

OPERATING SYSTEM

By :- Rahul kumar


What Is Operating system?

- An **operating system** is a piece of software that manages all the resources of a computer system, both hardware and software, and provides an environment in which the user can execute his/her programs in a convenient and efficient manner by hiding underlying complexity of the hardware.
- Act as interface between user and computer hardware.
User->Application Program -> Operating System -> Computer Hardware

Goals and need of OS :

- Act as resource manager
- Provide memory protection.
- Since OS interacts with the hardware, so normal program doesn't need to have hardware **interaction** code.
- Runs the program by providing isolation and protection.

Types of Operating System :

1. **Single process OS**, only 1 process executes at a time from the ready queue.
2. **Batch-processing OS** :- Single CPU used.
3. **Multi-Programming OS** increases CPU utilization by keeping multiple jobs (code and data) in the memory so that the CPU always has one to execute in case some job gets busy with I/O. Ex THE
4. **Multitasking OS** able to run more than one task with the help of time sharing and context switching. Ex ~~Windows~~  CTS, M17

5. **Multi-processing OS**, more than 1 CPU in a single computer. Increases reliability, Lesser process starvation. *ex - Windows*
6. **Distributed OS**, manages many bunches of resources, ≥ 1 CPUs, ≥ 1 memory, ≥ 1 GPUs, etc - Loosely connected autonomous, interconnected computer nodes. Ex LOCUS.
7. **RTOS** Real time error free, computations within tight-time boundaries. - Air Traffic control system, ROBOTS etc.

Components of OS

1. **Kernel**: A kernel is that part of the operating system which interacts directly with the hardware and performs the most crucial tasks. Heart of OS/Core component.
2. **User space**: Where application software runs, apps don't have privileged access to the underlying hardware. It interacts with kernel. They are GUI and CLI.

Function of Kernel :

1. **Process management**: a. Scheduling processes and threads on the CPUs. Creating & deleting both user and system process. Suspending and resuming processes. Providing mechanisms for process synchronization or process communication
2. **Memory management** : Allocating and deallocating memory space as per need. Keeping track of which part of memory are currently being used and by which process.
3. **File management** : Creating and deleting files. Creating and deleting directories to organize files
4. **I/O management**: to manage and control I/O operations and I/O devices

Types of Kernel :

1. **Monolithic kernel** : All functions are in kernel itself. Bulky in size. Memory required to run is high. Less reliable, one module crashes whole kernel is down. High performance as communication is fast. (Less user mode, kernel mode overheads) f. Eg. Linux, Unix, MS-DOS.
2. **Micro Kernel** : Only major functions are in **kernel**. i.e Memory mgmt. and Process mgmt. File mgmt. and IO mgmt. are in **User-space**. smaller in size. More Reliable. More stable. Performance is slow. **Overhead** switching b/w user mode and kernel mode. h. Eg. L4 Linux, Symbian OS, MINIX etc
3. **Hybrid Kernel**: Advantages of both worlds. (File mgmt. in User space and rest in Kernel space.) b. Combined approach. Eg. MacOS, Windows NT/7/10 f. IPC also happens but lesser overheads.

IPC (Inter Process Communication) :

- IPC is used to communicate between user mode and kernel mode. Apps interact with kernel using system calls.
- It is a mechanism using which user app interact with kernel for such service for which it does not have the permission to perform. Only way is to communicate from user mode to kernel mode.

Process and Threads :

- **Program** : It is an executable file which contains a certain set of instructions written to complete a specific job or operation on your computer. Is compiled code ready to be executed. Stored in disk.
- **Process** : Program under execution lies in RAM (Primary memory).
- **Threads** : When process is divided into different sub-tasks which can be executed **independently**. Light-weight process. It has its own path of execution. **parallelism** can be achieved.

Multi-Tasking	Multi-Threading
<ul style="list-style-type: none"> • Execution of more than 1 task simultaneously. 	<ul style="list-style-type: none"> • When process is divided into different sub-task which can be executed independently
<ul style="list-style-type: none"> • No. of CPU can be 1. 	<ul style="list-style-type: none"> • No. of CPU ≥ 1.
<ul style="list-style-type: none"> • Isolation and memory protection exists. Because for different process different memory has to be allocated by the OS. 	<ul style="list-style-type: none"> • No isolation and memory protection, resources are shared among threads of that process. OS allocates memory to a process; multiple threads of that process share the same memory and resources allocated to the process.
<ul style="list-style-type: none"> • Process are context switched . 	<ul style="list-style-type: none"> • Threads are context switched.

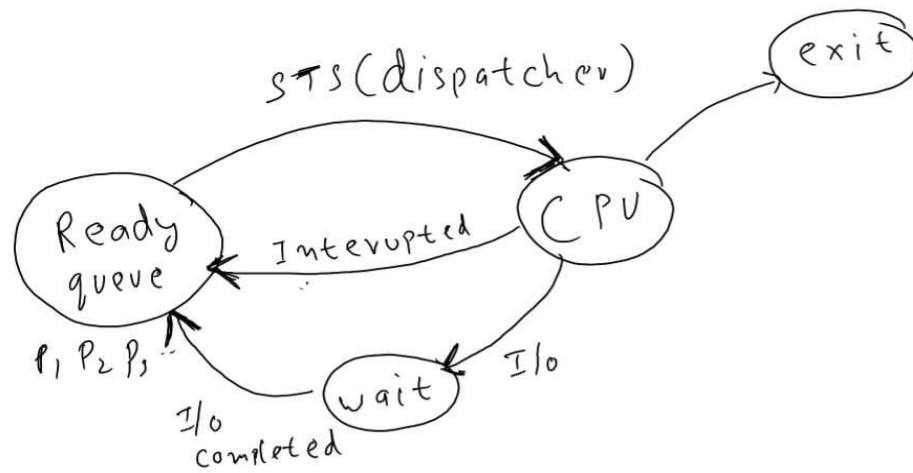
Note: context switching : When CPU switches from one process to another process than the kernel saves the context of the old process in its PCB (Process Control Block) restores the content of the new process scheduled to run. It is pure **overhead**. Speed varies from machine to machine.

Process Scheduling Algorithms

Some Common terms :-

1. **Process Scheduling:** Basics of multi-programming, by switching the CPU among the process the OS can make the computer more productive.
2. **Non-preemptive Scheduling:** Once CPU is allocated to a process, the process keeps the CPU until it is terminated, or it is in wait state. (I/O state).
3. **Preemptive Scheduling:** CPU is taken away from a process after time quantum expires as well as it is terminated.
4. **Throughput:** Number of process completed per unit time.
5. **Arrival Time (AT):** The time when process is arrived at the ready Queue.
6. **Brust Time (BT):** The time required by the process for its execution.
7. **Turnaround Time (TAT):** The time taken from the first time process enter the ready state till it terminates. $(CT-AT)$.
8. **Wait Time:** Time process spends waiting for the CPU $(TAT-BT)$.
9. **Response Time:** Time duration between process getting into ready queue and process getting CPU for 1st time.

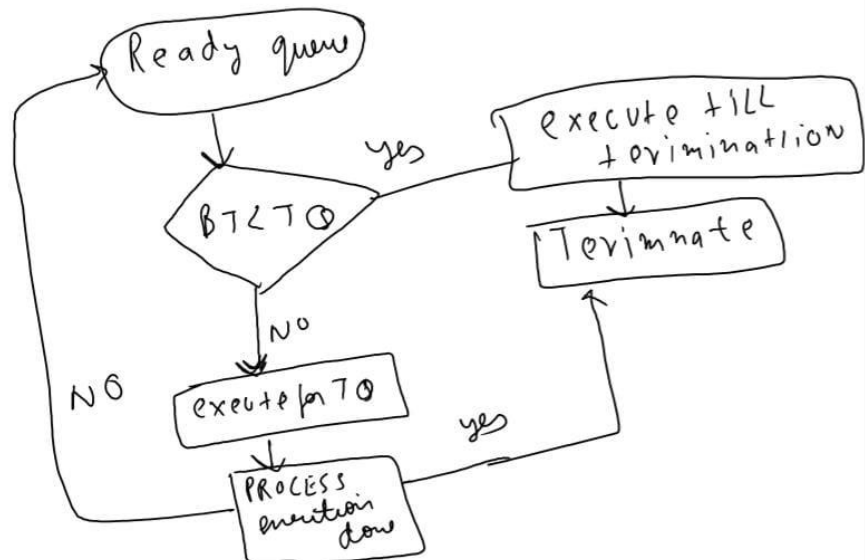
10. **Completion Time (CT):** Time taken till process gets terminated.



Scheduling Algorithms

1. **FCFS (First Come First Serve)** : Which ever process comes first in ready queue will be given the CPU 1st. Has convo effect.
2. **Shortest Job First (SJF) (Non-preemptive)** : Process with least BT will be dispatched to CPU first. Since Burst time estimation is impossible therefore it is impossible to implement.
3. **Shortest Job First (SJF) (Preemptive)** : Less starvation, no convo effect, optimal but impossible to implement.
4. **Priority Scheduling (Non-Preemptive)** : Priority is assigned to a process when it is created. SJF is special case when priority is inversely proportional to BT. *aging technique in priority scheduling.*
5. **Priority Scheduling (Preemptive)** : Current run state is pre-empted if next job(process) has higher priority. May casue indefinite waiting(starvation) for lower priority jobs.
6. **Round Robin Scheduling (RR)** : One of the most famous algo. FCFS but preemptive. It is designed for the time sharing system. Based on AT+ time quantum and not BT. No convo effect, easy to implement.

ROUND ROBIN :



7. **Multi-level queue scheduling (MLQ)** : Ready queue is divided into multiple queues depending upon priority. A process is permanently assigned to one of the queues (inflexible) based on some property of process, memory, size, process priority or process type. Each queue has its own scheduling algorithm. E.g., SP \rightarrow RR, IP \rightarrow RR & BP \rightarrow FCFS. If an IP process comes and BP process is currently executing than BP will be pre-empted. Therefore starvation and convo effect for lower-priority process.

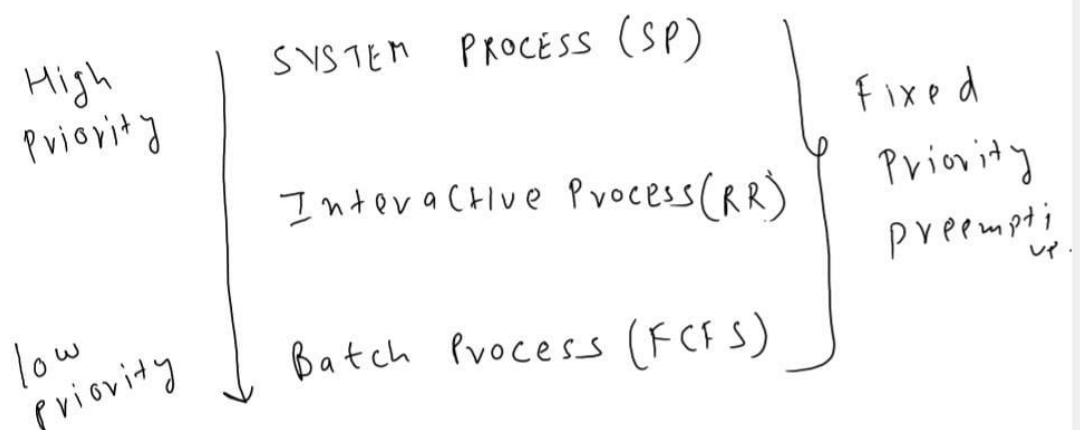


Fig: MLQ

Critical Section:

- The critical section refers to the segment of code where processes/threads access shared resources, such as common variables and files, and perform write operations on them.
- **Race Condition:** A race condition occurs when two or more threads can access shared data and they try to change it at the same time. Because the thread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data. Therefore, the result of the change in data is dependent on the thread scheduling algorithm, i.e., both threads are "racing" to access/change the data.
- **Solution to Race Condition**
 1. **Atomic operations:** Make Critical code section an atomic operation, i.e., Executed in one CPU cycle.
 2. **Semaphores**
 3. Mutual exclusion using locks.

Semaphores

- Synchronization method
- An integer that is equal to number of resources.
- Multiple threads can go and execute C.S concurrently.
- Allows multiple program threads to access the finite instances of resources whereas mutex allows multiple threads to access a single shared resource one at a time.
- Two types : Binary Semaphore : value can be 0 or 1. (Mutex/Locks)
- Counting Semaphore : Can range over an unrestricted domain.

P()/ wait \rightarrow C.S \rightarrow V()/ Signal.

Wait (Semaphore S)

```
{  
S=s-1;  
If(s<0)  
{  
Put process in wait or sleep state().  
}  
Else return // put the process in critical section.  
}
```

C.S execution.

Signal (Semaphore S)

```
{  
S=s+1;  
If(s<=0)  
{  
Select the process from suspended list or wait list and wake_up().  
}  
}
```

DeadLock

- In Multi-programming environment, we have several processes competing for finite number of resources
- Process requests a resource (R), if R is not available (taken by other process), process enters in a waiting state. Sometimes that waiting process is never able to change its state because the resource, it has requested is busy (forever), called DEADLOCK (DL)
- Two or more processes are waiting on some resource's availability, which will never be available as it is also busy with some other process. The Processes are said to be in Deadlock.

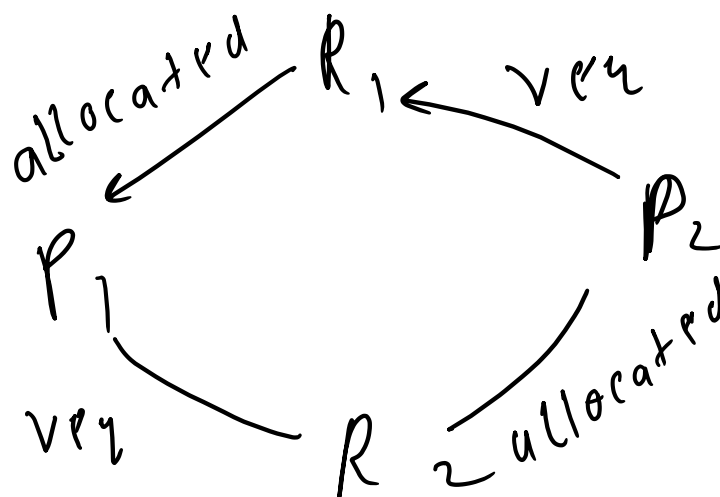


Fig deadlock condⁿ

DeadLock condition: (oop. Is deadlock avoidance).

1. **Mutual exclusion:** only 1 process at a time can use the resource, if another process request that resource, the requesting process must wait until the resource has been released.
2. **Hold&wait:** A process must be holding one resource and waiting to acquire additional resource that is held by another process.
3. **No pre-emption:** Resource must be voluntarily released by the process after completion of execution.

4. **Circular wait** : The wait of different resources by different process should be in such a way that it should form a cycle.

Recovery from dead Lock :

1. **Process termination**: Abort all DL processes . Abort one process at a time until DL cycle is eliminated.
2. **Resource pre-emption**: To eliminate DL, we successively preempt some resources from processes and give these resources to other processes until DL cycle is broken.

Deadlock Avoidance :

The kernel is given in advance about the info which resources will be use in its lifetime for a process. Schedule a process & its resource such that Deadlock never occur.

Safe State: A state is safe if the system can allocate resources to each process and still avoid DL.

Unsafe state:- A state in which process allocation can lead to deadlock.

Bankers Algo :- When a process request a set of resources, the system must determine when allocating there resource will leave the system in a safe state. If yes, then the resource is allocated. If No. then that process must wait till other process release enough resources.