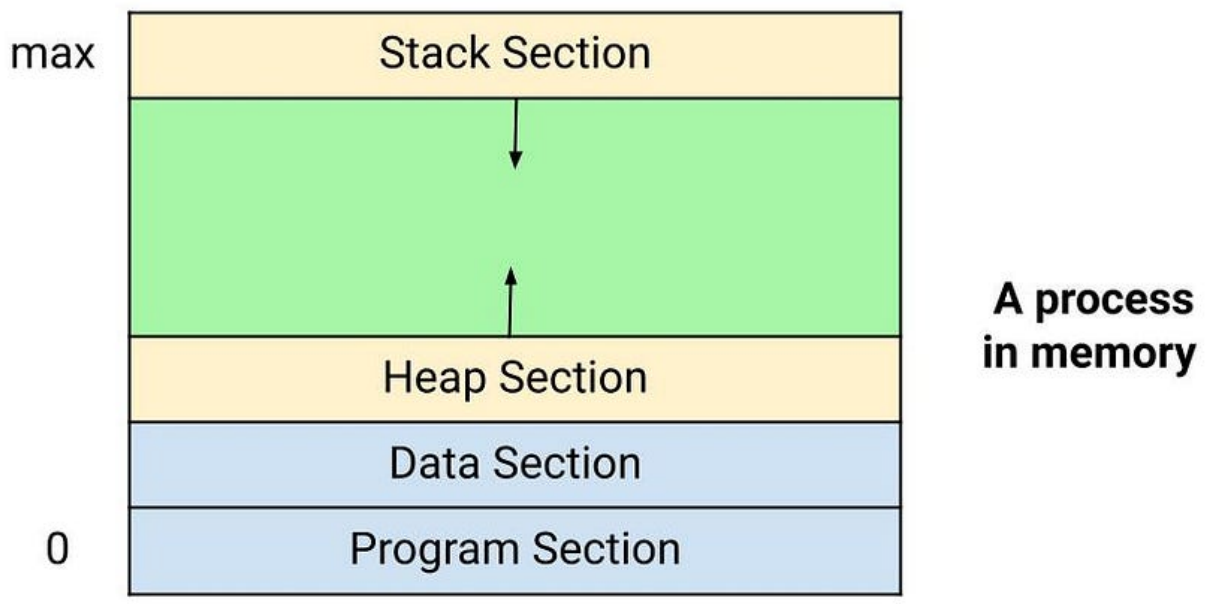
Process Management in Operating System (OS)

In this blog, we will learn about process management in Operating System and various related algorithms. Before looking into process management and how it works, let's start with the definition and various aspects of a process.

In simple words, a program in execution is called a **process**. It is an instance of a program that actually runs, i.e., an entity that can be assigned and executed on a processor. There are two essential elements of a process: program code and a set of data associated with that code.

Process memory is divided into four sections:

- **Program memory** stores the compiled program code.
- **Data section** stores global and static variables.
- **Heap section** is used to manage dynamic memory allocation inside our program. In other words, it is the portion of memory where dynamically allocated memory resides i.e., memory allocation via new or malloc and memory deallocation via delete or free, etc.
- **Stack section** stores local variables defined inside our program or function. Stack space is created for local variables when declared, and space is freed up when they go out of scope.



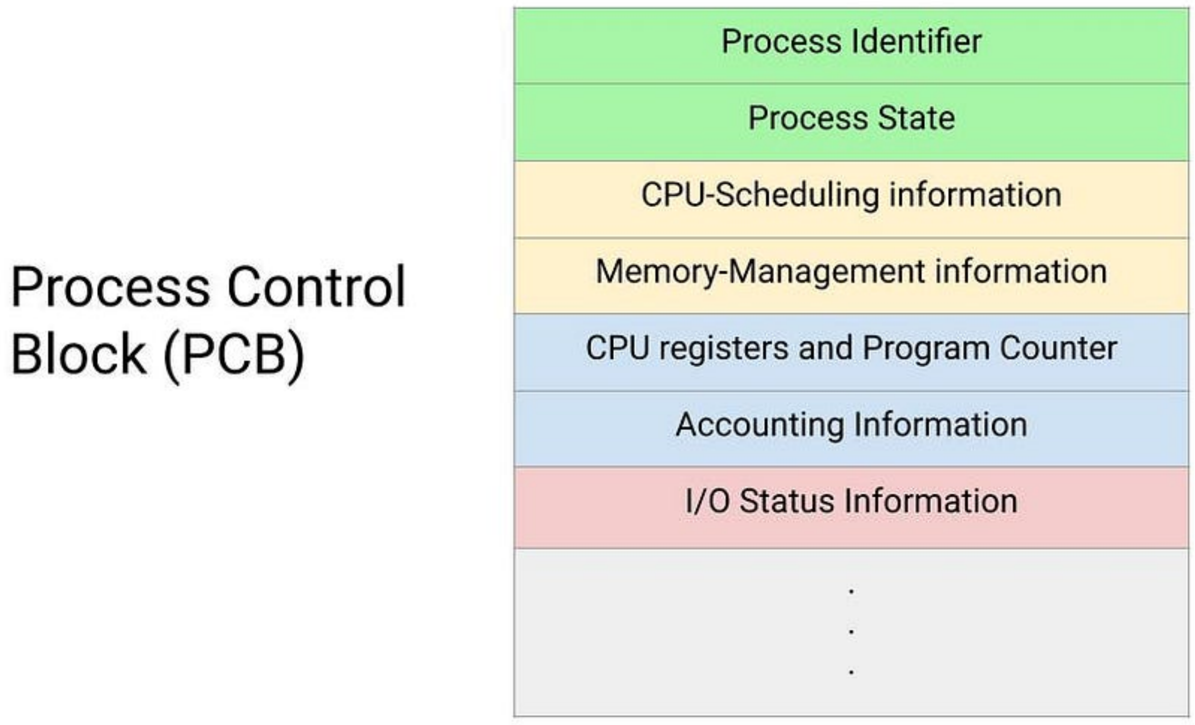
Note: Stack and heap sections start at opposite ends of the process free space and grow towards each other. When they meet, stack overflow error will occur or a call to new memory allocation will fail due to insufficient memory available in the heap section.

Process Control Block (PCB)

A program executing as a process is uniquely determined by various parameters. These parameters are stored in a Process Control Block (PCB). It is a data structure which holds the following information:

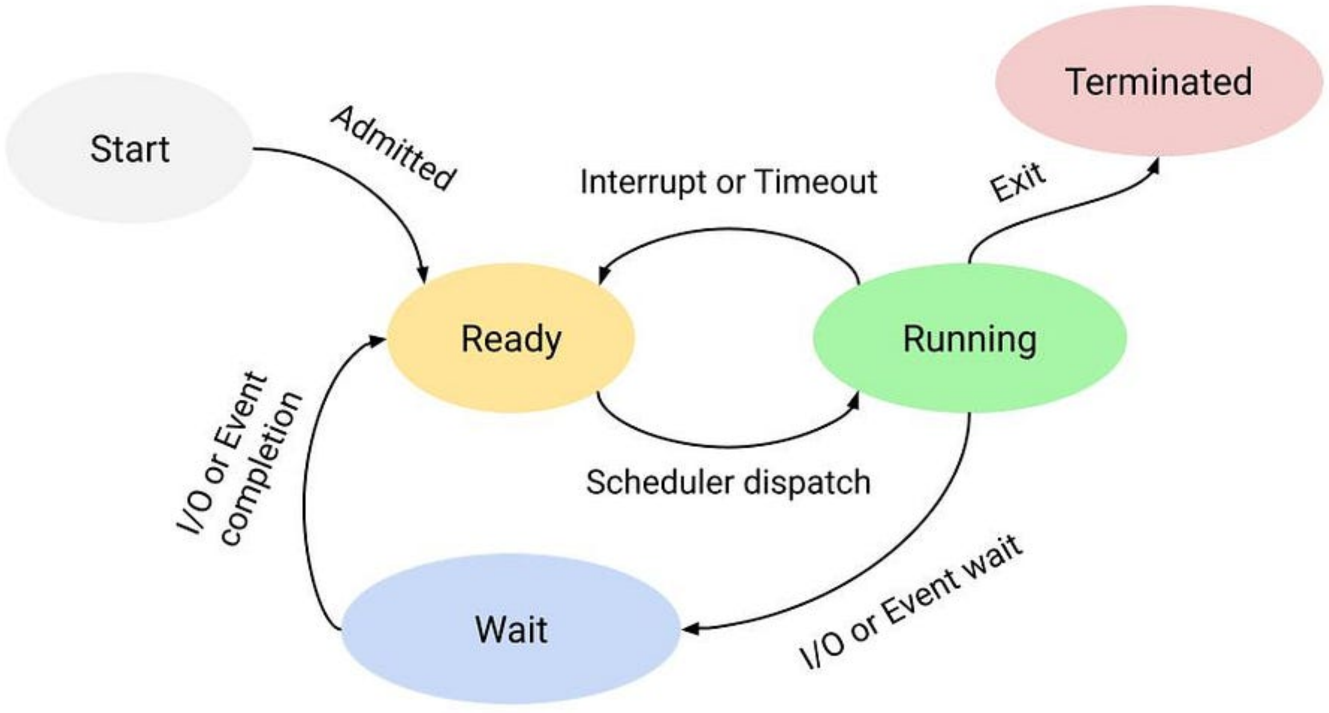
- **Process Id:** Unique id associated with every process to distinguish it from other processes.
- **Process state:** A process can be in a start, ready, running, wait and terminated state.
- **CPU scheduling information:** This is related to priority level relative to the other processes and pointers to scheduling queues.
- **CPU registers and program counter:** Need to be saved and restored when swapping processes in and out of the CPU.
- **I/O status information:** Includes I/O requests, I/O devices (e.g., disk drives) assigned to the process, and a list of files used by the process.

- **Memory management information:** Page tables or segment tables.
- **Accounting information:** User and kernel CPU time consumed, account numbers, limits, etc.


Process Control Block (PCB)
Process Identifier
Process State
CPU-Scheduling information
Memory-Management information
CPU registers and Program Counter
Accounting Information
I/O Status Information
.
.
.

States of a Process in Operating System

Now, we understood the process, where we defined one parameter of a process called **State**. Processes in the operating system can be in any of the five states: start, ready, running, wait, and terminated.



Let's understand these states and transition of a process from one state to another state:

- **Null** to **Start:** When a new process is created, it is said that the process is initiated or a process from NULL state has come to a Start state.
- **Start** to **Ready:** OS moves a process from Start state to the Ready state when it is prepared to execute an additional process. At this stage, process has all the resources available that it needs to run and waiting to get the CPU time for its execution.
- **Ready** to **Running:** At this transition, OS chooses one of the processes from Ready state using process scheduler or dispatcher. After this, CPU starts executing selected process in the running state.
- **Running** to **Terminated:** The currently running process is terminated by OS if it indicates that it has been completed.
- **Running** to **Ready:** Scheduler sets a specific time limit for executing any active process. But if the current process takes more time specified by the scheduler, it pushes it to the ready state again. The most critical reason for this transition is that it has achieved its maximum permissible period for uninterrupted execution. Almost all multiprogramming operating systems enforce this time constraint.

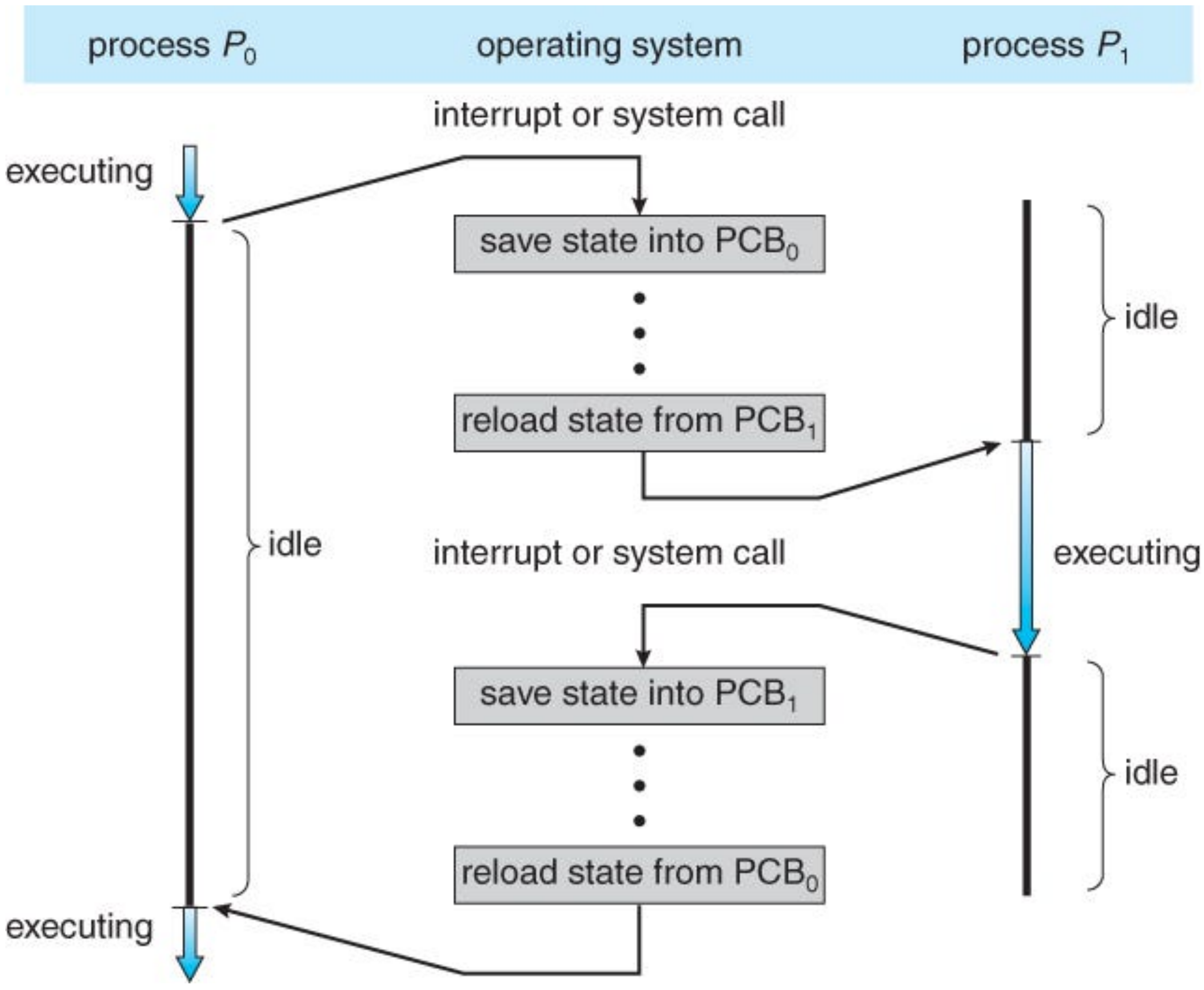
- **Running** to **Wait:** If a process wants anything for which it must wait, it is placed in the waiting state. Now process cannot run because it is waiting for some resource to become available or for some event to occur. For example, process may be waiting for keyboard input, disk access request, inter-process messages, a timer to go off, or a child process to finish.
- **Wait** to **Ready:** A process in the Waiting state is transferred to the Ready state when the event for which a process has been waiting gets completed.

Note: Some operating systems may have other states besides the ones listed here. Explore and think!

Execution of a process in Operating System

Operating system executes various activities in creating a process, which uses a process control block (PCB) to track the execution status of each process.

- To keep track of all processes, it assigns a process ID (PID) to each process to identify it uniquely. As discussed above, it also stores several other critical details in the process control block (PCB).
- Operating system updates information in the process PCB as process make transition from one state to another.
- To access PCB frequently, operating system keeps pointers to each process PCB in a process table.



Process Schedulers in Operating System

Process scheduling is critical for selecting and removing running process based on a particular strategy or algorithm. The main objectives of process scheduling are to keep CPU busy and deliver "acceptable" response times for all programs.

Multiprogramming operating systems allow more than one process to be loaded into the executable memory at a time, and loaded process shares CPU using time multiplexing.

Operating system has three types of process schedulers:

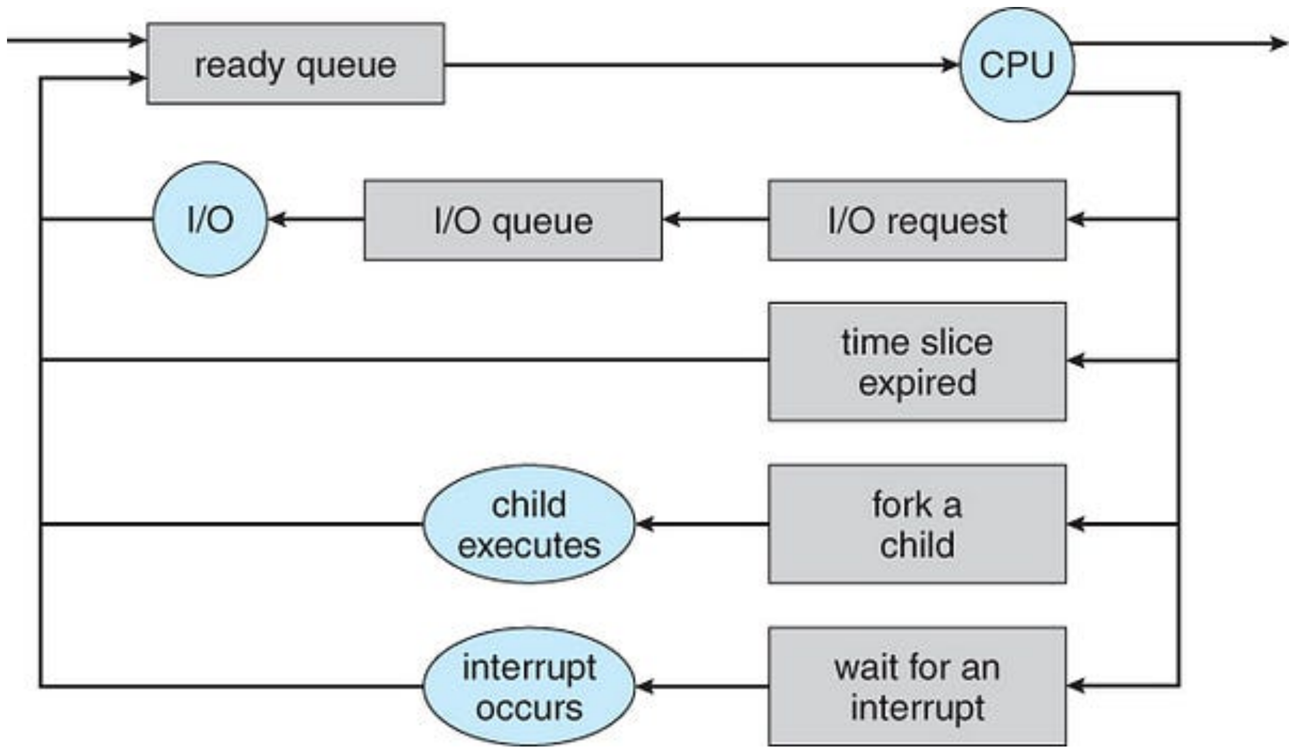
Long Term or Job Scheduler: This scheduler job is to bring new process to the Ready state. It determines which process is assigned to CPU for processing, selects processes from the queue, and loads them into memory for execution.

- Primary objective of job scheduler is to provide a balanced mix of jobs (such as I/O bound and processor bound) and control the degree of multiprogramming.
- It is a heavily loaded system, which can afford to take time to implement intelligent and advanced scheduling algorithms.

Short Term or CPU Scheduler: It is in charge of selecting one process from the ready state and scheduling it to the running state. They are also known as **Dispatchers**.

- CPU scheduler is responsible for ensuring no starvation due to high burst time processes.
- It runs frequently and quickly swaps one process from the running state to the ready state.

Medium Term Scheduler: It is in charge of swapping processes when a particular process is performing an I/O operation. If a running process makes an I/O request, it may be suspended. A process that has been suspended cannot make any progress toward completion. Suspended process is transferred to secondary storage to remove it from memory and make room for other processes. This is known as **switching.**



CPU Scheduling in Operating System

Now our aim would be to understand CPU Scheduling concept and why we need it.

Both I/O and CPU time is used in a typical procedure. Time spent waiting for I/O in an old operating system like MS-DOS is wasted, and CPU is free during this time. In multiprogramming modern operating systems, one process can use CPU while another waits for I/O.

CPU Scheduling determines which process will exclusively use CPU while another is paused. The goal is to ensure that whenever CPU is idle, OS chooses at least one of the programs in the ready queue to run. Here CPU scheduler will be in charge of the selection process. It chooses one of the processes in memory that are ready to run.

Types of CPU Scheduling

There are two major types of CPU Scheduling:

- **Preemptive Scheduling:** Tasks are assigned with their priorities in Preemptive Scheduling. Even if a lower priority task is still running, it is sometimes necessary to run a higher priority task before a lower priority task. In this situation, lower priority task is put on hold for a while and resumes when higher priority task is completed.
- **Non-Preemptive Scheduling:** CPU has been assigned to a particular process in this scheduling mechanism. Process that keeps CPU occupied will either switch context or terminate to relieve the CPU. It's the only method that works across a variety of hardware platforms. That's why, unlike preemptive scheduling, it doesn't require any particular hardware like timers.

Differences between Preemptive and Non-Preemptive Scheduling

- In pre-emptive scheduling, memory allocated in the main memory of CPU is for a limited time. It may be assigned to any other process depending on the state of the current process or priority of the new

incoming process. In non-Preemptive scheduling, memory is allocated to the same process until it is completed.
- In preemptive scheduling, if high-priority processes keep on coming, a low-priority process can be interrupted for an indefinite time. In non-preemptive scheduling, if any large process is processing, it will keep on processing and will not allow even a small process to process before it completes execution.
- In preemptive scheduling, it has to save all data of the process in a halted state so that it can continue from that point only, whereas no such requirements exist in non-preemptive scheduling.

Different Scheduling Algorithms in Operating System

Now, we will look at the various scheduling algorithms involved in process management one by one.

First Come First Serve (FCFS) Scheduling Algorithm

It is the most basic CPU scheduling algorithm, where a FIFO queue is used to manage the scheduling strategy. The idea is simple: a process that asks first to get the CPU allocation, get access to the CPU first.

PCB (Process Control Block) of the process is linked to the tail of the queue as it enters the ready queue. As a result, whenever a CPU becomes available, it is assigned to the process at the front of the queue.

Some important points of this method:

- It offers both non-preemptive and pre-emptive scheduling.
- Jobs are always executed on a first-come first-served basis.
- It is easy to use and implement.
- Poor performance and overall wait time are quite high.

A simple example of this algorithm: Suppose we have five processes p1, p2, p3, p4, and p5, and the ready queue receives them at times t1, t2, t3, t4, and t5 such that t1 < t2 < t3 < t4 < t5. So p1 arrived first in the ready queue, so it will be executed first, followed by p2, p3, p4, and p5, respectively.

Convoy Effect

In First Come First Serve type of algorithm, if a process with a large CPU burst time arrives before any small process, then the small process will get blocked by that large process, which is called Convoy Effect.

Shortest Job First(SJF) Scheduling Algorithm

SJF algorithm is a non-preemptive scheduling algorithm. This policy prioritises waiting process with the shortest execution time. Among all scheduling algorithms, Shortest Job First has the advantage of having the shortest average waiting time. It first sorts all processes by arrival time, creates a pool of processes and then chooses the process with the shortest burst times. After completing the process, it again selects the process with the shortest burst time from the pool.

Some important points of this method:

- Short processes are handled very quickly.
- System also has a low overhead because it only makes decisions when a process is finished, or a new one is added.
- When a new process is added, algorithm just has to compare the currently running process to the new one, ignoring any other processes waiting to run.
- Long processes can be postponed indefinitely if short processes are added regularly.

It is further categorized into two types:

1. **Preemptive priority scheduling:** Due to the arrival of a smaller CPU burst time process, the current process is sent to halted state and execution of the new process proceeds.
2. **Non-preemptive priority scheduling:** The current process is not disturbed due to the arrival of a smaller CPU bursts time process.

A simple example of this algorithm: Suppose we have five processes p1, p2, p3, p4, and p5, and the ready queue receives them at times t1, t2, t3, t4, and t5 such that t1 < t2 < t3 < t4 < t5. Now, you can assume the ready queue as the priority queue, which rearranges the incoming process based on CPU bursts time. Therefore, process with the least CPU burst time is delivered first, and so on.

Longest Job First Scheduling Algorithm

Longest Job First (LJF) is non-preemptive scheduling. This algorithm keeps track of the burst time of all processes accessible at the moment of arrival and then assigns processor to the process with the longest burst time. In this algorithm, once a process begins to run, it cannot be halted in the middle of its execution.

It organise processes in ascending order of their arrival time. Then, out of all processes that have arrived up to that point, it will choose one with the longest burst time. After that, it will process it throughout the duration of the burst. Until this process completes its execution, LJF monitors if any more processes arrive.

Some important points of this method:

- This algorithm reduces processing speed, resulting in a decrease in system efficiency and utilisation.
- The average waiting time and turn-around time for a particular set of procedures rise due to this approach.
- With this technique, a process with a short burst time may never get executed while system continues to run large processes.

A simple example of this algorithm: Similar to the above example, you can assume here that the same ready queue prioritises based on a larger CPU burst time i.e. out of those five processes, the one with the largest CPU burst time will be executed first.

Round Robin scheduling Algorithm

Round Robin is a CPU scheduling algorithm in which each process is cyclically assigned a set time slot. This looks similar to FCFS scheduling, except that it includes preemption, which allows system to transition between processes. In other words, Round Robin Strategy was created with time-sharing systems in mind.

Each process has a set amount of time assigned to it, and once that time period has passed, process is preempted and another process takes its place. As a result, all processes get an equal amount of CPU time. This algorithm performs best in terms of average response time.

Some important points of this method:

- Round robin is a pre-emptive algorithm similar to FCFS with some improvements.
- CPU is shifted to the next process after a fixed interval time.
- Widely used scheduling method in traditional OS.

A simple example of this algorithm

Suppose we have five processes p1, p2, p3, p4, and p5, with total execution times of t1, t2, t3, t4, and t5. Now, we have one extra factor, t (a time quanta), which will ensure equal CPU time sharing for each process. Suppose the first process p1 arrives, and after t time of execution, p1 will be preempted and wait for the remaining execution of (t1 — t) time.

At this stage, p1 will move to the wait state, where it can perform its I/O operation. Now CPU is released for the next process, p2. After completing the I/O operation, process p1 is pushed to the ready queue again for its next processing cycle. The data of process p1 will be saved for its execution till t time so that it can continue from the same state in the next cycle. The same goes for all the processes.

Priority Based Scheduling

Priority scheduling is one of the most often used priority scheduling methods. Here priority is assigned to each process, and the highest-priority process will be executed first. On a first-come first-served basis, processes of the same priority will be executed. Note: Prioritisation can be determined by memory limitations, time constraints, or other resource constraints.

Priority scheduling does not always set the priority as the inverse of the CPU burst time and memory. Instead, it can be set internally or externally, but the scheduling is done based on process priority.

Some important points of this method:

- In priority scheduling, there are chances of indefinite blocking or starvation.
- We can leverage the concept of aging to prevent starving of any process by increasing the priority of low-priority processes based on their waiting time.

It is also categorized into two types:

1. **Preemptive priority scheduling:** If due to the incoming of a higher priority process, the current process is sent to halted state and execution of the new process proceeds.
2. **Non-preemptive priority scheduling:** If due to an incoming higher priority process, the current process is not disturbed.

A simple example of this algorithm: Similar to the example of the FCFS algorithm, the processes are inserted in the ready queue but based on a priority now, which can be CPU burst time, memory constraints, etc., and then its execution follows similar to the FCFS algorithm.

Reference: Operating System Concepts by Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin

Thanks to **Vishal Das** for his contribution to creating the first version of the content. Please write in the message below if you find anything incorrect, or if you want to share more insight. Enjoy learning, Enjoy

algorithms!