WEEK 2

Q. Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.

1. SJF (pre-emptive &; non-pre-emptive)

2. Priority (pre-emptive &; non-pre-emptive)

3. Round Robin (Experiment with different quantum sizes for RR algorithm)

Write a single program and inside the same program write different functions for different scheduling algorithms.

Code:

```
#include <stdio.h>

#include <stdbool.h>


#define MAX_PROCESSES 10


struct Process {

    int pid;

    int arrival_time;

    int burst_time;

    int priority;

    int remaining_time;

    int turnaround_time;

    int waiting_time;

};
```

```
void sjf_nonpreemptive(struct Process processes[], int n) {
   // Sort the processes based on burst time in ascending order
   int i,j,count=0,m;
   for(i=0;i<n;i++)
   {
   if(processes[i].arrival_time==0)
   count++;
}
if(count==n||count==1)
{
if(count==n)
{
for (i = 0; i < n - 1; i++) {
       for (j = 0; j < n - i - 1; j++) {
          if (processes[j].burst_time > processes[j + 1].burst_time) {
             struct Process temp = processes[j];
             processes[j] = processes[j + 1];
             processes[j + 1] = temp;
          }
       }
   }
}
else
{
for (i = 1; i < n - 1; i++) {
       for (j = 1; j <= n - i - 1; j++) {
```

```
            if (processes[j].burst_time > processes[j + 1].burst_time) {
                struct Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }
}



}

    int total_time = 0;
    double total_turnaround_time = 0;
    double total_waiting_time = 0;

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

        total_turnaround_time += processes[i].turnaround_time;
        total_waiting_time += processes[i].waiting_time;
    }

    printf("Process\tTurnaround Time\tWaiting Time\n");
```

```
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround_time,
processes[i].waiting_time);
    }

    printf("Average Turnaround Time: %.2f\n", total_turnaround_time / n);
    printf("Average Waiting Time: %.2f\n", total_waiting_time / n);
}

void sjf_preemptive(struct Process processes[], int n) {
    int total_time = 0,i;
    int completed = 0;

    while (completed < n) {
        int shortest_burst = -1;
        int next_process = -1;

        for (i = 0; i < n; i++) {
            if (processes[i].arrival_time <= total_time &&
processes[i].remaining_time > 0) {
                if (shortest_burst == -1 || processes[i].remaining_time <
shortest_burst) {
                    shortest_burst = processes[i].remaining_time;
                    next_process = i;
                }
            }
        }
```

```
        if (next_process == -1) {
            total_time++;
            continue;
        }

        processes[next_process].remaining_time--;
        total_time++;

        if (processes[next_process].remaining_time == 0) {
            completed++;
            processes[next_process].turnaround_time = total_time -
processes[next_process].arrival_time;
            processes[next_process].waiting_time =
processes[next_process].turnaround_time -
processes[next_process].burst_time;
        }
    }

    double total_turnaround_time = 0;
    double total_waiting_time = 0;

    printf("Process\tTurnaround Time\tWaiting Time\n");
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround_time,
processes[i].waiting_time);
```

```
        total_turnaround_time += processes[i].turnaround_time;
        total_waiting_time += processes[i].waiting_time;
    }

    printf("Average Turnaround Time: %.2f\n", total_turnaround_time / n);
    printf("Average Waiting Time: %.2f\n", total_waiting_time / n);
}

void priority_nonpreemptive(struct Process processes[], int n) {
    // Sort the processes based on priority in ascending order
    int i,j,count=0,m;
    for(i=0;i<n;i++)
    {
    if(processes[i].arrival_time==0)
    count++;
}
if(count==n||count==1)
{
if(count==n)
{
for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (processes[j].priority > processes[j + 1].priority) {
                struct Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
```

```
            }
        }
    }
}

else
{
    for (i = 1; i < n - 1; i++) {
        for (j = 1; j <= n - i - 1; j++) {
            if (processes[j].priority > processes[j + 1].priority) {
                struct Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }
}
}

    int total_time = 0;
    double total_turnaround_time = 0;
    double total_waiting_time = 0;

    for (i = 0; i < n; i++) {
        total_time += processes[i].burst_time;
        processes[i].turnaround_time = total_time - processes[i].arrival_time;
```

```
        processes[i].waiting_time = processes[i].turnaround_time -
processes[i].burst_time;

        total_turnaround_time += processes[i].turnaround_time;
        total_waiting_time += processes[i].waiting_time;
    }

    printf("Process\tTurnaround Time\tWaiting Time\n");
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround_time,
processes[i].waiting_time);
    }

    printf("Average Turnaround Time: %.2f\n", total_turnaround_time / n);
    printf("Average Waiting Time: %.2f\n", total_waiting_time / n);
}

void priority_preemptive(struct Process processes[], int n) {
    int total_time = 0,i;
    int completed = 0;

    while (completed < n) {
        int highest_priority = -1;
        int next_process = -1;

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

```
        if (processes[i].arrival_time <= total_time &&
processes[i].remaining_time > 0) {
            if (highest_priority == -1 || processes[i].priority < highest_priority) {
                highest_priority = processes[i].priority;
                next_process = i;
            }
        }
    }

    if (next_process == -1) {
        total_time++;
        continue;
    }

    processes[next_process].remaining_time--;
    total_time++;

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

```
    double total_turnaround_time = 0;
    double total_waiting_time = 0;

    printf("Process\tTurnaround Time\tWaiting Time\n");
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround_time,
processes[i].waiting_time);

        total_turnaround_time += processes[i].turnaround_time;
        total_waiting_time += processes[i].waiting_time;
    }

    printf("Average Turnaround Time: %.2f\n", total_turnaround_time / n);
    printf("Average Waiting Time: %.2f\n", total_waiting_time / n);
}

void round_robin(struct Process processes[], int n, int quantum) {
    int total_time = 0,i;
    int completed = 0;

    while (completed < n) {
        for (i = 0; i < n; i++) {
            if (processes[i].arrival_time <= total_time &&
processes[i].remaining_time > 0) {
                if (processes[i].remaining_time <= quantum) {
                    total_time += processes[i].remaining_time;
                    processes[i].remaining_time = 0;
```

```
                processes[i].turnaround_time = total_time - processes[i].arrival_time;
                processes[i].waiting_time = processes[i].turnaround_time - processes[i].burst_time;
                completed++;
            } else {
                total_time += quantum;
                processes[i].remaining_time -= quantum;
            }
        }
    }
}

double total_turnaround_time = 0;
double total_waiting_time = 0;

printf("Process\tTurnaround Time\tWaiting Time\n");
for (i = 0; i < n; i++) {
    printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround_time, processes[i].waiting_time);

    total_turnaround_time += processes[i].turnaround_time;
    total_waiting_time += processes[i].waiting_time;
}

printf("Average Turnaround Time: %.2f\n", total_turnaround_time / n);
printf("Average Waiting Time: %.2f\n", total_waiting_time / n);
```

```
}

int main() {
    int n, quantum,i,choice;
    struct Process processes[MAX_PROCESSES];

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

    for (i = 0; i < n; i++) {
        printf("Process %d\n", i + 1);
        printf("Enter arrival time: ");
        scanf("%d", &processes[i].arrival_time);
        printf("Enter burst time: ");
        scanf("%d", &processes[i].burst_time);
        printf("Enter priority: ");
        scanf("%d", &processes[i].priority);
        processes[i].pid = i + 1;
        processes[i].remaining_time = processes[i].burst_time;
        processes[i].turnaround_time = 0;
        processes[i].waiting_time = 0;
    }
    while(1)
    {

    }
```

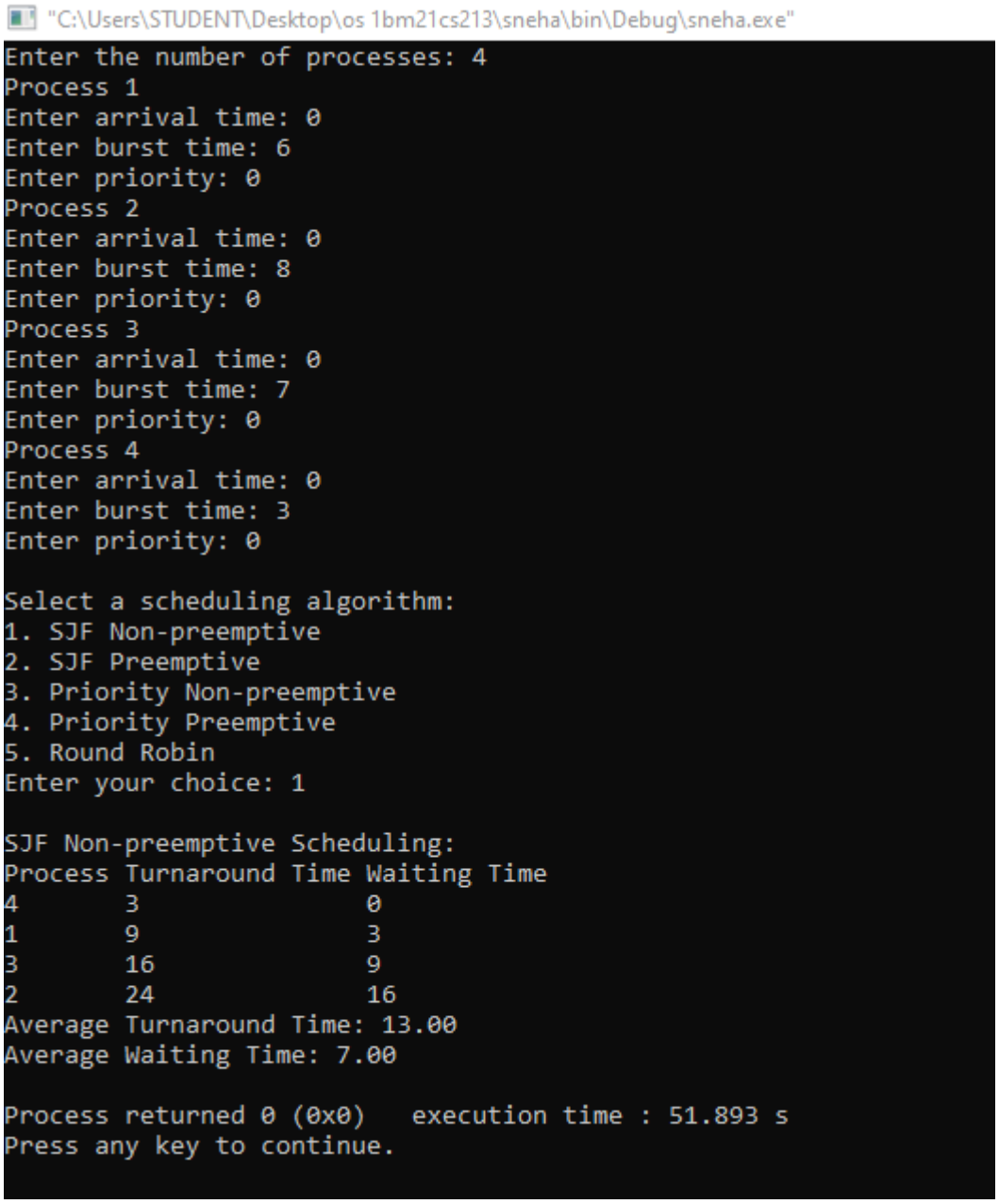
```
printf("\nSelect a scheduling algorithm:\n");
printf("1. SJF Non-preemptive\n");
printf("2. SJF Preemptive\n");
printf("3. Priority Non-preemptive\n");
printf("4. Priority Preemptive\n");
printf("5. Round Robin\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
   case 1:
      printf("\nSJF Non-preemptive Scheduling:\n");
      sjf_nonpreemptive(processes, n);
      break;
   case 2:
      printf("\nSJF Preemptive Scheduling:\n");
      sjf_preemptive(processes, n);
      break;
   case 3:
      printf("\nPriority Non-preemptive Scheduling:\n");
      priority_nonpreemptive(processes, n);
      break;
   case 4:
      printf("\nPriority Preemptive Scheduling:\n");
      priority_preemptive(processes, n);
      break;
```

```
        case 5:
            printf("\nEnter the quantum size for Round Robin: ");
            scanf("%d", &quantum);
            printf("\nRound Robin Scheduling (Quantum: %d):\n", quantum);
            round_robin(processes, n, quantum);
            break;
        default:
            printf("Invalid choice!\n");
            return 1;
    }

    return 0;
}
```

Output:


"C:\Users\STUDENT\Desktop\os 1bm21cs213\sneha\bin\Debug\sneha.exe"
Enter the number of processes: 4
Process 1
Enter arrival time: 0
Enter burst time: 6
Enter priority: 0
Process 2
Enter arrival time: 0
Enter burst time: 8
Enter priority: 0
Process 3
Enter arrival time: 0
Enter burst time: 7
Enter priority: 0
Process 4
Enter arrival time: 0
Enter burst time: 3
Enter priority: 0
Select a scheduling algorithm:
1. SJF Non-preemptive
2. SJF Preemptive
3. Priority Non-preemptive
4. Priority Preemptive
5. Round Robin
Enter your choice: 1
SJF Non-preemptive Scheduling:
Process Turnaround Time Waiting Time
4 3 0
1 9 3
3 16 9
2 24 16
Average Turnaround Time: 13.00
Average Waiting Time: 7.00
Process returned 0 (0x0) execution time : 51.893 s
Press any key to continue.

"C:\Users\STUDENT\Desktop\os 1bm21cs213\sneha\bin\Debug\sneha.exe"

```
Enter the number of processes: 4
Process 1
Enter arrival time: 0
Enter burst time: 8
Enter priority: 0
Process 2
Enter arrival time: 1
Enter burst time: 4
Enter priority: 0
Process 3
Enter arrival time: 2
Enter burst time: 9
Enter priority: 0
Process 4
Enter arrival time: 3
Enter burst time: 5
Enter priority: 0

Select a scheduling algorithm:
1. SJF Non-preemptive
2. SJF Preemptive
3. Priority Non-preemptive
4. Priority Preemptive
5. Round Robin
Enter your choice: 2

SJF Preemptive Scheduling:
Process Turnaround Time Waiting Time
1       17              9
2       4               0
3       24              15
4       7               2
Average Turnaround Time: 13.00
Average Waiting Time: 6.50

Process returned 0 (0x0)   execution time : 30.692 s
Press any key to continue.
```

"C:\Users\STUDENT\Desktop\os 1bm21cs213\sneha\bin\Debug\sneha.exe"

```
Enter the number of processes: 5
Process 1
Enter arrival time: 0
Enter burst time: 4
Enter priority: 4
Process 2
Enter arrival time: 1
Enter burst time: 3
Enter priority: 3
Process 3
Enter arrival time: 3
Enter burst time: 4
Enter priority: 1
Process 4
Enter arrival time: 6
Enter burst time: 2
Enter priority: 5
Process 5
Enter arrival time: 8
Enter burst time: 4
Enter priority: 2

Select a scheduling algorithm:
1. SJF Non-preemptive
2. SJF Preemptive
3. Priority Non-preemptive
4. Priority Preemptive
5. Round Robin
Enter your choice: 3

Priority Non-preemptive Scheduling:
Process Turnaround Time Waiting Time
1       4               0
3       5               1
5       4               0
2       14              11
4       11              9
Average Turnaround Time: 7.60
Average Waiting Time: 4.20

Process returned 0 (0x0)   execution time : 53.247 s
Press any key to continue.
```

"C:\Users\STUDENT\Desktop\os 1bm21cs213\sneha\bin\Debug\sneha.exe"

```
Enter the number of processes: 5
Process 1
Enter arrival time: 0
Enter burst time: 4
Enter priority: 4
Process 2
Enter arrival time: 1
Enter burst time: 3
Enter priority: 3
Process 3
Enter arrival time: 3
Enter burst time: 4
Enter priority: 1
Process 4
Enter arrival time: 6
Enter burst time: 2
Enter priority: 5
Process 5
Enter arrival time: 8
Enter burst time: 4
Enter priority: 2

Select a scheduling algorithm:
1. SJF Non-preemptive
2. SJF Preemptive
3. Priority Non-preemptive
4. Priority Preemptive
5. Round Robin
Enter your choice: 4

Priority Preemptive Scheduling:
Process Turnaround Time Waiting Time
1       15              11
2       7               4
3       4               0
4       11              9
5       4               0
Average Turnaround Time: 8.20
Average Waiting Time: 4.80

Process returned 0 (0x0)   execution time : 33.555 s
Press any key to continue.
```

"C:\Users\STUDENT\Desktop\os 1bm21cs213\sneha\bin\Debug\sneha.exe"

```
Enter the number of processes: 5
Process 1
Enter arrival time: 0
Enter burst time: 5
Enter priority: 0
Process 2
Enter arrival time: 0
Enter burst time: 3
Enter priority: 0
Process 3
Enter arrival time: 0
Enter burst time: 1
Enter priority: 0
Process 4
Enter arrival time: 0
Enter burst time: 2
Enter priority: 0
Process 5
Enter arrival time: 0
Enter burst time: 3
Enter priority: 0

Select a scheduling algorithm:
1. SJF Non-preemptive
2. SJF Preemptive
3. Priority Non-preemptive
4. Priority Preemptive
5. Round Robin
Enter your choice: 5

Enter the quantum size for Round Robin: 2

Round Robin Scheduling (Quantum: 2):
Process Turnaround Time Waiting Time
1       14              9
2       12              9
3       5               4
4       7               5
5       13              10
Average Turnaround Time: 10.20
Average Waiting Time: 7.40

Process returned 0 (0x0)   execution time : 51.361 s
Press any key to continue.
```