EXPERIMENT-3

Aim:

Write a program to perform priority scheduling.

Theory:

Code:

```
#include <bits/stdc++.h>

using namespace std;

struct Process {
    int id;
    int arrival_time;
    int burst_time;
    int priority;
    int completion_time;
    int turnaround_time;
    int waiting_time;
};

bool comparePriority(const Process& p1, const Process& p2) {
    return p1.priority < p2.priority;
}

void nonPreemptivePriorityScheduling(vector<Process>& processes) {
    int n = processes.size();

    sort(processes.begin(), processes.end(), comparePriority);

    int current_time = 0;
    for (int i = 0; i < n; i++) {
        processes[i].completion_time = current_time + processes[i].burst_time;
        processes[i].turnaround_time = processes[i].completion_time -
processes[i].arrival_time;
        processes[i].waiting_time = max(0, processes[i].turnaround_time -
processes[i].burst_time);

        current_time = processes[i].completion_time;
    }
}

void preemptivePriorityScheduling(vector<Process>& processes) {
    int n = processes.size();

    int current_time = 0;
    vector<bool> completed(n, false);

    // Creating a copy of burst time for each process
    vector<int> burst_times(n);
    for (int i = 0; i < n; i++) {
        burst_times[i] = processes[i].burst_time;
    }
```

```
    while (true) {
        int highest_priority = INT_MAX;
        int selected_process = -1;

        for (int i = 0; i < n; i++) {
            if (!completed[i] && processes[i].arrival_time <= current_time &&
processes[i].priority < highest_priority) {
                highest_priority = processes[i].priority;
                selected_process = i;
            }
        }

        if (selected_process == -1) {
            break;
        }

        processes[selected_process].completion_time = current_time + 1;
        burst_times[selected_process]--;

        if (burst_times[selected_process] == 0) {
            completed[selected_process] = true;
            processes[selected_process].turnaround_time =
processes[selected_process].completion_time -
processes[selected_process].arrival_time;
            processes[selected_process].waiting_time =
processes[selected_process].turnaround_time -
processes[selected_process].burst_time;
        }

        current_time++;
    }
}

void displayResult(const vector<Process>& processes) {
    cout << setw(10) << "Process" << setw(15) << "Arrival Time" << setw(15) <<
"Burst Time" << setw(15) << "Priority" << setw(15) << "Completion Time" <<
setw(15) << "Turnaround Time" << setw(15) << "Waiting Time" << endl;

    double total_waiting_time = 0;
    double total_turnaround_time = 0;

    for (const Process& p : processes) {
        cout << setw(10) << "P" << p.id << setw(15) << p.arrival_time <<
setw(15) << p.burst_time << setw(15) << p.priority << setw(15) <<
p.completion_time << setw(15) << p.turnaround_time << setw(15) <<
p.waiting_time << endl;

        total_waiting_time += p.waiting_time;
```

```
        total_turnaround_time += p.turnaround_time;
    }

    double avg_turnaround_time = total_turnaround_time / processes.size();
    double avg_waiting_time = total_waiting_time / processes.size();
    cout << "Average Turnaround Time: " << avg_turnaround_time << endl;
    cout << "Average Waiting Time: " << avg_waiting_time << endl;
}

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

    vector<Process> processes(n);

    cout << "Enter the arrival time, burst time, and priority for each
process:" << endl;
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        cout << "Process " << i + 1 << ":" << endl;
        cout << "Arrival Time: ";
        cin >> processes[i].arrival_time;
        cout << "Burst Time: ";
        cin >> processes[i].burst_time;
        cout << "Priority: ";
        cin >> processes[i].priority;
        cout << endl;
    }

    vector<Process> processesCopy;
    processesCopy = processes;

    // Performing preemptive priority scheduling
    preemptivePriorityScheduling(processes);

    cout << "Preemptive Priority Scheduling:" << endl;
    displayResult(processes);

    // Performing non-preemptive priority scheduling
    nonPreemptivePriorityScheduling(processesCopy);

    cout << "Non-Preemptive Priority Scheduling:" << endl;
    displayResult(processesCopy);

    return 0;
}
```

Output:

```
Enter the number of processes: 5
Enter the arrival time, burst time, and priority for each process:
Process 1:
Arrival Time: 0
Burst Time: 3
Priority: 3
Process 2:
Arrival Time: 1
Burst Time: 4
Priority: 2
Process 3:
Arrival Time: 2
Burst Time: 6
Priority: 4
Process 4:
Arrival Time: 3
Burst Time: 4
Priority: 6
Process 5:
Arrival Time: 5
Burst Time: 2
Priority: 10
Preemptive Priority Scheduling:
   Process   Arrival Time     Burst Time       PriorityCompletion TimeTurnaround Time   Waiting Time
        P1              0              3              3              7              7              4
        P2              1              4              2              5              4              0
        P3              2              6              4             13             11              5
        P4              3              4              6             17             14             10
        P5              5              2             10             19             14             12
Average Turnaround Time: 10
Average Waiting Time: 6.2
Non-Preemptive Priority Scheduling:
   Process   Arrival Time     Burst Time       PriorityCompletion TimeTurnaround Time   Waiting Time
        P2              1              4              2              4              3              0
        P1              0              3              3              7              7              4
        P3              2              6              4             13             11              5
P4              3              4              6             17             14             10
        P5              5              2             10             19             14             12
Average Turnaround Time: 9.8
Average Waiting Time: 6.2
```

EXPERIMENT-4

Aim:

Write a program to implement CPU scheduling for Round Robin.

Theory:

Code:

```
#include <bits/stdc++.h>

using namespace std;

struct Process {
    int id;
    int burst_time;
    int arrival_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
};

void roundRobin(vector<Process>& processes, int time_quantum) {
    int n = processes.size();

    vector<int> remaining_time(n);
    for (int i = 0; i < n; i++) {
        remaining_time[i] = processes[i].burst_time;
    }

    int current_time = 0;
    int completed = 0;
    while (completed < n) {
        for (int i = 0; i < n; i++) {
            if (remaining_time[i] > 0) {
                int execute_time = min(remaining_time[i], time_quantum);
                remaining_time[i] -= execute_time;
                current_time += execute_time;

                if (remaining_time[i] == 0) {
                    processes[i].completion_time = current_time;
                    completed++;
                }
            }
        }
    }

    for (int i = 0; i < n; i++) {
        processes[i].turnaround_time = processes[i].completion_time -
processes[i].arrival_time;
        processes[i].waiting_time = processes[i].turnaround_time -
processes[i].burst_time;
    }
}

void displayResult(const vector<Process>& processes) {
```

```
    cout << "Process\tBurst Time\tArrival Time\tCompletion Time\tTurnaround
Time\tWaiting Time" << endl;
    for (const Process& p : processes) {
        cout << p.id << "\t" << p.burst_time << "\t\t" << p.arrival_time <<
"\t\t" << p.completion_time << "\t\t" << p.turnaround_time << "\t\t" <<
p.waiting_time << endl;
    }

    double total_turnaround_time = 0;
    double total_waiting_time = 0;
    for (const Process& p : processes) {
        total_turnaround_time += p.turnaround_time;
        total_waiting_time += p.waiting_time;
    }

    double avg_turnaround_time = total_turnaround_time / processes.size();
    double avg_waiting_time = total_waiting_time / processes.size();
    cout << "Average Turnaround Time: " << avg_turnaround_time << endl;
    cout << "Average Waiting Time: " << avg_waiting_time << endl;
}

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

    vector<Process> processes(n);

    cout << "Enter the arrival time and burst time for each process:" << endl;
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        cout << "Process " << i + 1 << ":" << endl;
        cout << "Arrival Time: ";
        cin >> processes[i].arrival_time;
        cout << "Burst Time: ";
        cin >> processes[i].burst_time;
        cout << endl;
    }

    int time_quantum;
    cout << "Enter the time quantum: ";
    cin >> time_quantum;

    roundRobin(processes, time_quantum);
    displayResult(processes);

    return 0;
}
```

Output:

```
Enter the number of processes: 6
Enter the arrival time and burst time for each process:
Process 1:
Arrival Time: 0
Burst Time: 7

Process 2:
Arrival Time: 1
Burst Time: 4

Process 3:
Arrival Time: 2
Burst Time: 15

Process 4:
Arrival Time: 3
Burst Time: 11

Process 5:
Arrival Time: 4
Burst Time: 20

Process 6:
Arrival Time: 4
Burst Time: 9

Enter the time quantum: 5
Process Burst Time      Arrival Time    Completion Time Turnaround Time Waiting Time
1       7               0               31              31              24
2       4               1               9               8               4
3       15              2               55              53              38
4       11              3               56              53              42
5       20              4               66              62              42
6       9               4               50              46              37
Average Turnaround Time: 42.1667
Average Waiting Time: 31.1667
```