COMPUTER NETWORKS

UNIT-1

Computer System Architecture

Computer is an electronic machine that makes performing any task very easy. In computer, the CPU executes each instruction provided to it, in a series of steps, this series of steps is called **Machine Cycle**, and is repeated for each instruction. One machine cycle involves *fetching of instruction*, *decoding the instruction*, *transferring the data*, *executing the instruction*.

Computer system has five basic units that help the computer to perform operations, which are given below:

1. **Input Unit**

   Input unit connects the external environment with internal computer system. It provides data and instructions to the computer system. Commonly used input devices are *keyboard*, *mouse*, *magnetic tape* etc.

   Input unit perform following tasks:

   - Accept the data and instructions from the outside environment.
   - Convert it into machine language.
   - Supply the converted data to computer system.

2. **Output Unit**

   It connects the internal system of a computer to the external environment. It provides the results of any computation, or instructions to the outside world. Some output devices are *printers*, *monitor*etc.

3. **Storage Unit**

   This unit holds the data and instructions. It also stores the intermediate results before these are sent to the output devices. It also stores the data for later use.

   The storage unit of a computer system can be divided into two categories:

   - **Primary Storage**

     This memory is used to store the data which is being currently executed. It is used for temporary storage of data. The data is lost, when the computer is switched off. RAM is used as primary storage memory.

   - **Secondary Storage**

The secondary memory is slower and cheaper than primary memory. It is used for permanent storage of data. Commonly used secondary memory devices are hard disk, CD etc.

4. **Arithmetic Logical Unit**

   All the calculations are performed in ALU of the computer system. The ALU can perform basic operations such as addition, subtraction, division, multiplication etc. Whenever calculations are required, the control unit transfers the data from storage unit to ALU. When the operations are done, the result is transferred back to the storage unit.

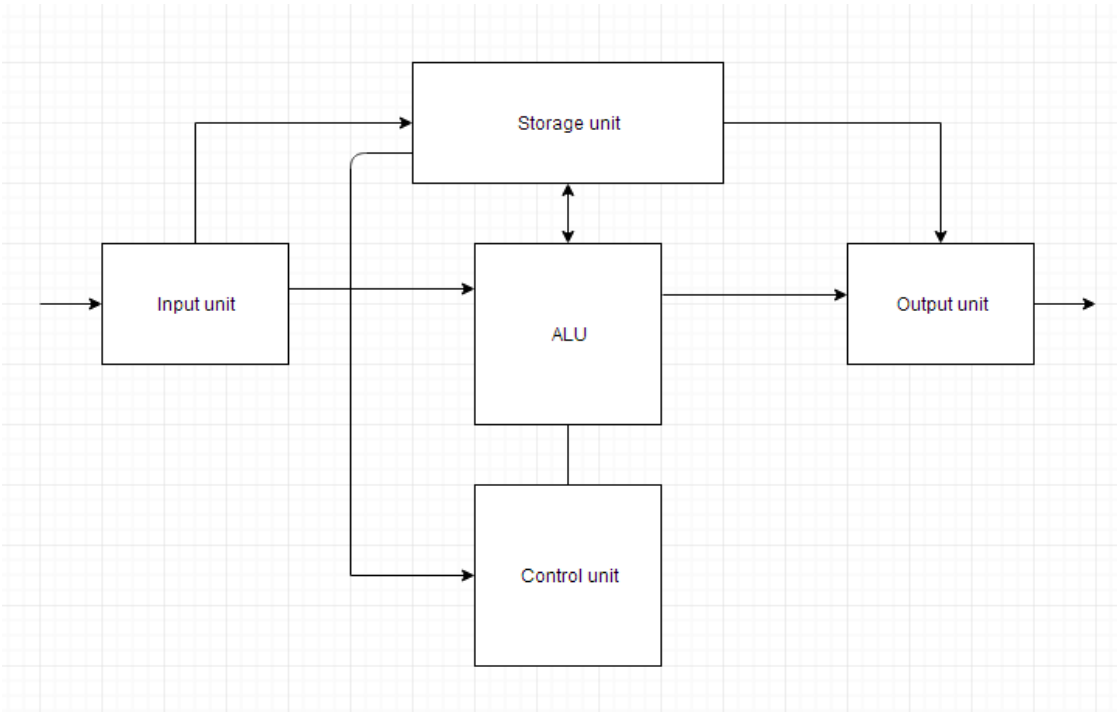
5. **Control Unit**

   It controls all other units of the computer. It controls the flow of data and instructions to and from the storage unit to ALU. Thus it is also known as central nervous system of the computer.

6. **CPU**

   It is Central Processing Unit of the computer. The control unit and ALU are together known as CPU. CPU is the brain of computer system. It performs following tasks:

   - It performs all operations.
   - It takes all decisions.
   - It controls all the units of computer.



Above figure shows the block diagram of a computer.

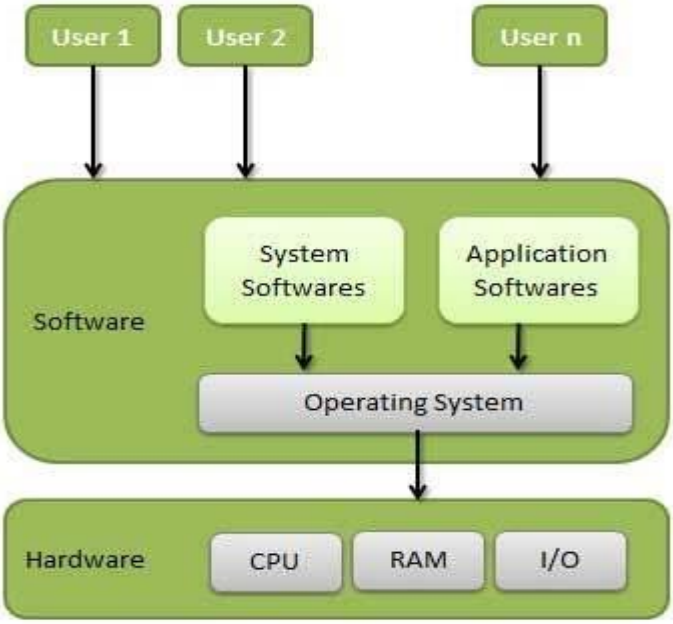
Operating System Structures

An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

Some popular Operating Systems include Linux, Windows, OS X, VMS, OS/400, AIX, z/OS, etc.

Definition

An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.



Following are some of important functions of an operating System.

- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job accounting

- Error detecting aids
- Coordination between other software and users

Operating System - Services

An Operating System provides services to both the users and to the programs.

- It provides programs an environment to execute.
- It provides users the services to execute the programs in a convenient manner.

Following are a few common services provided by an operating system −

- User Interface
- Program Execution
- File system manipulation
- Input / output Operations
- Communication
- Resource Allocation
- Error Detection
- Accounting
- Security and protection

User Interface of Operating System

Usually Operating system comes in three forms or types. Depending on the interface their types have been further sub divided. These are:

- Command line interface
- Batch based interface
- Graphical User Interface

Let’s get to know in brie about each of them.

Program Execution in Operating System

The operating system must have the capability to load a program into memory and execute that program. Furthermore, the program must be able to end its execution, either normally or abnormally / forcefully.

File System Manipulation in Operating System

Programs need has to be read and then write them as files and directories. File handling portion of operating system also allows users to create and delete files by specific name along with extension, search for a given file and / or list file information. Some programs comprise of permissions management for allowing or denying access to files or directories based on file ownership.

I/O operations in Operating System

A program which is currently executing may require I/O, which may involve file or other I/O device. For efficiency and protection, users cannot directly govern the I/O devices. So, the OS provide a means to do I/O Input / Output operation which means read or write operation with any file.

Communication System of Operating System

Process needs to swap over information with other process. Processes executing on same computer system or on different computer systems can communicate using operating system support. Communication between two processes can be done using shared memory or via message passing.

Resource Allocation of Operating System

When multiple jobs running concurrently, resources must need to be allocated to each of them. Resources can be CPU cycles, main memory storage, file storage and I/O devices. CPU scheduling routines are used here to establish how best the CPU can be used.

Error Detection

Errors may occur within CPU, memory hardware, I/O devices and in the user program. For each type of error, the OS takes adequate action for ensuring correct and consistent computing.

Accounting

This service of operating system keeps track of which users are using how much and what kinds of computer resources has been used for accounting or simply to accumulate usage statistics.

Protection and Security

Protection includes in ensuring all access to system resources in a controlled manner. For making a system secure, the user needs to authenticate him or her to the system before using (usually via login ID and password).

User Interface for Operating System

Usually Operating system comes in three forms or types. Depending on the interface their types have been further sub divided. These are:

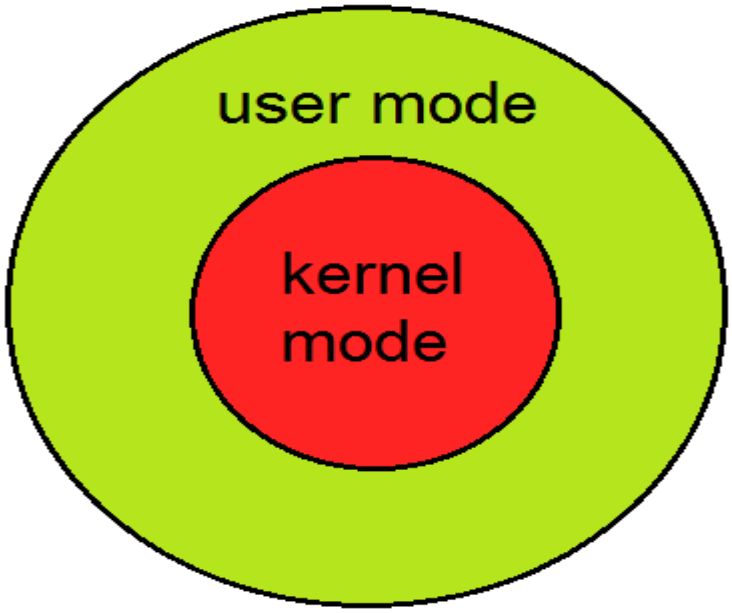
- Command line interface
- Batch based interface
- Graphical User Interface

Let's get to know in brie about each of them.

The command line interface (CLI) usually deals with using text commands and a technique for entering those commands. The batch interface (BI): commands and directives are used to manage those commands that are entered into files and those files get executed. Another type is the graphical user interface (GUI): which is a window system with a pointing device (like mouse or track-ball) to point to the I/O, choose from menus driven interface and to make choices viewing from a number of lists and a keyboard to entry the texts.

SYSTEM CALLS

To understand system calls, first one needs to understand the difference between **kernel mode** and **user mode** of a CPU. Every modern operating system supports these two modes.



Modes supported by the operating system

Kernel Mode

- When CPU is in **kernel mode**, the code being executed can access any memory address and any hardware resource.
- Hence kernel mode is a very privileged and powerful mode.
- If a program crashes in kernel mode, the entire system will be halted.

User Mode

- When CPU is in **user mode**, the programs don't have direct access to memory and hardware resources.
- In user mode, if any program crashes, only that particular program is halted.
- That means the system will be in a safe state even if a program in user mode crashes.
- Hence, most programs in an OS run in user mode.

System Call

When a program in user mode requires access to RAM or a hardware resource, it must ask the kernel to provide access to that resource. This is done via something called a **system call**.

When a program makes a system call, the mode is switched from user mode to kernel mode. This is called a **context switch**.

Then the kernel provides the resource which the program requested. After that, another context switch happens which results in change of mode from kernel mode back to user mode.

Generally, system calls are made by the user level programs in the following situations:

- Creating, opening, closing and deleting files in the file system.
- Creating and managing new processes.
- Creating a connection in the network, sending and receiving packets.
- Requesting access to a hardware device, like a mouse or a printer.

In a typical UNIX system, there are around 300 system calls. Some of them which are important ones in this context, are described below.

Types of System Calls

There are 5 different categories of system calls:

1. process control
2. file manipulation
3. device manipulation
4. information maintenance and
5. communication.

1. Process Control

A running program needs to be able to stop execution either normally or abnormally. When execution is stopped abnormally, often a dump of memory is taken and can be examined with a debugger.

2. File Management

Some common system calls are *create*, *delete*, *read*, *write*, *reposition*, or *close*. Also, there is a need to determine the file attributes – *get* and *set* file attribute. Many times the OS provides an API to make these system calls.

3. Device Management

Process usually require several resources to execute, if these resources are available, they will be granted and control returned to the user process. These resources are also thought of as devices. Some are physical, such as a video card, and others are abstract, such as a file.

User programs *request* the device, and when finished they *release* the device. Similar to files, we can *read*, *write*, and *reposition* the device.

4. Information Management

Some system calls exist purely for transferring information between the user program and the operating system. An example of this is *time*, or *date*.

The OS also keeps information about all its processes and provides system calls to report this information.

5. Communication

There are two models of interprocess communication, the message-passing model and the shared memory model.

- Message-passing uses a common mailbox to pass messages between processes.
- Shared memory use certain system calls to create and gain access to create and gain access to regions of memory owned by other processes. The two processes exchange information by reading and writing in the shared data.

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

OPERATING-SYSTEM STRUCTURES

1. System Components
2. Operating-System Services
3. System Calls
4. System Programs
5. System Structure
6. Virtual Machines
7. System Design and Implementation
8. Booting

Most operating systems support the following types of system components:

- Process Management
- Main-Memory Management
- Secondary-Storage Management
- I/O System Management
- File Management
- Protection System
- Networking
- Command-Interpreter System

Process Management

- A process is a program in execution. A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.
- The operating system is responsible for the following activities in connection with process management:
    - process creation and deletion.
    - process suspension and resumption (scheduling).
    - provision of mechanisms for:
        - process synchronization
        - process communication
- Process management is usually performed by the kernel.

Main-Memory Management

- Memory is a large array of words or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices.
- The operating system is responsible for the following activities in connection with memory management:
    - Keep track of which parts of memory are currently being used and by whom.

    - Decide which processes to load when memory space becomes available.
    - Allocate and deallocate memory space as needed.
      e.g. the C function 'malloc' (or 'New' in Pascal) allocates a specified amount of memory; this happens via an OS call. The functions 'free'(C) and 'Dispose'(Pascal) deallocate this memory.

I/O System Management

- The I/O system consists of:
    - A buffer-caching system
    - A general device-driver interface
    - Drivers for specific hardware devices (**device drivers**)
- Device Drivers
    - Must have access to I/O hardware
    - Must be able to handle interrupts.
    - Communicate with other parts of the OS (File system, Networking etc).

File Management

- A file is a collection of related information.
  Commonly, files represent programs (both source and object forms) and data.
  Files may also be used to represent devices (e.g. lpt1: in DOS).
- The operating system is responsible for the following activities in connection with file management:
    - File creation and deletion.
    - Directory creation and deletion.
    - Support of primitives for manipulating files and directories.
    - Mapping files onto secondary storage.
      e.g. free space allocation.
    - File backup on stable (non-volatile) storage media.

Protection System

- Protection refers to a mechanism for controlling access by programs, processes, or users to both system and user resources.
- Operating Systems commonly control access by using permissions. All system resources have an owner and a permission associated with them. Users may be combined into groups for the purpose of protection.
  e.g. in UNIX every file has an owner and a group.
  The following is a listing of all the information about a file.
  rwxr-xr-- martin staff 382983 Jan 18 10:20 notes305.html
  The first field is the protection information; it shows the permissions for the owner, then the group, then everybody else.

The first **rwx** means that the owner has read, write and execute permissions.
The next **r-x** means that the group has read and execute permissions.
The next **r--** means that all other users have only read permission.

The name of the owner of the file is **martin**; the name of the group for the file is **staff**; the length of the file is **382983** bytes; the file was created on **Jan 18** at **10:20** and the name of the file is: **notes305.html**

- There is usually a special user corresponding to the system administrator, this user has permission do do anything. On UNIX systems this user is called **root.**

Networking (Distributed Systems)

- A distributed system is a collection of processors that do not share memory or a clock. Each processor has its own local memory.
- The processors in the system are connected through a communication network.
- A distributed system provides user access to various system resources.
- Access to a shared resource allows:
  - Computation speed-up
  - Increased data availability
  - Enhanced reliability

Command-Interpreter System

- Many commands are given to the operating system by control statements which deal with:
  - process creation and management (e.g. running a program)
  - I/O handling (e.g. set terminal type)
  - secondary-storage management (e.g. format a disk)
  - main-memory management (e.g. specify virtual memory parameters)
  - file-system access (e.g. print file)
  - protection (e.g. set permissions)
  - networking (e.g. set IP address)
- The program that reads and interprets control statements is called variously:
  - command-line interpreter
  - shell (in UNIX)

Its function is to get and execute the next command statement.

Some operating systems have no command line interpreter and use a GUI for all system administration (e.g. NT).

Operating-System Services

- Program execution - ability to load a program into memory and to run it.
- I/O operations - since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.
- File-system manipulation - capability to read, write, create, and delete files.
- Communications - exchange of information between processes executing either on the same computer or on different systems tied together by a network. Implemented via shared memory or message passing.
- Error detection - ensure correct computing by detecting errors in the CPU and memory hardware, in I/O devices, or in user programs.

Additional operating-system functions exist not for helping the user, but rather for ensuring efficient system operation.

- Resource allocation - allocating resources to multiple users or multiple processes running at the same time.
- Accounting - keep track of and record which users use how much and what kinds of computer resources for account billing or for accumulating usage statistics.
- Protection - ensuring that all access to system resources is controlled.

System Calls

- System calls provide the interface between a running program and the operating system.
    - Generally available as an assembly-language instruction to generate a software interrupt. (e.g. INT 21h in DOS)
    - Systems programming languages such as C allow system calls to be made directly.
- Three general methods are used to pass parameters between a running program and the operating system:
    - Pass parameters in registers.
    - Store the parameters in a table in memory, and the table address is passed as a parameter in a register.
    - Push (store) the parameters onto the stack by the program, and pop off the stack by the operating system.

System Programs

- System programs provide a convenient environment for program development and execution. They can be divided into:
    - File manipulation

    - Status information
    - File modification
    - Programming-language support
    - Program loading and execution
    - Communications
    - Application programs
- Most users' view of the operation system is defined by system programs, not the actual system calls.

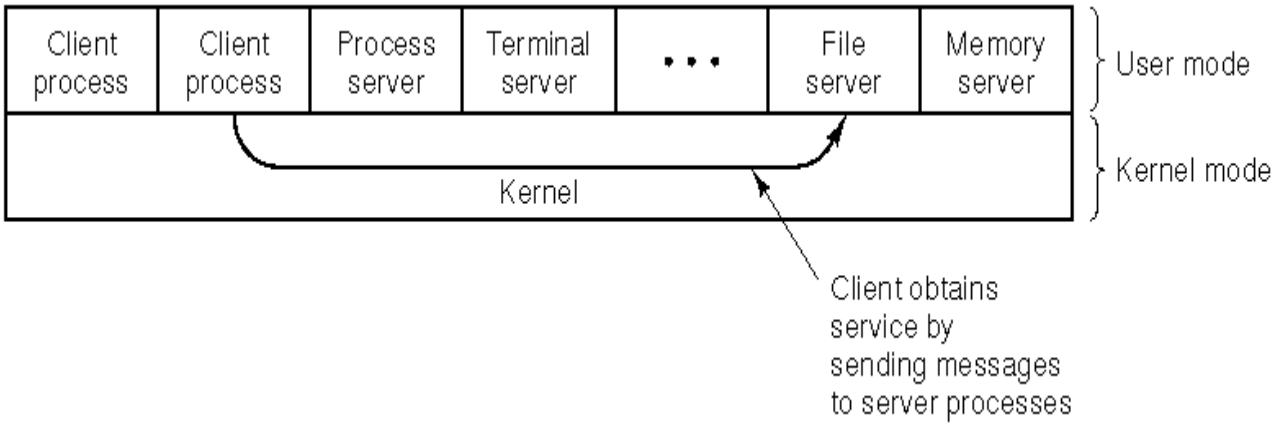
System Structure - Simple Approach

- MS-DOS - written to provide the most functionality in the least space; it was not divided into modules. MS-DOS has some structure, but its interfaces and levels of functionality are not well separated.
- UNIX - limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts:
    - the systems programs.
    - the kernel, which consists of everything below the system-call interface and above the physical hardware. Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level.

      Often this is called a **Monolithic Kernel**

      Many modern operating systems use a **Microkernel** - The kernel provides only the following minimal services.
        1. Interprocess communication.
        2. Memory management.
        3. Low level process management.
        4. Low Level I/O

      All other services are provided by user level processes.



System Structure - Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers.
- A layered design was first used in the THE operating system of Dijkstra in 1968. Its six layers are as follows:

Level 5: user programs

Level 4: buffering for input and output devices

Level 3: operator-console device driver

Level 2: memory management

Level 1: CPU scheduling

Level 0: hardware

Virtual Machines

- A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware.
- A virtual machine provides an interface identical to the underlying bare hardware.
- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.
- The resources of the physical computer are shared to create the virtual machines.

    - CPU scheduling can create the appearance that users have their own processor.
    - Spooling and a file system can provide virtual I/O.
    - A terminal serves as the virtual machine console.

Advantages and Disadvantages of Virtual Machines

- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.
- A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
- The virtual machine concept is difficult to implement due to the effort required to provide an exact duplicate of the underlying machine.

System Design Goals

- User goals - operating system should be convenient to use, easy to learn, reliable, safe, and fast.
- System goals - operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.

Mechanisms and Policies

- Mechanisms determine how to do something; policies decide what will be done.
- The separation of policy from mechanism is a very important principle; it allows maximum flexibility if policy decisions are to be changed later.

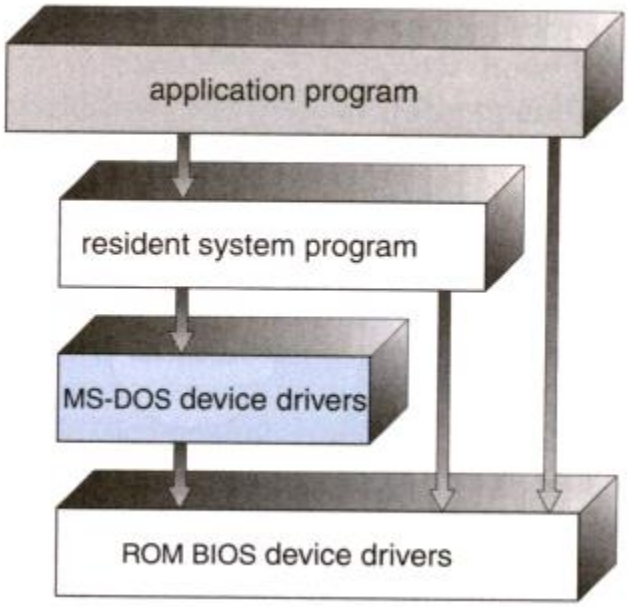
System Implementation

- Traditionally written in assembly language, operating systems are now mostly written in higher level languages.
- Code written in a high-level language:
    - can be written faster.
    - is more compact.
    - is easier to understand and debug.
- An operating system is far easier to port (move to some other hardware) if it is written in a high level language.
- The first OS to be written in a high-level language was UNIX, at the time there were very few suitable languages and so a new one was developed from an existing language called B - it was called C.

Booting

- Modern operating systems are designed to run on machines with a wide range of different hardware.
- Booting - starting a computer by loading the kernel.
- Bootstrap program - code stored in ROM that is able to locate the kernel, load it into memory, and start its execution.
- Once the kernel is loaded it must identify all the hardware present in the machine and load relevant device drivers.

Operating-System Structure

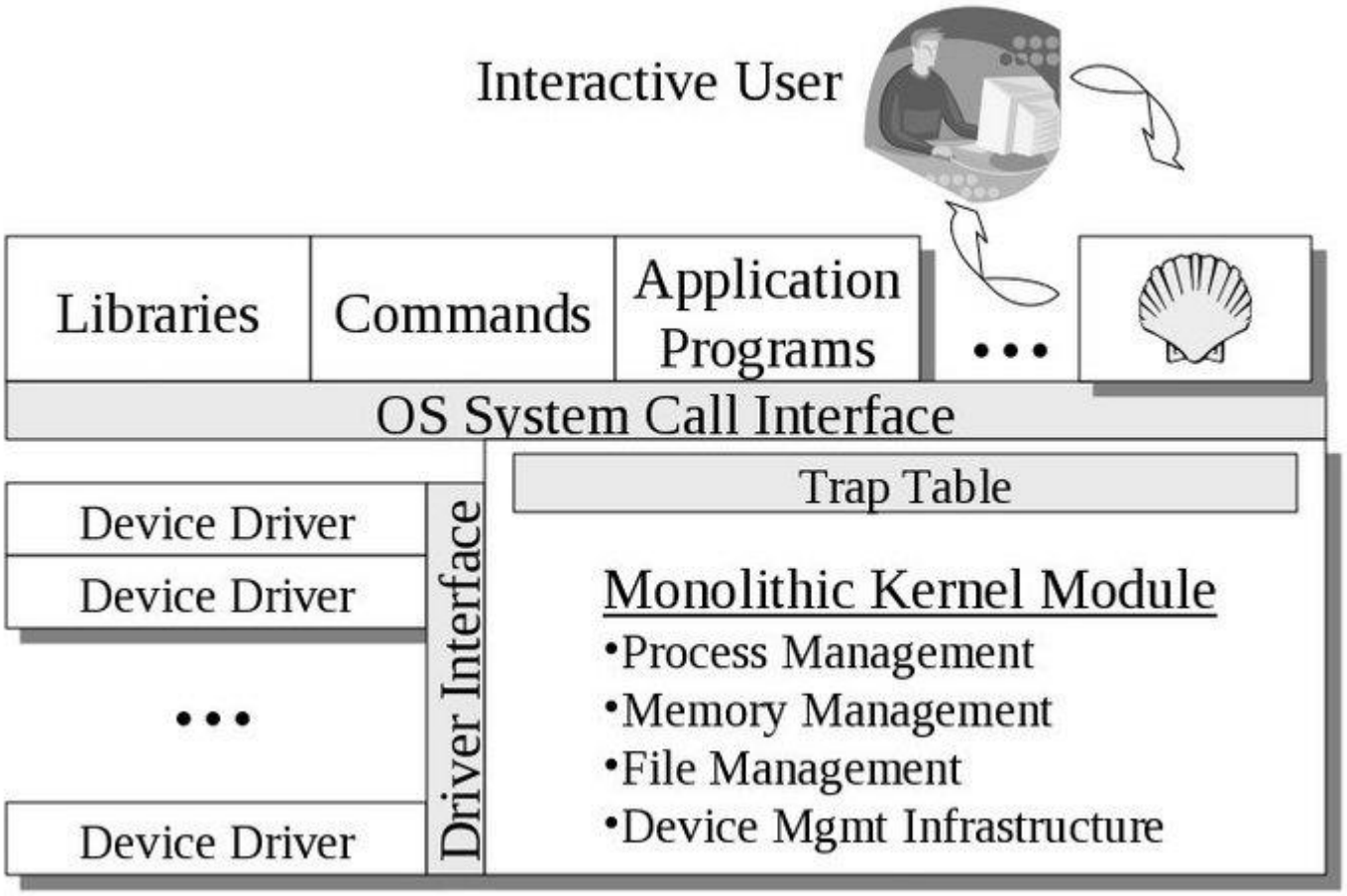


In MS-DOS, applications may bypass the operating system.

- Operating systems such as MS-DOS and the original UNIX did not have well-defined structures.
- There was no CPU Execution Mode (user and kernel), and so errors in applications could cause the whole system to crash.

1.8.2. Monolithic Approach

- Functionality of the OS is invoked with simple function calls within the kernel, which is one large program.
- Device drivers are loaded into the running kernel and become part of the kernel.

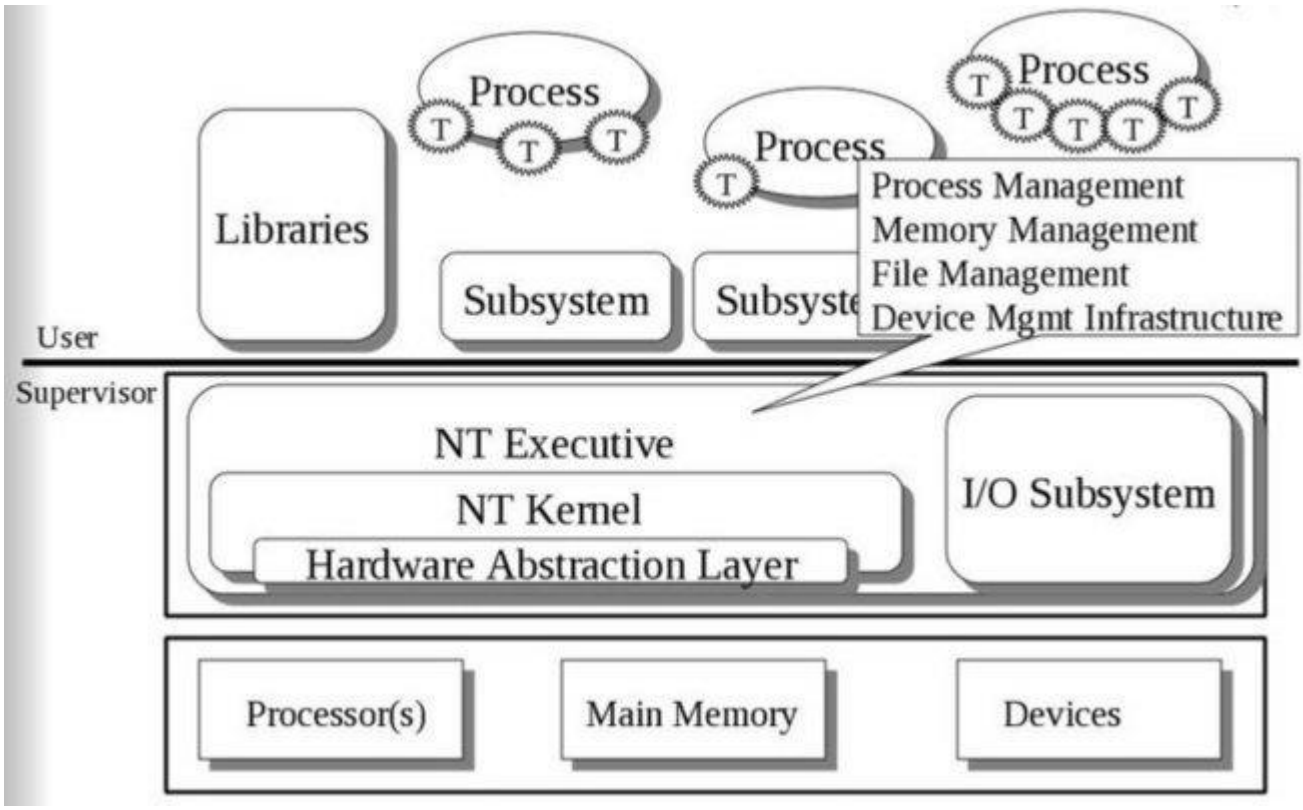


A monolithic kernel, such as Linux and other Unix systems.

1.8.3. Layered Approach

This approach breaks up the operating system into different layers.

- This allows implementers to change the inner workings, and increases modularity.
- As long as the external interface of the routines don't change, developers have more freedom to change the inner workings of the routines.
- With the layered approach, the bottom layer is the hardware, while the highest layer is the user interface.
    - The main *advantage* is simplicity of construction and debugging.
    - The main *difficulty* is defining the various layers.
    - The main *disadvantage* is that the OS tends to be less efficient than other implementations.



The Microsoft Windows NT Operating System. The lowest level is a monolithic kernel, but many OS components are at a higher level, but still part of the OS.

1.8.4. Microkernels

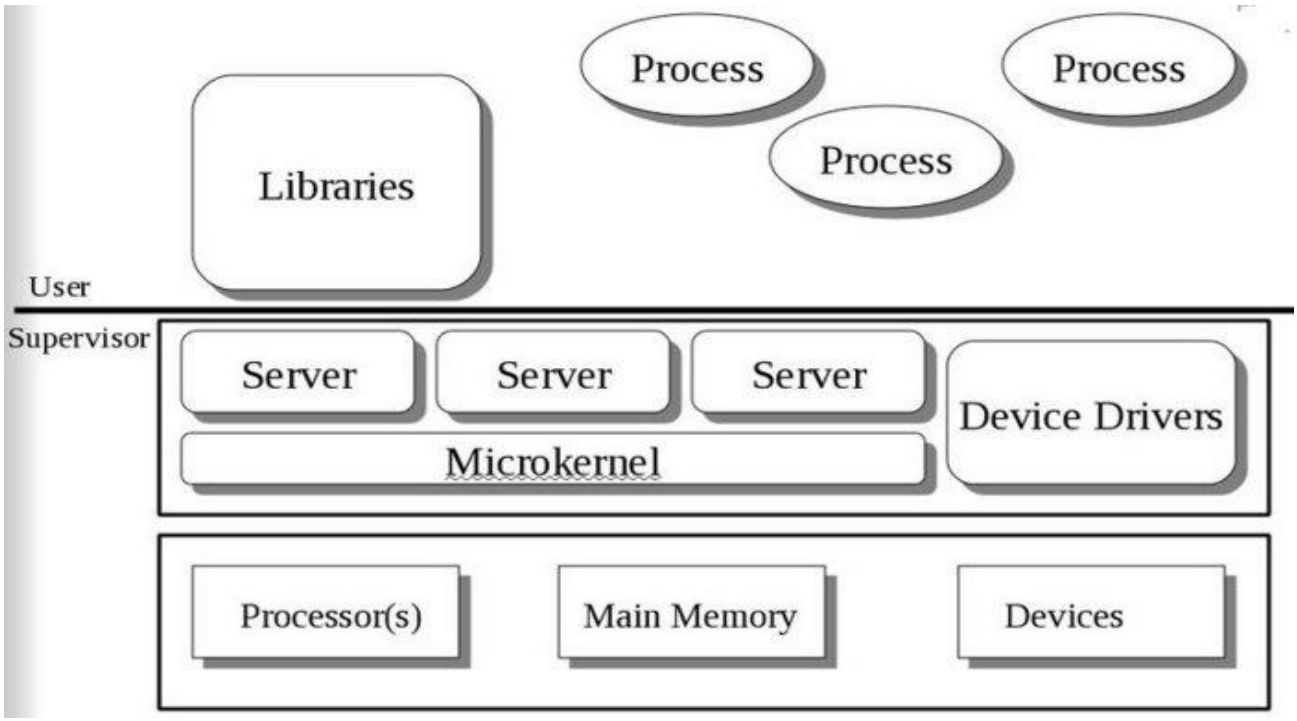
This structures the operating system by removing all nonessential portions of the kernel and implementing them as system and user level programs.

- Generally they provide minimal process and memory management, and a communications facility.
- Communication between components of the OS is provided by message passing.

The *benefits* of the microkernel are as follows:

- Extending the operating system becomes much easier.
- Any changes to the kernel tend to be fewer, since the kernel is smaller.
- The microkernel also provides more security and reliability.

Main *disadvantage* is poor performance due to increased system overhead from message passing.



A Microkernel architecture.

Note

- Andrew Tanenbaum's Minix is an example of a microkernel system. Minix was developed primarily to facilitate teaching graduate level operating system classes. Tanenbaum has authored several text books and is with VA University in Amsterdam.
- Another well known microkernel system is **Mach**, which was developed at Carnegie Mellon University in the mid-1980's. Mach was used as the low-level part of Apple OS X.

PROCESS

A process is a program in execution. Process is not as same as program code but a lot more than it. A process is an 'active' entity as opposed to program which is considered to be a 'passive' entity. Attributes held by process include hardware state, memory, CPU etc.

Process memory is divided into four sections for efficient working :

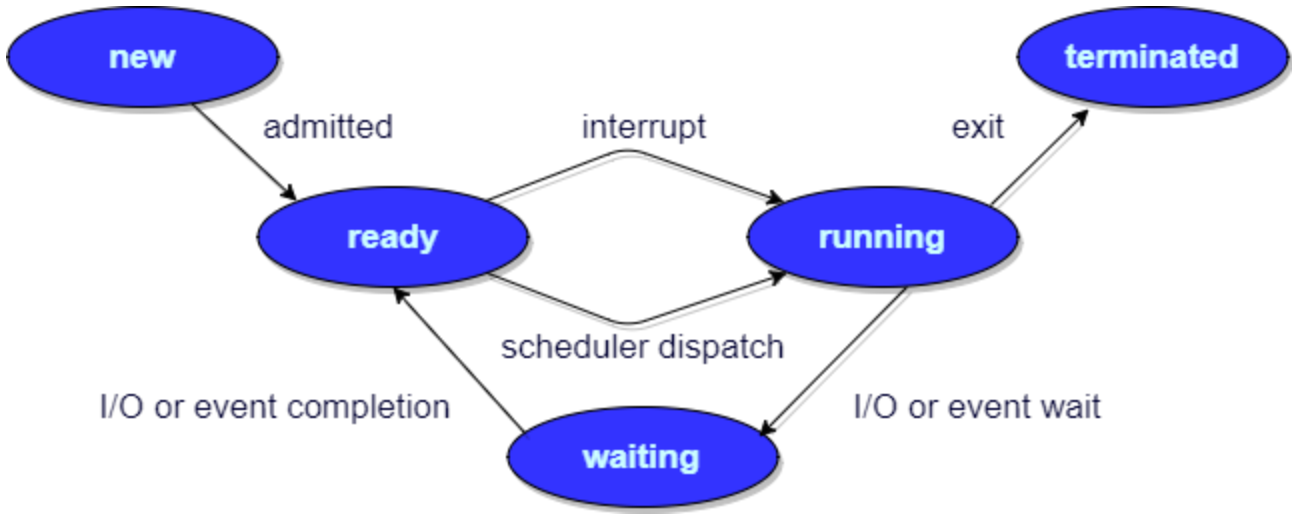
- The text section is made up of the compiled program code, read in from non-volatile storage when the program is launched.

- The data section is made up the global and static variables, allocated and initialized prior to executing the main.
- The heap is used for the dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.
- The stack is used for local variables. Space on the stack is reserved for local variables when they are declared.

PROCESS STATE

Processes can be any of the following states :

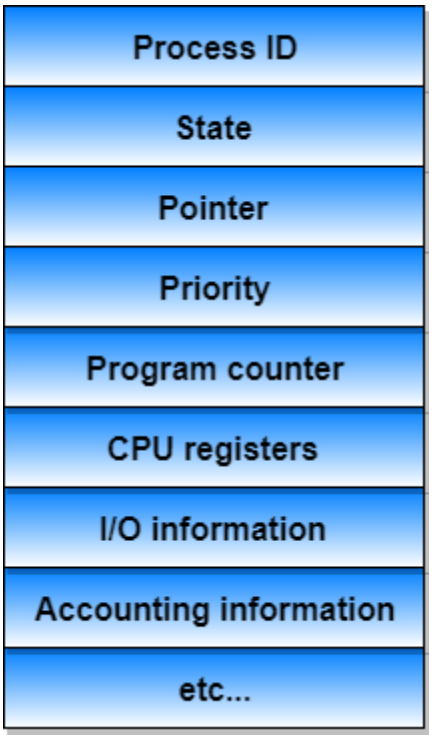
- **New** - The process is being created.
- **Ready** - The process is waiting to be assigned to a processor.
- **Running** - Instructions are being executed.
- **Waiting** - The process is waiting for some event to occur(such as an I/O completion or reception of a signal).
- **Terminated** - The process has finished execution.



PROCESS CONTROL BLOCK

There is a Process Control Block for each process, enclosing all the information about the process. It is a data structure, which contains the following :

- Process State - It can be running, waiting etc.
- Process ID and parent process ID.
- CPU registers and Program Counter. **Program Counter** holds the address of the next instruction to be executed for that process.
- CPU Scheduling information - Such as priority information and pointers to scheduling queues.
- Memory Management information - Eg. page tables or segment tables.
- Accounting information - user and kernel CPU time consumed, account numbers, limits, etc.
- I/O Status information - Devices allocated, open file tables, etc.



CPU SCHEDULING

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

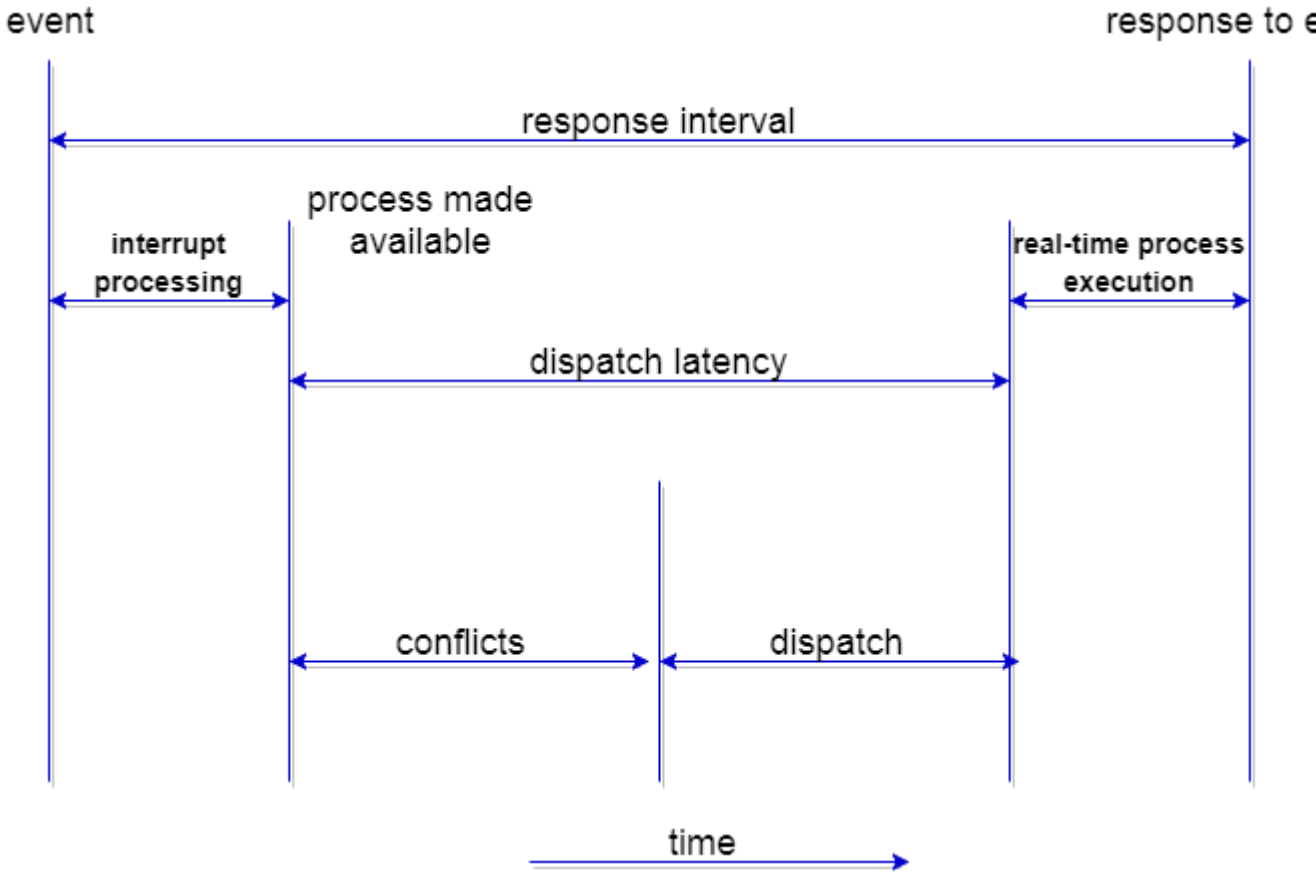
Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

Dispatcher

Another component involved in the CPU scheduling function is the **dispatcher**. The dispatcher is the module that gives control of the CPU to the process selected by the **short-term scheduler**. This function involves:

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program

The dispatcher should be as fast as possible, given that it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the **dispatch latency**. Dispatch Latency can be explained using the below figure:



Types of CPU Scheduling

CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the **running** state to the **waiting** state(for I/O request or invocation of wait for the termination of one of the child processes).
2. When a process switches from the **running** state to the **ready** state (for example, when an interrupt occurs).
3. When a process switches from the **waiting** state to the **ready** state(for example, completion of I/O).
4. When a process **terminates**.

In circumstances 1 and 4, there is no choice in terms of scheduling. A new process(if one exists in the ready queue) must be selected for execution. There is a choice, however in circumstances 2 and 3.

When Scheduling takes place only under circumstances 1 and 4, we say the scheduling scheme is **non-preemptive**; otherwise the scheduling scheme is **preemptive**.

Non-Preemptive Scheduling

Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

This scheduling method is used by the Microsoft Windows 3.1 and by the Apple Macintosh operating systems.

It is the only method that can be used on certain hardware platforms, because It does not require the special hardware(for example: a timer) needed for preemptive scheduling.

Preemptive Scheduling

In this type of Scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

SCHEDULING CRITERIA

There are many different criterias to check when considering the "best" scheduling algorithm :

- **CPU utilization**

To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

- **Throughput**

  It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

- **Turnaround time**

  It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process(Wall clock time).

- **Waiting time**

  The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

- **Load average**

  It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

- **Response time**

  Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

Scheduling Algorithms

We'll discuss four major scheduling algorithms here which are following :

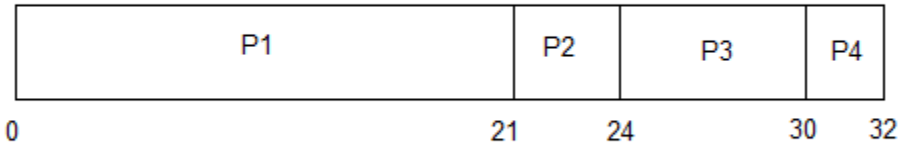
1. First Come First Serve(FCFS) Scheduling
2. Shortest-Job-First(SJF) Scheduling
3. Priority Scheduling
4. Round Robin(RR) Scheduling
5. Multilevel Queue Scheduling
6. Multilevel Feedback Queue Scheduling

First Come First Serve(FCFS) Scheduling

- Jobs are executed on first come, first serve basis.
- Easy to understand and implement.
- Poor in performance as average wait time is high.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The average waiting time will be = (0 + 21 + 24 + 30)/4 = 18.75 ms



This is the GANTT chart for the above processes

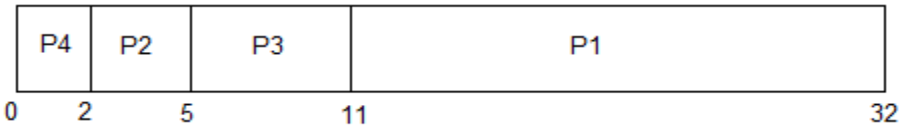
Shortest-Job-First(SJF) Scheduling

- Best approach to minimize waiting time.
- Actual time taken by the process is already known to processor.
- Impossible to implement.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

In Shortest Job First Scheduling, the shortest Process is executed first.

Hence the GANTT chart will be following :

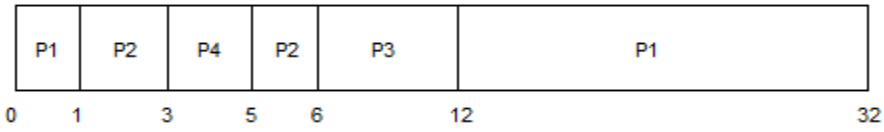


Now, the average waiting time will be = (0 + 2 + 5 + 11)/ 4 = 4.5 ms

In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with short burst time arrives, the existing process is preempted.

PROCESS	BURST TIME	ARRIVAL TIME
P1	21	0
P2	3	1
P3	6	2
P4	2	3

The GANTT chart for Preemptive Shortest Job First Scheduling will be,



The average waiting time will be, ((5-3) + (6-2) + (12-1))/4 = 4.25 ms

The average waiting time for preemptive shortest job first scheduling is less than both, non-preemptive SJF scheduling and FCFS scheduling.

Priority Scheduling

- Priority is assigned for each process.
- Process with highest priority is executed first and so on.
- Processes with same priority are executed in FCFS manner.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

PROCESS	BURST TIME	PRIORITY
P1	21	2
P2	3	1
P3	6	4
P4	2	3

The GANTT chart for following processes based on Priority scheduling will be,

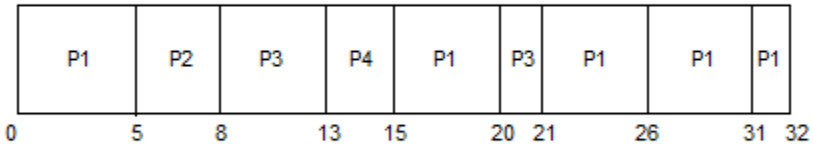
The average waiting time will be, (0 + 3 + 24 + 26)/4 = 13.25 ms

Round Robin(RR) Scheduling

- A fixed time is allotted to each process, called **quantum**, for execution.
- Once a process is executed for given time period that process is preemptied and other process executes for given time period.
- Context switching is used to save states of preemptied processes.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The GANTT chart for round robin scheduling will be,



The average waiting time will be, 11 ms.

Multilevel Queue Scheduling

Another class of scheduling algorithms has been created for situations in which processes are easily classified into different groups.

For example: A common division is made between foreground(or interactive) processes and background (or batch) processes. These two types of processes have different response-time requirements, and so might have different scheduling needs. In addition, foreground processes may have priority over background processes.

A multi-level queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.

For example: separate queues might be used for foreground and background processes. The foreground queue might be scheduled by Round Robin algorithm, while the background queue is scheduled by an FCFS algorithm.

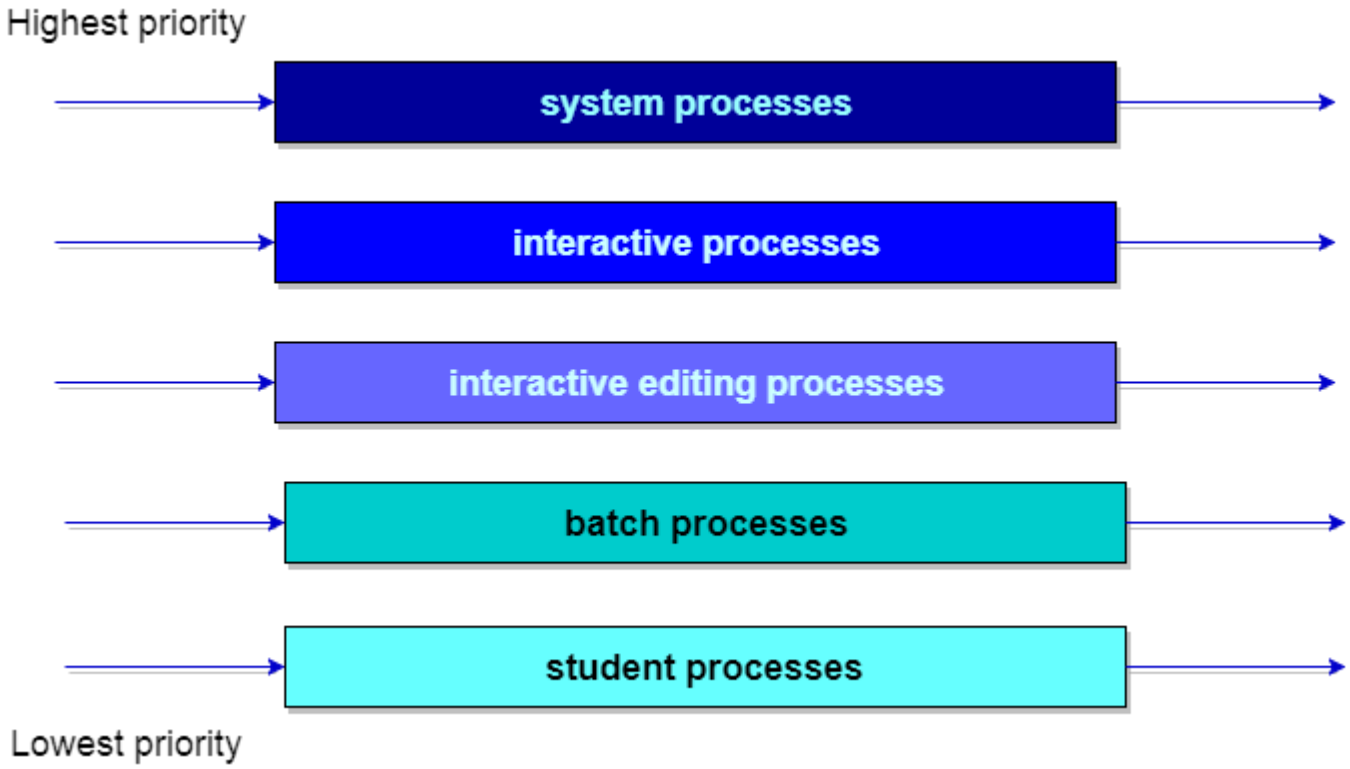
In addition, there must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling. **For example:** The foreground queue may have absolute priority over the background queue.

Let us consider an example of a multilevel queue-scheduling algorithm with five queues:

1. System Processes

2. Interactive Processes
3. Interactive Editing Processes
4. Batch Processes
5. Student Processes

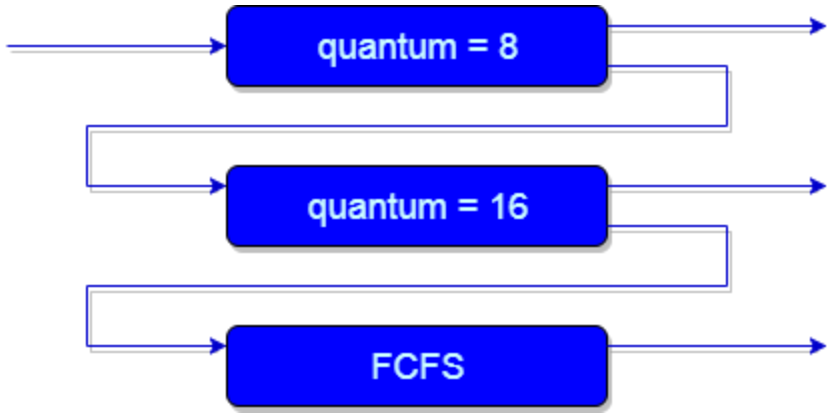
Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process will be preempted.



Multilevel Feedback Queue Scheduling

In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. This setup has the advantage of low scheduling overhead, but the disadvantage of being inflexible.

Multilevel feedback queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.



An example of a multilevel feedback queue can be seen in the below figure.

In general, a multilevel feedback queue scheduler is defined by the following parameters:

- The number of queues.
- The scheduling algorithm for each queue.
- The method used to determine when to upgrade a process to a higher-priority queue.
- The method used to determine when to demote a process to a lower-priority queue.
- The method used to determine which queue a process will enter when that process needs service.

The definition of a multilevel feedback queue scheduler makes it the most general CPU-scheduling algorithm. It can be configured to match a specific system under design. Unfortunately, it also requires some means of selecting values for all the parameters to define the best scheduler. Although a multilevel feedback queue is the **most general scheme**, it is also the **most complex**.

OVERVIEW OF MAIN MEMORY

Main Memory refers to a physical memory that is the internal memory to the computer. The word main is used to distinguish it from external mass storage devices such as disk drives. Main memory is also known as RAM. The computer is able to change only data that is in main memory. Therefore, every program we execute and every file we access must be copied from a storage device into main memory.

All the programs are loaded in the main memeory for execution. Sometimes complete program is loaded into the memory, but some times a certain part or routine of the program is loaded into the main memory only when it is called by the program, this mechanism is called Dynamic Loading, this enhance the performance.

Also, at times one program is dependent on some other program. In such a case, rather than loading all the dependent programs, CPU links the dependent programs to the main executing program when its required. This mechanism is known as Dynamic Linking.

Swapping

A process needs to be in memory for execution. But sometimes there is not enough main memory to hold all the currently active processes in a timesharing system. So, excess process are kept on disk and brought in to run dynamically. Swapping is the process of bringing in each process in main memory, running it for a while and then putting it back to the disk.

Contiguous Memory Allocation

In contiguous memory allocation each process is contained in a single contiguous block of memory. Memory is divided into several fixed size partitions. Each partition contains exactly one process. When a partition is free, a process is selected from the input queue and loaded into it. The free blocks of memory are known as *holes*. The set of holes is searched to determine which hole is best to allocate.

Memory Protection

Memory protection is a phenomenon by which we control memory access rights on a computer. The main aim of it is to prevent a process from accessing memory that has not been allocated to it. Hence prevents a bug within a process from affecting other processes, or the operating system itself, and instead results in a segmentation fault or storage violation exception being sent to the disturbing process, generally killing of process.

Memory Allocation

Memory allocation is a process by which computer programs are assigned memory or space. It is of three types :

1. **First Fit**

   The first hole that is big enough is allocated to program.

2. **Best Fit**

   The smallest hole that is big enough is allocated to program.

3. **Worst Fit**

   The largest hole that is big enough is allocated to program.

Fragmentation

Fragmentation occurs in a dynamic memory allocation system when most of the free blocks are too small to satisfy any request. It is generally termed as inability to use the available memory.

In such situation processes are loaded and removed from the memory. As a result of this, free holes exists to satisfy a request but is non contiguous i.e. the memory is fragmented into large no. Of small holes. This phenomenon is known as **External Fragmentation.**

Also, at times the physical memory is broken into fixed size blocks and memory is allocated in unit of block sizes. The memory allocated to a space may be slightly larger than the requested memory. The difference between allocated and required memory is known as **Internal fragmentation** i.e. the memory that is internal to a partition but is of no use.

Paging

A solution to fragmentation problem is Paging. Paging is a memory management mechanism that allows the physical address space of a process to be non-contagious. Here physical memory is divided into blocks of equal size called **Pages**. The pages belonging to a certain process are loaded into available memory frames.

Page Table

A Page Table is the data structure used by a virtual memory system in a computer operating system to store the mapping between *virtual address* and *physical addresses.*

Virtual address is also known as Logical address and is generated by the CPU. While Physical address is the address that actually exists on memory.

Segmentation

Segmentation is another memory management scheme that supports the user-view of memory. Segmentation allows breaking of the virtual address space of a single process into segments that may be placed in non-contiguous areas of physical memory.

Segmentation with Paging

Both paging and segmentation have their advantages and disadvantages, it is better to combine these two schemes to improve on each. The combined scheme is known as 'Page the Elements'.

Each segment in this scheme is divided into pages and each segment is maintained in a page table. So the logical address is divided into following 3 parts :

- Segment numbers(S)
- Page number (P)
- The displacement or offset number (D)

Virtual Memory

Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as large virtual memory is provided for user programs when a very small physical memory is there.

In real scenarios, most processes never need all their pages at once, for following reasons :

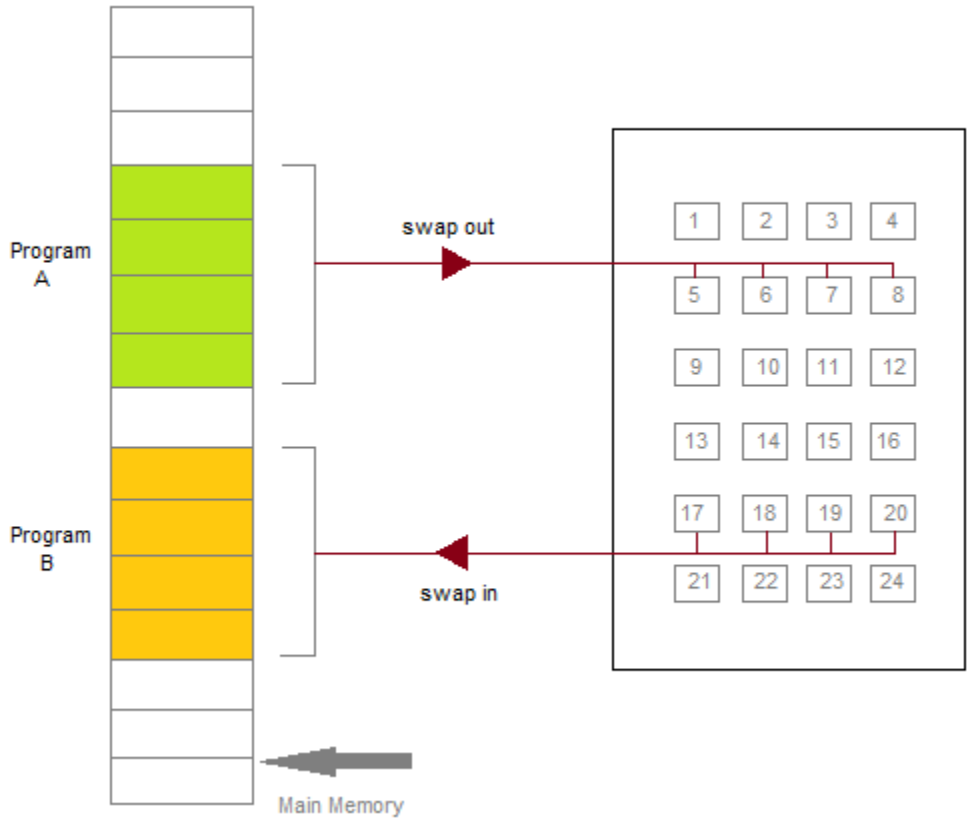
- Error handling code is not needed unless that specific error occurs, some of which are quite rare.
- Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
- Certain features of certain programs are rarely used.

Benefits of having Virtual Memory :

1. Large programs can be written, as virtual space available is huge compared to physical memory.
2. Less I/O required, leads to faster and easy swapping of processes.
3. More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.

Demand Paging

The basic idea behind demand paging is that when a process is swapped in, its pages are not swapped in all at once. Rather they are swapped in only when the process needs them(On demand). This is termed as lazy swapper, although a pager is a more accurate term.



Initially only those pages are loaded which will be required the process immediately.

The pages that are not moved into the memory, are marked as invalid in the page table. For an invalid entry the rest of the table is empty. In case of pages that are loaded in the memory, they are marked as valid along with the information about where to find the swapped out page.

When the process requires any of the page that is not loaded into the memory, a page fault trap is triggered and following steps are followed,

1. The memory address which is requested by the process is first checked, to verify the request made by the process.
2. If its found to be invalid, the process is terminated.
3. In case the request by the process is valid, a free frame is located, possibly from a free-frame list, where the required page will be moved.
4. A new operation is scheduled to move the necessary page from disk to the specified memory location. (This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime.)

5. When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to valid.
6. The instruction that caused the page fault must now be restarted from the beginning.

There are cases when no pages are loaded into the memory initially, pages are only loaded when demanded by the process by generating page faults. This is called **Pure Demand Paging**.

The only major issue with Demand Paging is, after a new page is loaded, the process starts execution from the beginning. Its is not a big issue for small programs, but for larger programs it affects performance drastically.

Page Replacement

As studied in Demand Paging, only certain pages of a process are loaded initially into the memory. This allows us to get more number of processes into the memory at the same time. but what happens when a process requests for more pages and no free memory is available to bring them in. Following steps can be taken to deal with this problem :

1. Put the process in the wait queue, until any other process finishes its execution thereby freeing frames.
2. Or, remove some other process completely from the memory to free frames.
3. Or, find some pages that are not being used right now, move them to the disk to get free frames. This technique is called **Page replacement** and is most commonly used. We have some great algorithms to carry on page replacement efficiently.

Basic Page Replacement

- Find the location of the page requested by ongoing process on the disk.
- Find a free frame. If there is a free frame, use it. If there is no free frame, use a page-replacement algorithm to select any existing frame to be replaced, such frame is known as **victim frame**.
- Write the victim frame to disk. Change all related page tables to indicate that this page is no longer in memory.

- Move the required page and store it in the frame. Adjust all related page and frame tables to indicate the change.
- Restart the process that was waiting for this page.

FIFO Page Replacement

- A very simple way of Page replacement is FIFO (First in First Out)
- As new pages are requested and are swapped in, they are added to tail of a queue and the page which is at the head becomes the victim.
- Its not an effective way of page replacement but can be used for small systems.

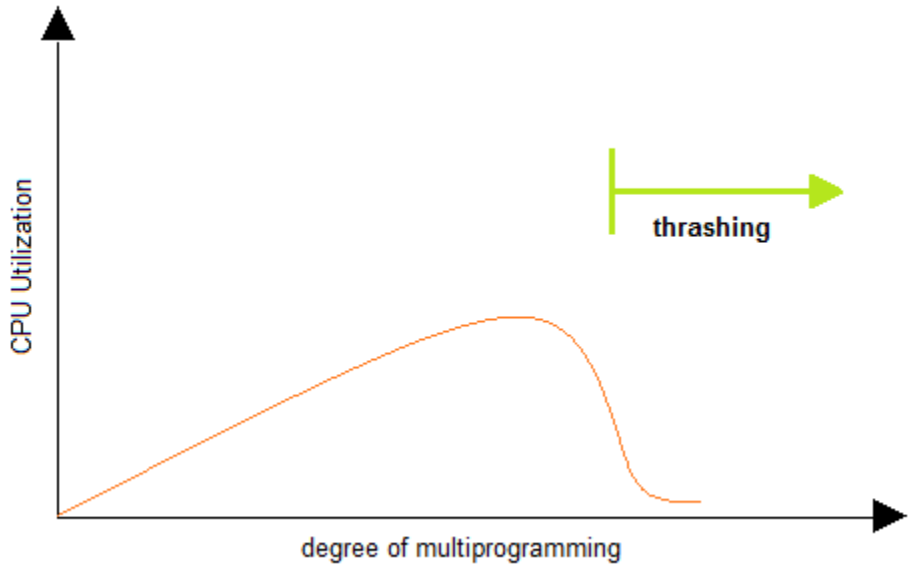
LRU Page Replacement

Below is a video, which will explain LRU Page replacement algorithm in details with an example.

Thrashing

A process that is spending more time paging than executing is said to be thrashing. In other words it means, that the process doesn't have enough frames to hold all the pages for its execution, so it is swapping pages in and out very frequently to keep executing. Sometimes, the pages which will be required in the near future have to be swapped out.

Initially when the CPU utilization is low, the process scheduling mechanism, to increase the level of multiprogramming loads multiple processes into the memory at the same time, allocating a limited amount of frames to each process. As the memory fills up, process starts to spend a lot of time for the required pages to be swapped in, again leading to low CPU utilization because most of the proccesses are waiting for pages. Hence the scheduler loads more processes to increase CPU utilization, as this continues at a point of time the complete system comes to a stop.



To prevent thrashing we must provide processes with as many frames as they really need "right now".
