UNIVERSITY OF ASIA PACIFIC

Course Title: Operating System Lab
Course Code: CSE 406

Date of Submission: 15th August,2025

Submitted by:

Name: Md.Sarjil Hasan

Reg: 22101168

Sec : D

Submitted to:

Atia Rahman Orthi

Lecturer

University of Asia Pacific

Problem Statement:

Priority Scheduling Algorithm.

Process ID	Priority	Arrival Time	Burst Time
P1	2	0	11
P2	0	5	28
P3	3	12	2
P4	1	2	10
P5	4	9	16

Completion Time (CT)
Turnaround Time (TAT) = CT – Arrival Time(AT)
Waiting Time (WT) = TAT – Burst Time(BT)

The algorithm should also compute and display the average waiting time and average turnaround time.

In modern operating systems, process scheduling plays a crucial role in improving CPU utilization, response time, throughput, and fairness. **Priority Scheduling** is a CPU scheduling algorithm where each process is assigned a priority value. The CPU is allocated to the process with the highest priority (lower number = higher priority in some conventions). This algorithm can be **preemptive** or **non-preemptive** depending on whether a newly arrived process with a higher priority can preempt the currently running process.

Objective:

To implement the **Priority Scheduling Algorithm** in C++ (or other language used in the lab).

To calculate **Waiting Time (WT)**, **Turnaround Time (TAT)**, and **Average Times**.

Algorithm Steps:

1.Input the number of processes, their burst time (BT), arrival time (AT), and priority (P).

2.At any given time, select the process with the highest priority (lowest value) among the processes that have arrived and are not yet completed.

3.Allocate CPU to that process until completion (non-preemptive) or until a higher-priority process arrives (preemptive).

4.Calculate:

- Completion Time (CT) = Time at which process finishes.
- Turnaround Time (TAT) = CT – AT.
- Waiting Time (WT) = TAT – BT.

5.Compute average WT and TAT.

Code:

Priority.cpp > main()

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    string pid[] = {"p1", "p2", "p3", "p4", "p5"};
    int priority[] = {2, 0, 3, 1, 4};
    int arrival[] = {0, 5, 12, 2, 9};
    int burst[] = {11, 28, 2, 10, 16};

    int n = 5;
    int start[5], finish[5], waiting[5], turnaround[5];
    bool completed[5] = {false};

    int time = 0, done = 0;
    vector<int> order;

    while (done < n)
    {
        int idx = -1;
        int bestPriority = INT_MAX;

        for (int i = 0; i < n; i++)
        {
            if (!completed[i] && arrival[i] <= time)
            {
                if (priority[i] < bestPriority)
                {
                    bestPriority = priority[i];
                    idx = i;
                }
            }
        }

```

```cpp
        if (idx == -1)
        {
            time++;
            continue;
        }

        start[idx] = time;
        finish[idx] = time + burst[idx];
        turnaround[idx] = finish[idx] - arrival[idx];
        waiting[idx] = turnaround[idx] - burst[idx];
        completed[idx] = true;
        time = finish[idx];
        order.push_back(idx);
        done++;
    }

    cout << left << setw(5) << "PID" << setw(10) << "Arrival"
         << setw(10) << "Burst" << setw(10) << "Priority"
         << setw(10) << "Start" << setw(10) << "Finish"
         << setw(12) << "Waiting" << setw(12) << "Turnaround" << "\n";

    double totalWT = 0, totalTAT = 0;
    for (int idx : order)
    {
        cout << left << setw(5) << pid[idx] << setw(10) << arrival[idx]
             << setw(10) << burst[idx] << setw(10) << priority[idx]
             << setw(10) << start[idx] << setw(10) << finish[idx]
             << setw(12) << waiting[idx] << setw(12) << turnaround[idx] << "\n";
        totalWT += waiting[idx];
        totalTAT += turnaround[idx];
    }
```

```cpp
    double totalWT = 0, totalTAT = 0;
    for (int idx : order)
    {
        cout << left << setw(5) << pid[idx] << setw(10) << arrival[idx]
             << setw(10) << burst[idx] << setw(10) << priority[idx]
             << setw(10) << start[idx] << setw(10) << finish[idx]
             << setw(12) << waiting[idx] << setw(12) << turnaround[idx] << "\n";
        totalWT += waiting[idx];
        totalTAT += turnaround[idx];
    }

    cout << "\nAverage Waiting Time: " << totalWT / n;
    cout << "\nAverage Turnaround Time: " << totalTAT / n << "\n";

    return 0;
}
```

Output:

```
67        totalTAT += turnaround[idx];

PROBLEMS    DEBUG CONSOLE    TERMINAL    OUTPUT    PORTS

Average Turnaround Time: 37.8
PS C:\Users\Sarjil\Desktop\lab4> cd "c:\Users\Sarjil\Desktop\lab4\" ; if ($?)
PID  Arrival   Burst     Priority  Start     Finish    Waiting     Turnaround
p1   0         11        2         0         11        0           11
p2   5         28        0         11        39        6           34
p4   2         10        1         39        49        37          47
p3   12        2         3         49        51        37          39
p5   9         16        4         51        67        42          58

Average Waiting Time: 24.4
Average Turnaround Time: 37.8
PS C:\Users\Sarjil\Desktop\lab4>
```

Results & Discussion:

1.Priority scheduling ensures that higher priority processes are executed first.
2.If two processes have the same priority, CPU is usually allocated on a First-Come-First-Served (FCFS) basis.
3.**Problem**: Priority scheduling may lead to starvation of low-priority processes.
4.**Solution**: Aging (gradually increasing priority of waiting processes).

Conclusion:

In this lab, we successfully implemented the **Priority Scheduling Algorithm**.We observed how process priorities affect CPU allocation and calculated **Waiting Time** and **Turnaround Time**.This algorithm is efficient for handling important tasks first but requires aging to prevent starvation.