INTRODUCTION TO OS

PROCESS & MULTITHREADING
Operating System Processes
Process Scheduling
CPU Scheduling
First Come First Serve
Shortest Job First
Priority Scheduling
Round Robin Scheduling
Multilevel Queue Scheduling
Multilevel Feedback Queue Scheduling
Comparision of Scheduling Algorithms
Introduction to Threads
Process Synchronization
Classical Synchronization Problems
Bounded Buffer Problem
Dining Philosophers Problem
Readers Writer Problem
Semaphores in OS
Deadlocks
Classical Problems of Synchronization
Deadlock Prevention in OS
Deadlock Avoidance in OS
Deadlock Detection and Recovery

CPU SCHEDULING

MEMORY MANAGEMENT

Shortest Job First(SJF) Scheduling

Shortest Job First scheduling works on the process with the shortest **burst time** or **duration** first.

- This is the best approach to minimize waiting time.
- This is used in Batch Systems.
- It is of two types:
  1. Non Pre-emptive
  2. Pre-emptive
- To successfully implement it, the burst time/duration time of the processes should be known to the processor in advance, which is practically not feasible all the time.
- This scheduling algorithm is optimal if all the jobs/processes are available at the same time. (either Arrival time is `0` for all, or Arrival time is same for all)

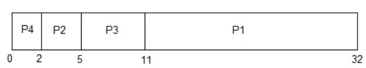
Non Pre-emptive Shortest Job First

Consider the below processes available in the ready queue for execution, with **arrival time** as `0` for all and given **burst times**.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

In Shortest Job First Scheduling, the shortest Process is executed first.

Hence the GANTT chart will be following :



Now, the average waiting time will be = (0 + 2 + 5 + 11)/4 = 4.5 ms

As you can see in the **GANTT chart** above, the process **P4** will be picked up first as it has the shortest burst time, then **P2**, followed by **P3** and at last **P1**.

We scheduled the same set of processes using the First come first serve algorithm in the previous tutorial, and got average waiting time to be `18.75 ms`, whereas with SJF, the average waiting time comes out `4.5 ms`.

Problem with Non Pre-emptive SJF

If the **arrival time** for processes are different, which means all the processes are not available in the ready queue at time `0`, and some jobs arrive after some time, in such situation, sometimes process with short burst time have to wait for the current process's execution to finish, because in Non Pre-emptive SJF, on arrival of a process with short duration, the existing job/process's execution is not halted/stopped to execute the short job first.

This leads to the problem of **Starvation**, where a shorter process has to wait for a long time until the current longer process gets executed. This happens if shorter jobs keep coming, but this can be solved using the concept of **aging**.

Pre-emptive Shortest Job First

In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with **short burst time** arrives, the existing process is preempted or removed from execution, and the shorter job is executed first.

Pre-emptive Shortest Job First Scheduling

The average waiting time will be,((5-3)+(6-2)+(12-1))/4=8.75

The average waiting time for preemptive shortest job first scheduling is less than both,non preemptive SJF scheduling and FCFS scheduling

As you can see in the **GANTT chart** above, as **P1** arrives first, hence it's execution starts immediately, but just after `1 ms`, process **P2** arrives with a **burst time** of `3 ms` which is less than the burst time of **P1**, hence the process **P1**(1 ms done, 20 ms left) is preemptied and process **P2** is executed.

As **P2** is getting executed, after `1 ms`, **P3** arrives, but it has a burst time greater than that of **P2**, hence execution of **P2** continues. But after another millisecond, **P4** arrives with a burst time of `2 ms`, as a result **P2**(2 ms done, 1 ms left) is preemptied and **P4** is executed.

After the completion of **P4**, process **P2** is picked up and finishes, then **P2** will get executed and at last **P1**.

The Pre-emptive SJF is also known as **Shortest Remaining Time First**, because at any given point of time, the job with the shortest remaining time is executed first.

Program for SJF Scheduling

In the below program, we consider the **arrival time** of all the jobs to be `0`.

Also, in the program, we will **sort** all the jobs based on their **burst time** and then execute them one by one, just like we did in FCFS scheduling program.

```
        total_tat = total_tat + tat[i];
        cout << " " << proc[i].pid << "\t\t"
             << proc[i].bt << "\t " << wt[i]
             << "\t\t " << tat[i] <<endl;
    }

    cout << "Average waiting time = "
         << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
         << (float)total_tat / (float)n;
}

// main function
int main()
{
    Process proc[] = {{1, 21}, {2, 3}, {3, 6}, {4, 2}};
    int n = sizeof proc / sizeof proc[0];

    // sorting processes by burst time.
    sort(proc, proc + n, comparison);
```

OUTPUT:

```
Order in which process gets executed
4 2 3 1
Processes  Burst time  Waiting time  Turn around time
 4            2       0             2
 2            3       2             5
 3            6       5             11
 1            21      11            32
Average waiting time = 4.5
Average turn around time = 12.5
```

Try implementing the program for SJF with variable **arrival time** for different jobs, yourself.

More...