Operating Systems

Week 2 - Lab

(25 June - 2021)

Student Name : *Gurram Harshavardhan Netha*
ID Number : *B171325*

Index

Shortest Job Next/First

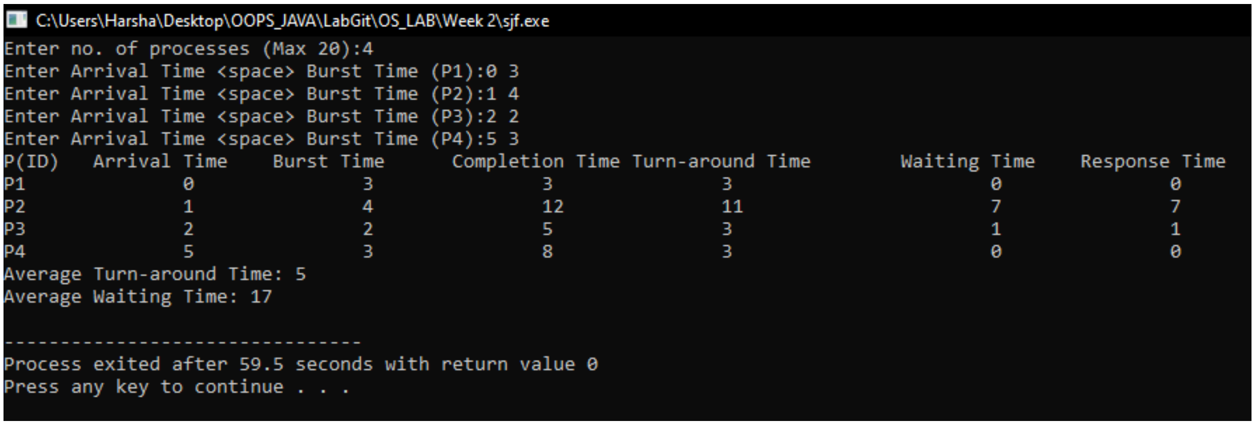
Problem Statement:

Implementation of Shortest Job First/Next CPU Scheduling algorithm in C language.

Input: Number of Processes.
Arrival and Burst times of all processes

Output: Completion Time, Turn-around time, Waiting time, Response time, Average turn-around time and Average Waiting time

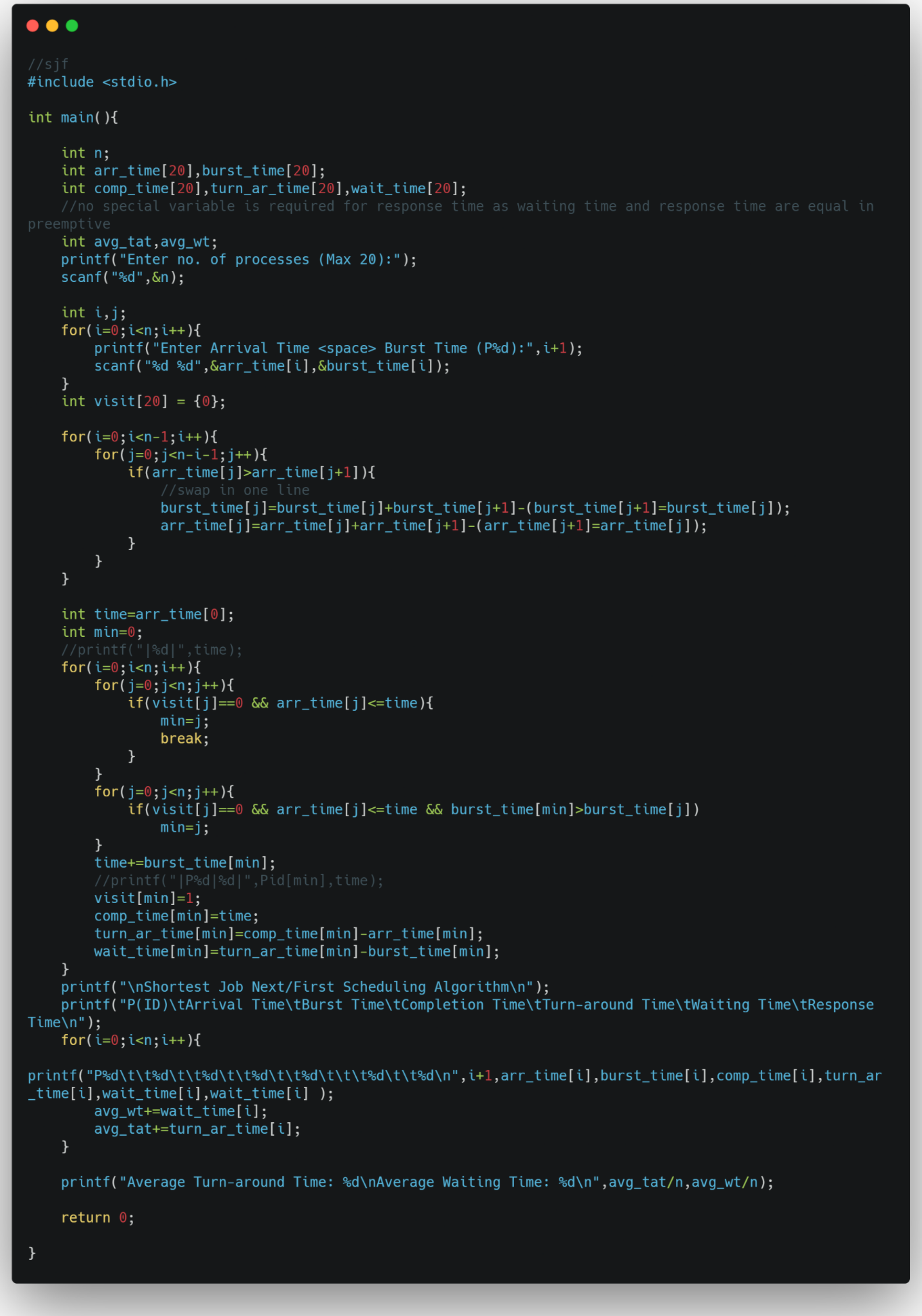
Execution Screenshot:


C:\Users\Harsha\Desktop\OOPS_JAVA\LabGit\OS_LAB\Week 2\sjf.exe

```
Enter no. of processes (Max 20):4
Enter Arrival Time <space> Burst Time (P1):0 3
Enter Arrival Time <space> Burst Time (P2):1 4
Enter Arrival Time <space> Burst Time (P3):2 2
Enter Arrival Time <space> Burst Time (P4):5 3
P(ID)   Arrival Time    Burst Time      Completion Time Turn-around Time        Waiting Time    Response Time
P1              0               3               3               3                       0               0
P2              1               4               12              11                      7               7
P3              2               2               5               3                       1               1
P4              5               3               8               3                       0               0
Average Turn-around Time: 5
Average Waiting Time: 17

--------------------------------
Process exited after 59.5 seconds with return value 0
Press any key to continue . . .
```

Code: Click on below image to inspect the code.



```
//sjf
#include <stdio.h>

int main(){

    int n;
    int arr_time[20],burst_time[20];
    int comp_time[20],turn_ar_time[20],wait_time[20];
    //no special variable is required for response time as waiting time and response time are equal in
preemptive
    int avg_tat,avg_wt;
    printf("Enter no. of processes (Max 20):");
    scanf("%d",&n);

    int i,j;
    for(i=0;i<n;i++){
        printf("Enter Arrival Time <space> Burst Time (P%d):",i+1);
        scanf("%d %d",&arr_time[i],&burst_time[i]);
    }
    int visit[20] = {0};

    for(i=0;i<n-1;i++){
        for(j=0;j<n-i-1;j++){
            if(arr_time[j]>arr_time[j+1]){
                //swap in one line
                burst_time[j]=burst_time[j]+burst_time[j+1]-(burst_time[j+1]=burst_time[j]);
                arr_time[j]=arr_time[j]+arr_time[j+1]-(arr_time[j+1]=arr_time[j]);
            }
        }
    }

    int time=arr_time[0];
    int min=0;
    //printf("|%d|",time);
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            if(visit[j]==0 && arr_time[j]<=time){
                min=j;
                break;
            }
        }
        for(j=0;j<n;j++){
            if(visit[j]==0 && arr_time[j]<=time && burst_time[min]>burst_time[j])
                min=j;
        }
        time+=burst_time[min];
        //printf("|P%d|%d|",Pid[min],time);
        visit[min]=1;
        comp_time[min]=time;
        turn_ar_time[min]=comp_time[min]-arr_time[min];
        wait_time[min]=turn_ar_time[min]-burst_time[min];
    }
    printf("\nShortest Job Next/First Scheduling Algorithm\n");
    printf("P(ID)\tArrival Time\tBurst Time\tCompletion Time\tTurn-around Time\tWaiting Time\tResponse
Time\n");
    for(i=0;i<n;i++){

printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d\t\t%d\n",i+1,arr_time[i],burst_time[i],comp_time[i],turn_ar
_time[i],wait_time[i],wait_time[i] );
        avg_wt+=wait_time[i];
        avg_tat+=turn_ar_time[i];
    }

    printf("Average Turn-around Time: %d\nAverage Waiting Time: %d\n",avg_tat/n,avg_wt/n);

    return 0;

}
```

Round Robin

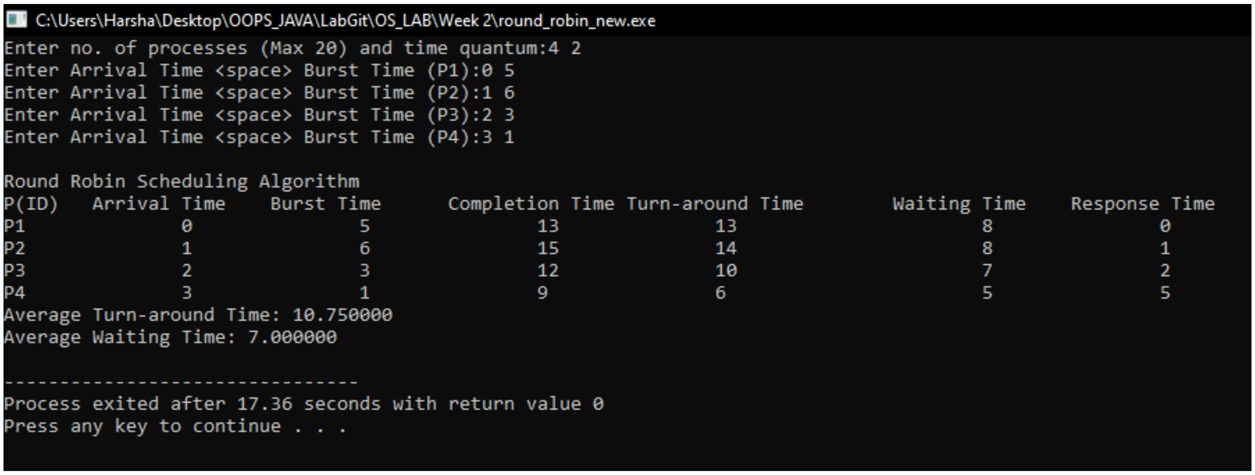
Problem Statement:

Implementation of Round Robin CPU Scheduling algorithm in C language.

Input: Number of Processes.
Arrival and Burst times of all processes

Output: Completion Time, Turn-around time, Waiting time, Response time, Average turn-around time and Average Waiting time

Execution Screenshot:

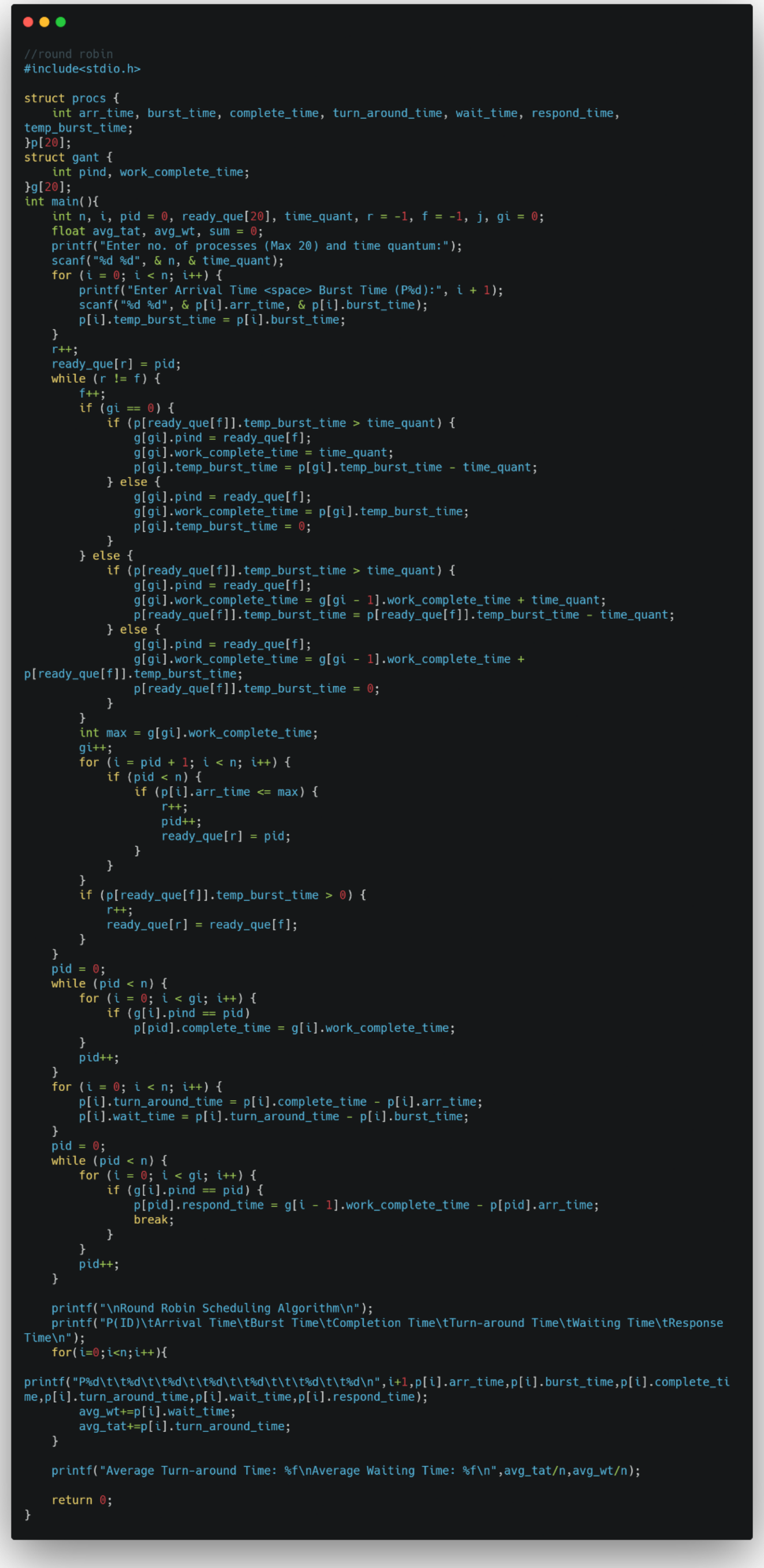

C:\Users\Harsha\Desktop\OOPS_JAVA\LabGit\OS_LAB\Week 2\round_robin_new.exe

```
Enter no. of processes (Max 20) and time quantum:4 2
Enter Arrival Time <space> Burst Time (P1):0 5
Enter Arrival Time <space> Burst Time (P2):1 6
Enter Arrival Time <space> Burst Time (P3):2 3
Enter Arrival Time <space> Burst Time (P4):3 1

Round Robin Scheduling Algorithm
P(ID)   Arrival Time    Burst Time      Completion Time Turn-around Time        Waiting Time    Response Time
P1              0               5               13              13                      8               0
P2              1               6               15              14                      8               1
P3              2               3               12              10                      7               2
P4              3               1               9               6                       5               5
Average Turn-around Time: 10.750000
Average Waiting Time: 7.000000

--------------------------------
Process exited after 17.36 seconds with return value 0
Press any key to continue . . .
```

Code: Click on below image to inspect the code



```c
//round robin
#include<stdio.h>

struct procs {
    int arr_time, burst_time, complete_time, turn_around_time, wait_time, respond_time,
temp_burst_time;
}p[20];
struct gant {
    int pind, work_complete_time;
}g[20];
int main(){
    int n, i, pid = 0, ready_que[20], time_quant, r = -1, f = -1, j, gi = 0;
    float avg_tat, avg_wt, sum = 0;
    printf("Enter no. of processes (Max 20) and time quantum:");
    scanf("%d %d", & n, & time_quant);
    for (i = 0; i < n; i++) {
        printf("Enter Arrival Time <space> Burst Time (P%d):", i + 1);
        scanf("%d %d", & p[i].arr_time, & p[i].burst_time);
        p[i].temp_burst_time = p[i].burst_time;
    }
    r++;
    ready_que[r] = pid;
    while (r != f) {
        f++;
        if (gi == 0) {
            if (p[ready_que[f]].temp_burst_time > time_quant) {
                g[gi].pind = ready_que[f];
                g[gi].work_complete_time = time_quant;
                p[gi].temp_burst_time = p[gi].temp_burst_time - time_quant;
            } else {
                g[gi].pind = ready_que[f];
                g[gi].work_complete_time = p[gi].temp_burst_time;
                p[gi].temp_burst_time = 0;
            }
        } else {
            if (p[ready_que[f]].temp_burst_time > time_quant) {
                g[gi].pind = ready_que[f];
                g[gi].work_complete_time = g[gi - 1].work_complete_time + time_quant;
                p[ready_que[f]].temp_burst_time = p[ready_que[f]].temp_burst_time - time_quant;
            } else {
                g[gi].pind = ready_que[f];
                g[gi].work_complete_time = g[gi - 1].work_complete_time +
p[ready_que[f]].temp_burst_time;
                p[ready_que[f]].temp_burst_time = 0;
            }
        }
        int max = g[gi].work_complete_time;
        gi++;
        for (i = pid + 1; i < n; i++) {
            if (pid < n) {
                if (p[i].arr_time <= max) {
                    r++;
                    pid++;
                    ready_que[r] = pid;
                }
            }
        }
        if (p[ready_que[f]].temp_burst_time > 0) {
            r++;
            ready_que[r] = ready_que[f];
        }
    }
    pid = 0;
    while (pid < n) {
        for (i = 0; i < gi; i++) {
            if (g[i].pind == pid)
                p[pid].complete_time = g[i].work_complete_time;
        }
        pid++;
    }
    for (i = 0; i < n; i++) {
        p[i].turn_around_time = p[i].complete_time - p[i].arr_time;
        p[i].wait_time = p[i].turn_around_time - p[i].burst_time;
    }
    pid = 0;
    while (pid < n) {
        for (i = 0; i < gi; i++) {
            if (g[i].pind == pid) {
                p[pid].respond_time = g[i - 1].work_complete_time - p[pid].arr_time;
                break;
            }
        }
        pid++;
    }

    printf("\nRound Robin Scheduling Algorithm\n");
    printf("P(ID)\tArrival Time\tBurst Time\tCompletion Time\tTurn-around Time\tWaiting Time\tResponse
Time\n");
    for(i=0;i<n;i++){

printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d\t\t%d\n",i+1,p[i].arr_time,p[i].burst_time,p[i].complete_ti
me,p[i].turn_around_time,p[i].wait_time,p[i].respond_time);
        avg_wt+=p[i].wait_time;
        avg_tat+=p[i].turn_around_time;
    }

    printf("Average Turn-around Time: %f\nAverage Waiting Time: %f\n",avg_tat/n,avg_wt/n);

    return 0;
}
```

Non-preemptive Priority Based

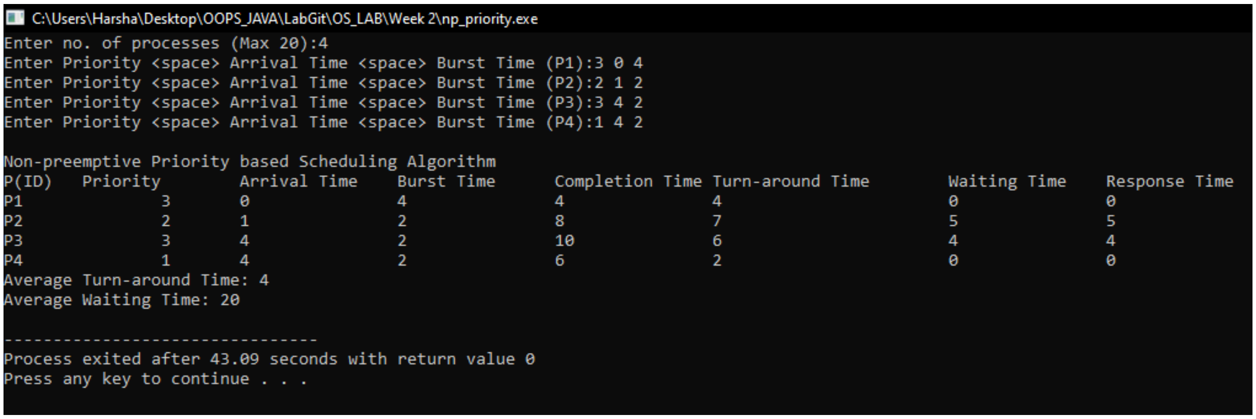
Problem Statement:

Implementation of Non-preemptive priority based CPU Scheduling algorithm in C language.

Input: Number of Processes.
Priority, Arrival time and Burst time of all processes

Output: Completion Time, Turn-around time, Waiting time, Response time, Average turn-around time and Average Waiting time

Execution Screenshot:



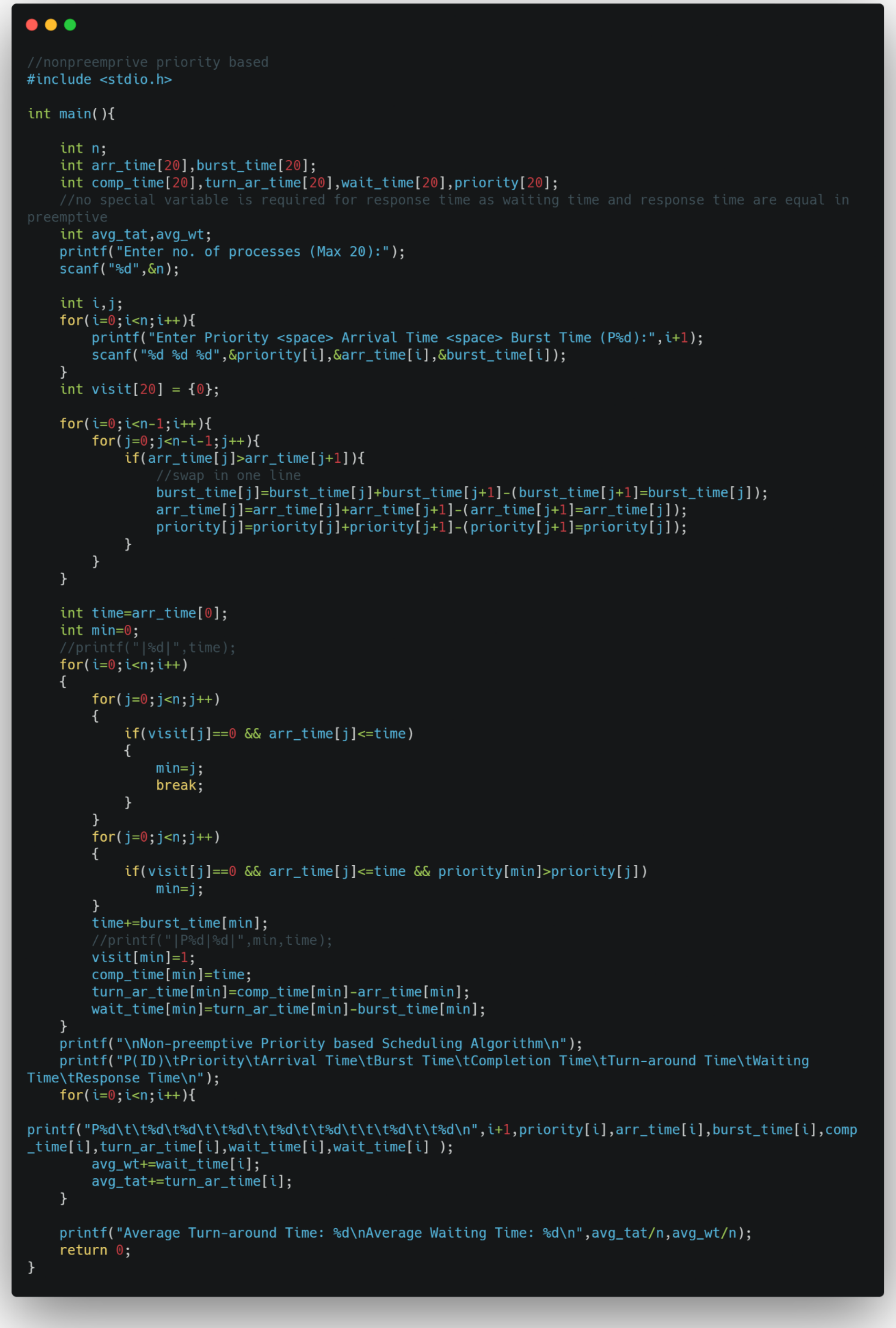
C:\Users\Harsha\Desktop\OOPS_JAVA\LabGit\OS_LAB\Week 2\np_priority.exe

```
Enter no. of processes (Max 20):4
Enter Priority <space> Arrival Time <space> Burst Time (P1):3 0 4
Enter Priority <space> Arrival Time <space> Burst Time (P2):2 1 2
Enter Priority <space> Arrival Time <space> Burst Time (P3):3 4 2
Enter Priority <space> Arrival Time <space> Burst Time (P4):1 4 2

Non-preemptive Priority based Scheduling Algorithm
P(ID)   Priority        Arrival Time    Burst Time      Completion Time Turn-around Time        Waiting Time    Response Time
P1              3       0               4               4               4                       0               0
P2              2       1               2               8               7                       5               5
P3              3       4               2               10              6                       4               4
P4              1       4               2               6               2                       0               0
Average Turn-around Time: 4
Average Waiting Time: 20

--------------------------------
Process exited after 43.09 seconds with return value 0
Press any key to continue . . .
```

Code: Click on below image to inspect the code.



```c
//nonpreemprive priority based
#include <stdio.h>

int main(){

    int n;
    int arr_time[20],burst_time[20];
    int comp_time[20],turn_ar_time[20],wait_time[20],priority[20];
    //no special variable is required for response time as waiting time and response time are equal in
preemptive
    int avg_tat,avg_wt;
    printf("Enter no. of processes (Max 20):");
    scanf("%d",&n);

    int i,j;
    for(i=0;i<n;i++){
        printf("Enter Priority <space> Arrival Time <space> Burst Time (P%d):",i+1);
        scanf("%d %d %d",&priority[i],&arr_time[i],&burst_time[i]);
    }
    int visit[20] = {0};

    for(i=0;i<n-1;i++){
        for(j=0;j<n-i-1;j++){
            if(arr_time[j]>arr_time[j+1]){
                //swap in one line
                burst_time[j]=burst_time[j]+burst_time[j+1]-(burst_time[j+1]=burst_time[j]);
                arr_time[j]=arr_time[j]+arr_time[j+1]-(arr_time[j+1]=arr_time[j]);
                priority[j]=priority[j]+priority[j+1]-(priority[j+1]=priority[j]);
            }
        }
    }

    int time=arr_time[0];
    int min=0;
    //printf("|%d|",time);
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            if(visit[j]==0 && arr_time[j]<=time)
            {
                min=j;
                break;
            }
        }
        for(j=0;j<n;j++)
        {
            if(visit[j]==0 && arr_time[j]<=time && priority[min]>priority[j])
                min=j;
        }
        time+=burst_time[min];
        //printf("|P%d|%d|",min,time);
        visit[min]=1;
        comp_time[min]=time;
        turn_ar_time[min]=comp_time[min]-arr_time[min];
        wait_time[min]=turn_ar_time[min]-burst_time[min];
    }
    printf("\nNon-preemptive Priority based Scheduling Algorithm\n");
    printf("P(ID)\tPriority\tArrival Time\tBurst Time\tCompletion Time\tTurn-around Time\tWaiting
Time\tResponse Time\n");
    for(i=0;i<n;i++){

printf("P%d\t\t%d\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d\t\t%d\n",i+1,priority[i],arr_time[i],burst_time[i],comp
_time[i],turn_ar_time[i],wait_time[i],wait_time[i] );
        avg_wt+=wait_time[i];
        avg_tat+=turn_ar_time[i];
    }

    printf("Average Turn-around Time: %d\nAverage Waiting Time: %d\n",avg_tat/n,avg_wt/n);
    return 0;
}
```

<u>Preemptive Priority Based</u>

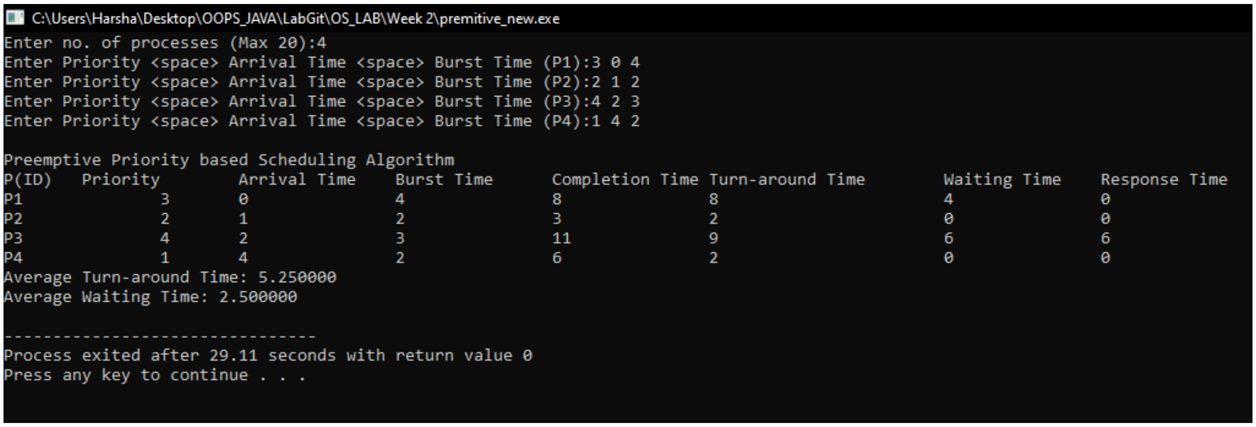
Problem Statement:

Implementation of Preemptive priority based CPU Scheduling algorithm in C language.

Input: Number of Processes.
Priority, Arrival time and Burst time of all processes

Output: Completion Time, Turn-around time, Waiting time, Response time, Average turn-around time and Average Waiting time

Execution Screenshot:



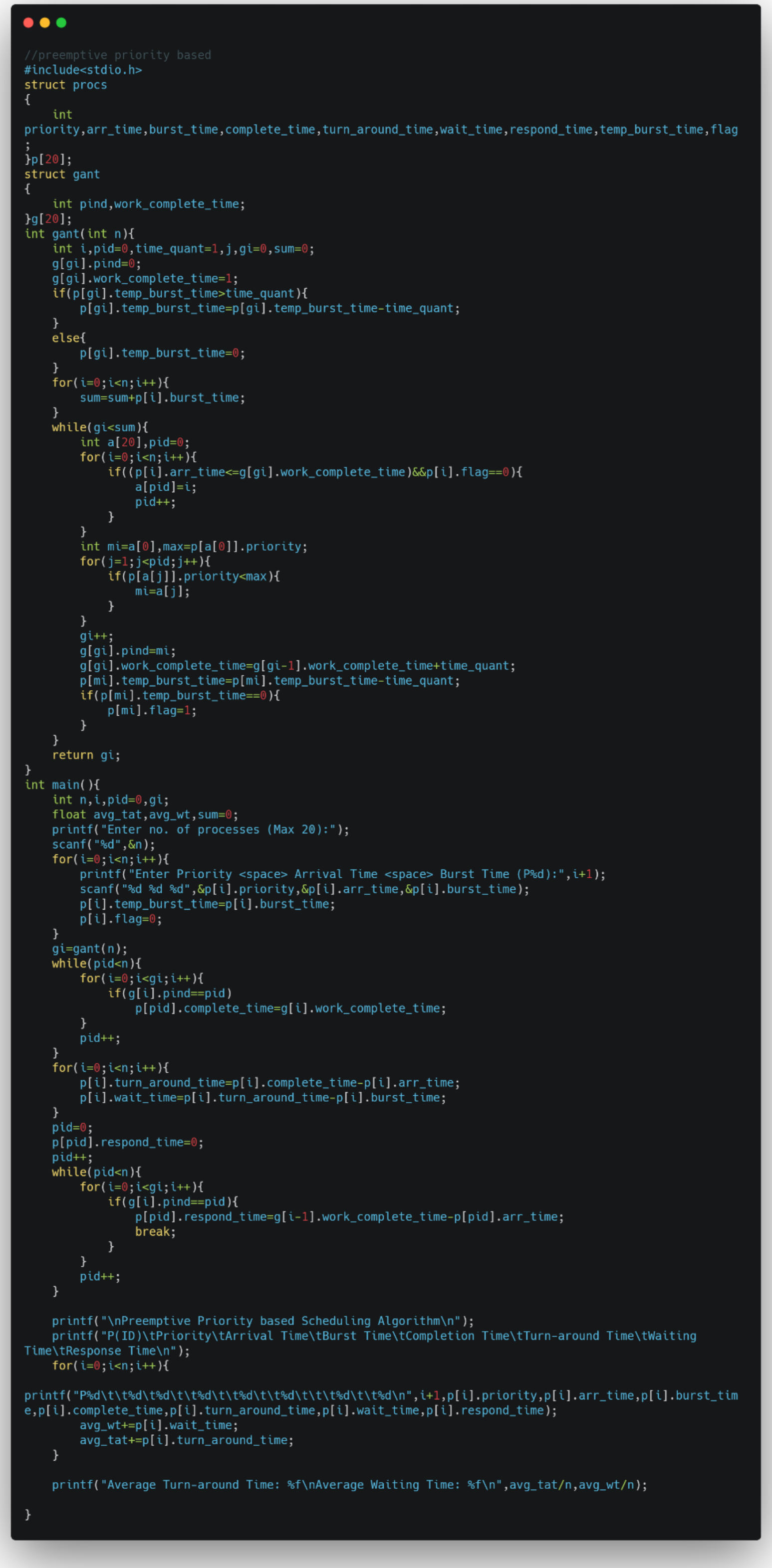
C:\Users\Harsha\Desktop\OOPS_JAVA\LabGit\OS_LAB\Week 2\premitive_new.exe

```
Enter no. of processes (Max 20):4
Enter Priority <space> Arrival Time <space> Burst Time (P1):3 0 4
Enter Priority <space> Arrival Time <space> Burst Time (P2):2 1 2
Enter Priority <space> Arrival Time <space> Burst Time (P3):4 2 3
Enter Priority <space> Arrival Time <space> Burst Time (P4):1 4 2

Preemptive Priority based Scheduling Algorithm
P(ID)   Priority        Arrival Time    Burst Time      Completion Time Turn-around Time        Waiting Time    Response Time
P1              3       0               4               8               8                       4               0
P2              2       1               2               3               2                       0               0
P3              4       2               3               11              9                       6               6
P4              1       4               2               6               2                       0               0
Average Turn-around Time: 5.250000
Average Waiting Time: 2.500000

--------------------------------
Process exited after 29.11 seconds with return value 0
Press any key to continue . . .
```

Code: Click on below image to inspect the code.



```
//preemptive priority based
#include<stdio.h>
struct procs
{
    int
priority,arr_time,burst_time,complete_time,turn_around_time,wait_time,respond_time,temp_burst_time,flag
;
}p[20];
struct gant
{
    int pind,work_complete_time;
}g[20];
int gant(int n){
    int i,pid=0,time_quant=1,j,gi=0,sum=0;
    g[gi].pind=0;
    g[gi].work_complete_time=1;
    if(p[gi].temp_burst_time>time_quant){
        p[gi].temp_burst_time=p[gi].temp_burst_time-time_quant;
    }
    else{
        p[gi].temp_burst_time=0;
    }
    for(i=0;i<n;i++){
        sum=sum+p[i].burst_time;
    }
    while(gi<sum){
        int a[20],pid=0;
        for(i=0;i<n;i++){
            if((p[i].arr_time<=g[gi].work_complete_time)&&p[i].flag==0){
                a[pid]=i;
                pid++;
            }
        }
        int mi=a[0],max=p[a[0]].priority;
        for(j=1;j<pid;j++){
            if(p[a[j]].priority<max){
                mi=a[j];
            }
        }
        gi++;
        g[gi].pind=mi;
        g[gi].work_complete_time=g[gi-1].work_complete_time+time_quant;
        p[mi].temp_burst_time=p[mi].temp_burst_time-time_quant;
        if(p[mi].temp_burst_time==0){
            p[mi].flag=1;
        }
    }
    return gi;
}
int main(){
    int n,i,pid=0,gi;
    float avg_tat,avg_wt,sum=0;
    printf("Enter no. of processes (Max 20):");
    scanf("%d",&n);
    for(i=0;i<n;i++){
        printf("Enter Priority <space> Arrival Time <space> Burst Time (P%d):",i+1);
        scanf("%d %d %d",&p[i].priority,&p[i].arr_time,&p[i].burst_time);
        p[i].temp_burst_time=p[i].burst_time;
        p[i].flag=0;
    }
    gi=gant(n);
    while(pid<n){
        for(i=0;i<gi;i++){
            if(g[i].pind==pid)
                p[pid].complete_time=g[i].work_complete_time;
        }
        pid++;
    }
    for(i=0;i<n;i++){
        p[i].turn_around_time=p[i].complete_time-p[i].arr_time;
        p[i].wait_time=p[i].turn_around_time-p[i].burst_time;
    }
    pid=0;
    p[pid].respond_time=0;
    pid++;
    while(pid<n){
        for(i=0;i<gi;i++){
            if(g[i].pind==pid){
                p[pid].respond_time=g[i-1].work_complete_time-p[pid].arr_time;
                break;
            }
        }
        pid++;
    }

    printf("\nPreemptive Priority based Scheduling Algorithm\n");
    printf("P(ID)\tPriority\tArrival Time\tBurst Time\tCompletion Time\tTurn-around Time\tWaiting
Time\tResponse Time\n");
    for(i=0;i<n;i++){

printf("P%d\t\t%d\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d\t\t%d\n",i+1,p[i].priority,p[i].arr_time,p[i].burst_tim
e,p[i].complete_time,p[i].turn_around_time,p[i].wait_time,p[i].respond_time);
        avg_wt+=p[i].wait_time;
        avg_tat+=p[i].turn_around_time;
    }

    printf("Average Turn-around Time: %f\nAverage Waiting Time: %f\n",avg_tat/n,avg_wt/n);

}
```