Write a C program to simulate the following non-pre-emptive 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)

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_PROCESSES 10

struct Process
{
   int pid;
   int arr_time;
   int burst_time;
   int priority;
   int rem_time;
   int tat;
   int wt;
};
void sjf_nonpreemptive(struct Process p[], int n)
{
   int i, j, count = 0, m;
   for (i = 0; i < n; i++)
   {
      if (p[i].arr_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 (p[j].burst_time > p[j + 1].burst_time)
                {
                    struct Process temp = p[j];
                    p[j] = p[j + 1];
                    p[j + 1] = temp;
                }
            }
        }
    }
    else
    {
        for (i = 1; i < n - 1; i++)
        {
            for (j = 1; j <= n - i - 1; j++)
            {
                if (p[j].burst_time > p[j + 1].burst_time)
                {
                    struct Process temp = p[j];
                    p[j] = p[j + 1];
                    p[j + 1] = temp;
                }
            }
        }
    }
}
```

```
    int total_time = 0;
    double total_tat = 0;
    double total_wt = 0;

    for (i = 0; i < n; i++)
    {
        total_time += p[i].burst_time;
        p[i].tat = total_time - p[i].arr_time;
        p[i].wt = p[i].tat - p[i].burst_time;

        total_tat += p[i].tat;
        total_wt += p[i].wt;
    }

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

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

void sjf_preemptive(struct Process p[], 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 (p[i].arr_time <= total_time && p[i].rem_time > 0)
        {
            if (shortest_burst == -1 || p[i].rem_time < shortest_burst)
            {
                shortest_burst = p[i].rem_time;
                next_process = i;
            }
        }
    }

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

    p[next_process].rem_time--;
    total_time++;

    if (p[next_process].rem_time == 0)
    {
        completed++;
        p[next_process].tat = total_time - p[next_process].arr_time;
```

```
            p[next_process].wt = p[next_process].tat - p[next_process].burst_time;
        }
    }

    double total_tat = 0;
    double total_wt = 0;

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

        total_tat += p[i].tat;
        total_wt += p[i].wt;
    }

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

void priority_nonpreemptive(struct Process p[], int n)
{
    int i, j, count = 0, m;
    for (i = 0; i < n; i++)
    {
        if (p[i].arr_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 (p[j].priority > p[j + 1].priority)
            {
                struct Process temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
}

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

```
        }
    }

    int total_time = 0;
    double total_tat = 0;
    double total_wt = 0;

    for (i = 0; i < n; i++)
    {
        total_time += p[i].burst_time;
        p[i].tat = total_time - p[i].arr_time;
        p[i].wt = p[i].tat - p[i].burst_time;

        total_tat += p[i].tat;
        total_wt += p[i].wt;
    }

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

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

void priority_preemptive(struct Process p[], 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 (p[i].arr_time <= total_time && p[i].rem_time > 0)
        {
            if (highest_priority == -1 || p[i].priority < highest_priority)
            {
                highest_priority = p[i].priority;
                next_process = i;
            }
        }
    }

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

    p[next_process].rem_time--;
    total_time++;

    if (p[next_process].rem_time == 0)
    {
```

```
            completed++;
            p[next_process].tat = total_time - p[next_process].arr_time;
            p[next_process].wt = p[next_process].tat - p[next_process].burst_time;
        }
    }

    double total_tat = 0;
    double total_wt = 0;

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

        total_tat += p[i].tat;
        total_wt += p[i].wt;
    }

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

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

    printf("\nGantt Chart: \n");
    while (completed < n)
    {
```

```
    for (i = 0; i < n; i++)
    {
        if (p[i].arr_time <= total_time && p[i].rem_time > 0)
        {
            if (p[i].rem_time <= quantum)
            {
                printf("P%d ", p[i].pid);
                total_time += p[i].rem_time;
                p[i].rem_time = 0;
                p[i].tat = total_time - p[i].arr_time;
                p[i].wt = p[i].tat - p[i].burst_time;
                completed++;
            }
            else
            {
                printf("P%d ", p[i].pid);
                total_time += quantum;
                p[i].rem_time -= quantum;
            }
        }
    }
}

double total_tat = 0;
double total_wt = 0;

printf("\n");
printf("\nProcess\tTurnaround Time\tWaiting Time\n");
for (i = 0; i < n; i++)
```

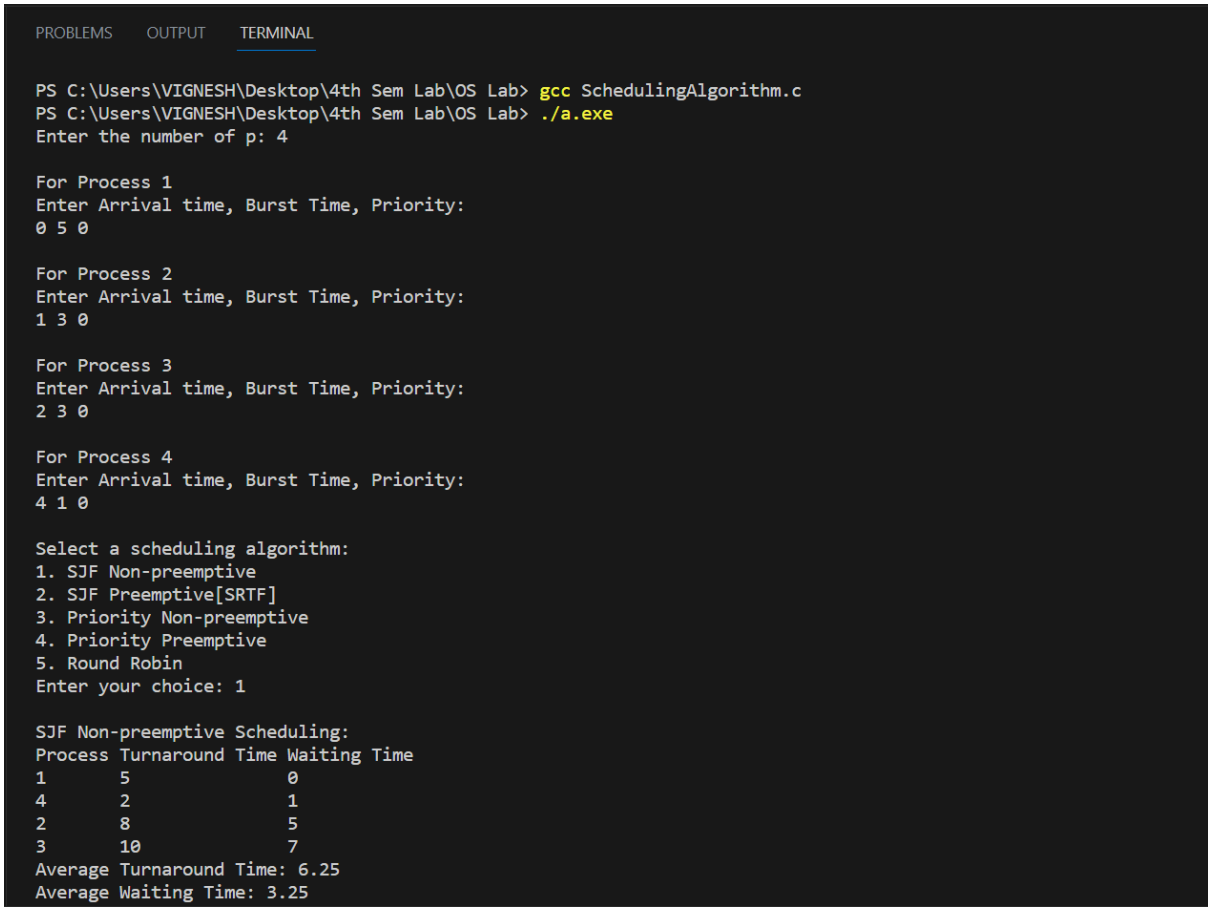
```
    {
        printf("%d\t%d\t\t%d\n", p[i].pid, p[i].tat, p[i].wt);

        total_tat += p[i].tat;
        total_wt += p[i].wt;
    }

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

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

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

    for (i = 0; i < n; i++)
    {
        printf("\nFor Process %d\n", i + 1);
        printf("Enter Arrival time, Burst Time, Priority:\n");
        scanf("%d%d%d",&p[i].arr_time,&p[i].burst_time,&p[i].priority);
        p[i].pid = i + 1;
        p[i].rem_time = p[i].burst_time;
        p[i].tat = 0;
        p[i].wt = 0;
    }
    printf("\nSelect a scheduling algorithm:\n");
```

```
printf("1. SJF Non-preemptive\n");
printf("2. SJF Preemptive[SRTF]\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(p, n);
   break;
case 2:
   printf("\nSJF Preemptive Scheduling:\n");
   sjf_preemptive(p, n);
   break;
case 3:
   printf("\nPriority Non-preemptive Scheduling:\n");
   priority_nonpreemptive(p, n);
   break;
case 4:
   printf("\nPriority Preemptive Scheduling:\n");
   priority_preemptive(p, 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(p, n, quantum);
            break;
        default:
            printf("Invalid choice!\n");
            return 1;
        }

        return 0;
}
```

OUTPUT:

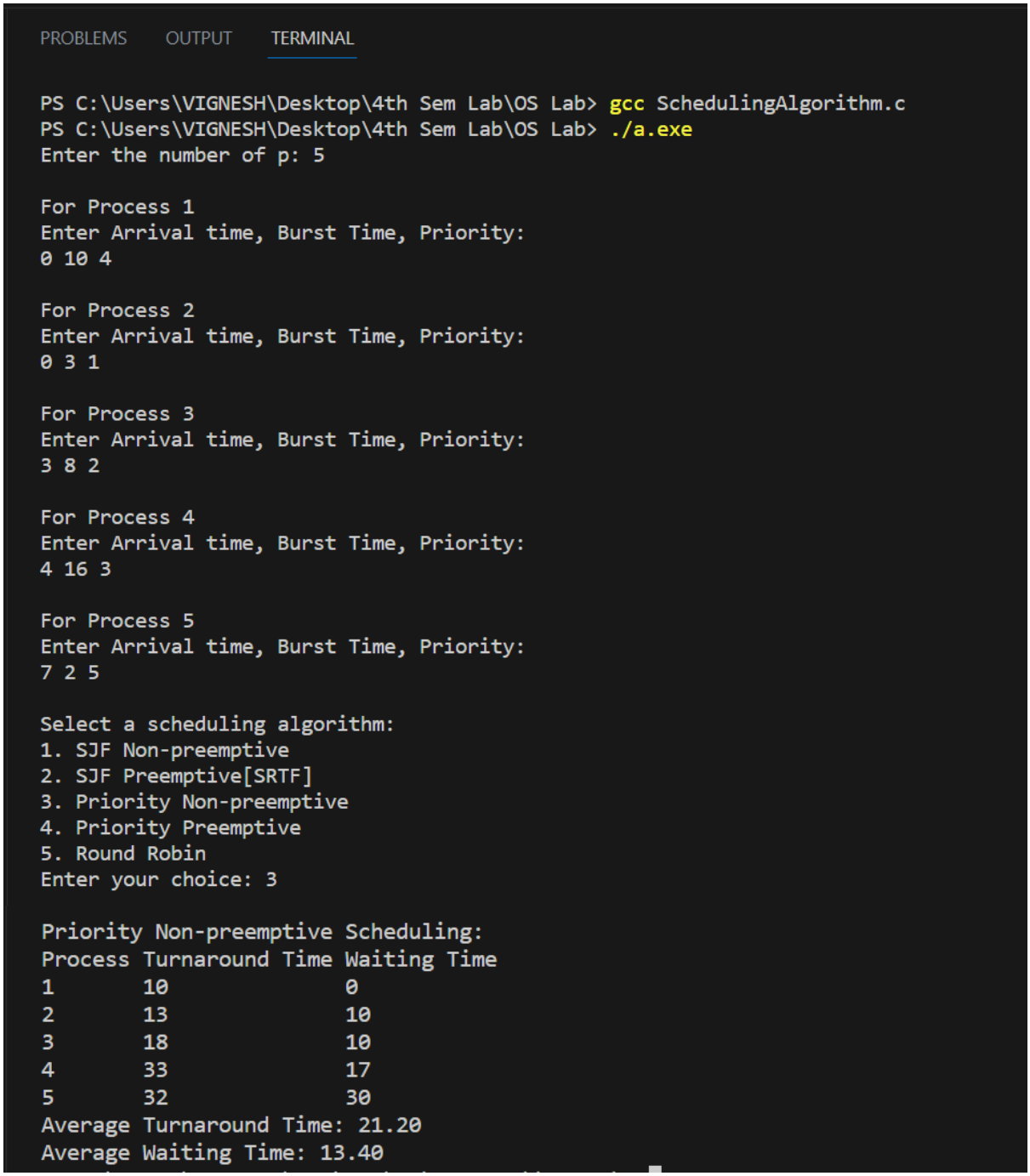
SJF Non-Pre-emptive



SJF Pre-emptive

```
PROBLEMS    OUTPUT    TERMINAL

PS C:\Users\VIGNESH\Desktop\4th Sem Lab\OS Lab> gcc SchedulingAlgorithm.c
PS C:\Users\VIGNESH\Desktop\4th Sem Lab\OS Lab> ./a.exe
Enter the number of p: 4

For Process 1
Enter Arrival time, Burst Time, Priority:
0 5 0

For Process 2
Enter Arrival time, Burst Time, Priority:
1 3 0

For Process 3
Enter Arrival time, Burst Time, Priority:
2 3 0

For Process 4
Enter Arrival time, Burst Time, Priority:
4 1 0

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

SJF Preemptive Scheduling:
Process Turnaround Time Waiting Time
1       12              7
2       3               0
3       6               3
4       1               0
Average Turnaround Time: 5.50
Average Waiting Time: 2.50
```

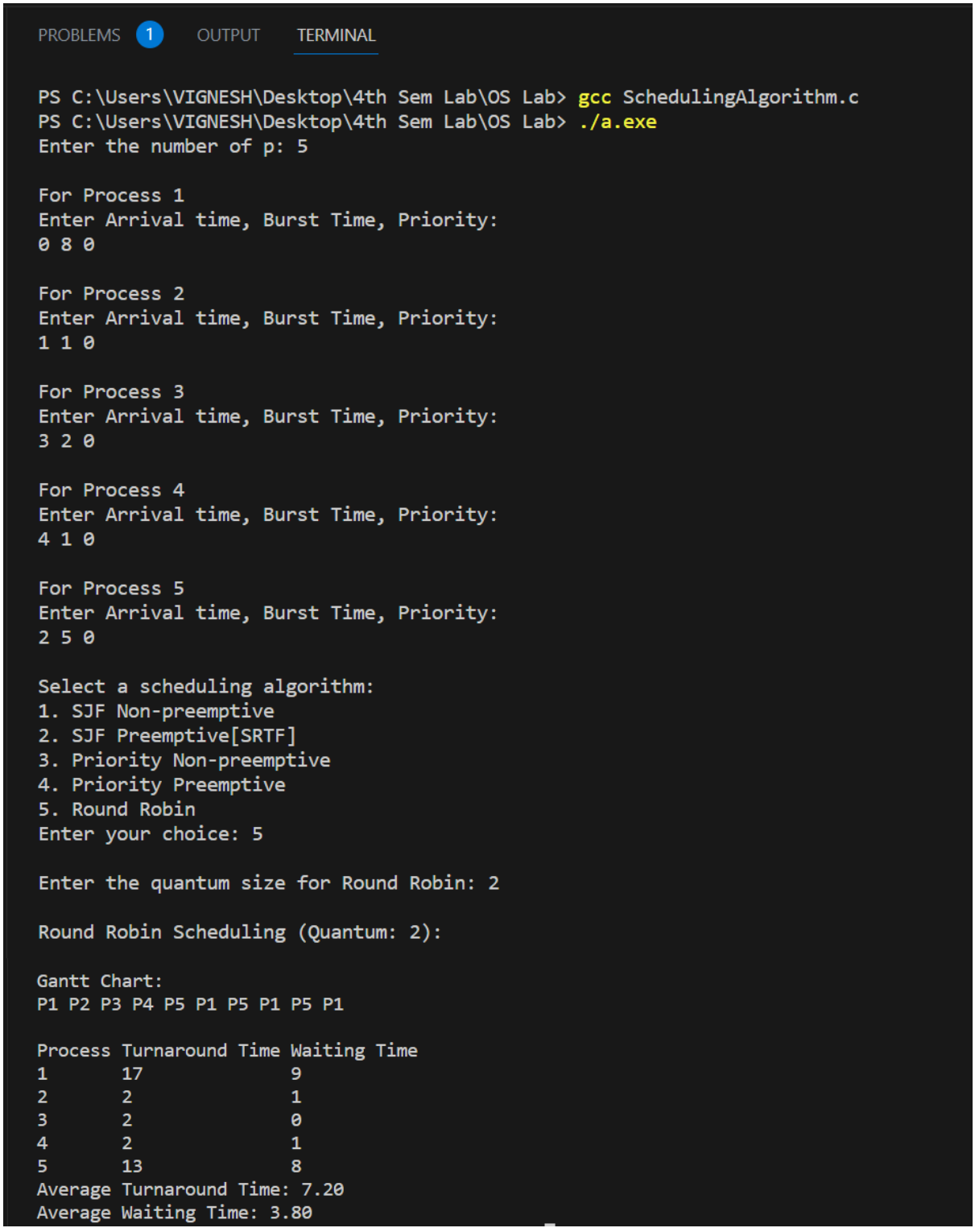
Priority Non-pre-emptive



Priority pre-emptive

```
PROBLEMS    OUTPUT    TERMINAL

PS C:\Users\VIGNESH\Desktop\4th Sem Lab\OS Lab> gcc SchedulingAlgorithm.c
PS C:\Users\VIGNESH\Desktop\4th Sem Lab\OS Lab> ./a.exe
Enter the number of p: 5

For Process 1
Enter Arrival time, Burst Time, Priority:
0 10 4

For Process 2
Enter Arrival time, Burst Time, Priority:
0 3 1

For Process 3
Enter Arrival time, Burst Time, Priority:
3 8 2

For Process 4
Enter Arrival time, Burst Time, Priority:
4 16 3

For Process 5
Enter Arrival time, Burst Time, Priority:
7 2 5

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

Priority Preemptive Scheduling:
Process Turnaround Time Waiting Time
1       37              27
2       3               0
3       8               0
4       23              7
5       32              30
Average Turnaround Time: 20.60
Average Waiting Time: 12.80
```

Round Robin


PROBLEMS 1
OUTPUT
TERMINAL
PS C:\Users\VIGNESH\Desktop\4th Sem Lab\OS Lab> gcc SchedulingAlgorithm.c
PS C:\Users\VIGNESH\Desktop\4th Sem Lab\OS Lab> ./a.exe
Enter the number of p: 5
For Process 1
Enter Arrival time, Burst Time, Priority:
0 8 0
For Process 2
Enter Arrival time, Burst Time, Priority:
1 1 0
For Process 3
Enter Arrival time, Burst Time, Priority:
3 2 0
For Process 4
Enter Arrival time, Burst Time, Priority:
4 1 0
For Process 5
Enter Arrival time, Burst Time, Priority:
2 5 0
Select a scheduling algorithm:
1. SJF Non-preemptive
2. SJF Preemptive[SRTF]
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):
Gantt Chart:
P1 P2 P3 P4 P5 P1 P5 P1 P5 P1
Process Turnaround Time Waiting Time
1 17 9
2 2 1
3 2 0
4 2 1
5 13 8
Average Turnaround Time: 7.20
Average Waiting Time: 3.80