Process Management in OS

A Program does nothing unless its instructions are executed by a CPU. A program in execution is called a process. In order to accomplish its task, process needs the computer resources.

There may exist more than one process in the system which may require the same resource at the same time. Therefore, the operating system has to manage all the processes and the resources in a convenient and efficient way.

Some resources may need to be executed by one process at one time to maintain the consistency otherwise the system can become inconsistent and deadlock may occur.

The operating system is responsible for the following activities in connection with Process Management

1. Scheduling processes and threads on the CPUs.
2. Creating and deleting both user and system processes.
3. Suspending and resuming processes.
4. Providing mechanisms for process synchronization.
5. Providing mechanisms for process communication.

Attributes of a process

The Attributes of the process are used by the Operating System to create the process control block (PCB) for each of them. This is also called context of the process. Attributes which are stored in the PCB are described below.

1. Process ID

When a process is created, a unique id is assigned to the process which is used for unique identification of the process in the system.

2. Program counter

A program counter stores the address of the last instruction of the process on which the process was suspended. The CPU uses this address when the execution of this process is resumed.

3. Process State

The Process, from its creation to the completion, goes through various states which are new, ready, running and waiting. We will discuss about them later in detail.

4. Priority

Every process has its own priority. The process with the highest priority among the processes gets the CPU first. This is also stored on the process control block.

5. General Purpose Registers

Every process has its own set of registers which are used to hold the data which is generated during the execution of the process.

6. List of open files

During the Execution, Every process uses some files which need to be present in the main memory. OS also maintains a list of open files in the PCB.

7. List of open devices

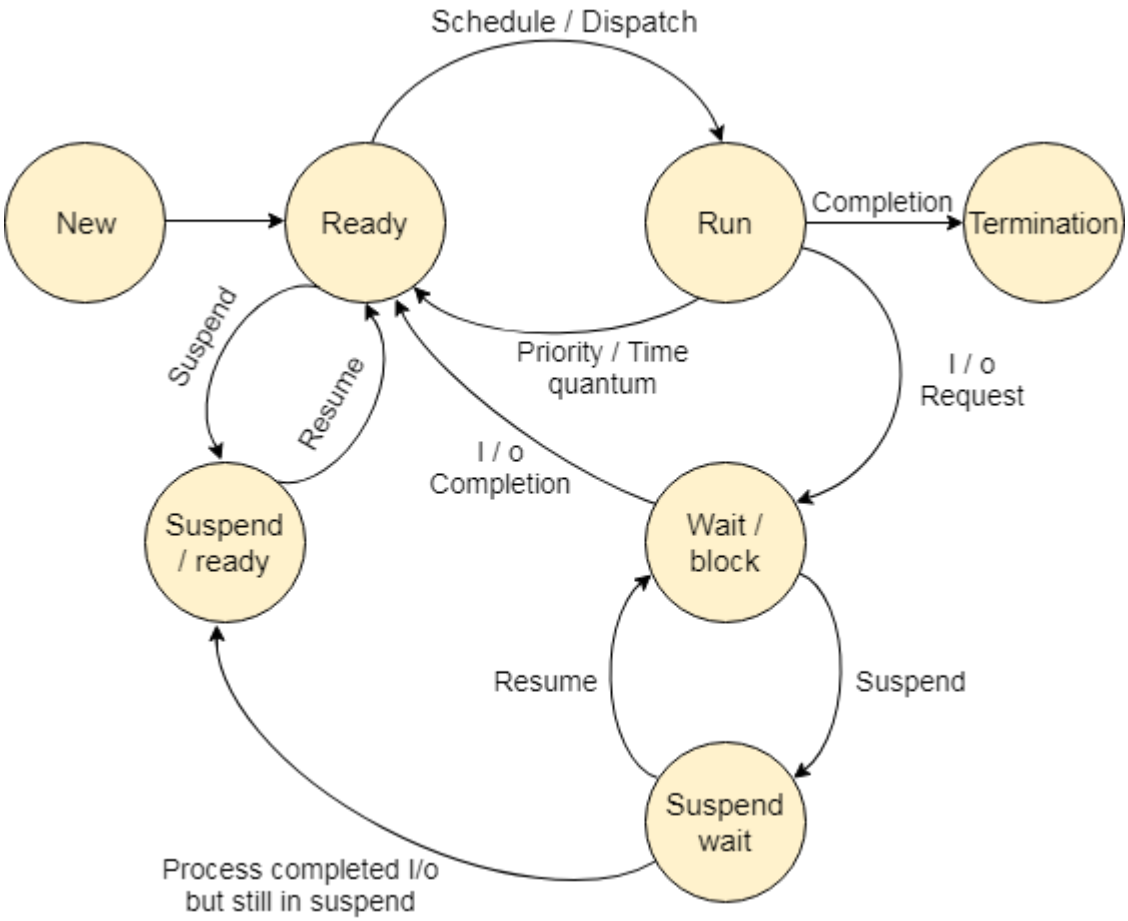
OS also maintain the list of all open devices which are used during the execution of the process.

Process ID
Program Counter
Process State
Priority
General Purpose Registers
List of Open Files
List of Open Devices

Process Attributes

Process States

State Diagram



The process, from its creation to completion, passes through various states. The minimum number of states is five.

The names of the states are not standardized although the process may be in one of the following states during execution.

1. New

A program which is going to be picked up by the OS into the main memory is called a new process. Ready

Whenever a process is created, it directly enters in the ready state, in which, it waits for the CPU to be assigned. The OS picks the new processes from the secondary memory and put all of them in the main memory.

The processes which are ready for the execution and reside in the main memory are called ready state processes. There can be many processes present in the ready state.

3. Running

One of the processes from the ready state will be chosen by the OS depending upon the scheduling algorithm. Hence, if we have only one CPU in our system, the number of running processes for a particular time will always be one. If we have n processors in the system then we can have n processes running simultaneously.

4. Block or wait

From the Running state, a process can make the transition to the block or wait state depending upon the scheduling algorithm or the intrinsic behavior of the process.

When a process waits for a certain resource to be assigned or for the input from the user then the OS move this process to the block or wait state and assigns the CPU to the other processes.

5. Completion or termination

When a process finishes its execution, it comes in the termination state. All the context of the process (Process Control Block) will also be deleted the process will be terminated by the Operating system.

6. Suspend ready

A process in the ready state, which is moved to secondary memory from the main memory due to lack of the resources (mainly primary memory) is called in the suspend ready state.

If the main memory is full and a higher priority process comes for the execution then the OS have to make the room for the process in the main memory by throwing the lower priority process out into the secondary memory. The suspend ready processes remain in the secondary memory until the main memory gets available.

7. Suspend wait

Instead of removing the process from the ready queue, it's better to remove the blocked process which is waiting for some resources in the main memory. Since it is already waiting for some resource to get available hence it is better if it waits in the secondary memory and make room for the higher priority process. These processes complete their execution once the main memory gets available and their wait is finished.

Operations on the Process

1. Creation

Once the process is created, it will be ready and come into the ready queue (main memory) and will be ready for the execution.

2. Scheduling

Out of the many processes present in the ready queue, the Operating system chooses one process and start executing it. Selecting the process which is to be executed next, is known as scheduling.

3. Execution

Once the process is scheduled for the execution, the processor starts executing it. Process may come to the blocked or wait state during the execution then in that case the processor starts executing the other processes.

4. Deletion/killing

Once the purpose of the process gets over then the OS will kill the process. The Context of the process (PCB) will be deleted and the process gets terminated by the Operating system. Process Scheduling in OS (Operating System)

Operating system uses various schedulers for the process scheduling described below.

1. Long term scheduler

Long term scheduler is also known as job scheduler. It chooses the processes from the pool (secondary memory) and keeps them in the ready queue maintained in the primary memory.

Long Term scheduler mainly controls the degree of Multiprogramming. The purpose of long term scheduler is to choose a perfect mix of IO bound and CPU bound processes among the jobs present in the pool.

If the job scheduler chooses more IO bound processes then all of the jobs may reside in the blocked state all the time and the CPU will remain idle most of the time. This will reduce the degree of Multiprogramming. Therefore, the Job of long term scheduler is very critical and may affect the system for a very long time.

2. Short term scheduler

Short term scheduler is also known as CPU scheduler. It selects one of the Jobs from the ready queue and dispatch to the CPU for the execution.

A scheduling algorithm is used to select which job is going to be dispatched for the execution. The Job of the short term scheduler can be very critical in the sense that if it selects job whose CPU burst time is very high then all the jobs after that, will have to wait in the ready queue for a very long time.

This problem is called starvation which may arise if the short term scheduler makes some mistakes while selecting the job.

3. Medium term scheduler

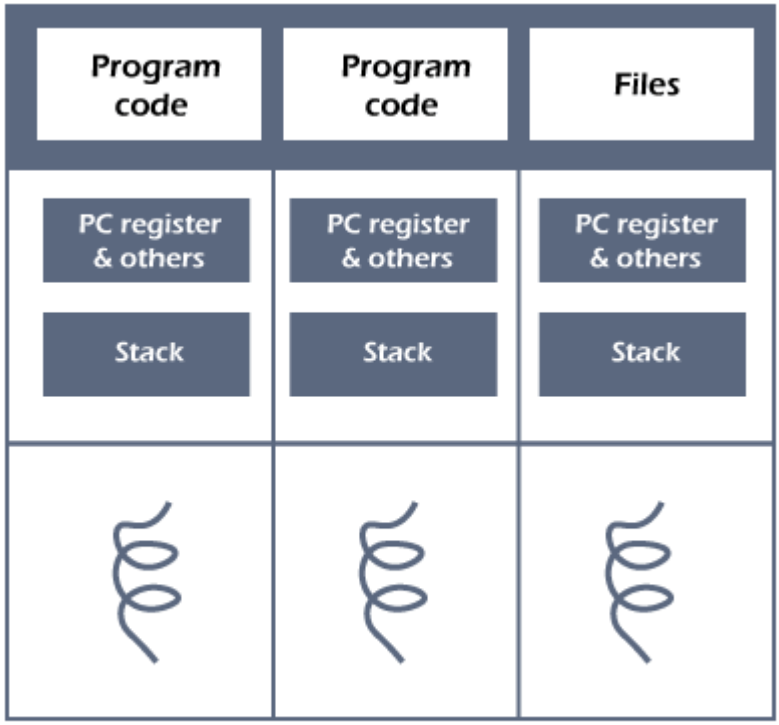
Medium term scheduler takes care of the swapped out processes.If the running state processes needs some IO time for the completion then there is a need to change its state from running to waiting.

Medium term scheduler is used for this purpose. It removes the process from the running state to make room for the other processes. Such processes are the swapped out processes and this procedure is called swapping. The medium term scheduler is responsible for suspending and resuming the processes.

It reduces the degree of multiprogramming. The swapping is necessary to have a perfect mix of processes in the ready queue.

Threads in Operating System (OS)

A thread is a single sequential flow of execution of tasks of a process so it is also known as thread of execution or thread of control. There is a way of thread execution inside the process of any operating system. Apart from this, there can be more than one thread inside a process. Each thread of the same process makes use of a separate program counter and a stack of activation records and control blocks. Thread is often referred to as a lightweight process.



Three threads of same process

The process can be split down into so many threads. **For example**, in a browser, many tabs can be viewed as threads. MS Word uses many threads - formatting text from one thread, processing input from another thread, etc.

Need of Thread:

- It takes far less time to create a new thread in an existing process than to create a new process.
- Threads can share the common data, they do not need to use Inter- Process communication.

- Context switching is faster when working with threads.
- It takes less time to terminate a thread than a process.

Types of Threads

In the operating system, there are two types of threads.

1. Kernel level thread.
2. User-level thread.

User-level thread

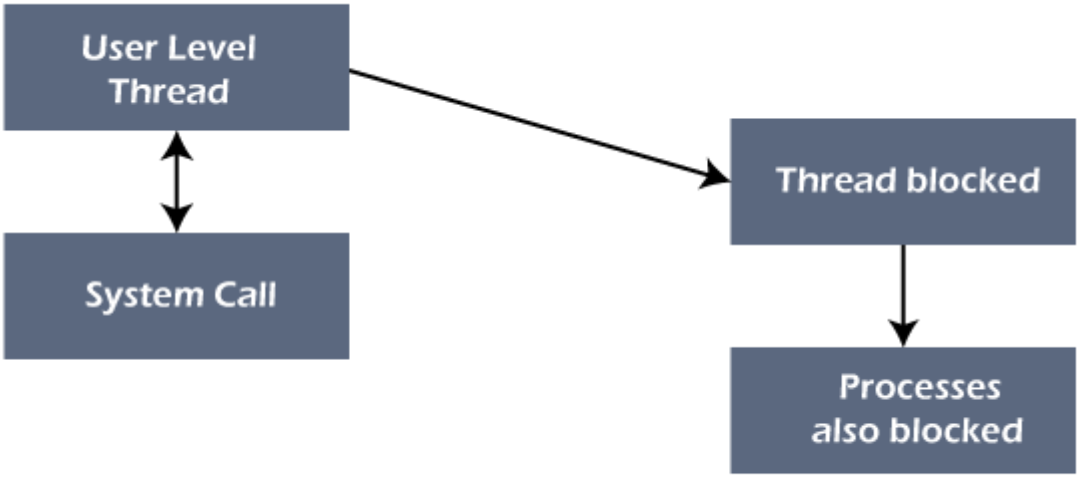
The operating system does not recognize the user-level thread. User threads can be easily implemented and it is implemented by the user. If a user performs a user-level thread blocking operation, the whole process is blocked. The kernel level thread does not know nothing about the user level thread. The kernel-level thread manages user-level threads as if they are single-threaded processes?examples: Java thread, POSIX threads, etc.

Advantages of User-level threads

1. The user threads can be easily implemented than the kernel thread.
2. User-level threads can be applied to such types of operating systems that do not support threads at the kernel-level.
3. It is faster and efficient.
4. Context switch time is shorter than the kernel-level threads.
5. It does not require modifications of the operating system.
6. User-level threads representation is very simple. The register, PC, stack, and mini thread control blocks are stored in the address space of the user-level process.
7. It is simple to create, switch, and synchronize threads without the intervention of the process.

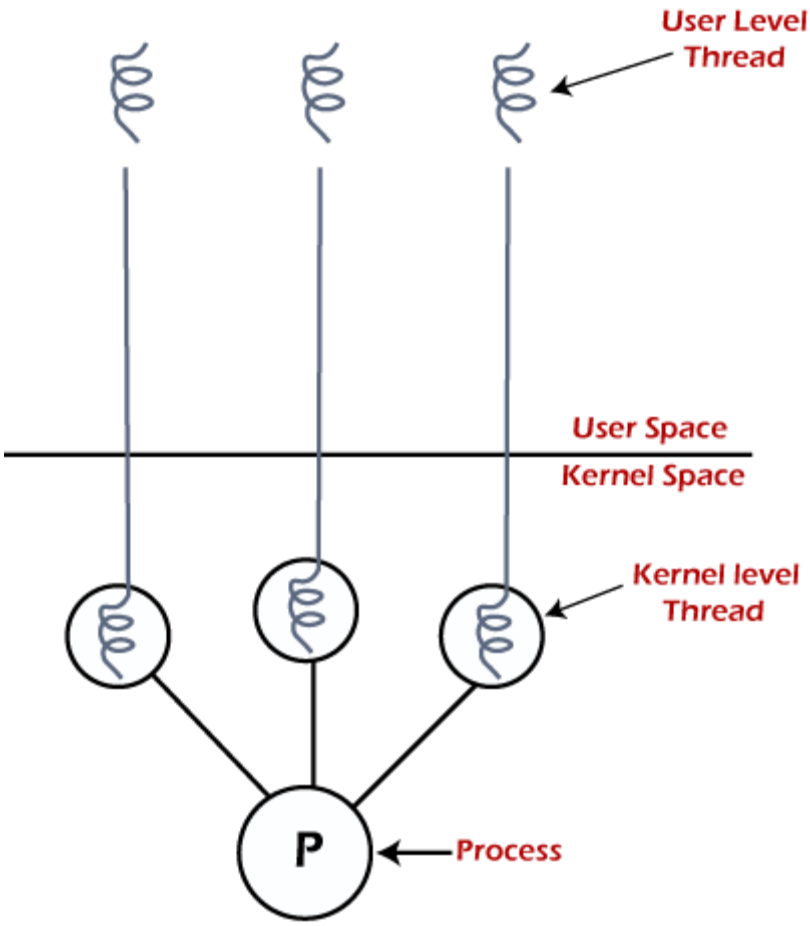
Disadvantages of User-level threads

1. User-level threads lack coordination between the thread and the kernel.
2. If a thread causes a page fault, the entire process is blocked.



Kernel level thread

The kernel thread recognizes the operating system. There is a thread control block and process control block in the system for each thread and process in the kernel-level thread. The kernel-level thread is implemented by the operating system. The kernel knows about all the threads and manages them. The kernel-level thread offers a system call to create and manage the threads from user-space. The implementation of kernel threads is more difficult than the user thread. Context switch time is longer in the kernel thread. If a kernel thread performs a blocking operation, the Banky thread execution can continue. Example: Window Solaris.



Advantages of Kernel-level threads

1. The kernel-level thread is fully aware of all threads.
2. The scheduler may decide to spend more CPU time in the process of threads being large numerical.
3. The kernel-level thread is good for those applications that block the frequency.

Disadvantages of Kernel-level threads

1. The kernel thread manages and schedules all threads.
2. The implementation of kernel threads is difficult than the user thread.
3. The kernel-level thread is slower than user-level threads.

Components of Threads

Any thread has the following components.

1. Program counter
2. Register set
3. Stack space

Benefits of Threads

- **Enhanced throughput of the system:** When the process is split into many threads, and each thread is treated as a job, the number of jobs done in the unit time increases. That is why the throughput of the system also increases.
- **Effective Utilization of Multiprocessor system:** When you have more than one thread in one process, you can schedule more than one thread in more than one processor.
- **Faster context switch:** The context switching period between threads is less than the process context switching. The process context switch means more overhead for the CPU.
- **Responsiveness:** When the process is split into several threads, and when a thread completes its execution, that process can be responded to as soon as possible.
- **Communication:** Multiple-thread communication is simple because the threads share the same address space, while in process, we adopt just a few exclusive communication strategies for communication between two processes.
- **Resource sharing:** Resources can be shared between all threads within a process, such as code, data, and files. Note: The stack and register cannot be shared between threads. There is a stack and register for each thread.

CPU Scheduling Algorithms in Operating Systems

In this tutorial, we will be learning about the CPU Scheduling Algorithms in Operating Systems. These are algorithms are very important topic in Operating Systems. This is because this CPU Scheduling Algorithms forms a base and foundation for the Operating Systems subject.

There are many processes which are going on in the Operating System. A task is a group of processes. Every task is executed by the Operating System. The Operating System divides the task into many processes. The final goal of the Operating System is completion of the task.

But there are some certain conditions which must be followed by the task. The conditions are that the Task must be finished in the quickest possible time with the limited resources which the Operating System have. This is the main motive of CPU Scheduling Algorithms.

CPU Scheduling

The CPU Scheduling is the process by which a process is executed by the using the resources of the CPU. The process also can wait due to the absence or unavailability of the resources. These processes make the complete use of Central Processing Unit.

The operating system must choose one of the processes in the list of ready-to-launch processes whenever the CPU gets idle. A temporary (CPU) scheduler does the selection. The Scheduler choose one of the ready-to-start memory processes to get the CPU.

Before, going to the Types of CPU Scheduling Algorithms, we are going to learn about the Basic Terminologies which are to be followed and used in the CPU Scheduling Algorithms by us.

1. Process ID

The Process ID is the first Thing is to be written while solving the problem. The Process ID acts like the name of the process. It is usually represented with numbers or P letter with numbers

Example:

1. 0, 1, 2, 3, ...
2. P0, P1, P2, P3 . . . . . .

Usually, we start the naming of the process from zero. We can also start the numbering from number 1 also. It is our interest

1. Arrival Time

The time which is required for the Process to enter the ready queue or the time when the Process is ready to be executed by the CPU. This Arrival Time can be represented as AT in short form. The Arrival Times is always positive or also zero.

2. Burst Time

The Time Slot which the Process requires to complete the Process is known as the Burst Time. The Burst Time can be represented as BT in short form. The Burst Times of a process is always greater than zero.

3. Completion Time

The Total Time required by the CPU to complete the process is known as Completion Time. The Completion Time can be represented as CT in short form. The Completion will always be greater than zero.

4. Turn Around Time

The time taken by the CPU since the Process has been ready to execute or since the process is in Ready Queue is known as Turn Around Time. The Turn Around Time can be calculated with the help of Completion Time and Arrival Time. The Turn Around Time can be represented as TAT in short form.

The Turn Around Time is the difference of Completion Time and Arrival Time.

Formula

1. TAT = CT - AT

Here, CT is Completion Time and AT is Arrival Time.

5. Waiting Time

The time the Process has been waiting to complete its process since the assignment of process for completion is known as Waiting Time. The Waiting Time can be represented as WT in short form. The Waiting Time can be calculated with the help of Turn Around Time and Burst Time.

The Waiting Time is the difference between Turn Around Time and Burst Time

Formula

1. WT = TAT - BT

6. Ready Queue

The Queue where all the processes are stored until the execution of the previous process. This ready queue is very important because there would be confusion in CPU when two same kinds of processes are being executed at the same time.

Then, in these kinds of conditions the ready queue comes into place and then, the its duty is fulfilled.

7. Gantt Chart

It is the place where all the already executed processes are stored. This is very useful for calculating Waiting Time, Completion Time, Turn Around Time.

Modes in CPU Scheduling Algorithms

There are two modes in CPU Scheduling Algorithms. They are:

1. Pre-emptive Approach
2. Non Pre-emptive Approach

In Pre Emptive-Approach the process once starts its execution then the CPU is allotted to the same process until the completion of process. There would be no shift of Processes by the Central Processing Unit. The complete CPU is allocated to the Process and there would be no change of CPU allocation until the process is complete.

In Non-Pre Emptive-Approach the process once stars its execution, then the CPU is not allotted to the same process until the completion of process. There is a shift of Processes by the Central Processing Unit. The complete CPU is allocated to the Process when only certain required conditions are achieved and there will be change of CPU allocation if there is break or False occurrence in the required conditions.

Types of CPU Scheduling Algorithms

- First Come First Serve
- Shortest Job First
- Priority Scheduling
- Round Robin Scheduling

First Come First Serve Scheduling Algorithm

This is the first type of CPU Scheduling Algorithms. Here, in this CPU Scheduling Algorithm we are going to learn how CPU is going to allot resources to the certain process.

Here, in the First Come First Serve CPU Scheduling Algorithm, the CPU allots the resources to the process in a certain order. The order is serial way. The CPU is allotted to the process in which it has occurred.

We can also say that First Come First Serve CPU Scheduling Algorithm follows First In First Out in Ready Queue.

First Come First Serve can be called as FCFS in short form.

Characteristics of FCFS (First Come First Serve):

- First Come First Serve can follow or can be executed in Pre emptive Approach or Non-Pre emptive Approach
- The Process which enters the Ready Queue is executed First. So, we say that FCFS follows First in First Out Approach.
- First Come First Come First Serve is only executed when the Arrival Time (AT) is greater than or equal to the Time which is at present.

Advantages

- Very easy to perform by the CPU
- Follows FIFO Queue Approach

Disadvantages

- First Come First Serve is not very efficient.
- First Come First Serve suffers because of Convoy Effect.

Examples

1. Process ID Process Name Arrival Time Burst Time
2. _
3. P 1 A 0 6
4. P 2 B 2 2
5. P 3 C 3 1
6. P 4 D 4 9
7. P 5 E 5 8

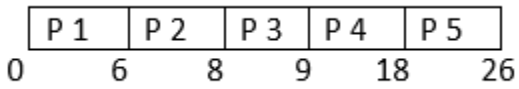
Now, we are going to apply FCFS (First Come First Serve) CPU Scheduling Algorithm for the above problem.

In FCFS, there is an exception that Arrival Times are not removed from the Completion Time.

Here, in First Come First Serve the basic formulas do not work. Only the basic formulas work.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	W
P 1	0	6	6	6	0
P 2	2	2	8	8	6
P 3	3	1	9	9	8
P4	4	9	18	18	9
P 5	5	8	26	26	18

Gantt Chart



Average Completion Time = The Total Sum of Completion Times which is divided by the total number of processes is known as Average Completion Time.

Average Completion Time =(CT1 + CT2 + CT3 + ... + CTn) / n

Average Completion Time

1. Average CT = (6 + 8 +9 + 18 + 26) / 5
2. Average CT = 67 / 5
3. Average CT = 13. 4

Average Turn Around Time = The Total Sum of Turn Around Times which is divided by the total number of processes is known as Average Turn Around Time.

Average Turn Around Time = (TAT1 + TAT2 + TAT3 + ... + TATn) / n

Average Turn Around Time

1. Average TAT = (6 + 8 + 9 +18 +26) / 5
2. Average TAT = 13.4

Average Waiting Time = The Total Sum of Waiting Times which is divided by the total number of processes is known as Average Waiting Time.

Average Waiting Time = (WT1 + WT2 + WT3 + ... + WTn) / n

Average Waiting Time

1. Average WT = (0 + 6 + 8 + 9 + 18) / 5
2. Average WT = 41 / 5
3. Average WT = 8.2

Convoy effect

Examples:

```
1. S. No    ProcessID      Process Name       Arrival Time     Burst Time
2. _ _ _   _ _ _ _ _ _    _ _ _ _ _ _ _     _ _ _ _ _ _ _   _ _ _ _ _ _ _
3. 1       P 1            A                 1               79
4. 2       P 2            B                 0               2
5. 3       P 3            C                 1               3
6. 4       P 4            D                 0               1
7. 5       P 5            E                 2               25
```

S. No	Process Id	Process Name	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around	Waiting Time (WT)

						Time (TAT)	
1	P 1	A	1	79	79	78	0
2	P 2	B	0	2	81	81	79
3	P 3	C	1	3	84	83	81
4	P 4	D	0	1	85	85	84
5	P 5	E	2	25	110	108	85
6	P 6	F	3	3	113	110	110

The Average Completion Time is:

Average CT = (79 + 81 + 84 + 85 + 110 + 113) / 6

Average CT = 92

The Average Waiting Time is:

Average WT = (0 + 79 + 81 + 84 + 85 + 110) /6

Average WT = 73.1666667

The Average Turn Around Time is:

Average TAT = (78 + 81 + 83 + 85 + 108 +110) / 6

Average TAT = 90.833334

The biggest problem in this is higher Completion Time, higher Waiting Time, higher Turn Around Time, lower efficiency.

The problem which we found in the above solution can be resolved by using a new CPU Scheduling Techniques named Shortest Job First (SJF).

Shortest Job First CPU Scheduling Algorithm

This is another type of CPU Scheduling Algorithms. Here, in this CPU Scheduling Algorithm we are going to learn how CPU is going to allot resources to the certain process.

The Shortest Job is heavily dependent on the Burst Times. Every CPU Scheduling Algorithm is basically dependent on the Arrival Times. Here, in this Shortest Job First CPU Scheduling Algorithm, the CPU allots its resources to the process which is in ready queue and the process which has least Burst Time.

If we face a condition where two processes are present in the Ready Queue and their Burst Times are same, we can choose any process for execution. In actual Operating Systems, if we face the same problem then sequential allocation of resources takes place.

Shortest Job First can be called as SJF in short form.

Characteristics:

- SJF (Shortest Job First) has the least average waiting time. This is because all the heavy processes are executed at the last. So, because of this reason all the very small, small processes are executed first and prevent starvation of small processes.
- It is used as a measure of time to do each activity.
- If shorter processes continue to be produced, hunger might result. The idea of aging can be used to overcome this issue.
- Shortest Job can be executed in Pre emptive and also non pre emptive way also.

Advantages

- SJF is used because it has the least average waiting time than the other CPU Scheduling Algorithms
- SJF can be termed or can be called as long term CPU scheduling algorithm.

Disadvantages

- Starvation is one of the negative traits Shortest Job First CPU Scheduling Algorithm exhibits.
- Often, it becomes difficult to forecast how long the next CPU request will take

Examples for Shortest Job First

1. Process ID Arrival Time Burst Time
2. _
3. P0 1 3
4. P1 2 6
5. P2 1 2
6. P3 3 7
7. P4 2 4
8. P5 5 5

Now, we are going to solve this problem in both pre emptive and non pre emptive way

We will first solve the problem in non pre emptive way.

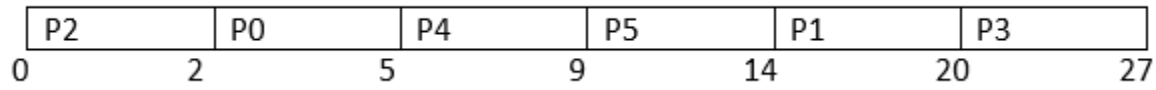
In Non pre emptive way, the process is executed until it is completed.

Non Pre Emptive Shortest Job First CPU Scheduling

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time TAT = CT - AT	Waiting = CT - B

P0	1	3	5	4	1
P1	2	6	20	18	12
P2	0	2	2	2	0
P3	3	7	27	24	17
P4	2	4	9	7	4
P5	5	5	14	10	5

Gantt Chart:



Now let us find out Average Completion Time, Average Turn Around Time, Average Waiting Time.

Average Completion Time:

1. Average Completion Time = (5 + 20 + 2 + 27 + 9 + 14) / 6
2. Average Completion Time = 77/6
3. Average Completion Time = 12.833

Average Waiting Time:

1. Average Waiting Time = (1 + 12 + 17 + 0 + 5 + 4) / 6
2. Average Waiting Time = 37 / 6
3. Average Waiting Time = 6.666

Average Turn Around Time:

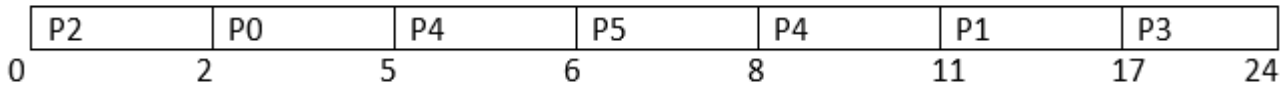
1. Average Turn Around Time = (4 +18 + 2 +24 + 7 + 10) / 6
2. Average Turn Around Time = 65 / 6
3. Average Turn Around Time = 6.166

Pre Emptive Approach

In Pre emptive way, the process is executed when the best possible solution is found.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time TAT = CT - AT	Waiting = CT - B
P0	1	3	5	4	1
P1	2	6	17	15	9
P2	0	2	2	2	0
P3	3	7	24	21	14
P4	2	4	11	9	5
P5	6	2	8	2	0

Gantt chart:



After P4 P5 is executed just because of the Pre Emptive Condition.

Now let us find out Average Completion Time, Average Turn Around Time, Average Waiting Time.

Average Completion Time

1. Average Completion Time = (5 + 17 + 2 + 24 + 11 +8) / 6
2. Average Completion Time = 67 / 6
3. Average Completion = 11.166

Average Turn Around Time

1. Average Turn Around Time = (4 +15 + 2 + 21 + 9 + 2) / 6
2. Average Turn Around Time = 53 / 6
3. Average Turn Around Time = 8.833

Average Waiting Time

1. Average Waiting Time = (1 + 9 + 0 + 14 + 5 + 0) /6
2. Average Waiting Time = 29 / 6
3. Average Waiting Time = 4.833

Priority CPU Scheduling

This is another type of CPU Scheduling Algorithms. Here, in this CPU Scheduling Algorithm we are going to learn how CPU is going to allot resources to the certain process.

The Priority CPU Scheduling is different from the remaining CPU Scheduling Algorithms. Here, each and every process has a certain Priority number.

There are two types of Priority Values.

- Highest Number is considered as Highest Priority Value
- Lowest Number is considered as Lowest Priority Value

Priority for Prevention The priority of a process determines how the CPU Scheduling Algorithm operates, which is a preemptive strategy. Since the editor has assigned equal importance to each function in this method, the most crucial steps must come first. The most significant CPU planning algorithm relies on the FCFS (First Come First Serve) approach when there is a conflict, that is, when there are many processors with equal value.

Characteristics

- Priority CPU scheduling organizes tasks according to importance.
- When a task with a lower priority is being performed while a task with a higher priority arrives, the task with the lower priority is replaced by the task with the higher priority, and the latter is stopped until the execution is finished.
- A process's priority level rises as the allocated number decreases.

Advantages

- The typical or average waiting time for Priority CPU Scheduling is shorter than First Come First Serve (FCFS).
- It is easier to handle Priority CPU scheduling
- It is less complex

Disadvantages

- The Starvation Problem is one of the Pre emptive Priority CPU Scheduling Algorithm's most prevalent flaws. Because of this issue, a process must wait a longer period of time to be scheduled into the CPU. The hunger problem or the starvation problem is the name given to this issue.

Examples:

Now, let us explain this problem with the help of an example of Priority Scheduling

1. S. No Process ID Arrival Time Burst Time Priority

2. ---   ------   -------   -------   -------
3. 1   P 1   0   5   5
4. 2   P 2   1   6   4
5. 3   P 3   2   2   0
6. 4   P 4   3   1   2
7. 5   P 5   4   7   1
8. 6   P 6   4   6   3

Here, in this problem the priority number with highest number is least prioritized.

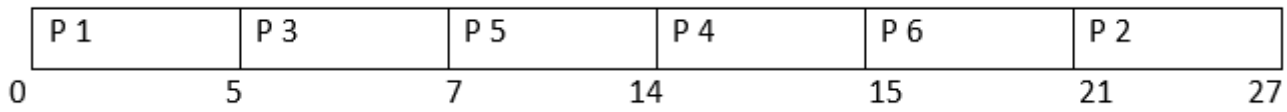
This means 5 has the least priority and 0 has the highest priority.

S. No	Process Id	Arrival Time	Burst Time		Priority	Completion Time Turn Around Time TAT = CT - AT	Waiting Time WT = TAT - BT
1	PP 1	P0	P5	P5	P5	P5	P0
2	PP 2	P1	P6	P4	P27	P26	P20
3	PP 3	P2	P2	P0	P7	P5	P3
4	PP 4	P3	P1	P2	P15	P12	P11
5	PP 5	P4	P7	P1	P14	P10	P3

6	P 6	P4	P6	P3	P21	P17	P11

Solution:

Gantt Chart:



Average Completion Time

1. Average Completion Time = (5 +27 +7 +15 +14 + 21) / 6
2. Average Completion Time = 89 / 6
3. Average Completion Time = 14.8333

Average Waiting Time

1. Average Waiting Time = (0 + 20 + 3 + 11 + 3 + 11) / 6
2. Average Waiting Time = 48 / 6
3. Average Waiting Time = 7

Average Turn Around Time

1. Average Turn Around Time = (5 + 26 + 5 + 11 + 10 + 17) / 6
2. Average Turn Around Time = 74 / 6
3. Average Turn Around Time = 12.333

Round Robin CPU Scheduling

Round Robin is a CPU scheduling mechanism those cycles around assigning each task a specific time slot. It is the First come, First served CPU Scheduling technique with preemptive mode. The Round Robin CPU algorithm frequently emphasizes the Time-Sharing method.

Round Robin CPU Scheduling Algorithm characteristics include:

- Because all processes receive a balanced CPU allocation, it is straightforward, simple to use, and starvation-free.
- One of the most used techniques for CPU core scheduling. Because the processes are only allowed access to the CPU for a brief period of time, it is seen as preemptive.

The benefits of round robin CPU Scheduling:

- Every process receives an equal amount of CPU time, therefore round robin appears to be equitable.
- To the end of the ready queue is added the newly formed process.

Examples:

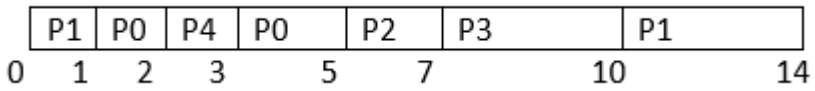
Problem

Process ID	Arrival Time	Burst Time
P0	1	3
P1	0	5
P2	3	2
P3	4	3
P4	2	1

Solution:

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	W
P0	1	3	5	4	1
P1	0	5	14	14	9
P2	3	2	7	4	2
P3	4	3	10	6	3
P4	2	1	3	1	0

Gantt Chart:



Average Completion Time = 7.8

Average Turn Around Time = 4.8

Average Waiting Time = 3

This is all about the CPU Scheduling Algorithms in Operating System.

Concurrency in Operating System

In this article, you will learn the concurrency in the operating system with its principles, issues, advantages and disadvantages.

What is Concurrency?

It refers to the execution of multiple instruction sequences at the same time. It occurs in an operating system when multiple process threads are executing concurrently. These threads can interact with one another via shared memory or message passing. Concurrency results in resource sharing, which causes issues like deadlocks and resource scarcity. It aids with techniques such as process coordination, memory allocation, and execution schedule to maximize throughput.

Principles of Concurrency

Today's technology, like multi-core processors and parallel processing, allows multiple processes and threads to be executed simultaneously. Multiple processes and threads can access the same memory space, the same declared variable in code, or even read or write to the same file.

The amount of time it takes a process to execute cannot be simply estimated, and you cannot predict which process will complete first, enabling you to build techniques to deal with the problems that concurrency creates.

Interleaved and overlapping processes are two types of concurrent processes with the same problems. It is impossible to predict the relative speed of execution, and the following factors determine it:

1. The way operating system handles interrupts
2. Other processes' activities
3. The operating system's scheduling policies

Problems in Concurrency

There are various problems in concurrency. Some of them are as follows:

1. Locating the programming errors

It's difficult to spot a programming error because reports are usually repeatable due to the varying states of shared components each time the code is executed.

2. Sharing Global Resources

Sharing global resources is difficult. If two processes utilize a global variable and both alter the variable's value, the order in which the many changes are executed is critical.

3. Locking the channel

It could be inefficient for the OS to lock the resource and prevent other processes from using it.

4. Optimal Allocation of Resources

It is challenging for the OS to handle resource allocation properly.

Issues of Concurrency

Various issues of concurrency are as follows:

1. Non-atomic

Operations that are non-atomic but interruptible by several processes may happen issues. A non-atomic operation depends on other processes, and an atomic operation runs independently of other processes.

2. Deadlock

In concurrent computing, it occurs when one group member waits for another member, including itself, to send a message and release a lock. Software and hardware locks are commonly used to arbitrate shared resources and implement process synchronization in parallel computing, distributed systems, and multiprocessing.

3. Blocking

A blocked process is waiting for some event, like the availability of a resource or completing an I/O operation. Processes may block waiting for resources, and a process may be blocked for a long time waiting for terminal input. If the process is needed to update some data periodically, it will be very undesirable.

4. Race Conditions

ADVERTISEMENT

A race problem occurs when the output of a software application is determined by the timing or sequencing of other uncontrollable events. Race situations can also happen in multithreaded software, runs in a distributed environment, or is interdependent on shared resources.

5. Starvation

A problem in concurrent computing is where a process is continuously denied the resources it needs to complete its work. It could be caused by errors in scheduling or mutual exclusion algorithm, but resource leaks may also cause it.

Concurrent system design frequently requires developing dependable strategies for coordinating their execution, data interchange, memory allocation, and execution schedule to decrease response time and maximize throughput.

Advantages and Disadvantages of Concurrency in Operating System

Various advantages and disadvantages of Concurrency in Operating systems are as follows:

Advantages

1. Better Performance

It improves the operating system's performance. When one application only utilizes the processor, and another only uses the disk drive, the time it takes to perform both apps simultaneously is less than the time it takes to run them sequentially.

2. Better Resource Utilization

It enables resources that are not being used by one application to be used by another.

3. Running Multiple Applications

It enables you to execute multiple applications simultaneously.

Disadvantages

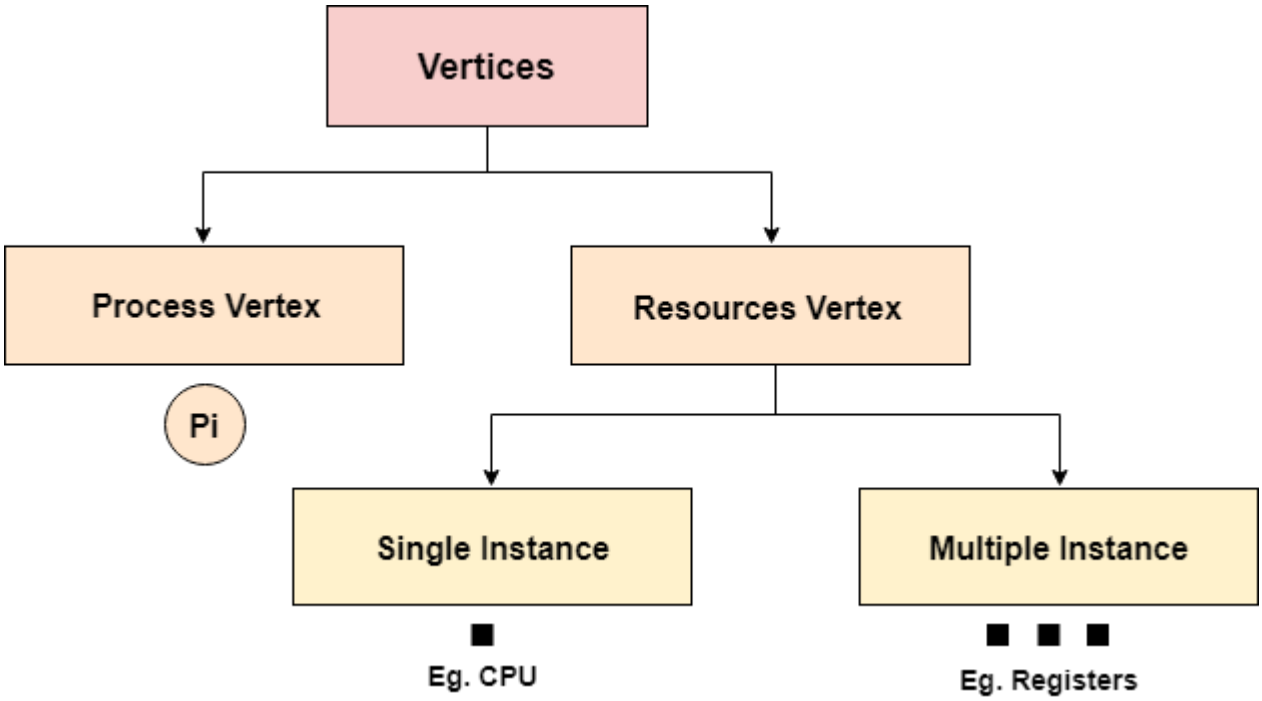
1. It is necessary to protect multiple applications from each other.
2. It is necessary to use extra techniques to coordinate several applications.
3. Additional performance overheads and complexities in OS are needed for switching between applications.

Resource Allocation Graph

The resource allocation graph is the pictorial representation of the state of a system. As its name suggests, the resource allocation graph is the complete information about all the processes which are holding some resources or waiting for some resources.

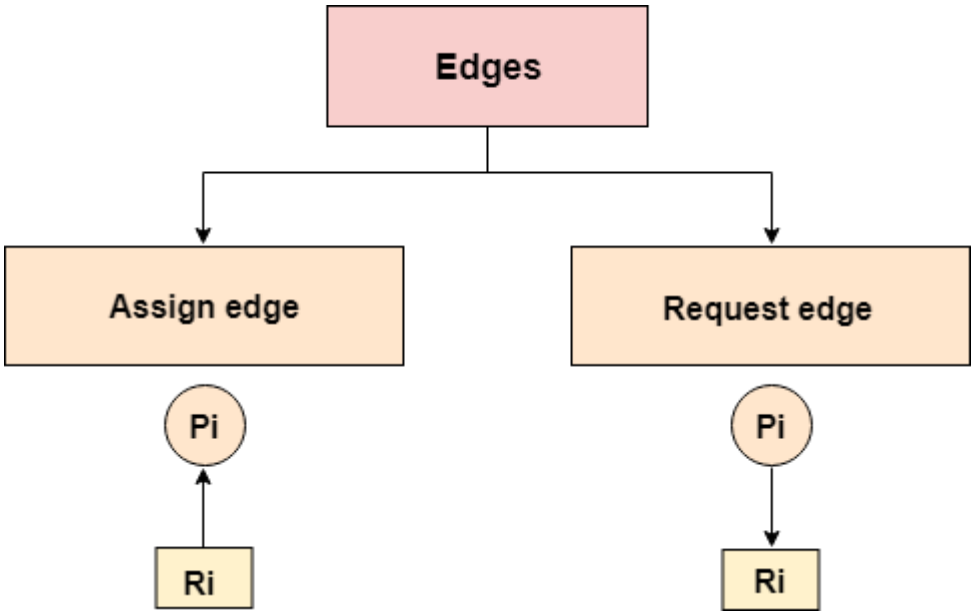
It also contains the information about all the instances of all the resources whether they are available or being used by the processes.

In Resource allocation graph, the process is represented by a Circle while the Resource is represented by a rectangle. Let's see the types of vertices and edges in detail.



Vertices are mainly of two types, Resource and process. Each of them will be represented by a different shape. Circle represents process while rectangle represents resource.

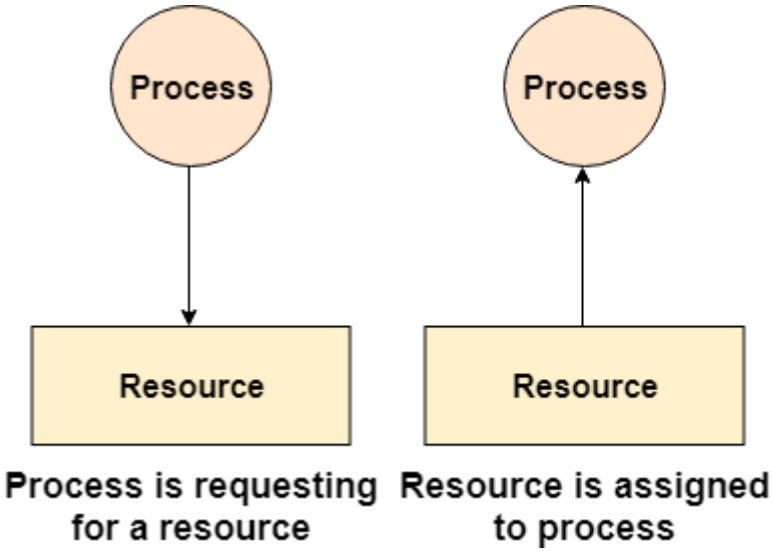
A resource can have more than one instance. Each instance will be represented by a dot inside the rectangle.



Edges in RAG are also of two types, one represents assignment and other represents the wait of a process for a resource. The above image shows each of them.

A resource is shown as assigned to a process if the tail of the arrow is attached to an instance to the resource and the head is attached to a process.

A process is shown as waiting for a resource if the tail of an arrow is attached to the process while the head is pointing towards the resource.

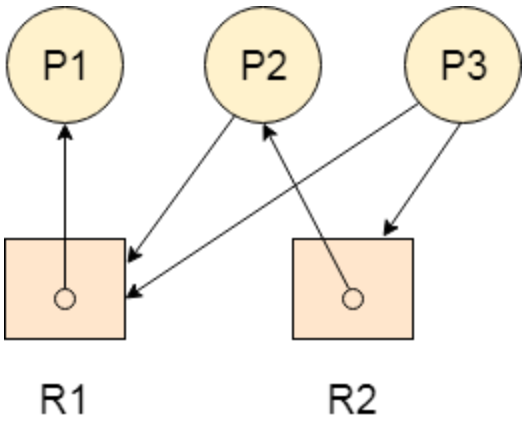


Example

Let'sconsider 3 processes P1, P2 and P3, and two types of resources R1 and R2. The resources are having 1 instance each.

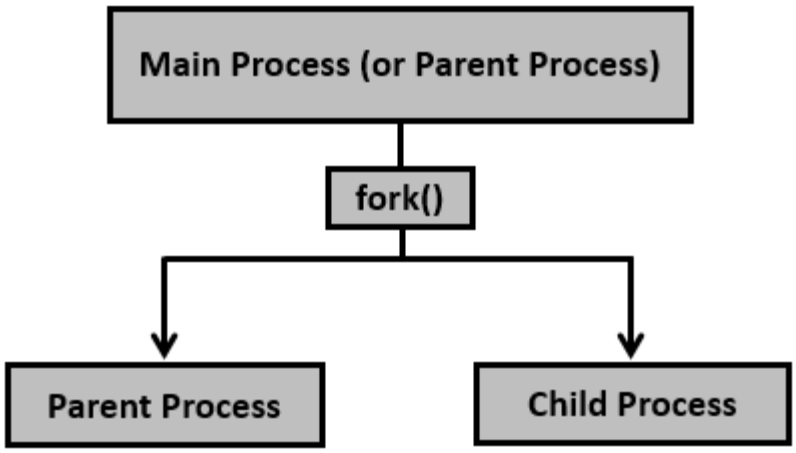
According to the graph, R1 is being used by P1, P2 is holding R2 and waiting for R1, P3 is waiting for R1 as well as R2.

The graph is deadlock free since no cycle is being formed in the graph.



Process creation is achieved through the **fork() system call.** The newly created process is called the child process and the process that initiated

it (or the process when execution is started) is called the parent process. After the fork() system call, now we have two processes - parent and child processes. How to differentiate them? Very simple, it is through their return values.



After creation of the child process, let us see the fork() system call details.

```
#include <sys/types.h>
#include <unistd.h>

pid_t fork(void);
```

Creates the child process. After this call, there are two processes, the existing one is called the parent process and the newly created one is called the child process.

The fork() system call returns either of the three values –

- Negative value to indicate an error, i.e., unsuccessful in creating the child process.
- Returns a zero for child process.
- Returns a positive value for the parent process. This value is the process ID of the newly created child process.

Let us consider a simple program.

```
File name: basicfork.c
```

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
   fork();
   printf("Called fork() system call\n");
   return 0;
}
```

Execution Steps

Compilation

gcc basicfork.c -o basicfork

Execution/Output

Called fork() system call

Called fork() system call

Note – Usually after fork() call, the child process and the parent process would perform different tasks. If the same task needs to be run, then for each fork() call it would run 2 power n times, where **n** is the number of times fork() is invoked.

In the above case, fork() is called once, hence the output is printed twice (2 power 1). If fork() is called, say 3 times, then the output would be printed 8 times (2 power 3). If it is called 5 times, then it prints 32 times and so on and so forth.

Having seen fork() create the child process, it is time to see the details of the parent and the child processes.

File name: pids_after_fork.c

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
   pid_t pid, mypid, myppid;
```

```
   pid = getpid();
   printf("Before fork: Process id is %d\n", pid);
   pid = fork();

   if (pid < 0) {
      perror("fork() failure\n");
      return 1;
   }

   // Child process
   if (pid == 0) {
      printf("This is child process\n");
      mypid = getpid();
      myppid = getppid();
      printf("Process id is %d and PPID is %d\n", mypid,
myppid);
   } else { // Parent process
      sleep(2);
      printf("This is parent process\n");
      mypid = getpid();
      myppid = getppid();
      printf("Process id is %d and PPID is %d\n", mypid,
myppid);
      printf("Newly created process id or child pid is %d\n",
pid);
   }
   return 0;
}
```

Compilation and Execution Steps

Before fork: Process id is 166629

This is child process

Process id is 166630 and PPID is 166629

Before fork: Process id is 166629

This is parent process

Process id is 166629 and PPID is 166628

Newly created process id or child pid is 166630

A process can terminate in either of the two ways –

- Abnormally, occurs on delivery of certain signals, say terminate signal.
- Normally, using _exit() system call (or _Exit() system call) or exit() library function.

The difference between _exit() and exit() is mainly the cleanup activity. The **exit()** does some cleanup before returning the control back to the kernel, while the **_exit()** (or _Exit()) would return the control back to the kernel immediately.

Consider the following example program with exit().

File name: atexit_sample.c

```
#include <stdio.h>
#include <stdlib.h>

void exitfunc() {
   printf("Called cleanup function - exitfunc()\n");
   return;
}

int main() {
   atexit(exitfunc);
   printf("Hello, World!\n");
   exit (0);
}
```

Compilation and Execution Steps

Hello, World!

Called cleanup function - exitfunc()

Consider the following example program with _exit().

File name: at_exit_sample.c

```
#include <stdio.h>
#include <unistd.h>

void exitfunc() {
   printf("Called cleanup function - exitfunc()\n");
   return;
```

```
}

int main() {
   atexit(exitfunc);
   printf("Hello, World!\n");
   _exit (0);
}
```

Drawbacks of Concurrency

- It is required to protect multiple applications from one another.
- It is required to coordinate multiple applications through additional mechanisms.
- Additional performance overheads and complexities in operating systems are required for switching among applications.
- Sometimes running too many applications concurrently leads to severely degraded performance.

Issues of Concurrency

- **Non-atomic:** Operations that are non-atomic but interruptible by multiple processes can cause problems.
- **Race conditions:** A race condition occurs of the outcome depends on which of several processes gets to a point first.
- **Blocking:** Processes can block waiting for resources. A process could be blocked for long period of time waiting for input from a terminal. If the process is required to periodically update some data, this would be very undesirable.
- **Starvation:** It occurs when a process does not obtain service to progress.
- **Deadlock:** It occurs when two processes are blocked and hence neither can proceed to execute.