Operating System

It is a system software and it acts as an interface between user and computer hardware.

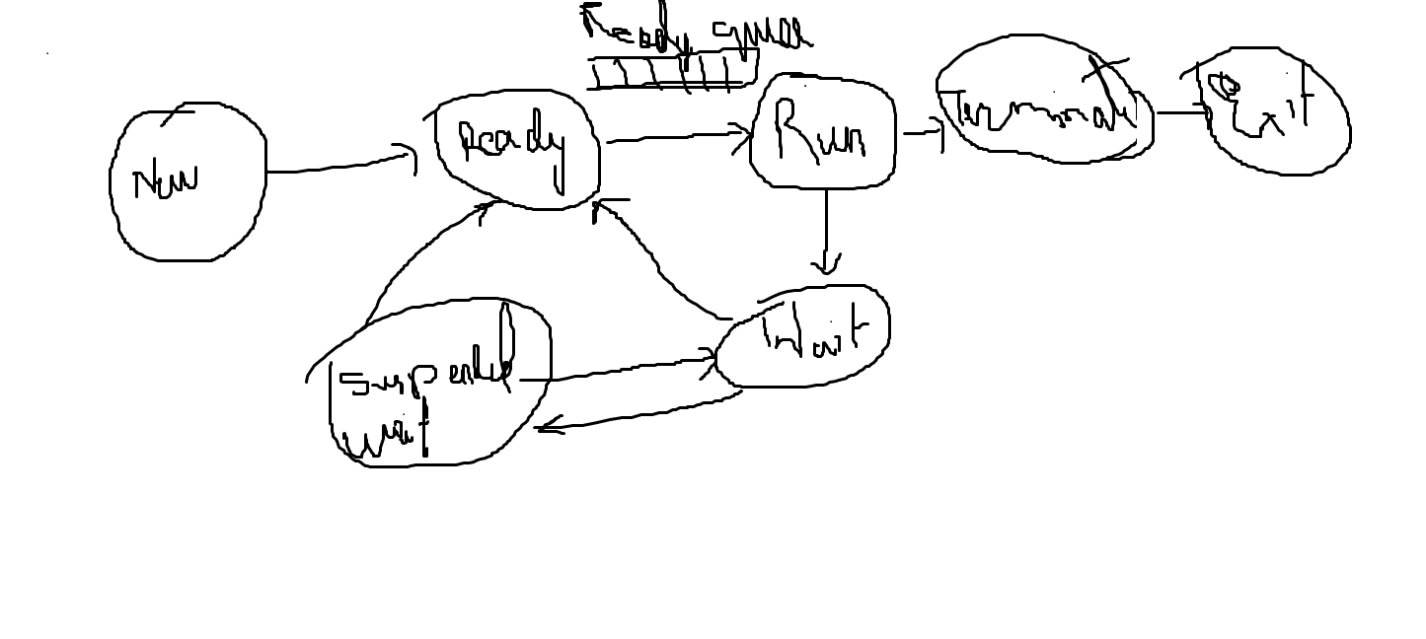
Functions of OS:

1. Process Management: Creating, exceuting and Terminating processes
2. Memory Management: allocation and deallocation
3. File Management: Creation, access, writing files
4. Device Management: Handling I/O Devices

Working of print function:

1. Printf function internally calls write() system call
2. Write() system call contains a set of statements
3. These statements are executed by OS with the help of Device Drivers
4. OS will access the hardware like monitor
5. Corresponding data will be printed on the console.

Process State Diagram:



In the new state, new process will be created.

New Process will be created by the fork() system call.

All newly created processes will be admitted into ready state,If the process is ready for the execution

Moving the processes from new state to ready state. It is the responsibility of Long Term Scheduler.

Now from the ready state, the process are taken and kept in a ready queue, It is the responsibility of short term scheduler.

Now from ready queue, one of the process will move to run state and the corresponding process will be executed by the CPU.

If the process is successfully executed, then corresponding process wil be terminated.

But if the process requires any other I/O devices, temporarily the corresponding process will move to Wait State.

Once the process acquires the I/O devices, then it will complete I/O operations and the corresponding process will move to Ready State from Wait State.

Based on the availability of the ready queue, the corresponding process will move to Run State and the process will be executed.

Dispatcher will move the process from ready queue to run state.

Preemptive and Non Preemptive Priorioty:-

In Preemptive priority, the process are executed based on priority.

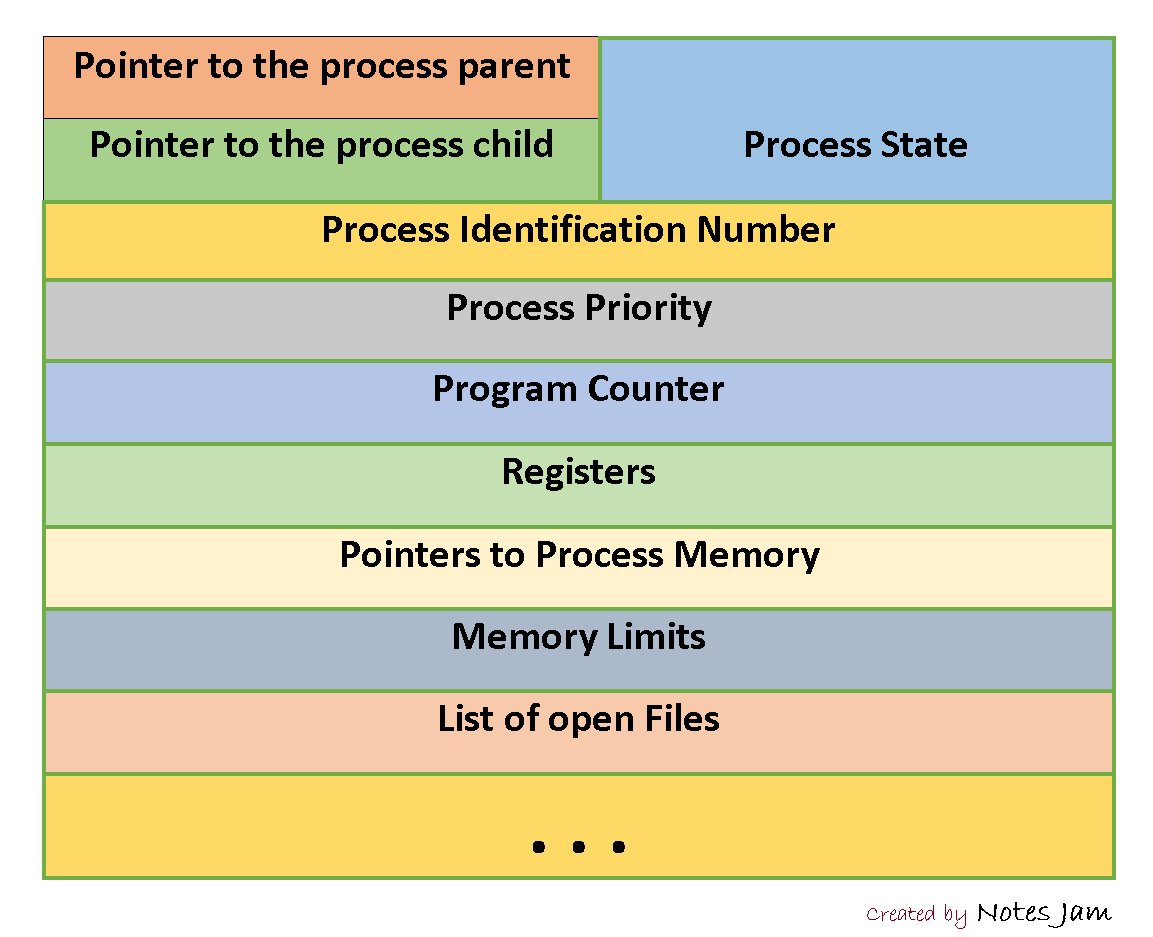
In NonPreemptive Priority, the process are executed based on the order of process creation. The process which is created first will be executed first.

The process with the lower priority is sent to wait state when a process with higher priority comes in Preemptive Priority.

Context Switching refers to switching from a lower priority task to a higher priority task or Switching from one process to another.

But we don’t directly switch tasks, We save the information related to lowest priority in one place like registers and program counter, then we can perform context switching.

PCB (Process Control Block):



Resource Allocation File

Process State: The state of the process will be indicated here. (Ready, Run , Wait, Etc.)

Process Identification Number: Every Process will have a unique number and a unique number will be created in New State by fork() system call.

Process Priority: Priority of the process will be indicated here. (Low, Medium, High)

Program Counter: It stores the address of the next instruction of the process.

Resource Allocation File: Any Resources are allocated to the process, that info will be maintained in the Resource allocation File.

Scheduling Algorithms:

We have different types of scheduling algorithms:

1. FIFO / First Come First Serve: Processes are executed in order of creation.
2. SJF / Shorted Job First: The process with least execution time will be executed first.
3. Priority Based Scheduling: The process with the highest priority will be executed first.
4. Round Robin Scheduling:

Each process will get equal chance of execution.

According to the time quantum, all the processes will be executed.

Non Preemptive:-

1. First Come First Serve
2. Shortest Job First
3. Priority

Preemptive:-

1. Shortest Job First
2. Priority
3. Round Robin
4. Multi Level Queue
5. Multi Level Feedback Queue

TurnAroundTime = Completion Time - Arrival Time.

Wait Time = Turn Around Time – Burst Time

FCFS / First Come First Serve:-

|  |  |  |
| --- | --- | --- |
| Process | Arrival Time | Burst Time |
| P1 | 0 | 5 |
| P2 | 1 | 3 |
| P3 | 2 | 8 |
| P4 | 3 | 6 |

Calculate Turnaround Time and Completion Time.

Solution using FCFS:-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P | AT | BT | CT | TAT |
| 1 | 0 | 5 | 5 | 5 |
| 2 | 1 | 3 | 8 | 7 |
| 3 | 2 | 8 | 16 | 14 |
| 4 | 3 | 6 | 22 | 19 |

Avg Turn Around Time:-

11.25

Completion Time:-

22

Solution for Shortest Job First:-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P | AT | BT | C | TAT |
| 1 | 0 | 5 | 5 | 5 |
| 2 | 1 | 3 | 8 | 7 |
| 3 | 2 | 8 | 22 | 20 |
| 4 | 3 | 6 | 14 | 11 |

Completion Time:

22

Avg. TAT:

10.75

Round Robin Scheduling:

Time Quantum: 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P | AT | BT | C | TAT |
| 1 | 0 | 5 | 16 | 16 |
| 2 | 1 | 3 | 7 | 6 |
| 3 | 2 | 8 | 20 | 18 |
| 4 | 3 | 6 | 22 | 19 |

1. P1 – 4 – Remaining 1
2. P2 – 3 – Done
3. P3 – 4 – Remaining 4
4. P4 – 4 – Remaining 2
5. P1 – 1 – Done
6. P3 – 4 Done
7. P4 – 2 Done

Preemptive Priority: (Shortest Job First)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Process | Arrival Time | Burst Time | CT | TAT | WT |
| 1 | 0 | 5 | 12 | 12 | 7 |
| 2 | 1 | 3 | 4 | 3 | 0 |
| 3 | 2 | 3 | 8 | 6 | 3 |
| 4 | 4 | 1 | 5 | 1 | 0 |

Calculate CT, TAT and Avg. TAT

CT:- 12

Avg. TAT:- 5.5

Fork() and VFork():-

| **S.No.** | **FORK()** | **VFORK()** |
| --- | --- | --- |
| 1. | In fork() system call, child and parent process have separate memory space. | While in vfork() system call, child and parent process share same address space. |
| 2. | The child process and parent process gets executed simultaneously. | Once child process is executed then parent process starts its execution. |
| 3. | The fork() system call uses copy-on-write as an alternative. | While vfork() system call does not use copy-on-write. |
| 4. | Child process does not suspend parent process execution in fork() system call. | Child process suspends parent process execution in vfork() system call. |
| 5. | Page of one process is not affected by page of other process. | Page of one process is affected by page of other process. |
| 6. | fork() system call is more used. | vfork() system call is less used. |
| 7. | There is wastage of address space. | There is no wastage of address space. |
| 8. | If child process alters page in address space, it is invisible to parent process. | If child process alters page in address space, it is visible to parent process. |

Fork() Function call:-

Fork() 🡪 In child execution will return 0

Fork() 🡪 in parent execution will return positive number (>0)

One fork() 🡪 2 Times output will be printed.

Void main(){

fork()

fork()

printf(“CCC”)

}

// CCC CCC CCC CCC

Fork() Fork()

0 0 -> Child Child

0 1 -> Child Parent

1 0 -> Parent Child

1 1 -> Parent Parent

Shared Data and It’s Problems:-

If one resource is shared by multiple processes, then it is called a shared resource.

Because of the shared resources, sometimes the data will be lost or corrupted.

To overcome the above problem, We have the concepts like Semaphore.

It is divided into types:

1. Binary Semaphore
2. Mutex (Mutual exclusion)
3. Counting Semaphore

Semaphore:- It is a token, It does not contain any info. It will just indicate whether the resource is available or not.

1 means resource available

0 means resource not available

Semaphore tells the processes whether a resource is available or not.

While a process is using a resource, Its semaphore indicates that it is not available.

After the process is done using the resource, The resource is freed and its semaphore indicates that its available.

If any process acquires the resource, the count will be decremented, then another process will go to wait state.

When a process releases the resource, the count will be updated / incremented, then another process acquires the resource.

**🔐 Definition and Purpose of Mutex**

**✅ Definition:**

A **Mutex** (short for **Mutual Exclusion**) is a **synchronization primitive** used in concurrent programming to **prevent multiple threads or processes from accessing a shared resource at the same time**.

**🎯 Purpose:**

The **main purpose** of a mutex is to **ensure data consistency and prevent race conditions** by allowing only **one thread** to access a critical section (shared resource) at a time.

**🔧 How Mutex Works:**

1. **Lock**: A thread **locks** the mutex before entering the critical section.
2. **Execute**: It safely accesses/updates the shared resource.
3. **Unlock**: After finishing, it **unlocks** the mutex so other threads can enter.

Priority Inversion:-

It is a scheduling problem where a lower priority task effectively delays or blocks a higher priority task by holding a resource the higher priority task needs.

This happens because the higher priority task cant execute until the lower priority task releases the resource.

Priority Inheritance is a technique used in operating systems, particularly in real time systems to prevent or mitigate priority inversion. Priority inversion occurs when a lower priority task holds a resource needed by a higher priority task causing the higher priority task to be blocked or delay.

Counting Semaphore:-

Binary semaphore will contain either 0 or 1. Counting semaphore will contain values from +infinity to -ve infinity.

A counting semaphore is a type of synchronization where we are using an integer value of more than one. It enables several processes to utilize a limited number of objects of a shared item. To value of the semaphore can be increased or decreased based on whether a process has released the resource or has acquired it.

Multiple resources – Multiple Processes – Counting Semaphore

Single resource – Multiple Processes – Binary Semaphore

Deadlock occurs when multiple processes require resources which the other has and none of the processes are able to finish the process.

In this situation, none of the processes are willing to let go of their resources.

To solve the deadlock, the processes must come to a solution.