Assignment 2

Name : Sujal Santosh Porte
PRN : 122B1B227

Code :

```
#include  <stdio.h>

#include <limits.h>

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

// Function to display Gantt Chart without repeating consecutive processes
void printGanttChart(int timeline[], int count) {
    printf("\nGantt Chart:\n|");
    int prev = -1;

    for (int i = 0; i < count; i++) {
        if (timeline[i] != prev) {

            printf(" P%d |", timeline[i]);
            prev = timeline[i];

        }
    }
    printf("\n");
}

// Function to print process table and averages
void printTable(struct Process processes[], int n) {
```

```
    int total_waiting = 0, total_turnaround = 0;
    printf("\n%-5s %-8s %-8s %-12s %-10s %-12s\n", "ID", "Arrival", "Burst", "Completion",
"Waiting", "Turnaround");

    for (int i = 0; i < n; i++) {

        printf("%-5d %-8d %-8d %-12d %-10d %-12d\n",


            processes[i].id, processes[i].arrival_time, processes[i].burst_time,
            processes[i].completion_time, processes[i].waiting_time,
processes[i].turnaround_time);

        total_waiting += processes[i].waiting_time;
        total_turnaround += processes[i].turnaround_time;
    }

    printf("\nAverage Waiting Time: %.2f\n", (float)total_waiting / n);
    printf("Average Turnaround Time: %.2f\n", (float)total_turnaround / n);

}

// First Come First Serve (FCFS) Scheduling
void FCFS(struct Process processes[], int n) {
    int time = 0, timeline[n], count = 0;
    for (int i = 0; i < n; i++) {
        if (time < processes[i].arrival_time)
            time = processes[i].arrival_time;


        timeline[count++] = processes[i].id;
        processes[i].completion_time = time + processes[i].burst_time;

        processes[i].turnaround_time = processes[i].completion_time - processes[i].arrival_time;
        processes[i].waiting_time = processes[i].turnaround_time - processes[i].burst_time;
        time = processes[i].completion_time;

    }
    printf("\nFCFS Scheduling:\n");

    printTable(processes, n);

    printGanttChart(timeline, count);
}
```

```
// Shortest Job First (SJF) Scheduling

void SJF(struct Process processes[], int n, int preemptive) {
    int time = 0, completed = 0, timeline[100], count = 0;

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

    while (completed != n) {

        int min_burst = INT_MAX, shortest = -1;
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= time && processes[i].remaining_time > 0 &&
processes[i].remaining_time < min_burst) {
                min_burst = processes[i].remaining_time;
                shortest = i;

            }
        }

        if (shortest == -1) {

            time++;
            Continue;

        }

        if (count == 0 || timeline[count - 1] != processes[shortest].id) {
            timeline[count++] = processes[shortest].id;
        }

        if (preemptive) {

            processes[shortest].remaining_time--;
            time++;
            if (processes[shortest].remaining_time == 0) {
                processes[shortest].completion_time = time;
                processes[shortest].turnaround_time = processes[shortest].completion_time -
processes[shortest].arrival_time;
                processes[shortest].waiting_time = processes[shortest].turnaround_time -
processes[shortest].burst_time;
                completed++;
            }
```

```
        } else {
            time += processes[shortest].remaining_time;
            processes[shortest].remaining_time = 0;

            processes[shortest].completion_time = time;

            processes[shortest].turnaround_time = processes[shortest].completion_time -
processes[shortest].arrival_time;
            processes[shortest].waiting_time = processes[shortest].turnaround_time -
processes[shortest].burst_time;
            completed++;
        }
    }

    printf("\nSJF Scheduling (%s):\n", preemptive ? "Preemptive" : "Non-Preemptive");

    printTable(processes, n);

    printGanttChart(timeline, count);
}

// Round Robin Scheduling
void RoundRobin(struct Process processes[], int n, int quantum) {
    int time = 0, remaining = n, i = 0, timeline[100], count = 0;
    for (int i = 0; i < n; i++)
        processes[i].remaining_time = processes[i].burst_time;

    while (remaining > 0) {
        if (processes[i].remaining_time > 0) {

            if (count == 0 || timeline[count - 1] != processes[i].id) {
                timeline[count++] = processes[i].id;

            }
            int exec = (processes[i].remaining_time > quantum) ? quantum :
processes[i].remaining_time;

            processes[i].remaining_time -= exec;
            time += exec;

            if (processes[i].remaining_time == 0) {
                remaining--;
                processes[i].completion_time = time;
```

```
            processes[i].turnaround_time = processes[i].completion_time -
processes[i].arrival_time;
            processes[i].waiting_time = processes[i].turnaround_time - processes[i].burst_time;
        }
    }
    i = (i + 1) % n;
  }

  printf("\nRound Robin Scheduling (Quantum = %d):\n", quantum);
  printTable(processes, n);
  printGanttChart(timeline, count);
}

// Priority Scheduling
void PriorityScheduling(struct Process processes[], int n, int preemptive) {

  int time = 0, completed = 0, timeline[100], count = 0;

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

  while (completed != n) {

    int min_priority = INT_MAX, highest_priority = -1;

    for (int i = 0; i < n; i++) {
      if (processes[i].arrival_time <= time && processes[i].remaining_time > 0 &&
processes[i].priority < min_priority) {

        min_priority = processes[i].priority;
        highest_priority = i;

      }
    }

    if (highest_priority == -1) {

      time++;
      continue;

    }

    if (count == 0 || timeline[count - 1] != processes[highest_priority].id) {
```

```
            timeline[count++] = processes[highest_priority].id;

        }

        if (preemptive) {
            Processes[highest_priority].remaining_time--;

            time++;
            if (processes[highest_priority].remaining_time == 0) {

                processes[highest_priority].completion_time = time;
                processes[highest_priority].turnaround_time =
processes[highest_priority].completion_time - processes[highest_priority].arrival_time;

                processes[highest_priority].waiting_time =
processes[highest_priority].turnaround_time - processes[highest_priority].burst_time;
                completed++;
            }
        } else {

time += processes[highest_priority].remaining_time;
            processes[highest_priority].remaining_time = 0
;
            processes[highest_priority].completion_time = time;
            processes[highest_priority].turnaround_time =
processes[highest_priority].completion_time - processes[highest_priority].arrival_time;

            processes[highest_priority].waiting_time = processes[highest_priority].turnaround_time -
processes[highest_priority].burst_time;

            completed++;
        }
    }

    printf("\nPriority Scheduling (%s):\n", preemptive ? "Preemptive" : "Non-Preemptive");

    printTable(processes, n);
    printGanttChart(timeline, count);
}



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

```
while (1) {
    printf("\nChoose Scheduling Algorithm:\n");
    printf("1. FCFS\n2. SJF\n3. Round Robin\n4. Priority Scheduling\n5. Exit\n");
    scanf("%d", &choice);

    if (choice == 5) break;

    printf("Enter number of processes: ");
    scanf("%d", &n);
    struct Process processes[n];

    for (int i = 0; i < n; i++) {
        printf("\nEnter Process ID: ");
        scanf("%d", &processes[i].id);
        printf("Enter Arrival Time: ");
        scanf("%d", &processes[i].arrival_time);
        printf("Enter Burst Time: ");
        scanf("%d", &processes[i].burst_time);
        if (choice == 4) {
            printf("Enter Priority: ");
            scanf("%d", &processes[i].priority);
        }
    }

    if (choice == 2 || choice == 4) {
        printf("Choose Mode: 1. Preemptive 2. Non-Preemptive\n");
        scanf("%d", &preemptive);
    }

    if (choice == 3) {
        printf("Enter Time Quantum: ");
        scanf("%d", &quantum);
    }

   switch (choice) {
        case 1: FCFS(processes, n); break;
        case 2: SJF(processes, n, preemptive == 1); break;
        case 3: RoundRobin(processes, n, quantum); break;
        case 4: PriorityScheduling(processes, n, preemptive == 1); break;
        default: printf("Invalid choice!\n");
    }
}
```

```
    return 0;
}
```

OUTPUT :

```
stealth@LAPTOP-M9JA1NRR:~$ gcc -w assign2_Mansi.c
stealth@LAPTOP-M9JA1NRR:~$ ./a.out

Choose Scheduling Algorithm:
1. FCFS
2. SJF
3. Round Robin
4. Priority Scheduling
5. Exit
1
Enter number of processes: 4

Enter Process ID: 1
Enter Arrival Time: 0
Enter Burst Time: 5

Enter Process ID: 2
Enter Arrival Time: 1
Enter Burst Time: 3

Enter Process ID: 3
Enter Arrival Time: 2
Enter Burst Time: 8

Enter Process ID: 4
Enter Arrival Time: 3
Enter Burst Time: 6

FCFS Scheduling:

ID    Arrival  Burst    Completion   Waiting    Turnaround
1     0        5        5            0          5
2     1        3        8            4          7
3     2        8        16           6          14
4     3        6        22           13         19

Average Waiting Time: 5.75
Average Turnaround Time: 11.25

Gantt Chart:
| P1 | P2 | P3 | P4 |
```

```
Choose Scheduling Algorithm:
1. FCFS
2. SJF
3. Round Robin
4. Priority Scheduling
5. Exit
2
Enter number of processes: 4

Enter Process ID: 1
Enter Arrival Time: 0
Enter Burst Time: 5

Enter Process ID: 2
Enter Arrival Time: 1
Enter Burst Time: 3

Enter Process ID: 3
Enter Arrival Time: 2
Enter Burst Time: 8

Enter Process ID: 4
Enter Arrival Time: 3
Enter Burst Time: 6
Choose Mode: 1. Preemptive 2. Non-Preemptive
1

SJF Scheduling (Preemptive):

ID    Arrival  Burst    Completion   Waiting    Turnaround
1     0        5        8            3          8
2     1        3        4            0          3
3     2        8        22           12         20
4     3        6        14           5          11

Average Waiting Time: 5.00
Average Turnaround Time: 10.50

Gantt Chart:
| P1 | P2 | P1 | P4 | P3 |
```

```
Choose Scheduling Algorithm:
1. FCFS
2. SJF
3. Round Robin
4. Priority Scheduling
5. Exit
2
Enter number of processes: 4

Enter Process ID: 1
Enter Arrival Time: 0
Enter Burst Time: 5

Enter Process ID: 2
Enter Arrival Time: 1
Enter Burst Time: 3

Enter Process ID: 3
Enter Arrival Time: 2
Enter Burst Time: 8

Enter Process ID: 4
Enter Arrival Time: 3
Enter Burst Time: 6
Choose Mode: 1. Preemptive 2. Non-Preemptive
2

SJF Scheduling (Non-Preemptive):

ID    Arrival  Burst    Completion   Waiting    Turnaround
1     0        5        5            0          5
2     1        3        8            4          7
3     2        8        22           12         20
4     3        6        14           5          11

Average Waiting Time: 5.25
Average Turnaround Time: 10.75

Gantt Chart:
| P1 | P2 | P4 | P3 |
```

```
Choose Scheduling Algorithm:
1. FCFS
2. SJF
3. Round Robin
4. Priority Scheduling
5. Exit
3
Enter number of processes: 4

Enter Process ID: 1
Enter Arrival Time: 0
Enter Burst Time: 5

Enter Process ID: 2
Enter Arrival Time: 1
Enter Burst Time: 3

Enter Process ID: 3
Enter Arrival Time: 2
Enter Burst Time: 8

Enter Process ID: 4
Enter Arrival Time: 3
Enter Burst Time: 6
Enter Time Quantum: 2

Round Robin Scheduling (Quantum = 2):

ID    Arrival  Burst    Completion   Waiting    Turnaround
1     0        5        16           11         16
2     1        3        11           7          10
3     2        8        22           12         20
4     3        6        20           11         17

Average Waiting Time: 10.25
Average Turnaround Time: 15.75

Gantt Chart:
| P1 | P2 | P3 | P4 | P1 | P2 | P3 | P4 | P1 | P3 | P4 | P3 |
```

```
4. Priority Scheduling
5. Exit
4
Enter number of processes: 4

Enter Process ID: 1
Enter Arrival Time: 0
Enter Burst Time: 5
Enter Priority: 2

Enter Process ID: 2
Enter Arrival Time: 1
Enter Burst Time: 3
Enter Priority: 1

Enter Process ID: 3
Enter Arrival Time: 2
Enter Burst Time: 8
Enter Priority: 3

Enter Process ID: 4
Enter Arrival Time: 3
Enter Burst Time: 6
Enter Priority: 2
Choose Mode: 1. Preemptive 2. Non-Preemptive
1

Priority Scheduling (Preemptive):

ID    Arrival  Burst    Completion   Waiting    Turnaround
1     0        5        8            3          8
2     1        3        4            0          3
3     2        8        22           12         20
4     3        6        14           5          11

Average Waiting Time: 5.00
Average Turnaround Time: 10.50

Gantt Chart:
| P1 | P2 | P1 | P4 | P3 |
```

```
4. Priority Scheduling
5. Exit
4
Enter number of processes: 4

Enter Process ID: 1
Enter Arrival Time: 0
Enter Burst Time: 5
Enter Priority: 2

Enter Process ID: 2
Enter Arrival Time: 1
Enter Burst Time: 3
Enter Priority: 1

Enter Process ID: 3
Enter Arrival Time: 2
Enter Burst Time: 8
Enter Priority: 3

Enter Process ID: 4
Enter Arrival Time: 3
Enter Burst Time: 6
Enter Priority: 2
Choose Mode: 1. Preemptive 2. Non-Preemptive
2

Priority Scheduling (Non-Preemptive):

ID    Arrival  Burst    Completion   Waiting    Turnaround
1     0        5        5            0          5
2     1        3        8            4          7
3     2        8        22           12         20
4     3        6        14           5          11

Average Waiting Time: 5.25
Average Turnaround Time: 10.75

Gantt Chart:
| P1 | P2 | P4 | P3 |
```

```
Choose Scheduling Algorithm:
1. FCFS
2. SJF
3. Round Robin
4. Priority Scheduling
5. Exit
5
stealth@LAPTOP-M9JA1NRR:~$
```