardware trigger to interrupt. Conversely, to **disarm** a device means to shut off or disconnect the hardware trigger from the interrupts. Each potential interrupting trigger has a separate arm bit. One arms a trigger if one is interested in interrupts from this source. Conversely, one disarms a trigger if one is not interested in interrupts from this source. To **enable** means to allow interrupts at this time. Conversely, to **disable** means to postpone interrupts until a later time. On the ARM Cortex-M processor there is one interrupt enable bit for the entire interrupt system. We disable interrupts if it is currently not convenient to accept interrupts. In particular, to disable interrupts we set the I bit in **PRIMASK**. In C, we enable and disable interrupts by calling the functions **EnableInterrupts()** and **DisableInterrupts()** respectively.

1.7. Concurrency

Concurrency is the tendency for things to happen at the same time in a system. Concurrency is a natural phenomenon, of course. In the real world, at any given time, many things are happening simultaneously. When we design software to monitor and control real-world systems, we must deal

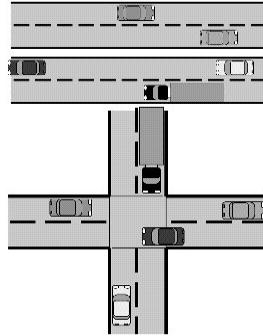


Figure 0.3 Example of concurrency at work

with this natural concurrency.

When dealing with concurrency issues in software systems, there are generally two aspects that are important: being able to detect and respond to external events occurring in a random order, and ensuring that these events are responded to in some minimum required interval.

In computer science, **deadlock** refers to a specific condition when two or more processes are each waiting for another to release a resource, or more than two processes are waiting for resources in a circular chain (see *Necessary conditions*). Deadlock is a common problem in multiprocessing where many processes share a specific type of mutually exclusive resource known as a *software*, or soft, lock. Computers intended for the *time-sharing* and/or *real-time* markets are often equipped with a *hardware lock* (or *hard lock*) which guarantees *exclusive access* to processes, forcing serialization. Deadlocks are particularly troubling because there is *no general* solution to avoid (soft) deadlocks.

This situation may be understood by an analogy with two people who are drawing diagrams, with only one pencil and one ruler between them. If one person takes the pencil and the other takes the ruler, a deadlock occurs when the person with the pencil needs the ruler and the person with the ruler needs the pencil, before he can give up the ruler. Both requests can't be satisfied, so a deadlock occurs.

Important to understand Concurrency

If the elevator system only had one elevator and that elevator had only to serve one passenger at a time, we might be tempted to think we could handle it with an ordinary sequential program. Even for this “simple” case, the program would require many branches to accommodate different conditions. For example, if the passenger never boarded and selected a

floor, we would want to reset the elevator to allow it to respond to another call.

The normal requirement to handle calls from multiple potential passengers and requests from multiple passengers exemplifies the external driving forces for concurrency we discussed earlier. Because the potential passengers lead their own concurrent lives, they place demands on the elevator at seemingly random times, no matter what the state of the elevator. It is extremely difficult to design a sequential program that can respond to any of these external events at any time while continuing to guide the elevator according to past decisions.

1.8. Scheduling and Dispatch

Almost all programs have some alternating cycle of CPU number crunching and waiting for I/O of some kind. (Even a simple fetch from memory takes a long time relative to CPU speeds.) In a simple system running a single process, the time spent waiting for I/O is wasted, and those CPU cycles are lost forever.

A scheduling system allows one process to use the CPU while another is waiting for I/O, thereby making full use of otherwise lost CPU cycles.

- ✚ Whenever the CPU becomes idle, it is the job of the CPU Scheduler (a.k.a. the short term scheduler) to select another process from the ready queue to run next.
- ✚ The storage structure for the ready queue and the algorithm used to select the next process are not necessarily a FIFO queue. There are several alternatives to choose from, as well as numerous adjustable parameters for each algorithm.

1.8.1. Preemptive Scheduling

- CPU scheduling decisions take place under one of four conditions:
 1. When a process switches from the running state to the waiting state, such as for an I/O request or invocation of the wait() system call.
 2. When a process switches from the running state to the ready state, for example in response to an interrupt.
 3. When a process switches from the waiting state to the ready state, say at completion of I/O or a return from wait().
 4. When a process terminates.
- For conditions 1 and 4 there is no choice - A new process must be selected.
- For conditions 2 and 3 there is a choice - To either continue running the current process, or select a different one.
- If scheduling takes place only under conditions 1 and 4, the system is said to be **nonpreemptive**, or **cooperative**. Under these conditions, once a process starts running it keeps running, until it either voluntarily blocks or until it finishes. Otherwise the system is said to be **preemptive**.

- Windows used non-preemptive scheduling up to Windows 3.x, and started using preemptive scheduling with Win95. Macs used non-preemptive prior to OSX, and preemptive since then. Note that preemptive scheduling is only possible on hardware that supports a timer interrupt.
- Note that pre-emptive scheduling can cause problems when two processes share data, because one process may get interrupted in the middle of updating shared data structures. Chapter 6 will examine this issue in greater detail.
- Preemption can also be a problem if the kernel is busy implementing a system call (e.g. updating critical kernel data structures) when the preemption occurs. Most modern UNIXes deal with this problem by making the process wait until the system call has either completed or blocked before allowing the preemption. Unfortunately this solution is problematic for real-time systems, as real-time response can no longer be guaranteed.
- Some critical sections of code protect themselves from concurrency problems by disabling interrupts before entering the critical section and re-enabling interrupts on exiting the section. Needless to say, this should only be done in rare situations, and only on very short pieces of code that will finish quickly, (usually just a few machine instructions.)

Dispatcher

- The **dispatcher** is the module that gives control of the CPU to the process selected by the scheduler. This function involves:
 - Switching context.
 - Switching to user mode.
 - Jumping to the proper location in the newly loaded program.
- The dispatcher needs to be as fast as possible, as it is run on every context switch. The time consumed by the dispatcher is known as **dispatch latency**.

1.9. The manner and importance of memory management

An **operating system (OS)** is software that manages computer hardware and software resources and provides common services for computer programs. The operating system is an essential component of the system software in a computer system. Application programs usually require an operating system to function.

Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources.

For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between programs and the computer hardware, although the application code is usually executed

directly by the hardware and frequently makes a system call to an OS function or be interrupted by it.

Among other things, a multiprogramming operating system kernel must be responsible for managing all system memory which is currently in use by programs. This ensures that a program does not interfere with memory already in use by another program. Since programs time share, each program must have independent access to memory.

Cooperative memory management, used by many early operating systems, assumes that all programs make voluntary use of the kernel's memory manager, and do not exceed their allocated memory. This system of memory management is almost never seen any more, since programs often contain bugs which can cause them to exceed their allocated memory. If a program fails, it may cause memory used by one or more other programs to be affected or overwritten. Malicious programs or viruses may purposefully alter another program's memory, or may affect the operation of the operating system itself. With cooperative memory management, it takes only one misbehaved program to crash the system.

Memory protection enables the kernel to limit a process' access to the computer's memory. Various methods of memory protection exist, including memory segmentation and paging. All methods require some level of hardware support (such as the 80286 MMU), which doesn't exist in all computers.

1.10. The manner and importance of a device management

In computing, a **file system** (or **filesystem**) is used to control how data is stored and retrieved. Without a file system, information placed in a storage area would be one large body of data with no way to tell where one piece of information stops and the next begins. By separating the data into individual pieces, and giving each piece a name, the information is easily separated and identified. Taking its name from the way paper-based information systems are named, each group of data is called a "file". The structure and logic rules used to manage the groups of information and their names is called a "file system".

There are many different kinds of file systems. Each one has different structure and logic, properties of speed, flexibility, security, size and more. Some file systems have been designed to be used for specific applications. For example, the ISO 9660 file system is designed specifically for optical discs.

File systems can be used on many different kinds of storage devices. Each storage device uses a different kind of media. The most common storage device in use today is a hard drive whose media is a disc that has been coated with a magnetic film. The film has ones and zeros 'written' on it

sending electrical pulses to a magnetic "read-write" head. Other media that are used are magnetic tape, optical disc, and flash memory. In some cases, the computer's main memory (RAM) is used to create a temporary file system for short term use.

1.11. Additional information

The most popular and widely-known operating system is Windows, a family of operating systems introduced by the Microsoft Corporation in 1985. Windows employs a graphical user interface (GUI), which eliminates the need for the user to learn complex commands. With a GUI, the user instructs the operating system by using a mouse to point and click icons that are displayed on the screen. The first version of Windows, released in November 1985, was designed as a GUI add-on for MS-DOS (short for Microsoft Disk Operating System), Microsoft's original command-line operating system. In order to communicate with the computer, DOS users had to type commands or instructions at a command prompt, and then the command-line interpreter executed those commands. DOS has limited use with modern computer systems and applications because it does not support multiple users or multitasking. It is also not as "user friendly" as a GUI operating system. By the late-1990s, most command-line operating systems had been replaced by user-friendly systems using a GUI.

By the time Windows 3.1 was released in 1991, Windows had gained significantly in market share, and from the mid-1990s on, Microsoft dominated the operating-system market. Microsoft released Windows 95 in August 1995. It was so well marketed and in such high demand that people bought the operating system even if they did not own a home computer. With each new release, from Windows 98 to Windows 2000 to Windows XP to Windows Vista, Microsoft gained popularity. By 2004, Microsoft commanded around 90 percent of the operating-system market. The latest Windows release, Windows Vista, was introduced worldwide in January 2007, five years after the introduction of its predecessor, Windows XP—the longest time between releases of Windows. Vista is only slowly gaining in popularity, while Windows XP remains the world's dominant operating system, commanding 79 percent market share in September 2007 (compared to only 7 percent for Vista).

The second-most popular family of operating systems is the Mac OS. Macintosh is often credited with introducing the GUI-style operating system; Mac OS was not the first graphical user interface, but it was the first successful one, largely because competitors were so expensive—the Xerox Alto, the first true GUI system, cost \$32,000 to build while the Macintosh retailed for \$2,500. Because of its relative affordability, it immediately became much more popular than other GUI computers. The first Mac OS, unnamed at the time but now known as "System 1," was

introduced in 1984. It had a desktop, windows, icons, a mouse, menus, and scrollbars.

Early versions of the Mac OS could only be used on Apple computers. In 1991, Apple introduced computers with PowerPC hardware; these computers could run either Windows or Mac OS. The latest version of the Mac OS, OS X, was first introduced in 2000. It can be used in computers with both PowerPC and Intel processors, giving it wider applicability and use. However, the Mac OS lags far behind Microsoft Windows. In 2007, it was the world's third-most popular operating system after Windows XP and Windows Vista but it commanded only slightly over 3 percent of the operating-system market.

UNIX is a multi-user, multitasking operating system, and was designed to be a small, flexible system used by computer programmers. UNIX is not considered to be very user-friendly for the average person due to its design. However, graphical user interfaces have been developed for UNIX to help alleviate the ease-of-use issue. Linux is a UNIX variant that runs on several different hardware platforms. Linus Torvalds, a student at the University of Helsinki in Finland, initially created it as a hobby. The kernel, at the heart of all Linux systems, is developed and released under the General Public License (GNU), and its source code is freely available to everyone. There are now hundreds of companies, organizations, and individuals that have released their own versions of operating systems based on the Linux kernel.

Because of its functionality, adaptability, and robustness, Linux is able to compete against the Unix and Microsoft operating systems. IBM, Hewlett-Packard, and other computer giants have embraced Linux and support its ongoing development. More than a decade after its initial release, Linux is being adopted worldwide mainly as a server platform. More and more people are starting to use Linux as a home and office desktop operating system. The operating system can also be incorporated directly into microchips in a process called "embedding." Many appliances and devices are now starting to use operating systems in this way.

1.12. The Role of OS in Computer Engineering

Operating systems (OS's) are important to software engineering for four fundamental and significant reasons. 1) OS's define the abstract machines in terms of which subsystems and application programs must be implemented and upon which they must be executed. 2) OS's play a significant role in defining the programming environment for development of subsystems and applications. 3) OS's have in the past motivated and been the experimental vehicles for the development of many of the fundamental concepts of software engineering. 4) OS's research is, of necessity, leading

the way in the development of the technology for the utilization of distributed systems and object oriented systems. This article attempts to place the concepts of OS's research and development in a software engineering perspective and to develop insights on the future contributions of OS's research to software engineering. Items 1) and 2) also explain why OS's and their characteristics are important not only to software engineers, but to all users of computer systems. The user who finds the OS of his computer system sometimes inconvenient to use should turn to the hardware reference manuals for his system and imagine directly using these incredibly detailed interfaces.