ACADEMIC TASK REPORT

NAME: SARIKA SINGH

REGISTRATION NUMBER: 11801556

EMAIL I'D: sarikasingh0852@gmail.com

GitHub : https://github.com/SharmaRithik/scheduling-program-to-implements-a-Queue

SECTION: K18CJ

Test Case

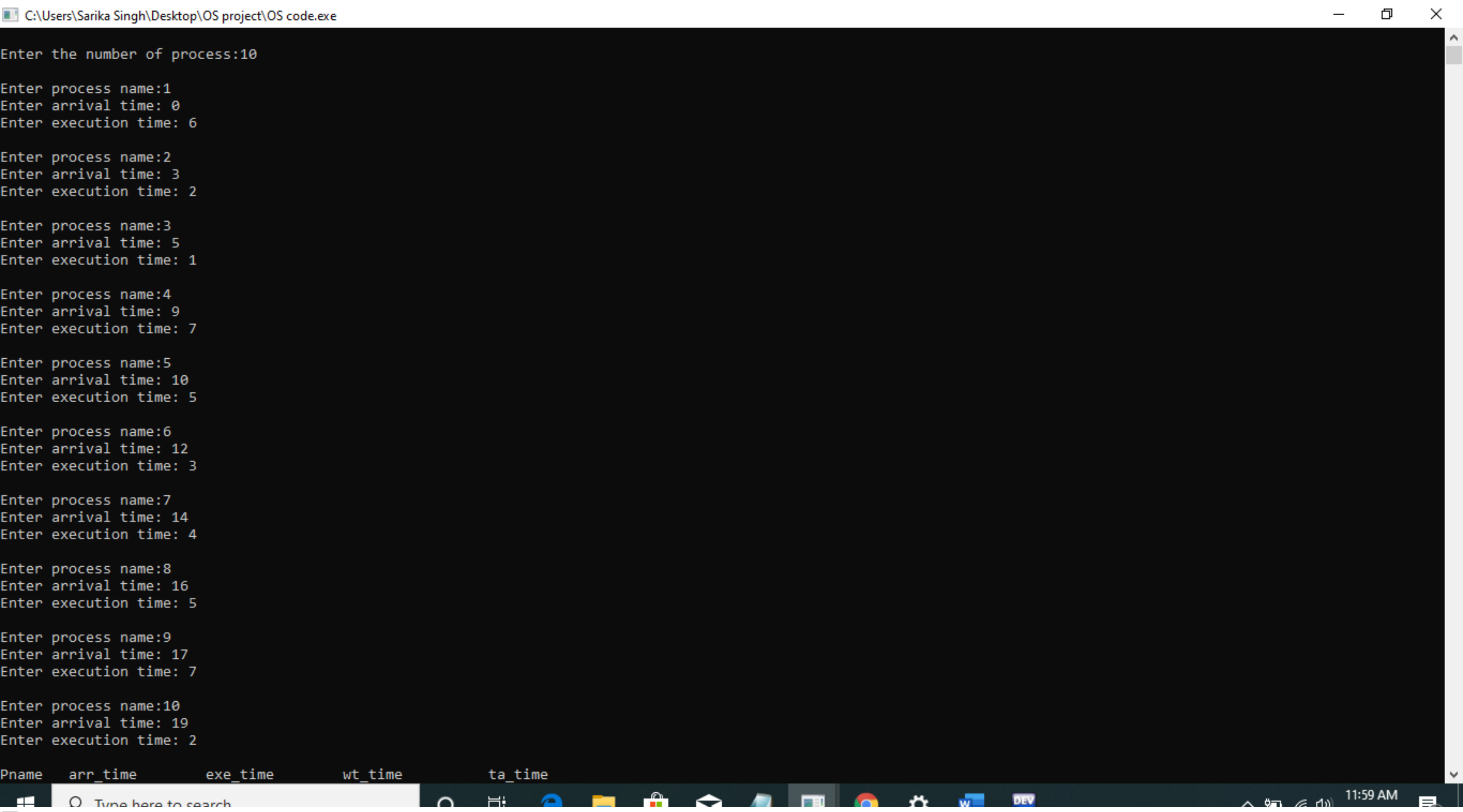
Sample Input:

Process	Arrival time	Burst time
P_1	0	6
P_2	3	2
P_3	5	1
P_4	9	7
P_5	10	5
P_6	12	3
P_7	14	4
P_8	16	5
P_9	17	7
P_{10}	19	2

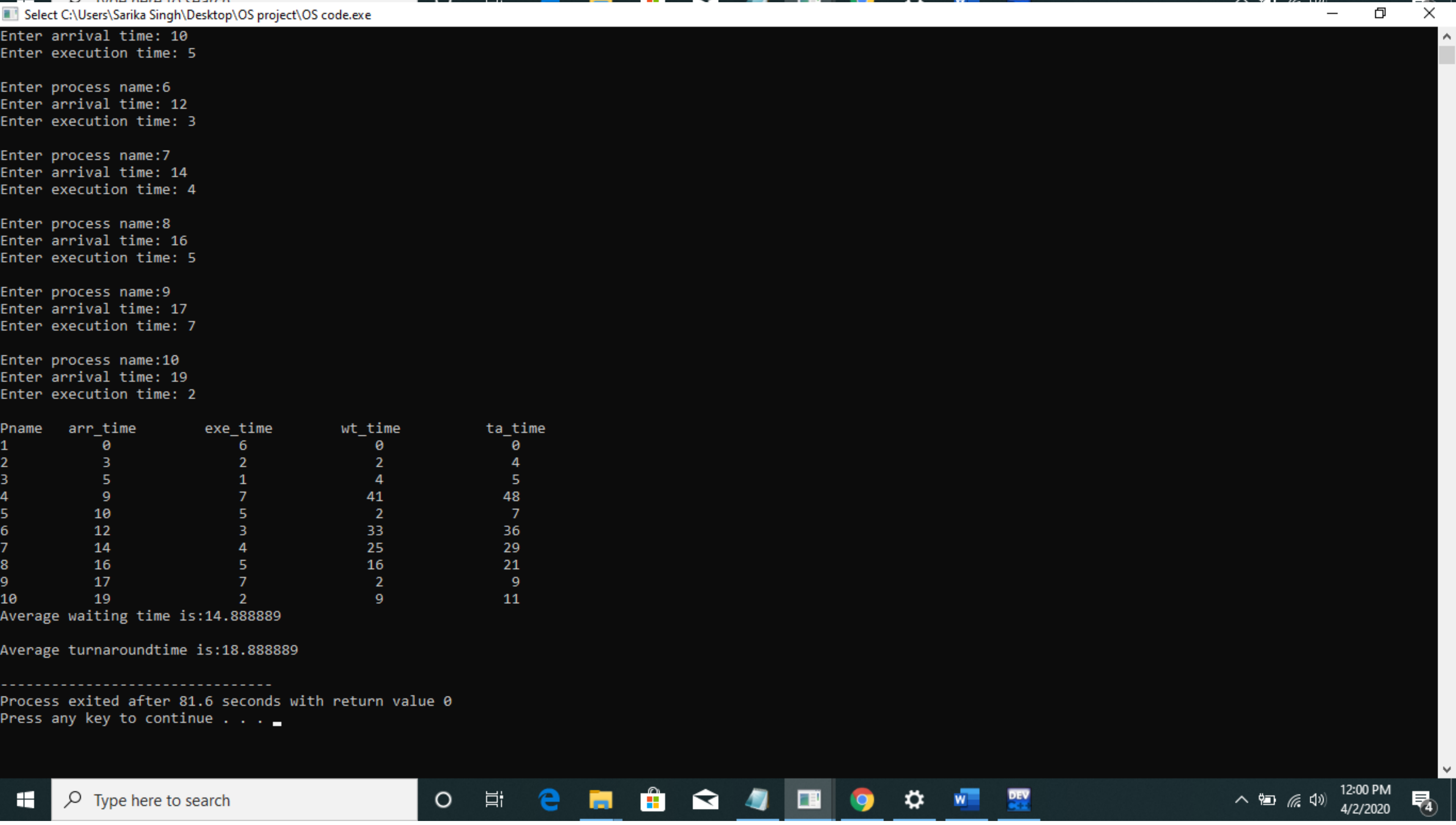
Expected Output:

Process	Arrival time	Execution time	Turnaround time	Waiting time
P_1	0	6	0	0
P_2	3	2	4	2
P_3	5	1	5	4
P_4	9	7	48	41
P_5	10	5	7	2
P_6	12	3	36	33
P_7	14	4	29	25
P_8	16	5	21	16
P_9	17	7	9	2
P_{10}	19	2	11	9

Average Turnaround time: 18.888889
Average Waiting time: 14.888889


C:\Users\Sarika Singh\Desktop\OS project\OS code.exe

```
Enter the number of process:10

Enter process name:1
Enter arrival time: 0
Enter execution time: 6

Enter process name:2
Enter arrival time: 3
Enter execution time: 2

Enter process name:3
Enter arrival time: 5
Enter execution time: 1

Enter process name:4
Enter arrival time: 9
Enter execution time: 7

Enter process name:5
Enter arrival time: 10
Enter execution time: 5

Enter process name:6
Enter arrival time: 12
Enter execution time: 3

Enter process name:7
Enter arrival time: 14
Enter execution time: 4

Enter process name:8
Enter arrival time: 16
Enter execution time: 5

Enter process name:9
Enter arrival time: 17
Enter execution time: 7

Enter process name:10
Enter arrival time: 19
Enter execution time: 2

Pname   arr_time        exe_time        wt_time          ta_time
```


Select C:\Users\Sarika Singh\Desktop\OS project\OS code.exe

```
Enter arrival time: 10
Enter execution time: 5

Enter process name:6
Enter arrival time: 12
Enter execution time: 3

Enter process name:7
Enter arrival time: 14
Enter execution time: 4

Enter process name:8
Enter arrival time: 16
Enter execution time: 5

Enter process name:9
Enter arrival time: 17
Enter execution time: 7

Enter process name:10
Enter arrival time: 19
Enter execution time: 2

Pname   arr_time        exe_time        wt_time          ta_time
1           0               6               0                0
2           3               2               2                4
3           5               1               4                5
4           9               7              41               48
5          10               5               2                7
6          12               3              33               36
7          14               4              25               29
8          16               5              16               21
9          17               7               2                9
10         19               2               9               11
Average waiting time is:14.888889

Average turnaroundtime is:18.888889

--------------------------------
Process exited after 81.6 seconds with return value 0
Press any key to continue . . .
```

12:00 PM
4/2/2020

Algorithm

Step -1: Declare array et[ ], arrt[ ], st[ ], ft[ ], wt[ ], ta[ ], et_copy[ ] pn[ ][ ], t[ ] and n, i, j, temp, avgwaitt, avgturna, totwt=0, totta=0

Step -2: Take input number of process in n.

Step -3: Repeat for int I = 0,1,2…. n-1

. Take arrival time and burst time of the process as input.

Step -4: Repeat step -5 to step - 7 i=0 to n-1

Step -5: Repeat step -6 to step - 7 j=0 to n-1

```
Step -6:   if(et[i]<et[j])

Step -7  temp=arrt[i];
         arrt[i]=arrt[j];
         arrt[j]=temp;
         temp=et[i];
         et[i]=et[j];
         et[j]=temp;
         strcpy(t,pn[i]);
         strcpy(pn[i],pn[j]);
         strcpy(pn[j],t);

Step- 8 Repeat step -9i=0 to n-1

Step -9   if(i==0)
               st[i]=arrt[i];
          else
               st[i]=ft[i-1];
               wt[i]=st[i]-arrt[i];
```

```
        ft[i]=st[i]+et[i];

        ta[i]=ft[i]-arrt[i];

        totwt+=wt[i];

        totta+=ta[i];
```

Step -10: Repeat for int I = 0,1,2…. n-1

. **A)** Calculate average waiting time and average burst time.

. **B)** Print all the Data in the form of table.

Step -11: Print Average waiting time and turnaround time

Time Complexity

Step -1 and Step -2 having time complexity constant time

complexity : O(1). Step -3 having time complexity no of

process : O(n).

Step -4 to Step-7 having time complexity two

iteration : O(n ^{2}). Step -8 to Step 9 having time

complexity : O(n).

Step -10 having time complexity no of

process : O(n).

Step -11 having time complexity : O(1).

Overall Time Complexity is : **O(n ^{2})**

Boundary Condition

* Time taken for checking and arranging the processes according to the shortest job is 2time unit.

* The variables are kept to integers and some values to float to give the output more exactly

* No character value is accepted and will result in dumping of the code

Problem in terms of Operating System Concept

We are given with a schedular which schedules the job by considering the arrival time of the process where arrival time if given as 0 is discarded or displayed as error. The schedular implements the shortest job first policy, but checks the queue of the processes after every process terminates and time taken for checking and arranging the process according to the shortest job is 2time unit. we need to find:

* Waiting time
* Turnaround time
* Average waiting time
* Average turnaround time

We are provided with the processes, arrival time and burst time.

Code Solution

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<string.h>

int main(){

    int et[20],arrt[10],n,i,j,temp,st[10],ft[10],wt[10],ta[10],et_copy[10];
    int totwt=0,totta=0;
    float avgwaitt,avgturna;
    char pn[10][10],t[10];

    printf("\nEnter the number of process:");
    scanf("%d",&n);
    for(i=0; i<n; i++)
    {
        printf("\nEnter process name:");
        //flushall();
        scanf("%s",pn[i]);
        printf("Enter arrival time: ");
        scanf("%d",&arrt[i]);
        printf("Enter execution time: ");
        scanf("%d",&et[i]);
        et_copy[i] = et[i];
    }
```

```
int current_time = -1;
int curr_job = -1;
int jobs_complete = 0;
int min_remaining_burst = 999999;
int is_sort = 0;
int nnn;
int num_error_free_process = n;

while(jobs_complete!=1){
    jobs_complete = 1;
    for(i=0;i<n;i++){
        //printf("Arrt = %d\n",arrt[i] );

        if(arrt[i] > 0){
            if(arrt[i] <= current_time){
                if(et[i] > 0 && et[i] < min_remaining_burst){
                    curr_job = i;
                    is_sort = 1;
                }
            }
            if(et[i]>0)
                jobs_complete = 0;
        }
    }
    //printf("current_time = %d\n", current_time);
    if(is_sort==1){
        current_time+=2;
        //printf("current_time + sort time = %d\n", current_time);
    }
    //printf("current_job = %d\n", curr_job);
    if(curr_job == -1){
        current_time++;
        continue;
    }
    if(current_time == -1)
        current_time = arrt[curr_job];

    st[curr_job] = current_time;
    current_time += et[curr_job];
    et[curr_job]=0;
    ta[curr_job] = current_time - arrt[curr_job];
    wt[curr_job] = ta[curr_job] - et_copy[curr_job];
    //printf("Job executed, time = %d\n", current_time);
    //scanf("%d", &nnn);
```

```
        curr_job = -1;
    }


    for(i=0;i<n;i++){
        if(arrt[i]<=0){
            ta[i] = 0;
            wt[i] = 0;
            num_error_free_process--;
        }
        totta += ta[i];
        totwt += wt[i];
    }


    avgwaitt=(float)totwt/num_error_free_process;
    avgturna=(float)totta/num_error_free_process;

    printf("\nPname\tarr_time\exc_time\twt_time\t\t ta_time");
    for(i=0; i<n; i++)
        printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],arrt[i],et_copy[i],wt[i],
ta[i]);
    printf("\nAverage waiting time is:%f\n",avgwaitt);
    printf("\nAverage turnaroundtime is:%f\n",avgturna);


    return 0;
}
```