DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF E&ME, NUST, RAWALPINDI

Operating Systems

Project Report

SUBMITTED TO:

Dr Mehwish Naseer

SUBMITTED BY:

Aleeza Rizwan	**Ibrahim Abdullah**	**Shaheer Afzal**
Reg# 456143	**Reg# 454188**	**Reg# 481560**
CE-45 Syndicate A	**CE-45 Syndicate A**	**CE-45 Syndicate A**

Submission Date: 6th May 2025.

Table of Contents:

Description **1**

Project Overview 1

Key Features 1

Workflow 2

Solution **2**

Code 2

Output 10

Conclusion **10**

1. **Develop a parallel application/Simulation using techniques and tools of the Operating Systems available in modern systems (such as Process Synchronization, Process Scheduling, Deadlock Management, etc.).**

Description of the Project:

Project Overview: This project is a console-based C++ application that simulates five different CPU scheduling algorithms, commonly taught in operating systems:

1. First Come First Serve (FCFS)
2. Shortest Job First (SJF) – Non-Preemptive
3. Shortest Job First (SJF) – Preemptive
4. Priority Scheduling – Non-Preemptive
5. Priority Scheduling – Preemptive

It allows users to input processes and visualize results through:

- A detailed execution table (showing completion, turnaround, and waiting times)
- A Gantt chart that shows execution sequence

.

Key Features:

1. **Multiple Scheduling Algorithms**
   - Supports both preemptive and non-preemptive strategies.
   - Dynamic selection via menu-based interface.

2. **Detailed Output**
   - Prints per-process metrics: arrival time, burst time, completion time, turnaround time, and waiting.
   - Displays a Gantt Chart showing execution timeline.

3. **Object-Oriented Design**
   - Makes the program modular and extensible for adding more algorithms.
   - Uses OOP principles:
     - Base class (SchedulerBase)
     - Derived classes like FCFS, SJF, etc

4. **Randomized and User-Controlled Input**
   - Processes are created with random burst times and sequential arrival times.
   - Priority input is taken only if needed based on algorithm.

Workflow:

1. **User Input Phase**
   - User is asked to enter:
     - Number of processes
     - Scheduling algorithm to simulate
     - (Optionally) priorities for each process

2. **Process Initialization**
   - Each process has:
     - Process ID
     - Random burst times
     - Sequential arrival time
     - User-input priority (if required)

3. **Simulation Phase**
   - The selected algorithm is applied
   - Processes are scheduled and executed.
   - Each algorithm calculates:
     - Completion time
     - Turnaround time = Completion – Arrival
     - Waiting time = Turnaround - Burst

4. **Result Display**
   - A Process Execution Table is printed.
   - A Gantt Chart visually shows the scheduling order and timing.

Solution:

Code: The C++ code for a process scheduler is given below:

```
//program for pre-emptive priority scheduling algorithm

#include <iostream>

#include <iomanip>

#include <string>
```

```
#include <algorithm>
#include <vector>
#include <random>

using namespace std;

struct Process
{
    int processID, arrivalTime, burstTime, priority;
    int remainingTime, completionTime, waitingTime, turnaroundTime;
    bool isCompleted;

    Process(int pid = 0)
    {
        processID = pid;
        arrivalTime = burstTime = priority = remainingTime = 0;
        completionTime = waitingTime = turnaroundTime = 0;
        isCompleted = false;
    }
};

class SchedulerBase
{
    protected:
    Process* processes;
    int n;
    vector<int> ganttStart, ganttPid;

    public:
    SchedulerBase(Process* p, int count)
    {
```

```
        n = count;
        processes = new Process[n];
        for (int i = 0; i < n; ++i) processes[i] = p[i];
    }

    virtual ~SchedulerBase()
    {
        delete[] processes;
    }

    virtual void run() = 0;

    void printTable()
    {
        cout << "\nProcess Execution Table:\n";
        cout << left << setw(12) << "Process ID"
             << setw(15) << "Arrival"
             << setw(15) << "Burst"
             << setw(12) << "Priority"
             << setw(18) << "Completion"
             << setw(18) << "Turnaround"
             << setw(15) << "Waiting" << "\n";
        for (int i = 0; i < n; ++i)
        {
            Process& p = processes[i];
            cout << left << setw(12) << ("P" + to_string(p.processID))
                 << setw(15) << p.arrivalTime
                 << setw(15) << p.burstTime
                 << setw(12) << p.priority
                 << setw(18) << p.completionTime
                 << setw(18) << p.turnaroundTime
```

```
            << setw(15) << p.waitingTime << "\n";
        }
    }

    void printGanttChart()
    {
        cout << "\nGantt Chart:\n ";
        for (size_t i = 0; i < ganttPid.size(); ++i) cout << "---------";
        cout << "\n|";
        for (int pid : ganttPid) cout << "   P" << pid << "   |";
        cout << "\n ";
        for (size_t i = 0; i < ganttPid.size(); ++i) cout << "---------";

        cout << "\n" << setw(9) << ganttStart[0];
        for (size_t i = 0; i < ganttStart.size(); ++i) {
            int next = (i + 1 < ganttStart.size()) ? ganttStart[i + 1] : getMaxCompletion();
            cout << setw(9) << next;
        }
        cout << "\n";
    }

    int getMaxCompletion() const
    {
        int maxCT = 0;
        for (int i = 0; i < n; ++i)
            if (processes[i].completionTime > maxCT)
                maxCT = processes[i].completionTime;
        return maxCT;
    }
};
```

```
class FCFS : public SchedulerBase
{
   public:
   FCFS(Process* p, int n) : SchedulerBase(p, n) {}

   void run() override
   {
      sort(processes, processes + n, [](Process a, Process b)
      {
         return a.arrivalTime < b.arrivalTime;
      });

      int currentTime = 0;
      for (int i = 0; i < n; ++i)
      {
         Process& p = processes[i];
         currentTime = max(currentTime, p.arrivalTime);
         ganttStart.push_back(currentTime);
         ganttPid.push_back(p.processID);

         currentTime += p.burstTime;
         p.completionTime = currentTime;
         p.turnaroundTime = p.completionTime - p.arrivalTime;
         p.waitingTime = p.turnaroundTime - p.burstTime;
      }
   }
};

class SJF : public SchedulerBase
{
   public:
```

```
SJF(Process* p, int n) : SchedulerBase(p, n) {}

void run() override
{
    int currentTime = 0, completed = 0;
    while (completed < n)
    {
        int idx = -1, minBT = 9999;
        for (int i = 0; i < n; ++i)
        {
            Process& p = processes[i];
            if (!p.isCompleted && p.arrivalTime <= currentTime && p.burstTime < minBT)
            {
                minBT = p.burstTime;
                idx = i;
            }
        }
        if (idx == -1)
        {
            currentTime++;
            continue;
        }

        Process& p = processes[idx];
        ganttStart.push_back(currentTime);
        ganttPid.push_back(p.processID);

        currentTime += p.burstTime;
        p.completionTime = currentTime;
        p.turnaroundTime = p.completionTime - p.arrivalTime;
        p.waitingTime = p.turnaroundTime - p.burstTime;
```

```
            p.isCompleted = true;
            completed++;
        }
    }
};

class PreemptiveSJF : public SchedulerBase
{
    public:
    PreemptiveSJF(Process* p, int n) : SchedulerBase(p, n) {}

    void run() override
    {
        int currentTime = 0, completed = 0, lastPid = -1;
        for (int i = 0; i < n; ++i) processes[i].remainingTime = processes[i].burstTime;

        while (completed < n)
        {
            int idx = -1, minRT = 9999;
            for (int i = 0; i < n; ++i)
            {
                Process& p = processes[i];
                if (!p.isCompleted && p.arrivalTime <= currentTime && p.remainingTime <
minRT && p.remainingTime > 0)
                {
                    minRT = p.remainingTime;
                    idx = i;
                }
            }
            if (idx == -1)
            {
```

```
            currentTime++; lastPid = -1;
            continue;
        }

        Process& p = processes[idx];
        if (lastPid != p.processID)
        {
            ganttStart.push_back(currentTime);
            ganttPid.push_back(p.processID);
            lastPid = p.processID;
        }

        p.remainingTime--;
        currentTime++;
        if (p.remainingTime == 0)
        {
            p.completionTime = currentTime;
            p.turnaroundTime = p.completionTime - p.arrivalTime;
            p.waitingTime = p.turnaroundTime - p.burstTime;
            p.isCompleted = true;
            completed++;
        }
    }
  }
};

class PriorityNonPreemptive : public SchedulerBase
{
  public:
  PriorityNonPreemptive(Process* p, int n) : SchedulerBase(p, n) {}
```

```
void run() override
{
    int currentTime = 0, completed = 0;
    while (completed < n)
    {
        int idx = -1, highestPriority = 9999;
        for (int i = 0; i < n; ++i)
        {
            Process& p = processes[i];
                if (!p.isCompleted && p.arrivalTime <= currentTime && p.priority <
highestPriority)
            {
                highestPriority = p.priority;
                idx = i;
            }
        }
        if (idx == -1)
        {
            currentTime++;
            continue;
        }

        Process& p = processes[idx];
        ganttStart.push_back(currentTime);
        ganttPid.push_back(p.processID);


        currentTime += p.burstTime;
        p.completionTime = currentTime;
        p.turnaroundTime = p.completionTime - p.arrivalTime;
        p.waitingTime = p.turnaroundTime - p.burstTime;
        p.isCompleted = true;
```

```
            completed++;
        }
    }
};

class PriorityPreemptive : public SchedulerBase
{
    public:
    PriorityPreemptive(Process* p, int n) : SchedulerBase(p, n) {}

    void run() override
    {
        int currentTime = 0, completed = 0, lastPid = -1;
        for (int i = 0; i < n; ++i) processes[i].remainingTime = processes[i].burstTime;

        while (completed < n)
        {
            int idx = -1, highestPriority = 9999;
            for (int i = 0; i < n; ++i)
            {
                Process& p = processes[i];
                 if (!p.isCompleted && p.arrivalTime <= currentTime && p.remainingTime > 0
&& p.priority < highestPriority)
                {
                    highestPriority = p.priority;
                    idx = i;
                }
            }

            if (idx == -1)
            {
```

```
            currentTime++;
            lastPid = -1;
            continue;
        }

        Process& p = processes[idx];
        if (lastPid != p.processID)
        {
            ganttStart.push_back(currentTime);
            ganttPid.push_back(p.processID);
            lastPid = p.processID;
        }

        p.remainingTime--;
        currentTime++;

        if (p.remainingTime == 0)
        {
            p.completionTime = currentTime;
            p.turnaroundTime = p.completionTime - p.arrivalTime;
            p.waitingTime = p.turnaroundTime - p.burstTime;
            p.isCompleted = true;
            completed++;
        }
      }
    }
};

int main()
{
    int n, choice;
```

```
cout << "Enter number of processes: ";
cin >> n;

cout << "\nSelect Scheduling Algorithm:\n";
cout << "1. FCFS\n2. SJF (Non-Preemptive)\n3. SJF (Preemptive)\n";
cout << "4. Priority (Non-Preemptive)\n5. Priority (Preemptive)\n";
cout << "Enter choice: ";
cin >> choice;

//creating processes
Process* processes = new Process[n];
for (int i = 0; i < n; ++i) {
    processes[i].processID = i + 1;
    processes[i].arrivalTime = i;
    processes[i].burstTime = rand() % 10 + 1;

    cout << "\nFor P" << processes[i].processID << ":\n";
    cout << "Arrival Time: " << processes[i].arrivalTime << endl;
    cout << "Burst Time: " << processes[i].burstTime << endl;

    //asking for priority only if needed
    if (choice == 4 || choice == 5)
    {
        cout << "Priority: ";
        cin >> processes[i].priority;
    }
    else
    {
        processes[i].priority = 0;
    }
}
```

```
    //creating appropriate scheduler
    SchedulerBase* scheduler = nullptr;
    switch (choice)
    {
        case 1: scheduler = new FCFS(processes, n); break;
        case 2: scheduler = new SJF(processes, n); break;
        case 3: scheduler = new PreemptiveSJF(processes, n); break;
        case 4: scheduler = new PriorityNonPreemptive(processes, n); break;
        case 5: scheduler = new PriorityPreemptive(processes, n); break;
        default: cout << "Invalid choice!\n"; delete[] processes; return 0;
    }

    scheduler->run();
    scheduler->printTable();
    scheduler->printGanttChart();

    delete scheduler;
    delete[] processes;

   return 0;
}
```

Output: The output for this code is given below:

```
Enter number of processes: 3

Select Scheduling Algorithm:
1. FCFS
2. SJF (Non-Preemptive)
3. SJF (Preemptive)
4. Priority (Non-Preemptive)
5. Priority (Preemptive)
Enter choice: 1

For P1:
Arrival Time: 0
Burst Time: 2

For P2:
Arrival Time: 1
Burst Time: 8

For P3:
Arrival Time: 2
Burst Time: 5

Process Execution Table:
Process ID  Arrival       Burst         Priority    Completion        Turnaround        Waiting
P1          0             2             0           2                 2                 0
P2          1             8             0           10                9                 1
P3          2             5             0           15                13                8

Gantt Chart:
 ---------------------------
|   P1   |   P2   |   P3   |
 ---------------------------
0        2        10       15
```

First Come First Serve

```
Enter number of processes: 3

Select Scheduling Algorithm:
1. FCFS
2. SJF (Non-Preemptive)
3. SJF (Preemptive)
4. Priority (Non-Preemptive)
5. Priority (Preemptive)
Enter choice: 2

For P1:
Arrival Time: 0
Burst Time: 2

For P2:
Arrival Time: 1
Burst Time: 8

For P3:
Arrival Time: 2
Burst Time: 5

Process Execution Table:
Process ID  Arrival       Burst         Priority    Completion        Turnaround        Waiting
P1          0             2             0           2                 2                 0
P2          1             8             0           15                14                6
P3          2             5             0           7                 5                 0

Gantt Chart:
 ---------------------------
|   P1   |   P3   |   P2   |
 ---------------------------
0        2        7        15
```

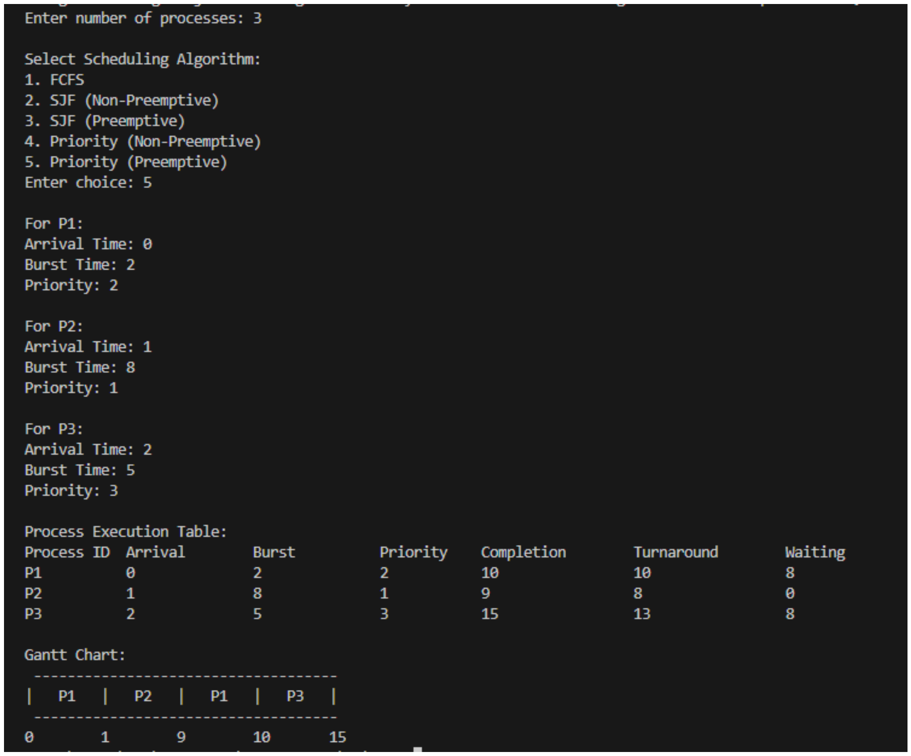
Shortest Job First (Non-Preemptive)

```
Enter number of processes: 3

Select Scheduling Algorithm:
1. FCFS
2. SJF (Non-Preemptive)
3. SJF (Preemptive)
4. Priority (Non-Preemptive)
5. Priority (Preemptive)
Enter choice: 3

For P1:
Arrival Time: 0
Burst Time: 2

For P2:
Arrival Time: 1
Burst Time: 8

For P3:
Arrival Time: 2
Burst Time: 5

Process Execution Table:
Process ID  Arrival       Burst         Priority    Completion        Turnaround        Waiting
P1          0             2             0           2                 2                 0
P2          1             8             0           15                14                6
P3          2             5             0           7                 5                 0

Gantt Chart:
 ---------------------------
|   P1   |   P3   |   P2   |
 ---------------------------
0        2        7        15
```

Shortest Job First (Preemptive)

```
Enter number of processes: 3

Select Scheduling Algorithm:
1. FCFS
2. SJF (Non-Preemptive)
3. SJF (Preemptive)
4. Priority (Non-Preemptive)
5. Priority (Preemptive)
Enter choice: 4

For P1:
Arrival Time: 0
Burst Time: 2
Priority: 3

For P2:
Arrival Time: 1
Burst Time: 8
Priority: 1

For P3:
Arrival Time: 2
Burst Time: 5
Priority: 6

Process Execution Table:
Process ID  Arrival       Burst         Priority    Completion        Turnaround        Waiting
P1          0             2             3           2                 2                 0
P2          1             8             1           10                9                 1
P3          2             5             6           15                13                8

Gantt Chart:
 ---------------------------
|   P1   |   P2   |   P3   |
 ---------------------------
0        2        10       15
```

Priority (Non-Preemptive)



```
Enter number of processes: 3

Select Scheduling Algorithm:
1. FCFS
2. SJF (Non-Preemptive)
3. SJF (Preemptive)
4. Priority (Non-Preemptive)
5. Priority (Preemptive)
Enter choice: 5

For P1:
Arrival Time: 0
Burst Time: 2
Priority: 2

For P2:
Arrival Time: 1
Burst Time: 8
Priority: 1

For P3:
Arrival Time: 2
Burst Time: 5
Priority: 3

Process Execution Table:
Process ID  Arrival       Burst         Priority    Completion       Turnaround       Waiting
P1          0             2             2           10               10               8
P2          1             8             1           9                8                0
P3          2             5             3           15               13               8

Gantt Chart:
 -----------------------------------
|  P1   |  P2   |  P1   |  P3   |
 -----------------------------------
0       1       9       10      15
```

Priority (Preemptive)

<u>Conclusion:</u>

This project effectively demonstrates how different CPU scheduling algorithms behave under various conditions. It provides both visual clarity and accurate metric computation, making it a useful tool for students learning operating systems.

By using object-oriented programming, it ensures:

- Clean separation between logic
- Easier debugging and expansion
- Reusability of components

The code is well-structured for enhancements like:

- Adding Round Robin algorithm
- Supporting I/O-bound processes
- GUI-based visualization (future scope)