OS LAB

Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.

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

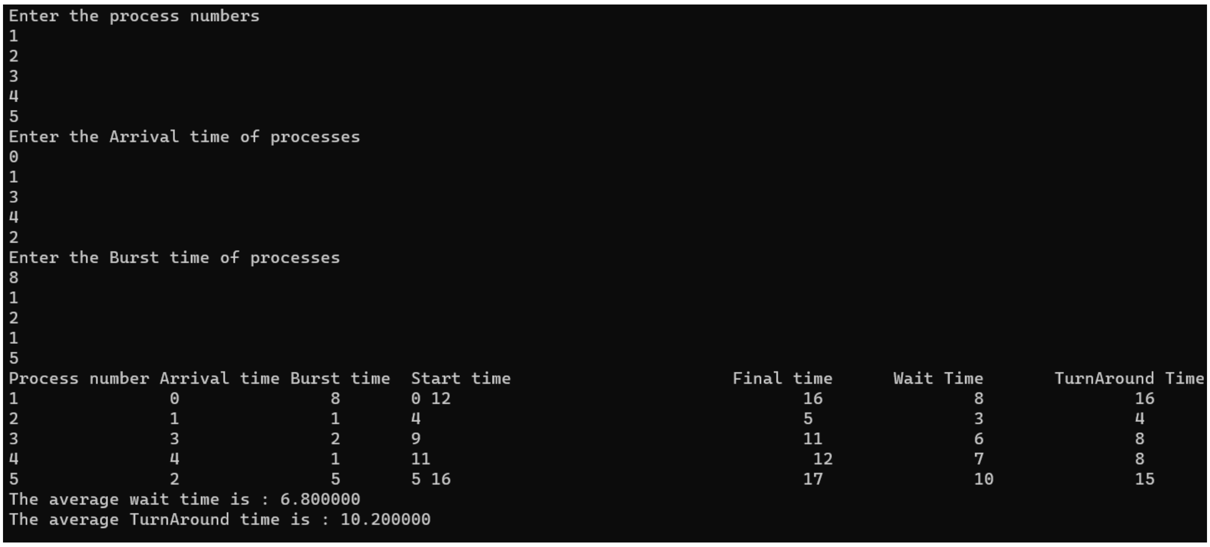
```
#include<stdio.h>
#include<limits.h>
#include<stdbool.h>

struct P{
int AT,BT,ST[20],WT,FT,TAT,pos;
};

int quant;
int main()
{
   int n,i,j;
   printf("Enter the no. of processes :");
   scanf("%d",&n);
   struct P p[n];
   printf("Enter the quantum \n");
   scanf("%d",&quant);
   printf("Enter the process numbers \n");
   for(i=0;i<n;i++)
   scanf("%d",&(p[i].pos));
   printf("Enter the Arrival time of processes \n");
   for(i=0;i<n;i++)
   scanf("%d",&(p[i].AT));
   printf("Enter the Burst time of processes \n");
```

```
for(i=0;i<n;i++)
scanf("%d",&(p[i].BT));
int c=n,s[n][20];
float time=0,mini=INT_MAX,b[n],a[n];
int index=-1;
for(i=0;i<n;i++)
   {
   b[i]=p[i].BT;
   a[i]=p[i].AT;
   for(j=0;j<20;j++)
   {
      s[i][j]=-1;
   }
}

int tot_wt,tot_tat;
tot_wt=0;
tot_tat=0;
bool flag=false;
while(c!=0)
{
   mini=INT_MAX;
   flag=false;
   for(i=0;i<n;i++)
   {
      float p=time+0.1;
      if(a[i]<=p && mini>a[i] && b[i]>0)
```

```
    {
        index=i;
        mini=a[i];
        flag=true;

    }
}

if(!flag)
{
    time++;
    continue;
}
j=0;
while(s[index][j]!=-1)
{
    j++;
}
if(s[index][j]==-1)
{
    s[index][j]=time;
    p[index].ST[j]=time;
}

if(b[index]<=quant)
{
time+=b[index];
```

```
    b[index]=0;
    }
    else
    {
        time+=quant;
        b[index]-=quant;
    }
    if(b[index]>0)
    {
        a[index]=time+0.1;
    }
    if(b[index]==0)
    {
        c--;
        p[index].FT=time;
        p[index].WT=p[index].FT-p[index].AT-p[index].BT;
        tot_wt+=p[index].WT;
        p[index].TAT=p[index].BT+p[index].WT;
        tot_tat+=p[index].TAT;
    }

}

printf("Process number ");
printf("Arrival time ");
printf("Burst time ");
printf("\tStart time");
j=0;
while(j!=10)
{
    j+=1;
    printf(" ");
}
printf("\t\tFinal time");
printf("\tWait Time ");
printf("\tTurnAround Time \n");
for(i=0;i<n;i++)
{
    printf("%d \t\t",p[i].pos);
    printf("%d \t\t",p[i].AT);
    printf("%d \t",p[i].BT);
    j=0;
    int v=0;
    while(s[i][j]!=-1)
    {
        printf("%d ",p[i].ST[j]);
        j++;
        v+=3;
    }
    while(v!=40)
    {
        printf(" ");
        v+=1;
        }
        printf("%d \t\t",p[i].FT);
        printf("%d \t\t",p[i].WT);
        printf("%d \n",p[i].TAT);


    }
    double avg_wt,avg_tat;
    avg_wt=tot_wt/(float)n;
    avg_tat=tot_tat/(float)n;
    printf("The average wait time is : %lf\n",avg_wt);
    printf("The average TurnAround time is : %lf\n",avg_tat);
    return 0;
}
```

OUTPUT



```
Enter the process numbers
1
2
3
4
5
Enter the Arrival time of processes
0
1
3
4
2
Enter the Burst time of processes
8
1
2
1
5
Process number Arrival time Burst time  Start time                      Final time      Wait Time       TurnAround Time
1               0               8       0 12                                   16              8               16
2               1               1       4                                      5               3               4
3               3               2       9                                      11              6               8
4               4               1       11                                      12             7               8
5               2               5       5 16                                   17              10              15
The average wait time is : 6.800000
The average TurnAround time is : 10.200000
```

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)

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

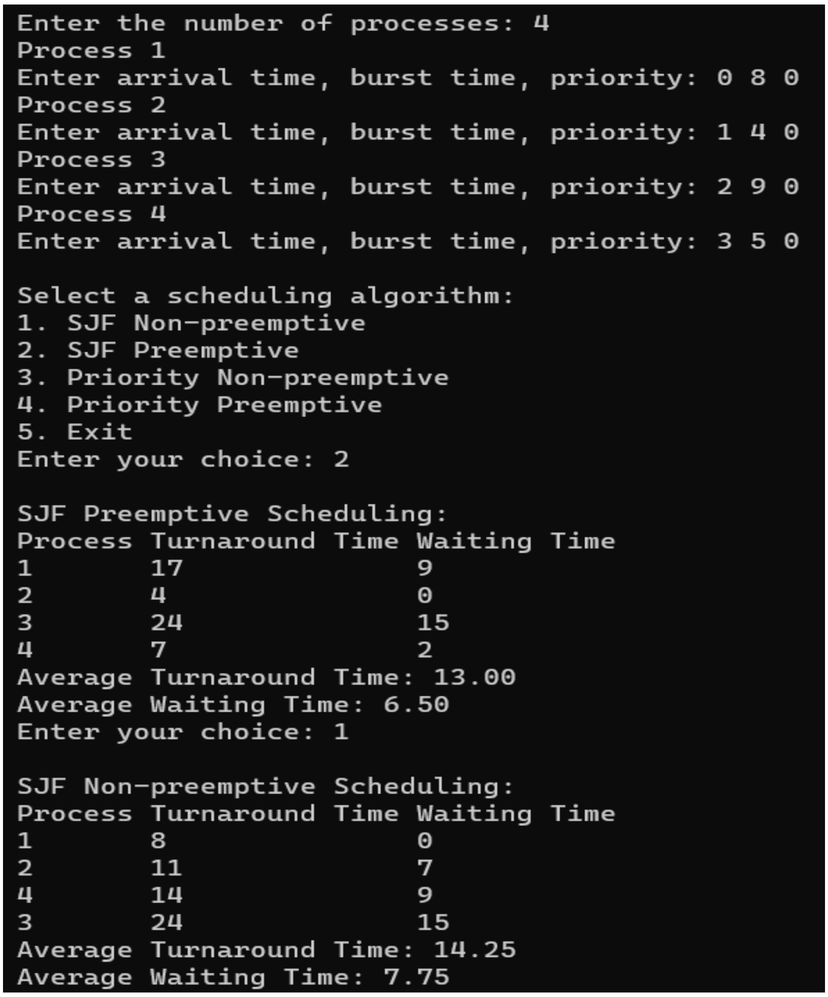
```
#include<stdlib.h>

#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) {
```

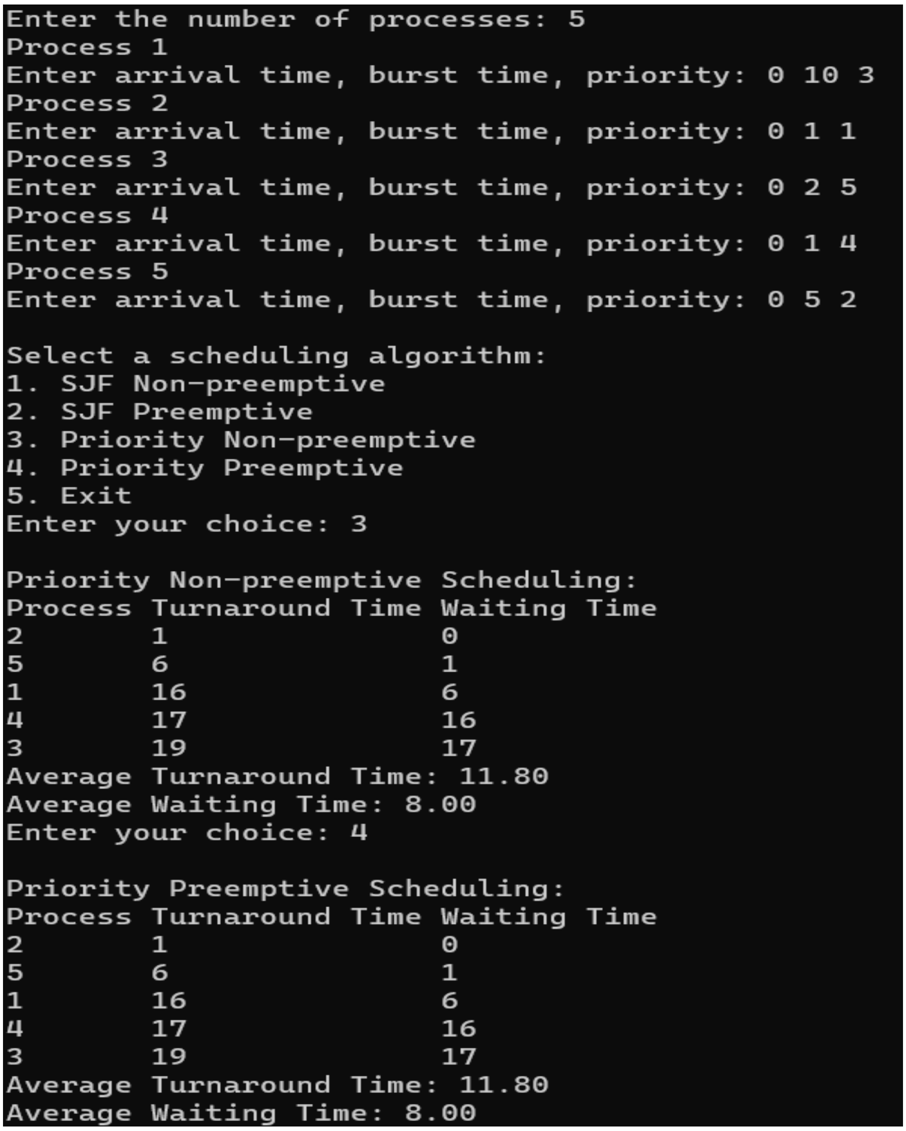
```
                if (highest_priority == -1 || processes[i].priority < highest_priority) {
                    highest_priority = processes[i].priority;
                    next_process = i;
                }
            }
        }

        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);
}
```

```
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, burst time, priority: ");
        scanf("%d %d %d", &processes[i].arrival_time, &processes[i].burst_time,
&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;
    }
    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. Exit\n");
    while(1)
    {
        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: exit(0); break;
            default:
                printf("Invalid choice!\n");
                return 1;
        }
    }
    return 0;
}
```

OUTPUT



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

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

SJF Non-preemptive Scheduling:
Process Turnaround Time Waiting Time
1       8               0
2       11              7
4       14              9
3       24              15
Average Turnaround Time: 14.25
Average Waiting Time: 7.75
```



```
Enter the number of processes: 5
Process 1
Enter arrival time, burst time, priority: 0 10 3
Process 2
Enter arrival time, burst time, priority: 0 1 1
Process 3
Enter arrival time, burst time, priority: 0 2 5
Process 4
Enter arrival time, burst time, priority: 0 1 4
Process 5
Enter arrival time, burst time, priority: 0 5 2

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

Priority Non-preemptive Scheduling:
Process Turnaround Time Waiting Time
2       1               0
5       6               1
1       16              6
4       17              16
3       19              17
Average Turnaround Time: 11.80
Average Waiting Time: 8.00
Enter your choice: 4

Priority Preemptive Scheduling:
Process Turnaround Time Waiting Time
2       1               0
5       6               1
1       16              6
4       17              16
3       19              17
Average Turnaround Time: 11.80
Average Waiting Time: 8.00
```