Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Technology, Pune-37

(An autonomous institute of Savitribai Phule Pune University)

OS Lab 4

Year	Third
Branch	AIDS
Division	AI-A
Batch	3
PRN	12320083
Roll no.	69
Name	Durvesh Chaudhari

PROBLEM STATEMENT

Write a program to compute the finish time, turnaround time, and waiting time for the following algorithms:
a) First come First serve
b) Shortest Job First (Preemptive and Non Preemptive)
c) Priority (Preemptive and Non Preemptive)
d) Round robin

Code:-

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>

#define MAX 10

struct Process {
    int pid;       // Process ID
    int burst;     // Burst Time
    int arrival;   // Arrival Time
    int priority; // Priority
    int waiting;   // Waiting Time
    int turnaround; // Turnaround Time
    int remaining; // Remaining Time
};

// Function to display Gantt Chart
void printGanttChart(int gantt[], int time[], int n) {
    printf("Gantt Chart:\n|");
    for (int i = 0; i < n; i++) {
        printf(" P%d |", gantt[i]);
    }
    printf("\n");
    for (int i = 0; i <= n; i++) {
        printf("%d\t", time[i]);
    }
    printf("\n");
}

void fcfs(struct Process p[], int n) {
    int wait_sum = 0, turnaround_sum = 0;
    p[0].waiting = 0;

    int gantt[MAX], time[MAX + 1];
    int current_time = 0;

    for (int i = 0; i < n; i++) {
        gantt[i] = p[i].pid;
        time[i] = current_time;
```

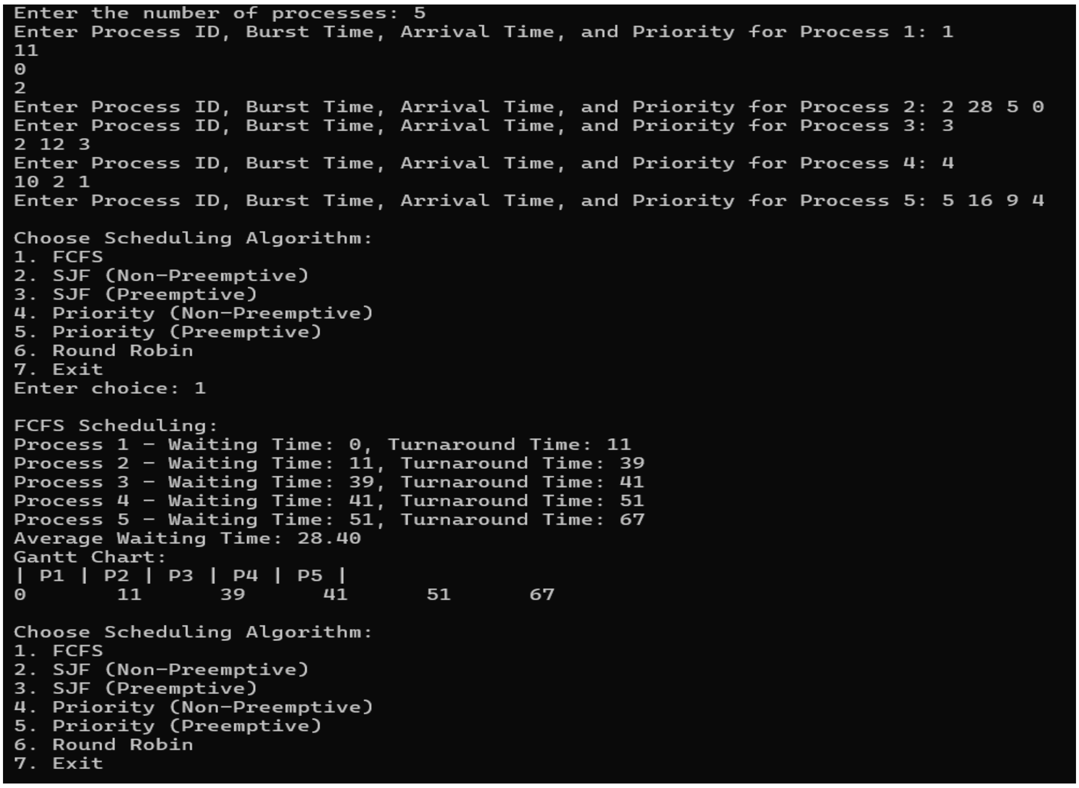
```
     if (i != 0)
            p[i].waiting = p[i - 1].waiting + p[i - 1].burst;
        wait_sum += p[i].waiting;
        current_time += p[i].burst;
    }
    time[n] = current_time;

    for (int i = 0; i < n; i++) {
        p[i].turnaround = p[i].waiting + p[i].burst;
        turnaround_sum += p[i].turnaround;
    }

    printf("\nFCFS Scheduling:\n");
    for (int i = 0; i < n; i++)
        printf("Process %d - Waiting Time: %d, Turnaround Time:
%d\n", p[i].pid, p[i].waiting, p[i].turnaround);

    printf("Average Waiting Time: %.2f\n", (float)wait_sum / n);
    printGanttChart(gantt, time, n);
}

void sjf_non_preemptive(struct Process p[], int n) {
    int wait_sum = 0, turnaround_sum = 0;
    bool completed[MAX] = { false };
    int current_time = 0, completed_count = 0;

    int gantt[MAX], time[MAX + 1];
    int g_idx = 0;

    while (completed_count != n) {
        int idx = -1, min_burst = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (p[i].arrival <= current_time && !completed[i] &&
p[i].burst < min_burst) {
                min_burst = p[i].burst;
                idx = i;
            }
        }

        if (idx == -1) {
            current_time++;
        } else {
            p[idx].waiting = current_time - p[idx].arrival;
            wait_sum += p[idx].waiting;
            gantt[g_idx] = p[idx].pid;
            time[g_idx] = current_time;
            g_idx++;
            current_time += p[idx].burst;
            p[idx].turnaround = p[idx].waiting + p[idx].burst;
            turnaround_sum += p[idx].turnaround;
            completed[idx] = true;
            completed_count++;
        }
    }
    time[g_idx] = current_time;

    printf("\nNon-Preemptive SJF Scheduling:\n");
    for (int i = 0; i < n; i++)
        printf("Process %d - Waiting Time: %d, Turnaround Time: 
%d\n", p[i].pid, p[i].waiting, p[i].turnaround);

    printf("Average Waiting Time: %.2f\n", (float)wait_sum / n);
    printGanttChart(gantt, time, g_idx);
}

void sjf_preemptive(struct Process p[], int n) {
    int wait_sum = 0, turnaround_sum = 0;
    int current_time = 0, completed_count = 0;
    int min_burst_idx;
    int min_remaining_time;

    int gantt[MAX * 2], time[MAX * 2 + 1];
    int g_idx = 0;

    while (completed_count < n) {
        min_remaining_time = INT_MAX;
        min_burst_idx = -1;
        for (int i = 0; i < n; i++) {
            if (p[i].arrival <= current_time && p[i].remaining > 
0 && p[i].remaining < min_remaining_time) {
                min_remaining_time = p[i].remaining;
                min_burst_idx = i;
            }
        }

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

        if (g_idx == 0 || gantt[g_idx - 1] != 
p[min_burst_idx].pid) {
            gantt[g_idx] = p[min_burst_idx].pid;
            time[g_idx] = current_time;
            g_idx++;
        }

        p[min_burst_idx].remaining--;
        current_time++;
```

```
          if (p[min_burst_idx].remaining == 0) {
            completed_count++;
            p[min_burst_idx].turnaround = current_time -
p[min_burst_idx].arrival;
            p[min_burst_idx].waiting =
p[min_burst_idx].turnaround - p[min_burst_idx].burst;
            wait_sum += p[min_burst_idx].waiting;
            turnaround_sum += p[min_burst_idx].turnaround;
        }
    }
    time[g_idx] = current_time;

    printf("\nPreemptive SJF Scheduling:\n");
    for (int i = 0; i < n; i++)
        printf("Process %d - Waiting Time: %d, Turnaround Time:
%d\n", p[i].pid, p[i].waiting, p[i].turnaround);

    printf("Average Waiting Time: %.2f\n", (float)wait_sum / n);
    printGanttChart(gantt, time, g_idx);
}

void priority_non_preemptive(struct Process p[], int n) {
    int wait_sum = 0, turnaround_sum = 0;
    bool completed[MAX] = { false };
    int current_time = 0, completed_count = 0;

    int gantt[MAX], time[MAX + 1];
    int g_idx = 0;

    while (completed_count != n) {
        int idx = -1, highest_priority = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (p[i].arrival <= current_time && !completed[i] &&
p[i].priority < highest_priority) {
                highest_priority = p[i].priority;
                idx = i;
            }
        }

        if (idx == -1) {
            current_time++;
        } else {
            p[idx].waiting = current_time - p[idx].arrival;
            wait_sum += p[idx].waiting;
            gantt[g_idx] = p[idx].pid;
            time[g_idx] = current_time;
            g_idx++;
            current_time += p[idx].burst;
            p[idx].turnaround = p[idx].waiting + p[idx].burst;
            turnaround_sum += p[idx].turnaround;
            completed[idx] = true;
            completed_count++;
        }
    }
    time[g_idx] = current_time;

    printf("\nNon-Preemptive Priority Scheduling:\n");
    for (int i = 0; i < n; i++)
        printf("Process %d - Waiting Time: %d, Turnaround Time:
%d\n", p[i].pid, p[i].waiting, p[i].turnaround);

    printf("Average Waiting Time: %.2f\n", (float)wait_sum / n);
    printGanttChart(gantt, time, g_idx);
}

void priority_preemptive(struct Process p[], int n) {
    int wait_sum = 0, turnaround_sum = 0;
    int current_time = 0, completed_count = 0;

    int gantt[MAX * 2], time[MAX * 2 + 1];
    int g_idx = 0;

    while (completed_count < n) {
        int idx = -1, highest_priority = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (p[i].arrival <= current_time && p[i].remaining >
0 && p[i].priority < highest_priority) {
                highest_priority = p[i].priority;
                idx = i;
            }
        }

        if (idx == -1) {
            current_time++;
        } else {
            if (g_idx == 0 || gantt[g_idx - 1] != p[idx].pid) {
                gantt[g_idx] = p[idx].pid;
                time[g_idx] = current_time;
                g_idx++;
            }

            p[idx].remaining--;
            current_time++;

            if (p[idx].remaining == 0) {
                completed_count++;
                p[idx].turnaround = current_time -
p[idx].arrival;
                p[idx].waiting = p[idx].turnaround -
p[idx].burst;
```

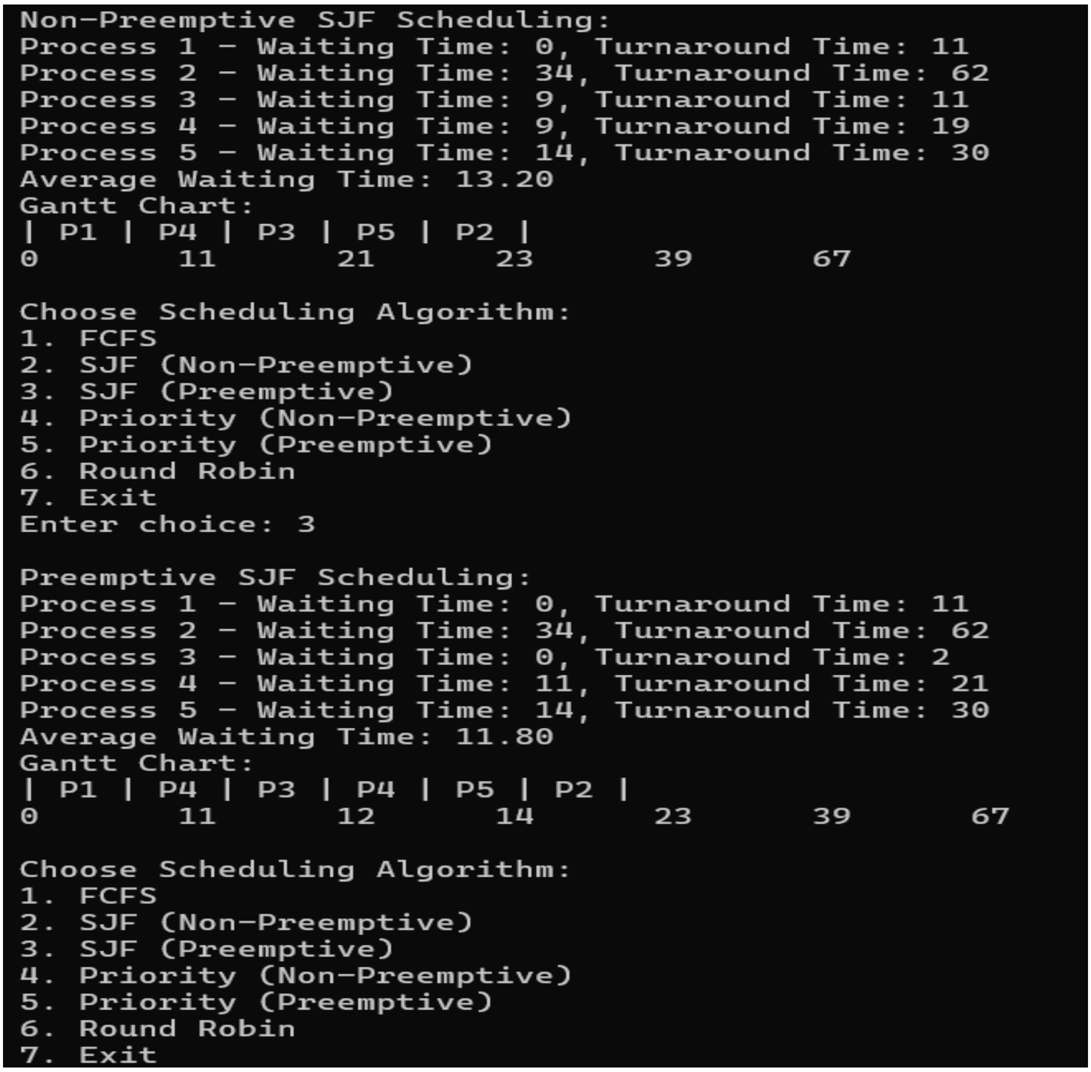
```
                wait_sum += p[idx].waiting;
                turnaround_sum += p[idx].turnaround;
            }
        }
    }
    time[g_idx] = current_time;

    printf("\nPreemptive Priority Scheduling:\n");
    for (int i = 0; i < n; i++)
        printf("Process %d - Waiting Time: %d, Turnaround Time:
%d\n", p[i].pid, p[i].waiting, p[i].turnaround);

    printf("Average Waiting Time: %.2f\n", (float)wait_sum / n);
    printGanttChart(gantt, time, g_idx);
}

// Function to reset remaining times
void reset_processes(struct Process p[], int n) {
    for (int i = 0; i < n; i++)
        p[i].remaining = p[i].burst;
}

void round_robin(struct Process p[], int n, int quantum) {
    int wait_sum = 0, turnaround_sum = 0;
    int remaining[MAX];
    int current_time = 0;
    int gantt[MAX * 2], time[MAX * 2 + 1];
    int g_idx = 0;
    bool done;

    // Initialize remaining burst times
    for (int i = 0; i < n; i++) {
        remaining[i] = p[i].burst;
    }

    while (1) {
        done = true;

        for (int i = 0; i < n; i++) {
            if (remaining[i] > 0) {
                done = false; // There is a pending process

                if (remaining[i] > quantum) {
                    current_time += quantum;
                    remaining[i] -= quantum;
                } else {
                    current_time += remaining[i];
                    p[i].waiting = current_time - p[i].burst -
p[i].arrival;
                    remaining[i] = 0;
```

```
               }

                if (g_idx == 0 || gantt[g_idx - 1] != p[i].pid) {
                    gantt[g_idx] = p[i].pid;
                    time[g_idx] = current_time - remaining[i];
                    g_idx++;
                }
            }
        }

        if (done) break;
    }

    // Calculate turnaround time and total waiting time
    for (int i = 0; i < n; i++) {
        p[i].turnaround = p[i].burst + p[i].waiting;
        wait_sum += p[i].waiting;
        turnaround_sum += p[i].turnaround;
    }

    time[g_idx] = current_time;

    printf("\nRound Robin Scheduling (Quantum = %d):\n",
quantum);
    for (int i = 0; i < n; i++) {
        printf("Process %d - Waiting Time: %d, Turnaround Time:
%d\n", p[i].pid, p[i].waiting, p[i].turnaround);
    }

    printf("Average Waiting Time: %.2f\n", (float)wait_sum / n);
    printGanttChart(gantt, time, g_idx);
}

int main() {
    int n, choice, quantum;
    struct Process p[MAX];

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        printf("Enter Process ID, Burst Time, Arrival Time, and
Priority for Process %d: ", i + 1);
        scanf("%d %d %d %d", &p[i].pid, &p[i].burst,
&p[i].arrival, &p[i].priority);
        p[i].remaining = p[i].burst;
    }

    do {
      printf("\nChoose Scheduling Algorithm:\n1. FCFS\n2. SJF
(Non-Preemptive)\n3. SJF (Preemptive)\n4. Priority
(Non-Preemptive)\n5. Priority (Preemptive)\n6. Round Robin\n7.
Exit\nEnter choice: ");
        scanf("%d", &choice);

        reset_processes(p, n);  // Reset remaining times for all
processes

        switch (choice) {
            case 1: fcfs(p, n); break;
            case 2: sjf_non_preemptive(p, n); break;
            case 3: sjf_preemptive(p, n); break;
            case 4: priority_non_preemptive(p, n); break;
            case 5: priority_preemptive(p, n); break;
            case 6:
                printf("Enter the quantum time: ");
                scanf("%d", &quantum);
                round_robin(p, n, quantum);
                break;
            case 7: printf("Exiting...\n"); break;
            default: printf("Invalid choice! Please try
again.\n");
        }
    } while (choice != 7);

    return 0;
}
```

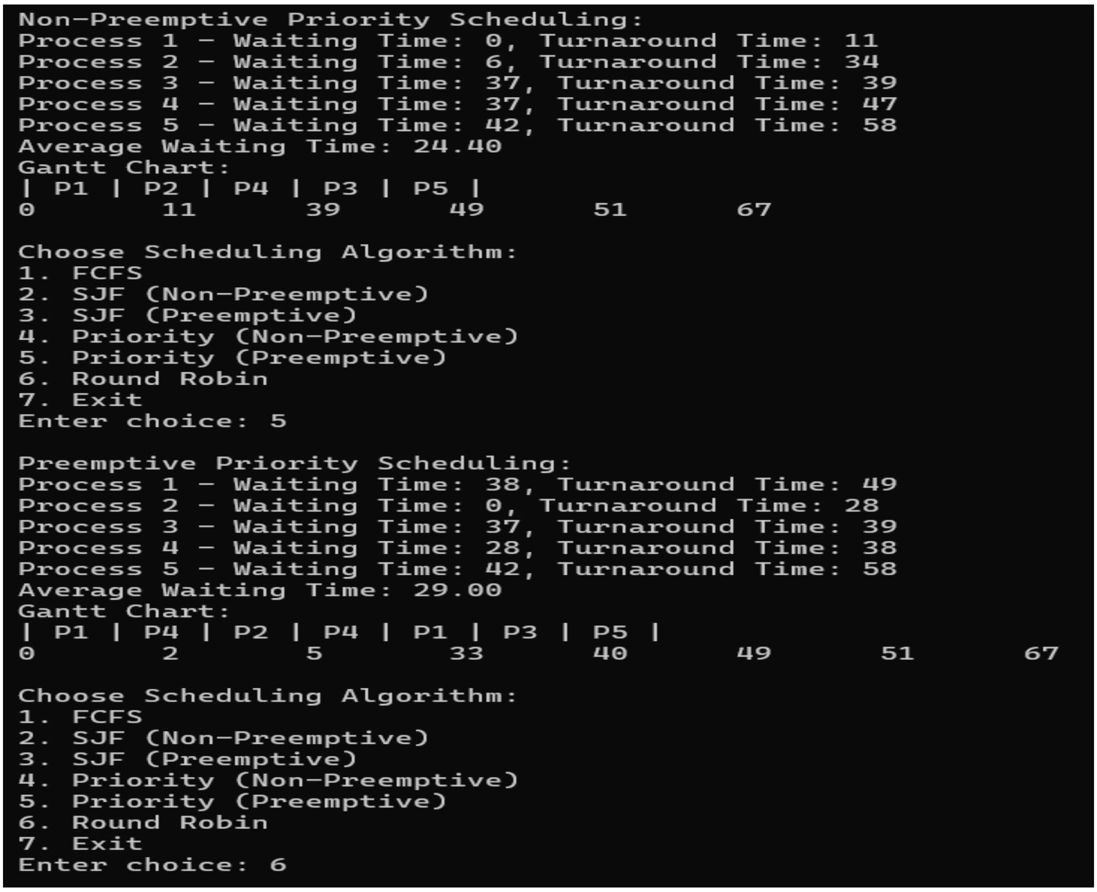
Output:



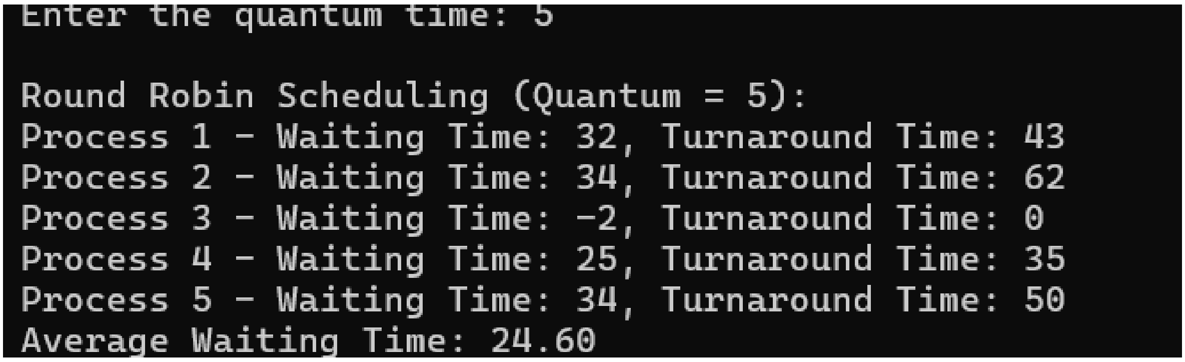
```
Enter the number of processes: 5
Enter Process ID, Burst Time, Arrival Time, and Priority for Process 1: 1
11
0
2
Enter Process ID, Burst Time, Arrival Time, and Priority for Process 2: 2 28 5 0
Enter Process ID, Burst Time, Arrival Time, and Priority for Process 3: 3
2 12 3
Enter Process ID, Burst Time, Arrival Time, and Priority for Process 4: 4
10 2 1
Enter Process ID, Burst Time, Arrival Time, and Priority for Process 5: 5 16 9 4

Choose Scheduling Algorithm:
1. FCFS
2. SJF (Non-Preemptive)
3. SJF (Preemptive)
4. Priority (Non-Preemptive)
5. Priority (Preemptive)
6. Round Robin
7. Exit
Enter choice: 1

FCFS Scheduling:
Process 1 - Waiting Time: 0, Turnaround Time: 11
Process 2 - Waiting Time: 11, Turnaround Time: 39
Process 3 - Waiting Time: 39, Turnaround Time: 41
Process 4 - Waiting Time: 41, Turnaround Time: 51
Process 5 - Waiting Time: 51, Turnaround Time: 67
Average Waiting Time: 28.40
Gantt Chart:
| P1 | P2 | P3 | P4 | P5 |
0       11      39      41      51      67

Choose Scheduling Algorithm:
1. FCFS
2. SJF (Non-Preemptive)
3. SJF (Preemptive)
4. Priority (Non-Preemptive)
5. Priority (Preemptive)
6. Round Robin
7. Exit
```



```
Non-Preemptive SJF Scheduling:
Process 1 - Waiting Time: 0, Turnaround Time: 11
Process 2 - Waiting Time: 34, Turnaround Time: 62
Process 3 - Waiting Time: 9, Turnaround Time: 11
Process 4 - Waiting Time: 9, Turnaround Time: 19
Process 5 - Waiting Time: 14, Turnaround Time: 30
Average Waiting Time: 13.20
Gantt Chart:
| P1 | P4 | P3 | P5 | P2 |
0       11      21      23      39      67

Choose Scheduling Algorithm:
1. FCFS
2. SJF (Non-Preemptive)
3. SJF (Preemptive)
4. Priority (Non-Preemptive)
5. Priority (Preemptive)
6. Round Robin
7. Exit
Enter choice: 3

Preemptive SJF Scheduling:
Process 1 - Waiting Time: 0, Turnaround Time: 11
Process 2 - Waiting Time: 34, Turnaround Time: 62
Process 3 - Waiting Time: 0, Turnaround Time: 2
Process 4 - Waiting Time: 11, Turnaround Time: 21
Process 5 - Waiting Time: 14, Turnaround Time: 30
Average Waiting Time: 11.80
Gantt Chart:
| P1 | P4 | P3 | P4 | P5 | P2 |
0       11      12      14      23      39      67

Choose Scheduling Algorithm:
1. FCFS
2. SJF (Non-Preemptive)
3. SJF (Preemptive)
4. Priority (Non-Preemptive)
5. Priority (Preemptive)
6. Round Robin
7. Exit
```



```
Non-Preemptive Priority Scheduling:
Process 1 - Waiting Time: 0, Turnaround Time: 11
Process 2 - Waiting Time: 6, Turnaround Time: 34
Process 3 - Waiting Time: 37, Turnaround Time: 39
Process 4 - Waiting Time: 37, Turnaround Time: 47
Process 5 - Waiting Time: 42, Turnaround Time: 58
Average Waiting Time: 24.40
Gantt Chart:
| P1 | P2 | P4 | P3 | P5 |
0       11      39      49      51      67

Choose Scheduling Algorithm:
1. FCFS
2. SJF (Non-Preemptive)
3. SJF (Preemptive)
4. Priority (Non-Preemptive)
5. Priority (Preemptive)
6. Round Robin
7. Exit
Enter choice: 5

Preemptive Priority Scheduling:
Process 1 - Waiting Time: 38, Turnaround Time: 49
Process 2 - Waiting Time: 0, Turnaround Time: 28
Process 3 - Waiting Time: 37, Turnaround Time: 39
Process 4 - Waiting Time: 28, Turnaround Time: 38
Process 5 - Waiting Time: 42, Turnaround Time: 58
Average Waiting Time: 29.00
Gantt Chart:
| P1 | P4 | P2 | P4 | P1 | P3 | P5 |
0       2       5       33      40      49      51      67

Choose Scheduling Algorithm:
1. FCFS
2. SJF (Non-Preemptive)
3. SJF (Preemptive)
4. Priority (Non-Preemptive)
5. Priority (Preemptive)
6. Round Robin
7. Exit
Enter choice: 6
```



```
Enter the quantum time: 5

Round Robin Scheduling (Quantum = 5):
Process 1 - Waiting Time: 32, Turnaround Time: 43
Process 2 - Waiting Time: 34, Turnaround Time: 62
Process 3 - Waiting Time: -2, Turnaround Time: 0
Process 4 - Waiting Time: 25, Turnaround Time: 35
Process 5 - Waiting Time: 34, Turnaround Time: 50
Average Waiting Time: 24.60
```