Batch: A1	**Roll No.: 16010123012**
Experiment No. 05	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

TITLE: Implementation of Priority Based CPU Scheduling Algorithms – Pre-emptive and Non Pre-emptive.

AIM: Implementation of Priority Based CPU Scheduling Algorithms – Pre-emptive Non Preemptive

---

Expected Outcome of Experiment:

CO 2 Illustrate and analyse the Process, threads, process scheduling and thread scheduling

---

Books/ Journals/ Websites referred:

1. **Silberschatz A., Galvin P., Gagne G. "Operating Systems Principles", Willey Eight edition.**

2. **Achyut S. Godbole , Atul Kahate "Operating Systems" McGraw Hill Third Edition.**

3. **William Stallings, "Operating System Internal & Design Principles", Pearson.**

4. **Andrew S. Tanenbaum, "Modern Operating System", Prentice Hall.**

Pre Lab/ Prior Concepts:

Priority scheduling in OS is the scheduling algorithm that schedules processes according to the priority assigned to each of the processes. Higher priority processes are executed before lower priority processes.

What is a Priority Scheduling Algorithm?

The Operating System has a major function of deciding the order in which processes/tasks will have access to the processor and the amount of processing time each of these processes or tasks will have. This function of the operating system is termed process scheduling.

Under process scheduling, the tasks that the operating system performs are:

- Keeping track of the status of the processes
- Allocation of processor to processes
- De-allocation of the processor to processes.

All these processes are stored by the operating system in a process table along with an ID allotted to every process (PID), to help keep track of all of them. And, to keep track of their current state the operating system uses a Process Control Block (PCB). The information is updated in the control block as and when the state of the process changes.

But did you wonder what is the criteria that the Operating System uses for the scheduling of these processes? Which process must be allocated with CPU resources first?

Well, there are multiple ways in which the Operating System makes these decisions such as execution of the process that takes the least time first, or execution according to the order in which they requested access to the processor, and so on. One such important criterion is the priority of the process.

In priority scheduling in OS, processes are executed on the basis of their priority. The jobs/processes with higher priority are executed first. Naturally, you might want to know how the priority of processes is decided.

Priority of processes depends on some factors such as:

Time limit

Memory requirements of the process

Ratio of average I/O to average CPU burst time

There can be more factors on the basis of which the priority of a process/job is determined. This priority is assigned to the processes by the scheduler.

These priorities of processes are represented as simple integers in a fixed range such as 0 to 7, or maybe 0 to 4095. These numbers depend on the type of system.

Now that we know that each job is assigned a priority on the basis of which it is scheduled, we can move ahead with its working and types.

Types of Priority Scheduling Algorithms

There are two types of priority scheduling algorithms in OS:

Non-Preemptive Scheduling

In this type of scheduling:

If during the execution of a process, another process with a higher priority arrives for execution, even then the currently executing process will not be disturbed.

The newly arrived high priority process will be put in next for execution since it has higher priority than the processes that are in a waiting state for execution.

All the other processes will remain in the waiting queue to be processed. Once the execution of the current process is done, the high-priority process will be given the CPU for execution.

Preemptive Scheduling

`Preemptive Scheduling as opposed to non-preemptive scheduling will preempt (stop and store the currently executing process) the currently running process if a higher priority process enters the waiting state for execution and will execute the higher priority process first and then resume executing the previous process.

Characteristics of Priority Scheduling Algorithm

It is a scheduling algorithm that schedules the incoming processes on the basis of priority.

Operating systems use it for performing batch processes

If there exist two jobs/processes in the ready state (ready for execution) that have the same priority, then priority scheduling executes the processes on a first come first serve basis. For every job that exists, we have a priority number assigned to it that indicates its priority level.

If the integer value of the priority number is low, it means that the process has a higher priority. (low number = high priority).

Description of the application to be implemented:

Pre-emptive:

Step 1: Input Process Details
1. Start
2. Take input for the number of processes n
3. For each process, take input:
   - Arrival Time (AT)
   - Burst Time (BT)
   - Priority (Higher value = Higher priority)

- Assign Process ID (PID)

4. Sort processes by arrival time in ascending order

Step 2: Find Completion Time (CT) Using Priority Scheduling

1. Initialize current_time = 0
2. Create a boolean array is_completed[n], initialized to `false`
3. For each process (count = 0 to n-1):
   - Find the process with the highest priority among those that have arrived (AT ≤ current_time) and are not yet completed
   - If no process is available, increment current_time and continue
   - Otherwise:
     - Execute the selected process for its burst time (BT)
     - Update its completion time (CT = current_time + BT)
     - Mark the process as completed
     - Update current_time

Step 3: Calculate and Display Results

1. Compute Turnaround Time (TAT) and Waiting Time (WT)
2. Compute Average Turnaround Time (Avg TAT) and Average Waiting Time (Avg WT)
3. Print Process ID, Arrival Time, Burst Time, Priority, Completion Time, Turnaround Time Waiting Time, Avg TAT and Avg WT
4. End

Non Pre-emptive:

Step 1: Input Process Details

1. Start
2. Take input for the number of processes n
3. For each process, take input:
   - Arrival Time (AT)
   - Burst Time (BT)
   - Priority (Higher value means higher priority)
   - Assign Process ID (PID)
   - Initialize Remaining Burst Time to Burst Time

Step 2: Find Completion Time Using Priority Preemptive Scheduling

1. Sort the processes in ascending order of Arrival Time (AT)
2. Initialize current_time = 0, completed = 0
3. Create a priority queue (min-heap) to store available processes, sorted by:
   - Higher priority first (higher value = higher priority)
   - If priority values are the same, choose the process with earlier arrival time
4. While (completed < n):
   - Add all processes that have arrived (`AT ≤ current_time`) to the priority queue

- If no process is available, increment current_time and continue
- Select the process with the highest priority (from the priority queue)
- Execute the process for 1 unit of time (preemptive execution)
- Decrement its remaining time
- If the process completes (remaining time = 0):
  - Set Completion Time (CT) = current_time
  - Compute Turnaround Time (TAT) = CT - AT
  - Compute Waiting Time (WT) = TAT - BT
  - Remove the process from the queue
  - Increment completed count
- If the process is not completed, push it back into the queue

Step 3: Calculate and Display Results
1. Compute Average Turnaround Time (Avg TAT) and Average Waiting Time (Avg WT)
2. Print Process ID, Arrival Time, Burst Time, Priority, Completion Time, Turnaround Time Waiting Time, Avg TAT and Avg WT
3. End

Implementation details:
Pre-emptive

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

struct Process
{
  int pid;
  float arrivalTime;
  float burstTime;
  float remainingTime;
  float completionTime;
  float turnaroundTime;
  float waitingTime;
  int priority;
};

struct comparePriority
{
  bool operator()(Process &a, Process &b)
  {
    if (a.priority == b.priority)
      return a.arrivalTime > b.arrivalTime;
    return a.priority < b.priority
  }
```

```
};

void findCompletionTime(Process proc[], int n)
{
  float current_time = 0;
  int completed = 0;
  int i = 0;
  priority_queue<Process, vector<Process>, comparePriority> pq;
  sort(proc, proc + n, [](const Process &a, const Process &b)
       { return a.arrivalTime < b.arrivalTime; });

  while (completed < n)
  {
    while (i < n && proc[i].arrivalTime <= current_time)
    {
      pq.push(proc[i]);
      i++;
    }

    if (pq.empty())
    {
      current_time++;
      continue;
    }

    Process currProc = pq.top();
    pq.pop();

    currProc.remainingTime--;
    current_time++;

    if (currProc.remainingTime == 0)
    {
      completed++;
      currProc.completionTime = current_time;
      currProc.turnaroundTime = currProc.completionTime -
currProc.arrivalTime;
      currProc.waitingTime = currProc.turnaroundTime -
currProc.burstTime;
    }
    if (currProc.remainingTime > 0)
    {
      pq.push(currProc);
    }
```

```
    else
    {
      proc[currProc.pid - 1] = currProc;
    }
  }
}

void printAvgTime(Process proc[], int n)
{
  float total_wt = 0, total_tat = 0;
  for (int i = 0; i < n; i++)
  {
    total_wt += proc[i].waitingTime;
    total_tat += proc[i].turnaroundTime;
  }
  cout << "Average turnaround time = " << total_tat / n << endl;
  cout << "Average waiting time = " << total_wt / n << endl;
}

int main()
{
  int n;
  cout << "Enter the number of processes: ";
  cin >> n;
  Process proc[n];

  for (int i = 0; i < n; i++)
  {
    cout << "Enter the arrival time of process " << i + 1 << ": ";
    cin >> proc[i].arrivalTime;
    cout << "Enter the burst time of process " << i + 1 << ": ";
    cin >> proc[i].burstTime;
    cout << "Enter the priority of process " << i + 1 << ": ";
    cin >> proc[i].priority;

    proc[i].pid = i + 1;
    proc[i].remainingTime = proc[i].burstTime;
  }
  findCompletionTime(proc, n);
  cout << endl;
  cout << "PN\tAT\tBT\tPriority\tCT\tTAT\tWT" << endl;
  for (int i = 0; i < n; i++)
  {
```

```
    cout << proc[i].pid << "\t" << proc[i].arrivalTime << "\t" <<
proc[i].burstTime << "\t"
         << proc[i].priority << "\t\t" << proc[i].completionTime << "\t"
         << proc[i].turnaroundTime << "\t" << proc[i].waitingTime <<
endl;
  }
  printAvgTime(proc, n);

  return 0;
}
```

```
Enter the number of processes: 4
Enter the arrival time of process 1: 0
Enter the burst time of process 1: 5
Enter the priority of process 1: 10
Enter the arrival time of process 2: 1
Enter the burst time of process 2: 4
Enter the priority of process 2: 20
Enter the arrival time of process 3: 2
Enter the burst time of process 3: 2
Enter the priority of process 3: 30
Enter the arrival time of process 4: 4
Enter the burst time of process 4: 1
Enter the priority of process 4: 40

PN      AT      BT      Priority        CT      TAT     WT
1       0       5       10              12      12      7
2       1       4       20              8       7       3
3       2       2       30              4       2       0
4       4       1       40              5       1       0
Average turnaround time = 5.5
Average waiting time = 2.5
```

Non Pre-emptive

```
#include <bits/stdc++.h>
#define endl '\n'
using namespace std;

struct Process
{
  int pid;
  float arrival_time;
  float burst_time;
  float completion_time;
  float turnaround_time;
```

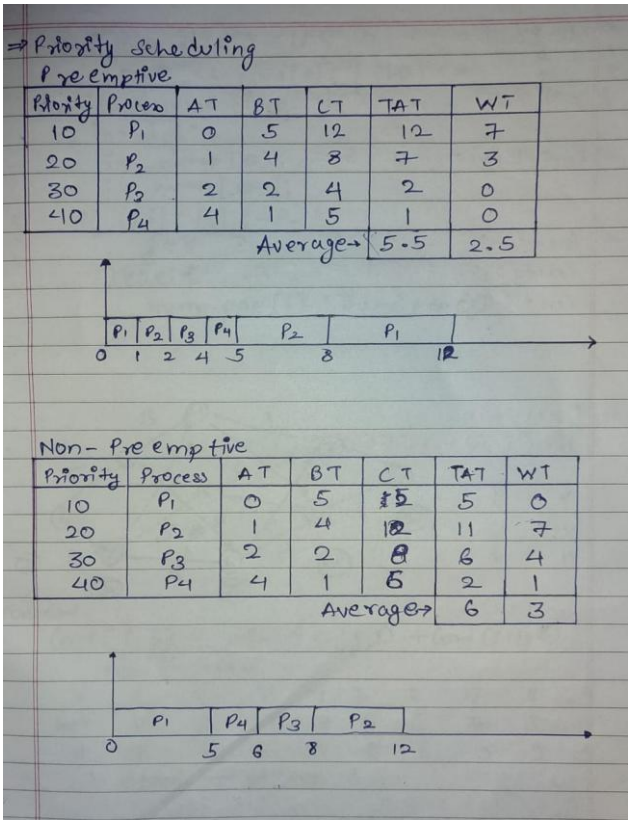
```
  float waiting_time;
  int priority;
};

void findCompletionTime(Process proc[], int n)
{
  float current_time = 0;
  vector<bool> is_completed(n, false);

  for (int count = 0; count < n; count++)
  {
    int idx = -1;
    float max_priority = -1;

    for (int i = 0; i < n; i++)
    {
      if (!is_completed[i] && proc[i].arrival_time <= current_time)
      {
        if (proc[i].priority > max_priority)
        {
          max_priority = proc[i].priority;
          idx = i;
        }
      }
    }

    if (idx == -1)
    {
      current_time++;
      continue;
    }

    current_time += proc[idx].burst_time;
    proc[idx].completion_time = current_time;
    is_completed[idx] = true;
  }
}

void findTurnAroundTime(Process proc[], int n)
{
  for (int i = 0; i < n; i++)
  {
    proc[i].turnaround_time = proc[i].completion_time -
proc[i].arrival_time;
```

```
  }
}

void findWaitingTime(Process proc[], int n)
{
  for (int i = 0; i < n; i++)
  {
    proc[i].waiting_time = proc[i].turnaround_time - proc[i].burst_time;
  }
}

void findAvgTime(Process proc[], int n)
{
  float total_wt = 0, total_tat = 0;
  for (int i = 0; i < n; i++)
  {
    total_wt += proc[i].waiting_time;
    total_tat += proc[i].turnaround_time;
  }
  cout << "Average turnaround time = " << total_tat / n << endl;
  cout << "Average waiting time = " << total_wt / n << endl;
}

int main()
{
  int n;
  cout << "Enter the number of processes: ";
  cin >> n;

  Process proc[n];

  for (int i = 0; i < n; i++)
  {
    cout << "Enter the arrival time of process " << i + 1 << ": ";
    cin >> proc[i].arrival_time;
    cout << "Enter the burst time of process " << i + 1 << ": ";
    cin >> proc[i].burst_time;
    cout << "Enter the priority of process " << i + 1 << "(higher value
means higher priority): ";
    cin >> proc[i].priority;
    proc[i].pid = i + 1;
  }

  sort(proc, proc + n, [](Process a, Process b)
```

```
      { return a.arrival_time < b.arrival_time; });

  findCompletionTime(proc, n);
  findTurnAroundTime(proc, n);
  findWaitingTime(proc, n);

  cout << "PN\tAT\tBT\tPriority\tCT\tTAT\tWT" << endl;
  for (int i = 0; i < n; i++)
  {
    cout << proc[i].pid << "\t" << proc[i].arrival_time << "\t" <<
proc[i].burst_time
         << "\t" << proc[i].priority << "\t\t" << proc[i].completion_time
<< "\t"
         << proc[i].turnaround_time << "\t" << proc[i].waiting_time <<
endl;
  }

  findAvgTime(proc, n);
  return 0;
}
```

```
Enter the number of processes: 4
Enter the arrival time of process 1: 0
Enter the burst time of process 1: 5
Enter the priority of process 1(higher value means higher priority): 10
Enter the arrival time of process 2: 1
Enter the burst time of process 2: 4
Enter the priority of process 2(higher value means higher priority): 20
Enter the arrival time of process 3: 2
Enter the burst time of process 3: 2
Enter the priority of process 3(higher value means higher priority): 30
Enter the arrival time of process 4: 4
Enter the burst time of process 4: 1
Enter the priority of process 4(higher value means higher priority): 40
PN      AT      BT      Priority        CT      TAT     WT
1       0       5       10              5       5       0
2       1       4       20              12      11      7
3       2       2       30              8       6       4
4       4       1       40              6       2       1
Average turnaround time = 6
Average waiting time = 3


...Program finished with exit code 0
```

Lab – Work:



⇒ Priority Scheduling

Preemptive

Priority	Process	AT	BT	CT	TAT	WT
10	P_1	0	5	12	12	7
20	P_2	1	4	8	7	3
30	P_3	2	2	4	2	0
40	P_4	4	1	5	1	0
				Average→	5.5	2.5

Non-Preemptive

Priority	Process	AT	BT	CT	TAT	WT
10	P_1	0	5	5	5	0
20	P_2	1	4	12	11	7
30	P_3	2	2	8	6	4
40	P_4	4	1	6	2	1
				Average→	6	3

Conclusion:

I have successfully implemented Preemptive and Non-Preemptive Priority Scheduling algorithms, gaining insights into how priority levels impact process execution. In preemptive scheduling, higher-priority processes interrupt lower-priority ones, ensuring faster execution but increasing context switching overhead. Non-preemptive scheduling is simpler but may delay urgent tasks. By analyzing turnaround and waiting times, I observed how scheduling policies affect CPU efficiency and response time, reinforcing the need to balance fairness, efficiency, and priority in process management.

Post Lab Descriptive Questions

1. Consider a set of processes with their respective arrival and burst times given in milliseconds.

Process	Priority	Arrival Time	Burst Time
P1	1	0	4
P2	2	0	3
P3	1	6	7
P4	3	11	4
P5	2	12	2

A. **Apply the Priority based CPU scheduling algorithm.**

- Draw the Gantt Chart.

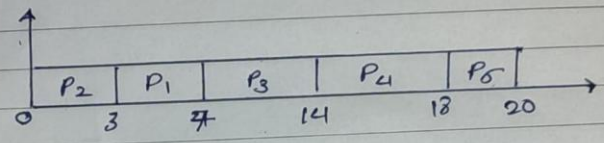
- Calculate the Average Turnaround Time.
- Calculate the Average Waiting Time.

B. **Analyze the results and comment on the performance of pre-emptive and non pre-emptive for the given process set.**

Preemptive and Non-Preemptive Priority Scheduling result in the same average turnaround time and average waiting time for this specific process set. Preemptive scheduling introduces context switching overhead but ensures higher-priority processes get immediate CPU access. Non-preemptive scheduling is simpler and avoids context switching but may delay urgent tasks if a lower-priority process is running.

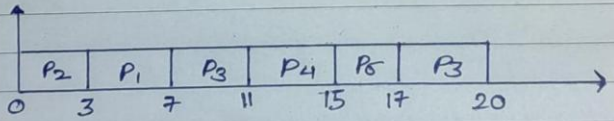
→ Non Preemptive

Process	Priority	AT	BT	CT	TAT	WT
P_1	1	0	4	7	7	3
P_2	2	0	3	3	3	0
P_3	1	6	7	14	8	1
P_4	8	11	4	18	7	3
P_5	2	12	2	20	8	6
				Average →	6.6	2.6



→ Preemptive

Process	Priority	AT	BT	CT	TAT	WT
P_1	1	0	4	7	7	3
P_2	2	0	3	3	3	0
P_3	1	6	7	20	14	7
P_4	3	11	4	15	4	0
P_5	2	12	2	17	5	3
				Average →	6.6	2.6



Date: 22 / 03 / 2025 **Signature of faculty in-charge**