1.9 EVOLUTION OF OPERATING SYSTEM

An operating system acts as an intermediary between the user of a computer and the computer hardware.

The evolution of operating system is explained at various stages.

- Serial Processing
- Simple Batch Systems
- Multi programmed batch systems.
- Time sharing systems

1.9.1 Serial Processing

- With the earliest computers, from the late 1940s to the mid-1950s, the programmer interacted directly with the computer hardware; there was no OS.
- These computers were run from a console consisting of display lights, toggle switches, some form of input device, and a printer.
- Programs in machine code were loaded via the input device (e.g., a card reader).
- If an error halted the program, the error condition was indicated by the lights.
- If the program proceeded to a normal completion, the output appeared on the printer.

These early systems presented two main problems:

- **Scheduling**: Most installations used a hardcopy sign-up sheet to reserve computer time. A user might sign up for an hour and finish in 45 minutes; this would result in wasted computer processing time. On the other hand, the user might run into problems, not finish in the allotted time, and be forced to stop before resolving the problem.
- **Setup time**: A single program, called a job, could involve loading the compiler plus the high-level language program (source program) into memory, saving the compiled program (object program) and then loading and linking together the object program and common functions. Thus, a considerable amount of time was spent just in setting up the program to run.
- This mode of operation could be termed serial processing, reflecting the fact that users have access to the computer in series.

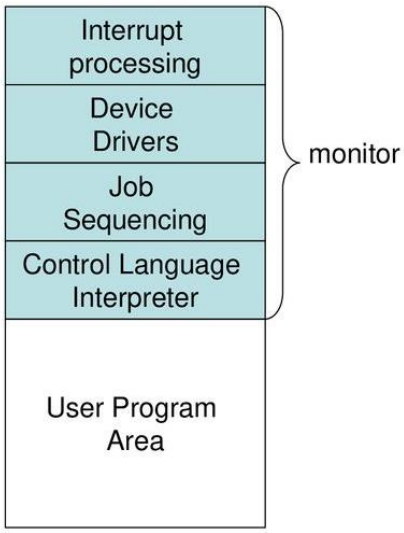
1.9.2 Simple Batch Systems

To improve utilization, the concept of a batch OS was developed.

- The central idea behind the simple batch-processing scheme is the use of a piece of software known as the monitor.
- With this type of OS, the user no longer has direct access to the processor.
- Instead, the user submits the job on cards or tape to a computer operator, who batches the jobs together sequentially and places the entire batch on an input device, for use by the monitor.
- Each program is constructed to branch back to the monitor when it completes processing, at which point the monitor automatically begins loading the next program.

Monitor point of view: The monitor controls the sequence of events. For this to be so, much of the monitor must always be in main memory and available for execution. That portion is referred to as the resident monitor. The rest of the monitor consists of utilities and common functions that are loaded as subroutines to the user program at the beginning of any job that requires them. The monitor reads in jobs one at a time from the input device (typically a card reader or magnetic tape drive). As it is read in, the current job is placed in the user program area, and control is passed to this job. When the job is completed, it returns control to the monitor, which immediately reads in the next job. The results of each job are sent to an output device, such as a printer, for delivery to the user.

Processor point of view: At a certain point, the processor is executing instructions from the portion of main memory containing the monitor. These instructions cause the next job to be read into another portion of main memory. Once a job has been read in, the processor will encounter a branch instruction in the monitor that instructs the processor to continue execution at the start of the user program. The processor will then execute the instructions in the user program until it encounters an ending or error.



- With each job, instructions are included in a primitive form of job control language (JCL).
- This is a special type of programming language used to provide instructions to the monitor.

Certain other hardware features are also desirable

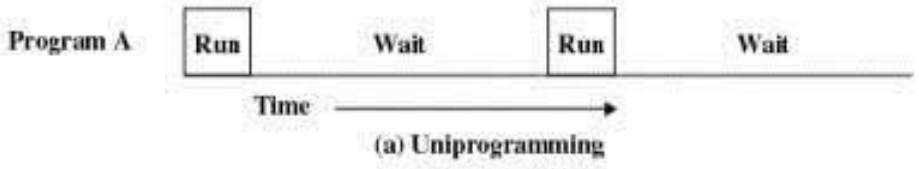
- Memory protection
- Timer
- Privileged instructions
- Interrupts

• A user program executes in a user mode, in which certain areas of memory are protected from the user's use and in which certain instructions may not be executed.

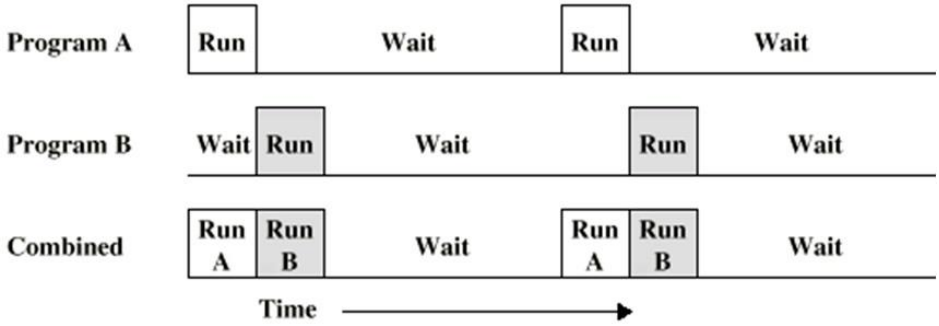
• The monitor executes in a system mode, or what has come to be called kernel mode, in which privileged instructions may be executed and in which protected areas of memory may be accessed.

1.9.3 Multi programmed Batch Systems

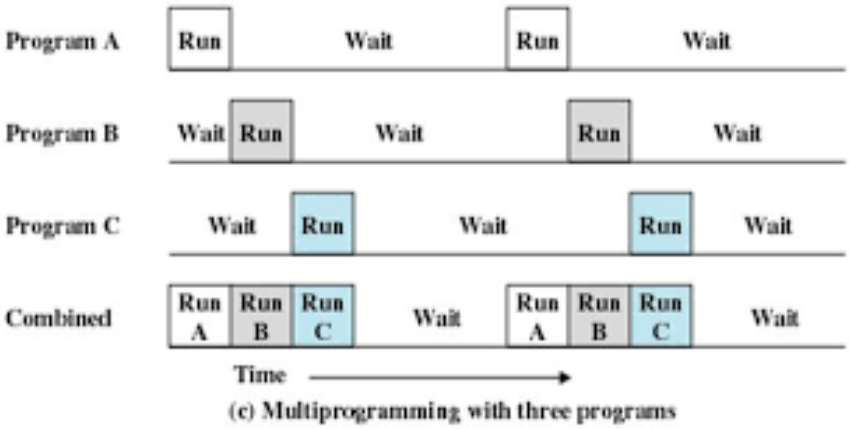
- Even with the automatic job sequencing provided by a simple batch OS, the processor is often idle.
- The approach used to improve processor utilization is known as multi programming, or multitasking.
- It is the central theme of modern operating systems.



(a) Uniprogramming



Multiprogrammed with two programs



(c) Multiprogramming with three programs

1.9.4 Time-Sharing Systems:

- In time sharing systems the processor time is shared among multiple users.
- In a time-sharing system, multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation.
- If there are n users actively requesting service at one time, each user will only see on the average 1/n of the effective computer capacity.

Batch Multiprogramming Vs Time Sharing systems

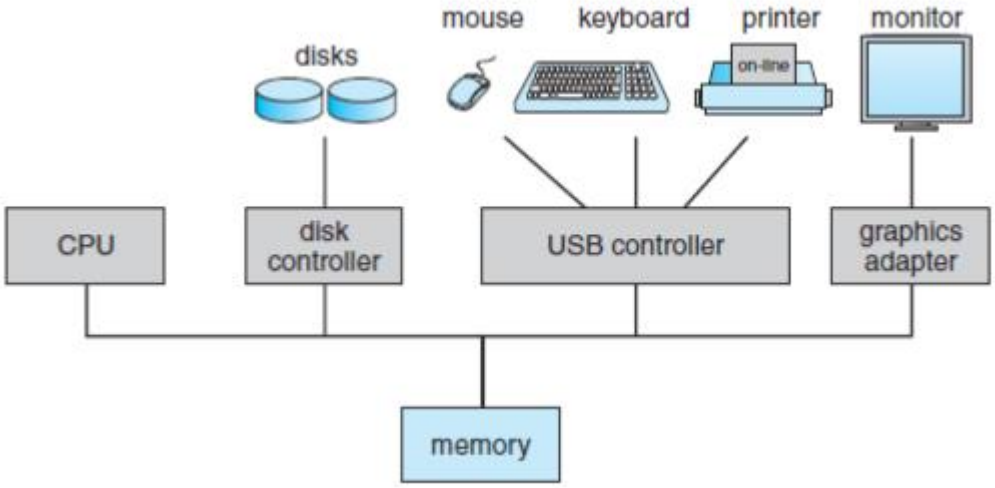
	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

1.10 COMPUTER SYSTEM ORGANIZATION:

Computer system organization deals with the structure of the computer system.

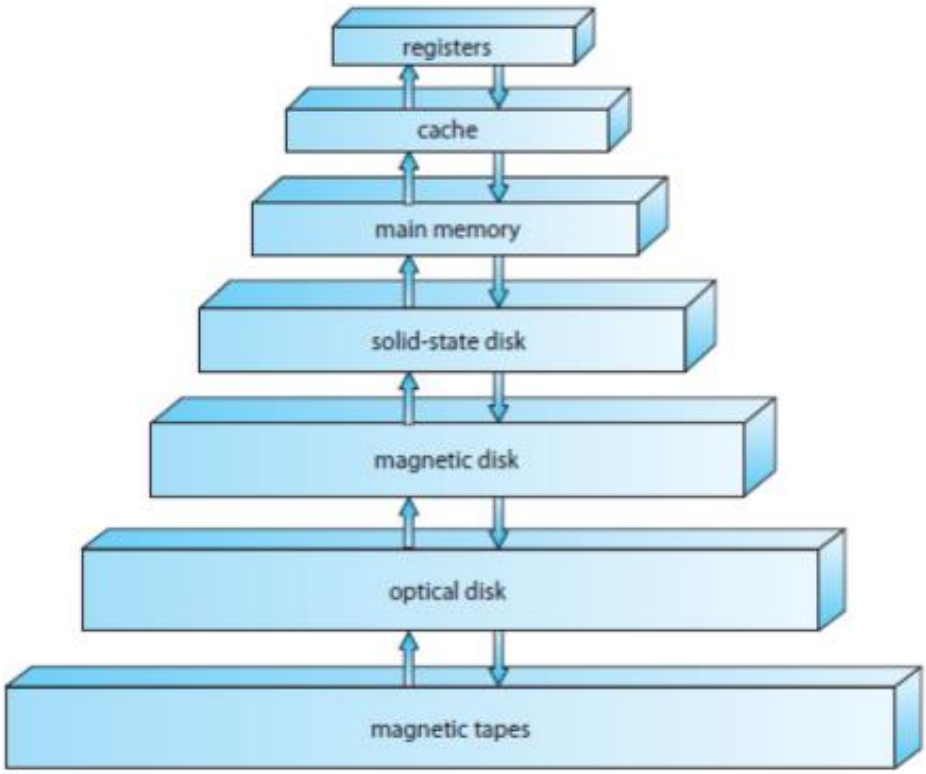
Computer system operation:

- A modern general-purpose computer system consists of one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory.
- For a computer to start running when it is powered up or rebooted—it needs to have an initial program to run. This initial program is called as the Bootstrap program.
- It is stored within the computer hardware in read-only memory (ROM) or electrically erasable programmable read-only memory (EEPROM), known by the general term firmware.
- The bootstrap loader: It initializes all aspects of the system, from CPU registers to device controllers to memory contents.
- The bootstrap program loads the operating system and start executing that system.
- Once the kernel is loaded and executing, it can start providing services to the system and its users.



Storage structure:

- The CPU can load instructions only from memory, so any programs to run must be stored in main memory.
- Main memory commonly is implemented in a semiconductor technology called dynamic random-access memory
- ROM is a read only memory that is used to store the static programs such as bootstrap loader.
- Ideally, we want the programs and data to reside in main memory permanently.
- Main memory is usually too small to store all needed programs and data permanently
- Main memory is a volatile storage device that loses its contents when power is turned off or otherwise lost.
- Most computer systems provide secondary storage as an extension of main memory.
- Volatile storage loses its contents when the power to the device is removed so that the data must be written to nonvolatile storage for safekeeping.
- Caches can be installed to improve performance.



I/O Structure:

A large portion of operating system code is dedicated to managing I/O, both because of its importance to the reliability and performance of a system.

- A general-purpose computer system consists of CPUs and multiple device controllers that are connected through a common bus. Each device controller is in charge of a specific type of device.
- The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage
- Operating systems have a device driver for each device controller.
- To start an I/O operation, the device driver loads the appropriate registers within the device controller.
- The controller starts the transfer of data from the device to its local buffer. Once the transfer of data is complete, the device controller informs the device driver via an interrupt that it has finished its operation. This is called as interrupt driven I/O.

- The direct memory access I/O technique transfers a block of data directly to or from its own buffer storage to memory, with no intervention by the CPU. Only one interrupt is generated per block, to tell the device driver that the operation has completed,