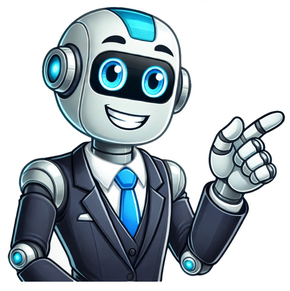
I'm not a robot



LAST UPDATED ON: SEPTEMBER 16, 2024 In this tutorial we will understand the priority scheduling algorithm, how it works and its advantages and disadvantages. In the Shortest Job First scheduling algorithm, the priority of a process is generally the inverse of the CPU burst time, i.e. the larger the burst time the lower is the priority of that process. In case of priority scheduling the priority is not always set as the inverse of the CPU burst time, rather it can be internally or externally set, but yes the scheduling is done on the basis of priority of the process where the process which is most urgent is processed first, followed by the ones with lesser priority in order. Processes with same priority are executed in FCFS manner. The priority of process, when internally defined, can be decided based on memory requirements, time limits ,number of open files, ratio of I/O burst to CPU burst etc. Whereas, external priorities are set based on criteria outside the operating system, like the importance of the process, funds paid for the computer resource use, makrte factor etc. Types of Priority Scheduling Algorithm Priority scheduling can be of two types: Preemptive Priority Scheduling: If the new process arrived at the ready queue has a higher priority than the currently running process, the CPU is preempted, which means the processing of the current process is stoped and the incoming new process with higher priority gets the CPU for its execution. Non-Preemptive Priority Scheduling: In case of non-preemptive priority scheduling algorithm if a new process arrives with a higher priority than the current running process, the incoming process is put at the head of the ready queue, which means after the execution of the current process it will be processed. Example of Priority Scheduling Algorithm Consider the below table fo processes with their respective CPU burst times and the priorities. As you can see in the GANTT chart that the processes are given CPU time just on the basis of the priorities. Problem with Priority Scheduling Algorithm In priority scheduling algorithm, the chances of indefinite blocking or starvation. A process is considered blocked when it is ready to run but has to wait for the CPU as some other process is running currently. But in case of priority scheduling if new higher priority processes keeps coming in the ready queue then the processes waiting in the ready queue with lower priority may have to wait for long durations before getting the CPU for execution. In 1973, when the IBM 7904 machine was shut down at MIT, a low-priority process was found which was submitted in 1967 and had not yet been run. Using Aging Technique with Priority Scheduling To prevent starvation of any process, we can use the concept of aging where we keep on increasing the priority of low-priority process based on the its waiting time. For example, if we decide the aging factor to be 0.5 for each day of waiting, then if a process with priority 20(which is comparitively low priority) comes in the ready queue. After one day of waiting, its priority is increased to 19.5 and so on. Doing so, we can ensure that no process will have to wait for indefinite time for getting CPU time for processing. Implementing Priority Scheduling Algorithm in C++ Implementing priority scheduling algorithm is easy. All we have to do is to sort the processes based on their priority and CPU burst time, and then apply FCFS Algorithm on it. Here is the C++ code for priority scheduling algorithm: // Implementation of Priority scheduling algorithm #include using namespace std; struct Process { // this is the process ID int pid; // the CPU burst time int bt; // priority of the process int priority; }; // sort the processes based on priority bool sortProcesses(Process a, Process b) { return (a.priority > b.priority); } // Function to find the waiting time for all processes void findWaitingTime(Process proc[], int n, int wt[]) { // waiting time for first process is 0 wt[0] = 0; // calculating waiting time for (int i = 1; i < n ; i++) wt[i] = proc[i-1].bt + wt[i-1] ; } // Function to calculate turn around time void findTurnAroundTime(Process proc[], int n, int wt[], int tat[]) { // calculating turnaround time by adding // bt[i] + wt[i] for (int i = 0; i < n ; i++) tat[i] = proc[i].bt + wt[i]; } //Function to calculate average time void findavgTime(Process proc[], int n) { int wt[n], tat[n], total_wt = 0, total_tat = 0; //Function to find waiting time of all processes findWaitingTime(proc, n, wt); //Function to find turn around time for all processes findTurnAroundTime(proc, n, wt, tat); //Display processes along with all details cout