# PREEMPTIVE SCHEDULING ALGORITHM

Preemptive scheduling is used when a process switches from running state to ready state or from the waiting state to ready state. The resources (mainly CPU cycles) are allocated to the process for a limited amount of time and then taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining. That process stays in the ready queue till it gets its next chance to execute.

**Algorithms based on preemptive scheduling :**

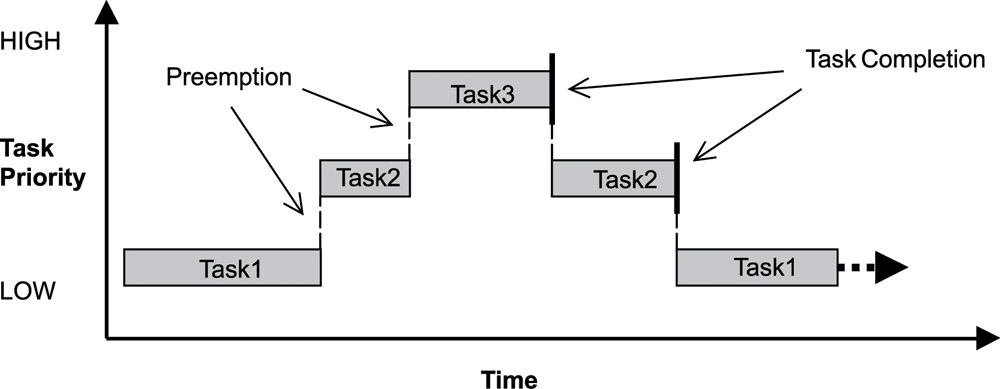
* Round Robin (RR)
* Shortest Remaining Time First (SRTF)
* Priority (preemptive version)

**Priority (preemptive version)**

In Preemptive Priority Scheduling, at the time of arrival of a process in the ready queue, its Priority is compared with the priority of the other processes present in the ready queue as well as with the one which is being executed by the CPU at that point of time. The One with the highest priority among all the available processes will be given the CPU next.

The difference between preemptive priority scheduling and non-preemptive priority scheduling is that, in the preemptive priority scheduling, the job which is being executed can be stopped at the arrival of a higher priority job.

Once all the jobs get available in the ready queue, the algorithm will behave as non-preemptive priority scheduling, which means the job scheduled will run till the completion and no preemption will be done.



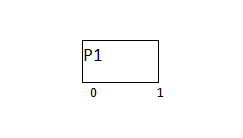
Example

There are 7 processes P1, P2, P3, P4, P5, P6 and P7 given. Their respective priorities, Arrival Times and Burst times are given in the table below.

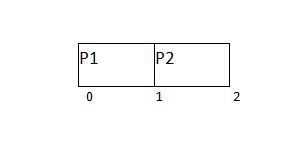
|  |  |  |  |
| --- | --- | --- | --- |
| **Process Id** | **Priority** | **Arrival Time** | **Burst Time** |
| 1 | 2(L) | 0 | 1 |
| 2 | 6 | 1 | 7 |
| 3 | 3 | 2 | 3 |
| 4 | 5 | 3 | 6 |
| 5 | 4 | 4 | 5 |
| 6 | 10(H) | 5 | 15 |
| 7 | 9 | 15 | 8 |

### GANTT chart Preparation

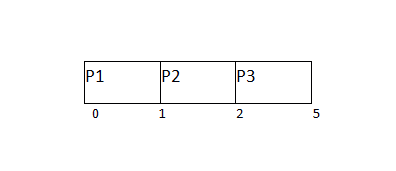
At time 0, P1 arrives with the burst time of 1 units and priority 2. Since no other process is available hence this will be scheduled till next job arrives or its completion (whichever is lesser).



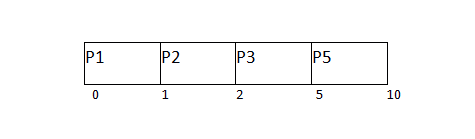
At time 1, P2 arrives. P1 has completed its execution and no other process is available at this time hence the Operating system has to schedule it regardless of the priority assigned to it.



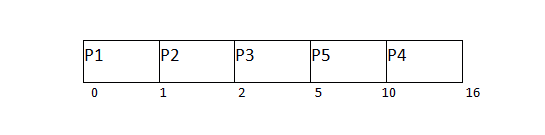
The Next process P3 arrives at time unit 2, the priority of P3 is higher to P2. Hence the execution of P2 will be stopped and P3 will be scheduled on the CPU.



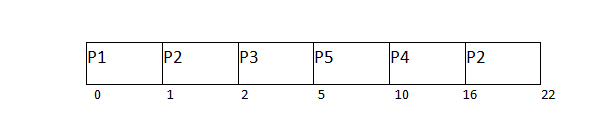
During the execution of P3, three more processes P4, P5 and P6 becomes available. Since, all these three have the priority lower to the process in execution so PS can't preempt the process. P3 will complete its execution and then P5 will be scheduled with the priority highest among the available processes.



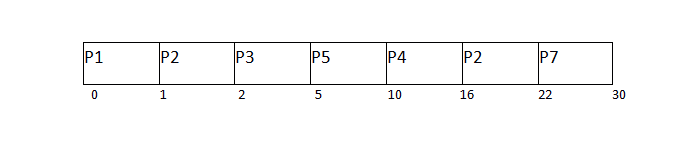
Meanwhile the execution of P5, all the processes got available in the ready queue. At this point, the algorithm will start behaving as Non Preemptive Priority Scheduling. Hence now, once all the processes get available in the ready queue, the OS just took the process with the highest priority and execute that process till completion. In this case, P4 will be scheduled and will be executed till the completion.



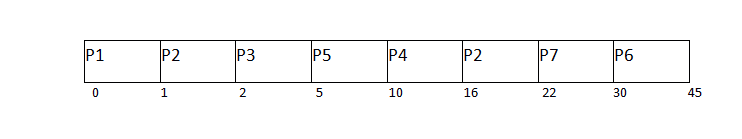
Since P4 is completed, the other process with the highest priority available in the ready queue is P2. Hence P2 will be scheduled next.



P2 is given the CPU till the completion. Since its remaining burst time is 6 units hence P7 will be scheduled after this.



The only remaining process is P6 with the least priority, the Operating System has no choice unless of executing it. This will be executed at the last.



The Completion Time of each process is determined with the help of GANTT chart. The turnaround time and the waiting time can be calculated by the following formula.

1. Turnaround Time = Completion Time - Arrival Time
2. Waiting Time = Turn Around Time - Burst Time

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Process Id** | **Priority** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turnaround Time** | **Waiting Time** |
| 1 | 2 | 0 | 1 | 1 | 1 | 0 |
| 2 | 6 | 1 | 7 | 22 | 21 | 14 |
| 3 | 3 | 2 | 3 | 5 | 3 | 0 |
| 4 | 5 | 3 | 6 | 16 | 13 | 7 |
| 5 | 4 | 4 | 5 | 10 | 6 | 1 |
| 6 | 10 | 5 | 15 | 45 | 40 | 25 |
| 7 | 9 | 6 | 8 | 30 | 24 | 16 |

# ALGORITHM :

completed = 0

current\_time = 0

while(completed != n) {

    find process with maximum priority time among process that are in ready queue at current\_time

    if(process found) {

        if(process is getting CPU for the first time) {

            start\_time = current\_time

        }

        burst\_time = burst\_time - 1

        current\_time = current\_time + 1

        if(burst\_time == 0) {

            completion\_time  = current\_time

            turnaround\_time = completion\_time - arrival\_time

            waiting\_time = turnaround\_time - burst\_time

            response\_time = start\_time - arrival\_time

            mark process as completed

            completed++

        }

    }

    else {

        current\_time++

    }

}

# IMPLEMENTATION:

#include <iostream>

#include <iomanip>

#include <string.h>

using namespace std;

struct process {

    int pid;

    int arrival\_time;

    int burst\_time;

    int priority;

    int start\_time;

    int completion\_time;

    int turnaround\_time;

    int waiting\_time;

    int response\_time;

};

int main() {

    int n;

    struct process p[100];

    float avg\_turnaround\_time;

    float avg\_waiting\_time;

    float avg\_response\_time;

    int total\_turnaround\_time = 0;

    int total\_waiting\_time = 0;

    int total\_response\_time = 0;

    int total\_idle\_time = 0;

    int burst\_remaining[100];

    int is\_completed[100];

    memset(is\_completed,0,sizeof(is\_completed));

    cout<<"Enter the number of processes: ";

    cin>>n;

    for(int i = 0; i < n; i++) {

        cout<<"Enter arrival time of process "<<i+1<<": ";

        cin>>p[i].arrival\_time;

        cout<<"Enter burst time of process "<<i+1<<": ";

        cin>>p[i].burst\_time;

        cout<<"Enter priority of the process "<<i+1<<": ";

        cin>>p[i].priority;

        p[i].pid = i+1;

        burst\_remaining[i] = p[i].burst\_time;

        cout<<endl;

    }

    int current\_time = 0;

    int completed = 0;

    int prev = 0;

    while(completed != n) {

        int idx = -1;

        int mx = -1;

        for(int i = 0; i < n; i++) {

            if(p[i].arrival\_time <= current\_time && is\_completed[i] == 0) {

                if(p[i].priority > mx) {

                    mx = p[i].priority;

                    idx = i;

                }

                if(p[i].priority == mx) {

                    if(p[i].arrival\_time < p[idx].arrival\_time) {

                        mx = p[i].priority;

                        idx = i;

                    }

                }

            }

        }

        if(idx != -1) {

            if(burst\_remaining[idx] == p[idx].burst\_time) {

                p[idx].start\_time = current\_time;

                total\_idle\_time += p[idx].start\_time - prev;

            }

            burst\_remaining[idx] -= 1;

            current\_time++;

            prev = current\_time;

            if(burst\_remaining[idx] == 0) {

                p[idx].completion\_time = current\_time;

                p[idx].turnaround\_time = p[idx].completion\_time - p[idx].arrival\_time;

                p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time;

                p[idx].response\_time = p[idx].start\_time - p[idx].arrival\_time;

                total\_turnaround\_time += p[idx].turnaround\_time;

                total\_waiting\_time += p[idx].waiting\_time;

                total\_response\_time += p[idx].response\_time;

                is\_completed[idx] = 1;

                completed++;

            }

        }

        else {

             current\_time++;

        }

    }

    avg\_turnaround\_time = (float) total\_turnaround\_time / n;

    avg\_waiting\_time = (float) total\_waiting\_time / n;

    avg\_response\_time = (float) total\_response\_time / n;

    cout<<endl<<endl;

    cout<<"#P\t"<<"AT\t"<<"BT\t"<<"PRI\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;

    for(int i = 0; i < n; i++) {

        cout<<p[i].pid<<"\t"<<p[i].arrival\_time<<"\t"<<p[i].burst\_time<<"\t"<<p[i].priority<<"\t"<<p[i].start\_time<<"\t"<<p[i].completion\_time<<"\t"<<p[i].turnaround\_time<<"\t"<<p[i].waiting\_time<<"\t"<<p[i].response\_time<<"\t"<<"\n"<<endl;

    }

    cout<<"Average Turnaround Time = "<<avg\_turnaround\_time<<endl;

    cout<<"Average Waiting Time = "<<avg\_waiting\_time<<endl;

    cout<<"Average Response Time = "<<avg\_response\_time<<endl;

}