Operating System Assignment (CSE316)

Name: Deepanshu Ujjwal

Reg number: 11807132

Roll number: B-61

Question 1:

Explanation: The code uses multiple features of C language, it has struct, file handling things and memory allocation. The CPU burst is read as a first line in text file CPU_BURST.txt. Then the number is validated if it is positive or not. The P1, P2, P3, P4 and P5 are only taken in consideration if all of them are positive, i.e. the positive Burst Time only, as Burst Time cannot be negative. Program has multiple struct data structures used, use of for and while loop. After successful inputs of the Burst time of the 5 process, the program then calculate the average waiting time and average turn around time.

As mentioned is the problem, SJF is used in the below program and the arrival time for all the processes are considered same.

Shortest Job First (SJF)

1. The process with the lowest execution time is executed first.
2. It can be either preemptive or non preemptive.

Complexity of program: O(logn)

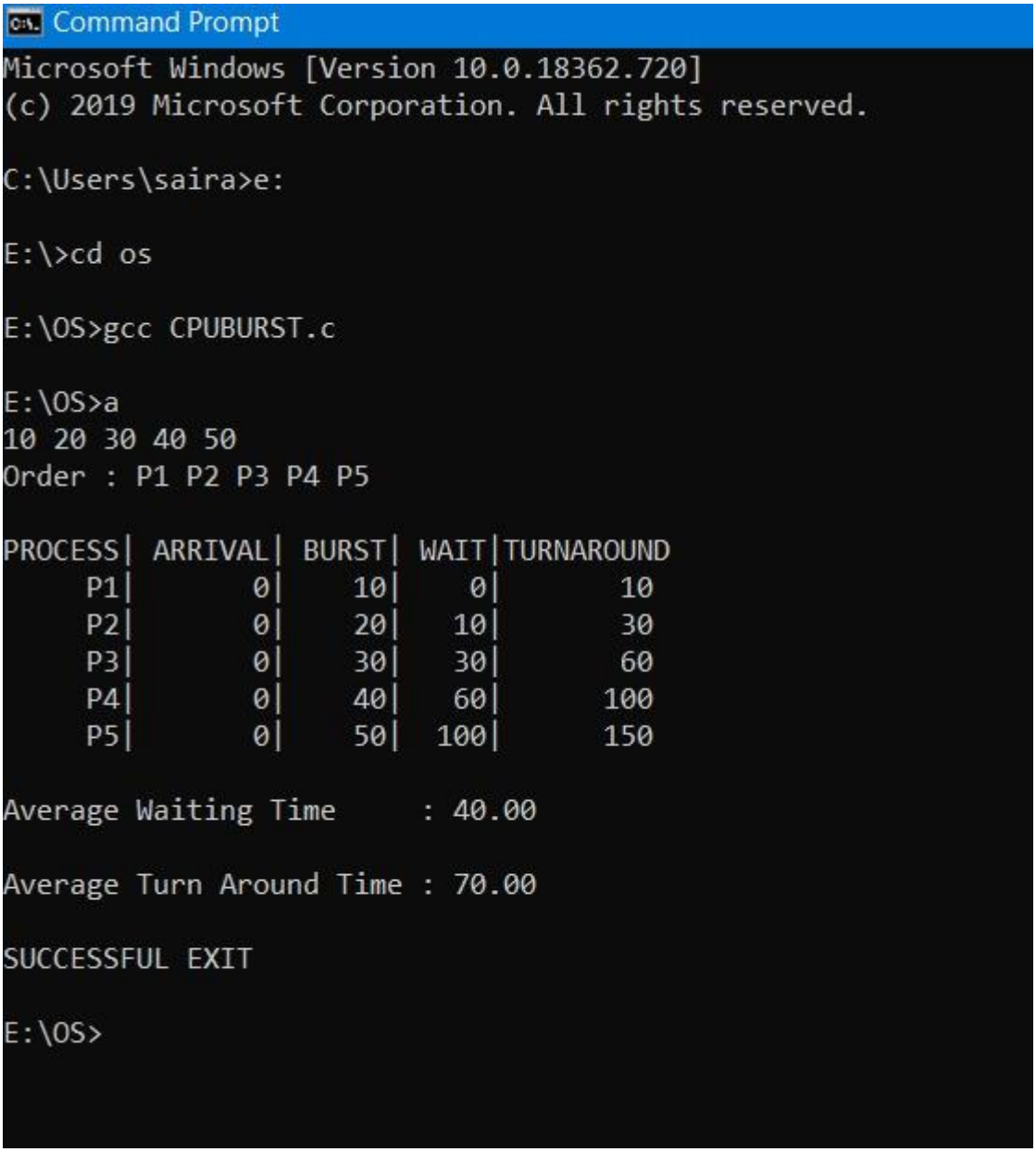
Code:

```
#include<stdi
o.h>
                #include<stdlib.h>
                #include<unistd.h>

                #define FILE_NAME "CPU_BURST.txt"

                struct Process{
                       int at,bt,wt,tat;
                       char name[4];
                };
```

```
struct Process initialize(int at,int bt,int name){
        struct Process X;
        X.bt = bt;
        X.at = at;
        sprintf(X.name,"P%d",name+1);

        return X;
}

int main(){

        FILE *fp = fopen(FILE_NAME,"r");

        if(!fp)
                return -1*printf("FILE OPEN ERROR!\n");

    int d,i,j,count=0;

    int *queue = (int*)malloc(sizeof(int));

    //inputs are space separated integers on a single line of a txt file
located in the same directory
    while(EOF != fscanf(fp,"%d ",&d )){
        printf("%d ",d);
        queue = (int*)realloc(queue,(count+1)*sizeof(int));
        queue[count++] = d;
    }
    fclose(fp);

        //int queue[] = {3,1,3,2,4,5};

        struct Process P[count];

        for(i=0; i<count; i++)
                P[i] = initialize(0,queue[i],i);

        //sort
        for(i=1; i<count; i++){
                for(j=0; j<count-i; j++){
                        if(P[j].bt>P[j+1].bt){
                                struct Process temp = P[j];
                                P[j] = P[j+1];
                                P[j+1] = temp;
                        }
                }
```

```
	}

	//FCFS non-preemptive [same arrival time]
	//after sorting we can apply FCFS which will result in SJF]
	printf("\nOrder : ");

	int elapsed_time=0;
	for(i=0; i<count; i++){
		P[i].wt = elapsed_time;
		P[i].tat= P[i].wt+P[i].bt;
		elapsed_time += P[i].bt;

		printf("%s ",P[i].name);
	}
	//sort again
	for(i=1; i<count; i++){
		for(j=0; j<count-i; j++){
			if(P[j].name[1]>P[j+1].name[1]){
				struct Process temp = P[j];
				P[j] = P[j+1];
				P[j+1] = temp;
			}
		}
	}
	printf("\n\n%7s|%8s|%6s|%5s|%s\n","PROCESS","ARRIVAL","BURST","WAI
T","TURNAROUND");

	int total_wt=0,total_tt=0;

	for(i=0; i<count; i++){
		total_wt+=P[i].wt;
		total_tt+=P[i].tat;

	printf("%7s|%8d|%6d|%5d|%9d\n",P[i].name,P[i].at,P[i].bt,P[i].wt,P
[i].tat);
	}

	printf("\nAverage Waiting Time     : %.2f\n",total_wt*1.0/count);
	printf("\nAverage Turn Around Time : %.2f\n",total_tt*1.0/count);

	return 0*printf("\nSUCCESSFUL EXIT\n");
}
```

Test Case for Question 1:

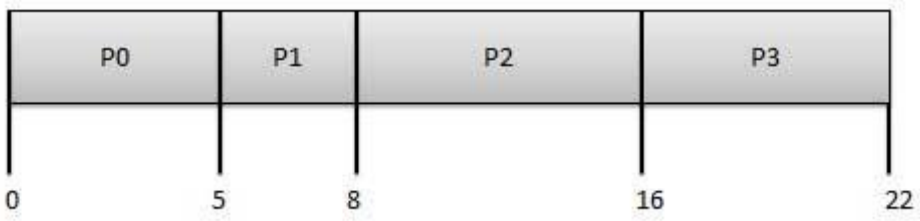


Question 2: A uniprocessor system has n number of CPU intensive processes, each process has its own requirement of CPU burst. The process with lowest CPU burst is given the highest priority. A late arriving higher priority process can preempt a currently running process with lower priority. Simulate a scheduler that is scheduling the processes in such a way that higher priority process is never starved due to the execution of lower priority process. What should be its average waiting time and average turnaround time if no two processes are arriving at same time.

Explanation: The following C program regarding above problem comprises of concept of struct, loops and array data structure.

The description of the problem is following: There is a uniprocessor that takes n number of CPU processes, each process requires multiples resources from the CPU. Each processes has their own Burst time, and the process having the lowest Burst Time is given the highest priority and will preempt the other process of lower priority, if required. The program performs the task according to SJF and then it finally calculates the average waiting time and turnaround time of the computation. Shortest Job First with Non-Preemption and lowest Burst Time criteria is used.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Complexity of program: O(logn)

Code:

```
#include<std
io.h>
                int n;
                struct process
                {

                int p_no;

                int arrival_t,burst_t,ct,wait_t,taround_time,p;

                int flag;
                 }p_list[100];
                 void Sorting()
                 {
                struct process p;

                int i, j;

                for(i=0;i<n-1;i++)

                {

                for(j=i+1;j<n;j++)

                {

                if(p_list[i].arrival_t > p_list[j].arrival_t)

                {

                p = p_list[i];

                p_list[i] = p_list[j];

                p_list[j] = p;

                }

                }

                }
                 }
                 int main()
                 {

                int i,t=0,b_t=0,peak;
```

```
int a[10];

float wait_time = 0, taround_time = 0, avg_w_t=0, avg_taround_time=0;

printf("enter the no. of processes: ");

scanf("%d",&n);

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

{

p_list[i].p_no = i+1;

printf("\nEnter Details For P%d process:-\n", p_list[i].p_no);
printf("Enter Arrival Time: ");
scanf("%d", &p_list[i].arrival_t );
printf("Enter Burst Time: ");
scanf("%d", &p_list[i].burst_t);
p_list[i].flag = 0;
b_t = b_t + p_list[i].burst_t;
}
Sorting();
for(int i=0;i<n;i++)
{
a[i]=p_list[i].burst_t;
}
p_list[9].burst_t = 9999;
for(t = p_list[0].arrival_t; t <= b_t+1;)
{
peak = 9;
for(i=0;i<n;i++)
{
if(p_list[i].arrival_t <= t && p_list[i].burst_t < p_list[peak].burst_t &&
p_list[i].flag != 1)
{
peak = i;
}
if(p_list[peak].burst_t==0 && p_list[i].flag != 1)
{
p_list[i].flag = 1;
p_list[peak].ct=t;p_list[peak].burst_t=9999;
printf("P%d completes in %d\n",p_list[i].p_no,p_list[peak].ct);
}
}
```

```
t++;
(p_list[peak].burst_t)--;
}
for(i=0;i<n;i++)
{
p_list[i].taround_time=(p_list[i].ct)-(p_list[i].arrival_t);
avg_taround_time=avg_taround_time+p_list[i].taround_time;
p_list[i].wait_t=((p_list[i].taround_time)-a[i]);
avg_w_t=avg_w_t+p_list[i].wait_t;
}
printf("PNO\tAT\tCT\tTA\tWTt\n");
for(i=0;i<n;i++)
{
printf("P%d\t%d\t%d\t%d\t%d\n",p_list[i].p_no,p_list[i].arrival_t,p_list[i]
.ct,p_list[i].taround_time
,p_list[i].wait_t);
 }
printf("Average Turn around Time: %f\t\n\n",avg_taround_time);
printf("Average Waiting Time :\t %f\t\n",avg_w_t);
}   //end of program
```

Test Case for Problem 2:

```
enter the no. of processes: 3

Enter Details For P1 process:-
Enter Arrival Time: 1
Enter Burst Time: 3

Enter Details For P2 process:-
Enter Arrival Time: 2
Enter Burst Time: 4

Enter Details For P3 process:-
Enter Arrival Time: 1
Enter Burst Time: 2
P3 completes in 3
P1 completes in 7
P2 completes in 10
PNO     AT      CT      TA      WTt
P1      1       7       6       3
P3      1       3       2       0
P2      2       10      8       4
Average Turn around Time: 16.000000

Average Waiting Time :   7.000000

--------------------------------
Process exited after 25.75 seconds with return value 34
Press any key to continue . . .
```