OPERATING SYSTEM

Shahzad Ali Rana
Lecturer GCUF
03006641562
/shahzad.rana.127

History of Operating Systems

Historically operating systems have been tightly related to the computer architecture, it is good idea to study the history of operating systems from the architecture of the computers on which they run.

Operating systems have evolved through a number of distinct phases or generations which corresponds roughly to the decades.

The 1940's - First Generations

The earliest electronic digital computers had no operating systems. Machines of the time were so primitive that programs were often entered one bit at time on rows of mechanical switches (plug boards). Programming languages were unknown (not even assembly languages). Operating systems were unheard of.

The 1950's - Second Generation

By the early 1950's, the routine had improved somewhat with the introduction of punch cards. The General Motors Research Laboratories implemented the first operating systems in early 1950's for their IBM 701. The system of the 50's generally ran one job at a time. These were called single-stream batch processing systems because programs and data were submitted in groups or batches.

The 1960's - Third Generation

The systems of the 1960's were also batch processing systems, but they were able to take better advantage of the computer's resources by running several jobs at once. So operating systems designers developed the concept of multiprogramming in which several jobs are in main memory at once; a processor is switched from job to job as needed to keep several jobs advancing while keeping the peripheral devices in use.

For example, on the system with no multiprogramming, when the current job paused to wait for other I/O operation to complete, the CPU simply sat idle until the I/O finished. The solution for this problem that evolved was to partition memory into several pieces, with a different job in each partition. While one job was waiting for I/O to complete, another job could be using the CPU.

Another major feature in third-generation operating system was the technique called spooling (simultaneous peripheral operations on line). In spooling, a high-speed device like a disk interposed between a running program and a low-speed device involved with the program in input/output. Instead of writing directly to a printer, for example, outputs are written to the disk. Programs can run to completion faster, and other programs can be initiated sooner when the printer becomes available, the outputs may be printed.

Note that spooling technique is much like thread being spun to a spool so that it may be later be unwound as needed.

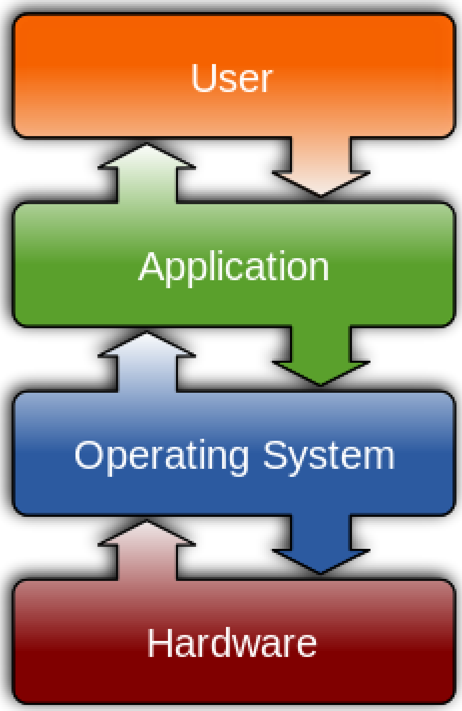
Another feature present in this generation was time-sharing technique, a variant of multiprogramming technique, in which each user has an on-line (i.e., directly connected) terminal. Because the user is present and interacting with the computer, the computer system must respond quickly to user requests, otherwise user productivity could suffer. Timesharing systems were developed to multiprogramming large number of simultaneous interactive users.

Fourth Generation

With the development of LSI (Large Scale Integration) circuits, chips, operating system entered in the system entered in the personal computer and the workstation age. Microprocessor technology evolved to the point that it becomes possible to build desktop computers as powerful as the mainframes of the 1970s. Two operating systems have dominated the personal computer scene: MS-DOS, written by Microsoft, Inc. for the IBM PC and other machines using the Intel 8088 CPU and its successors, and UNIX, which is dominant on the large personal computers using the Motorola 6899 CPU family.

Operating Systems Goals

When a computer is turned on, the hardware loads code from a set location in permanent memory (generally the disk drive) into main memory. Those instructions load the operating system code and start it up. Once running, the operating system is responsible for starting up application programs and managing the hardware resources that they need. Indeed, most programs are not allowed to interact directly with the hardware - instead, the operating system mediates all interactions between programs and the hardware.



Hardware Allocation

Most of the hardware resources a computer has are **scarce**, meaning they are in limited supply: The CPU can only do so much work at any given time. There is a physical limit to how much information can fit into main memory (RAM). Only one program can be printing to a printer at any given time.

The operating system is in charge of **allocating** this scarce hardware - deciding who gets what when and preventing any one program from monopolizing the hardware resources.

Hardware Abstraction

Computer hardware is diverse and complicated. There is much different kind of storage drives that can be hooked up to a computer - flash, hard drive, CD drive, SSD, etc... And of those styles of drive comes in thousands of makes and models. Each of those different drives stores information in different ways, often scattered pieces of information across the drive.

Imagine you are writing a program and want to read a document from a drive. Without an operating system, you would have to write code to talk to a drive, track down all the chunks of information and assemble them into one block in memory. Then you would have to make sure that code worked with thousands of different kinds of drives.

Instead of that nightmare situation, the operating system provides an **abstraction** for dealing with hardware - instead of talking to a drive about chunks of information, your program can talk to the operating system about *"opening a file"*. Because the operating system knows how to look up and retrieve the information that corresponds to that *file* on a wide variety of devices, your program is freed up from worrying about all of the messy details.

Common Interface

Related to the job of hardware abstraction is providing tools to other programs to interact with users in a consistent and effective manner. Operating systems provide application programming interfaces (APIs) that include tools for doing things like drawing windows. Rather than each application designer deciding what windows should look like, what font to use in the title bar, where to put the close button, etc... The operating system provides code that takes care of those details. Not only does this make programming an application easier, it also helps enforce a consistent experience for the user.

We could fill books with all the ways that operating systems provide allocation of an abstraction for computer hardware, but on the following pages we will focus on two fundamental resources: processors and main memory.

Organization of a Computer System

As shown in Figure 1.1, the major high-level components of a computer system are:
1. Hardware, which provides basic computing resources (CPU, memory, I/O devices).
2. Operating system, which manages the use of the hardware among the various application programs for the various users and provides the user a relatively simple machine to use.
3. Applications programs that define the ways in which system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs).

4. Users, which include people, machines, other computers.

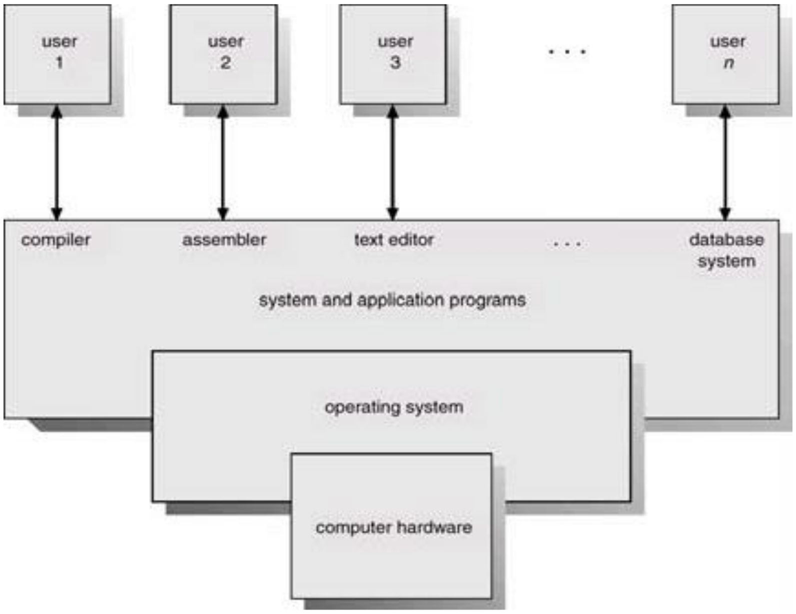


Figure 1.1. High-level components of a computer system

Purpose of a Computer: (Setting the Stage for OS Concepts and Principles)
Computer systems consist of software and hardware that are combined to provide a tool to implement solutions for specific problems in an efficient manner and to execute Programs. Figure 1.2 shows the general organization of a contemporary computer system and how various system components are interconnected.

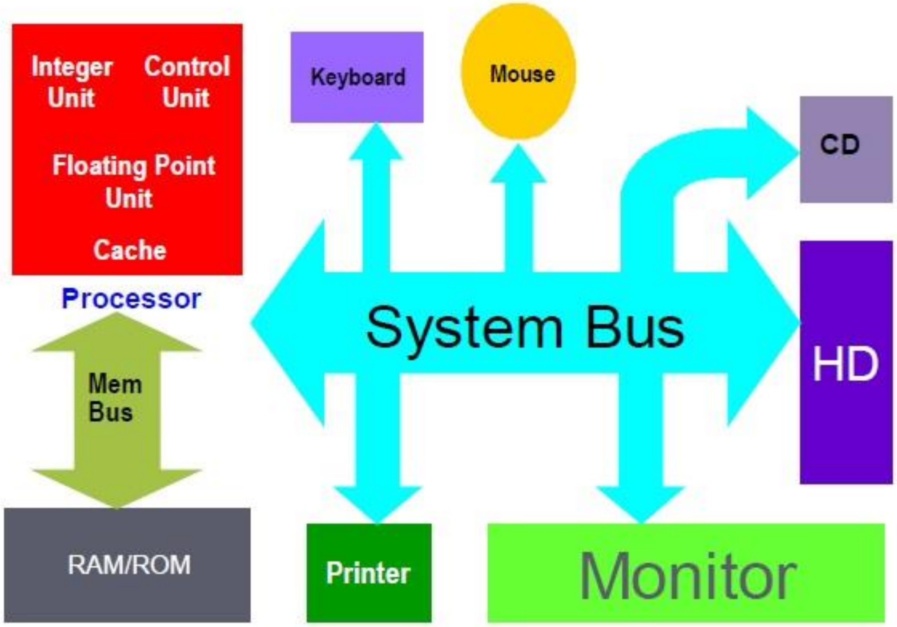


Figure 1.2. Organization of a Computer System

Viewing things closely will reveal that the primary purpose of a computer system is to generate executable programs and execute them. The following are some of the main issues involved in performing these tasks.

1. Storing an executable on a secondary storage device such as hard disk
2. Loading executable from disk into the main memory
3. Setting the CPU state appropriately so that program execution could begin
4. Creating multiple cooperating processes, synchronizing their access to shared data, and allowing them to communicate with each other the above issues require the operating system to provide the following services and much more:

- Manage secondary storage devices
- Allocate appropriate amount of disk space when files are created
- Deallocate space when files are removing
- Insure that a new file does not overwrite an existing file
- Schedule disk requests
- Manage primary storage
- Allocate appropriate amount of memory space when programs are to be loaded into the memory for executing
- Deallocate space when processes terminate
- Insure that a new process is not loaded on top of an existing process
- Insure that a process does not access memory space that does not belong to it
- Minimize the amount of unused memory space
- Allow execution of programs larger in size than the available main memory
- Manage processes
- Allow simultaneous execution of processes by scheduling the CPU(s)
- Prevent deadlocks between processes
- Insure integrity of shared data
- Synchronize executions of cooperating processes

- Allow a user to manage his/her files and directories properly
- User view of directory structure
- Provide a mechanism that allows users to protect their files and directories

In this course, we will discuss in detail these operating system services (and more) with a particular emphasis on the UNIX and Linux operating systems. See the course outline for details of topics and lecture schedule.

What is an Operating System?

“An operating system (OS) is system software that manages computer hardware and software resources and provides common services for computer programs.”

“An operating system is a powerful, and usually large, program that controls and manages the hardware and other software on a computer. All computers and computer-like devices have operating systems, including your laptop, tablet, desktop, smart phone, smart watch, router... you name it.”

There are two views about this. The **top-down** view is that it is a program that acts as an intermediary between a user of a computer and the computer hardware, and makes the computer system convenient to use. It is because of the operating system that users of a computer system don’t have to deal with computer’s hardware to get their work done. Users can use simple commands to perform various tasks and let the operating system do the difficult work of interacting with computer hardware. Thus, you can use a command like copy file1 file2 to copy ‘file1’ to ‘file2’ and let the operating system communicate with the controller(s) of the disk that contain(s) the two files.

A computer system has many hardware and software resources that may be required to solve a problem: CPU time, memory space, files storage space, I/O devices etc. The operating system acts as the manager of these resources, facing numerous and possibly conflicting requests for resources, the operating system must decide how (and when) to allocate (and deallocate) them to specific programs and users so that it can operate the computer system efficiently, fairly, and securely.

So, the **bottom-up** view is that operating system is a resource manager who manages the hardware and software resources in the computer system. A slightly different view of an operating system emphasizes the need to control the various I/O devices and programs. An operating system is a control program that manages the execution of user programs to prevent errors and improper use of a computer.

Classification of Operating systems

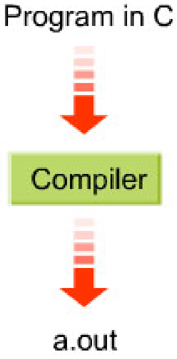
- **Multi-user:** Allows two or more users to run programs at the same time. Some operating systems permit hundreds or even thousands of concurrent users.
- **Multiprocessing :** Supports running a program on more than one CPU.
- **Multitasking :** Allows more than one program to run concurrently.
- **Multithreading :** Allows different parts of a single program to run concurrently.
- **Real time:** Responds to input instantly. General-purpose operating systems, such as DOS and UNIX, are not real-time.

Introduction About Process Management Memory Management

Process Management

As we know a process is a program in execution. To understand the importance of this definition, let's imagine that we have written a program called my_prog.c in C. On execution, this program may read in some data and output some data. Note that when a program is written and a file is prepared, it is still a script. It has no dynamics of its own i.e., it cannot cause any input processing or output to happen. Once we compile, and still later when we run this program, the intended operations take place. In other words, a program is a text script with no dynamic behavior. When a program is in execution, the script is acted upon. It can result in engaging a processor for some processing and it can also engage in I/O operations. It is for this reason a process is differentiated from program. While the program is a text script, a program in execution is a process.

In other words, To begin with let us define what a "process" is and in which way a process differs from a program. A process is an executable entity – it's a program in execution. When we compile a C language program we get an a.out file which is an executable file. When we seek to run this file – we see the program in execution. Every process has its instruction sequence. Clearly, therefore, at any point in time there is a current instruction in execution.

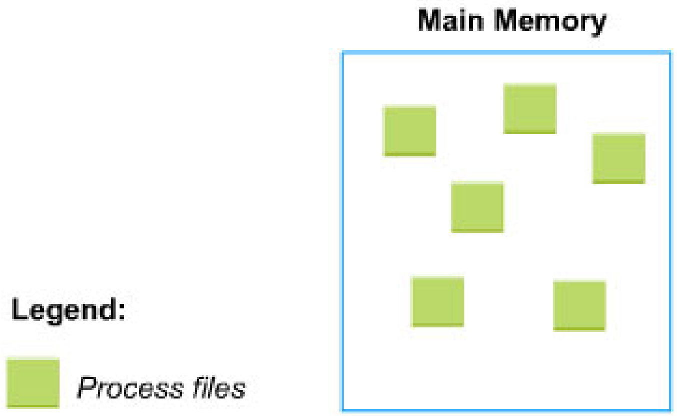


A program counter determines helps to identify the next instruction in the sequence. So process must have an inherent program counter. Referring back to the C language program Processes get created, may have to be suspended awaiting an event like completing a certain I/O. A process terminates when the task it is defined for is completed. During the life time of a process it may seek memory dynamically. In fact, the malloc instruction in C precisely does that. In any case, from the stand point of OS a process should be memory resident and, therefore, needs to be stored in specific area within the main memory. Processes during their life time may also seek to use I/O devices.

For instance, an output may have to appear on a monitor or a printed output may be needed. In other words, process management requires not only making the processor available for execution but, in addition, allocates main memory, files and IO. The process management component then requires coordination with the main memory management, secondary memory management, as well as, files and I/O.

Memory Management:

As we observed earlier in the systems operate using Von-Neumann's stored program concept. The basic idea is that an instruction sequence is required to be stored before it can be executed. Therefore, every executable file needs to be stored in the main memory. In addition, we noted that modern systems support multi-programming. This means that more than one executable process may be stored in the main memory. If there are several programs residing in the memory, it is imperative that these be appropriately assigned specific areas.



The OS needs to select one amongst these to execute. Further these processes have their data areas associated with them and may even dynamically seek more data areas. In other words, the OS must cater to allocating and de-allocating memory to processes as well as to the data required by these processes. All processes need files for their operations and the OS must manage these as well.

Files and IO Management:

On occasions processes need to operate on files. Typical file operations are:

1. Create: To create a file in the environment of operation
2. Open: To open an existing file from the environment of operation.
3. Read: To read data from an opened file.
4. Write: To write into an existing file or into a newly created file or it may be to modify or append or write into a newly created file.
5. Append: Like write except that writing results in appending to an existing file.
6. Modify or rewrite: Essentially like write – results in rewriting a file.
7. Delete: This may be to remove or disband a file from further use.

OS must support all of these and many other file operations. For instance, there are other file operations like which applications or users may be permitted to access the files. Files may be "owned" or "shared". There are file operations that permit a user to specify this. Also, files may be of different types. For instance, we have already seen that we may have executable and text files. In addition, there may be image files or audio files. Later in this course you will learn about various file types and the file operations on more details. For now it suffices to know that one major task OSs perform related to management of files.