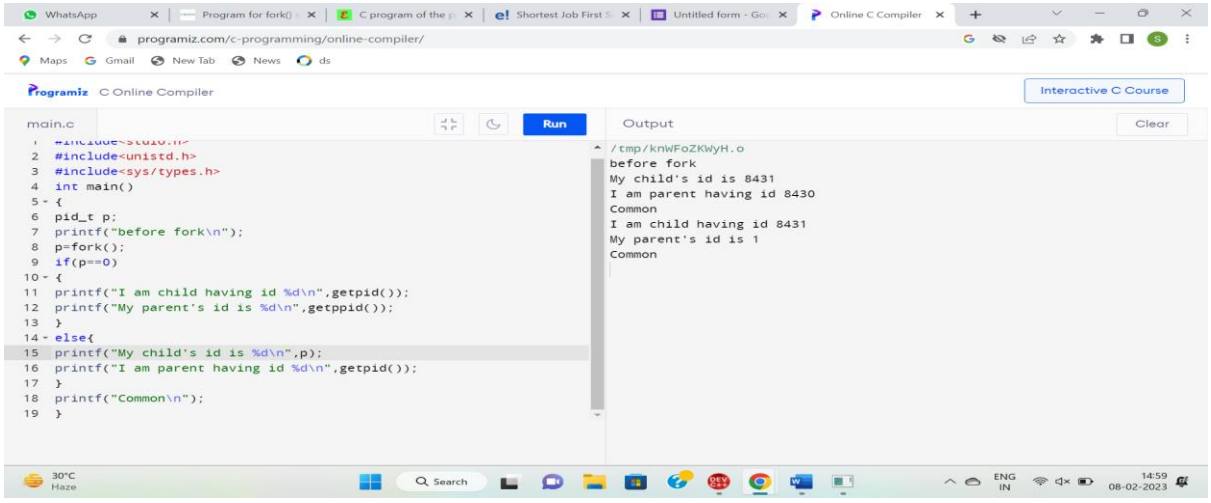
**1.Create a new process by invoking the appropriate system call. Get the process identifier of the currently running process and its respective parent using system calls and display the same using a C program.**

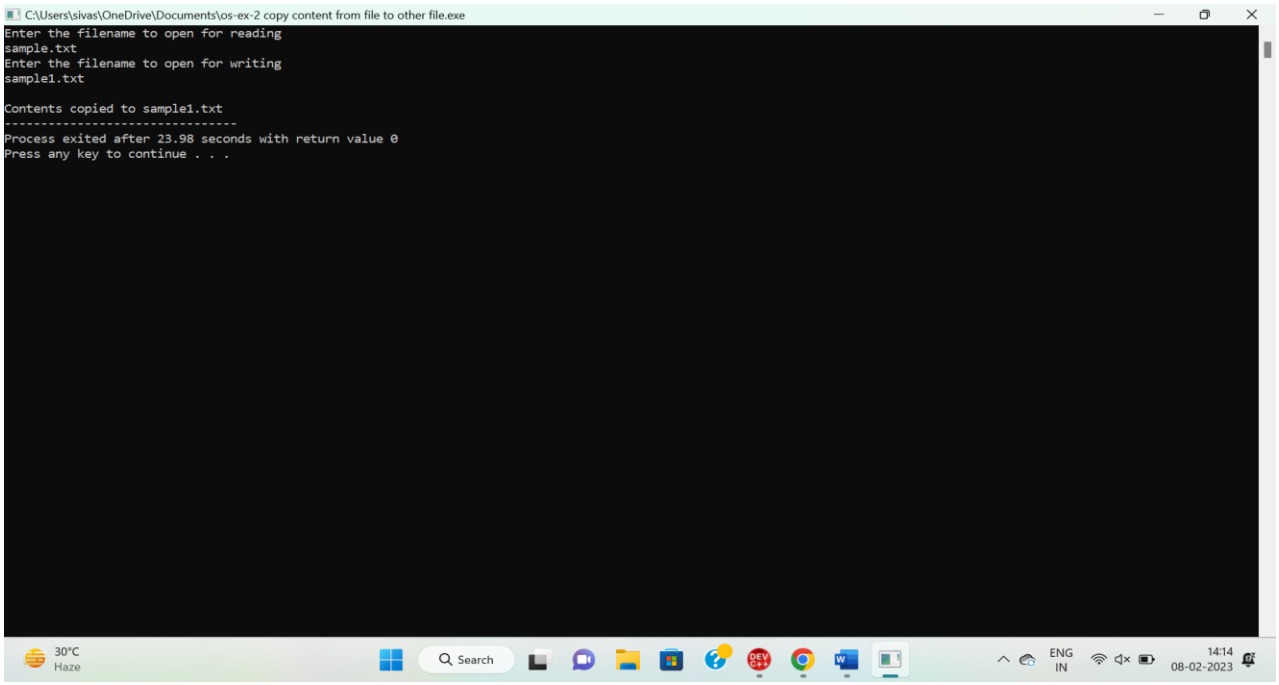
```
#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

int main()

{

pid_t p;

printf("before fork\n");

p=fork();

if(p==0)

{

printf("I am child having id %d\n",getpid());

printf("My parent's id is %d\n",getppid());

}

else{

printf("My child's id is %d\n",p);

printf("I am parent having id %d\n",getpid());

}

printf("Common\n");

}
```



**2. Identify the system calls to copy the content of one file to another and illustrate the same using a C program.**

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	FILE *fptr1,*fptr2;
	char filename[100],c;
	printf("Enter the filename to open for reading \n");
	scanf("%s", filename);
	fptr1 = fopen(filename,"r");
	if (fptr1==NULL)
	{
		printf("Cannot open file %s \n",filename);
		exit(0);
	}
	printf("Enter the filename to open for writing \n");
	scanf("%s",filename);
	fptr2 = fopen(filename,"w");
	if (fptr2 == NULL)
	{
		printf("Cannot open file %s \n",filename);
		exit(0);
	}
	c = fgetc(fptr1);
	while (c!=EOF)
	{
		fputc(c,fptr2);
		c = fgetc(fptr1);
	}
```

```
    printf("\nContents copied to %s",filename);
    fclose(fptr1);
    fclose(fptr2);
    return 0;
}
```



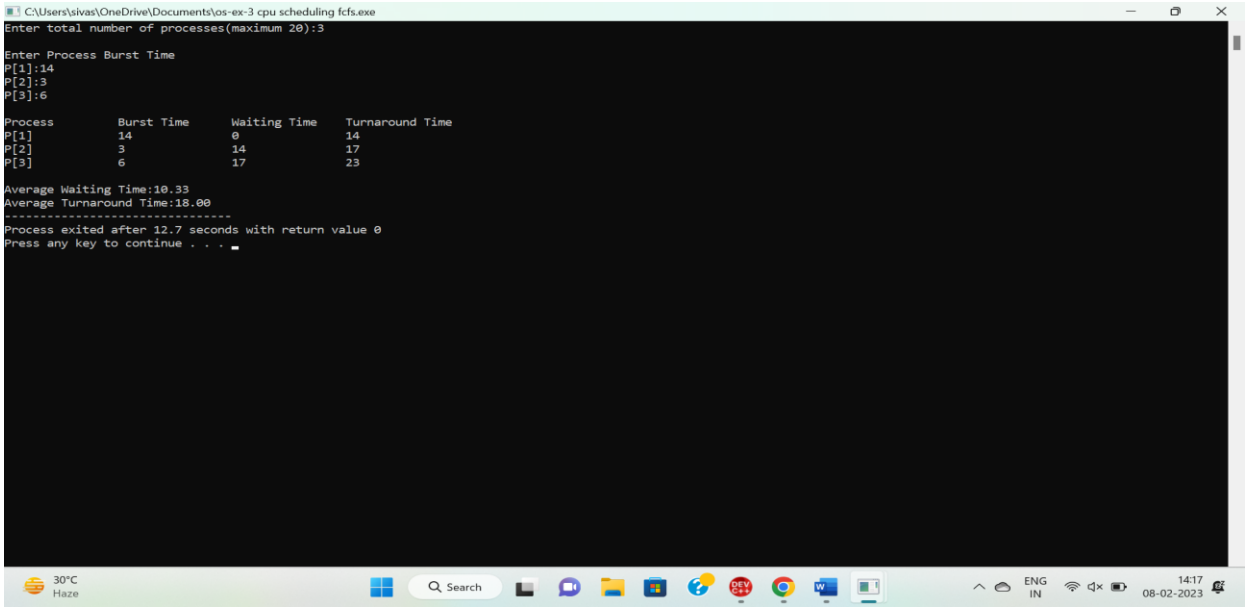

**3. Design a CPU scheduling program with C using First Come First Served technique with the following considerations.**

**a. All processes are activated at time 0.**

**b. Assume that no process waits on I/O devices.**

```
#include<stdio.h>
int main()
{
int n,bt[20],wt[20],tat[20],i,j; float avwt=0,avtat=0;
printf("Enter total number of processes(maximum 20):");
scanf("%d",&n);
printf("\nEnter Process Burst Time\n");
```

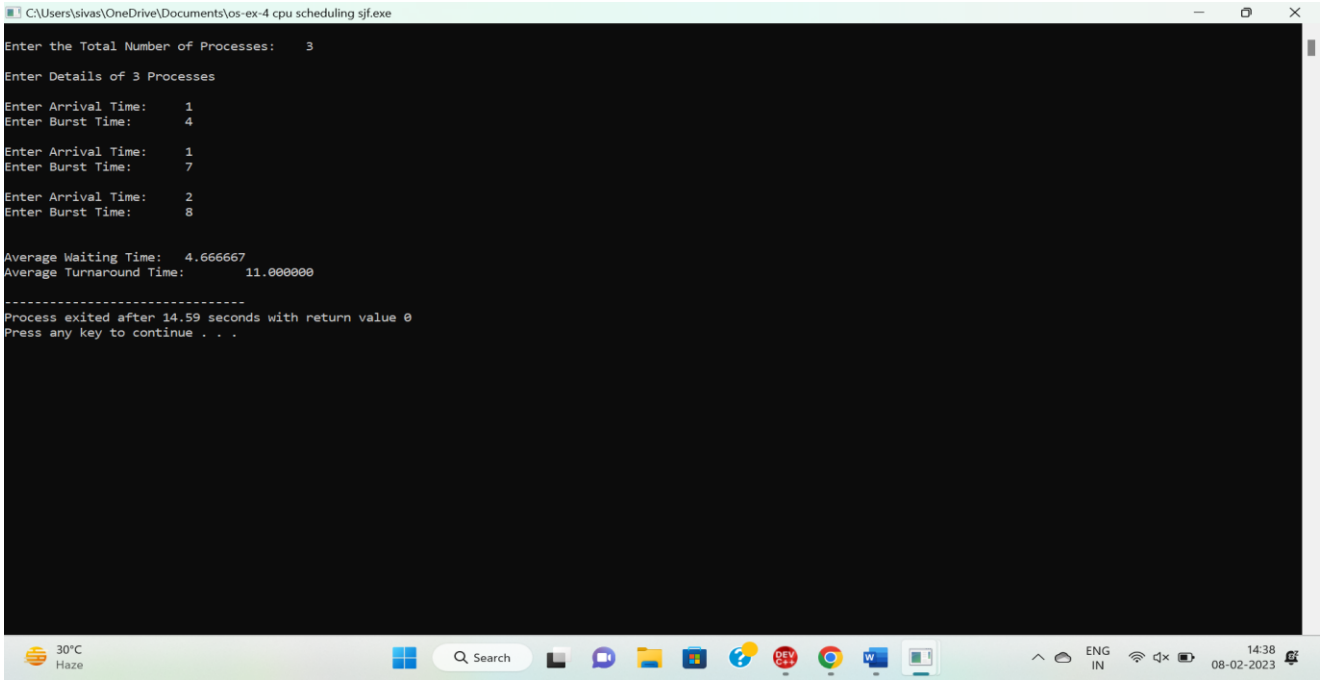
```
for(i=0;i<n;i++)
{
printf("P[%d]:",i+1);
scanf("%d",&bt[i]);
}
 wt[0]=0;
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
}
printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i]; avwt+=wt[i];
avtat+=tat[i];printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
} avwt/=i; avtat/=i;printf("\n\nAverage Waiting Time:%.2f",avwt);
printf("\nAverage Turnaround Time:%.2f",avtat);
return 0;
}
```




**4. Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next.**

```
#include <stdio.h>

int main()

{

    int arrival_time[10], burst_time[10], temp[10];

    int i, smallest, count = 0, time, limit;

    double wait_time = 0, turnaround_time = 0, end;

    float average_waiting_time, average_turnaround_time;

    printf("\nEnter the Total Number of Processes:\t");

    scanf("%d", &limit);

    printf("\nEnter Details of %d Processes\n", limit);

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

    {

        printf("\nEnter Arrival Time:\t");

        scanf("%d", &arrival_time[i]);

        printf("Enter Burst Time:\t");

        scanf("%d", &burst_time[i]);

        temp[i] = burst_time[i];

    }
```

```
    burst_time[9] = 9999;
    for(time = 0; count != limit; time++)
    {
        smallest = 9;
        for(i = 0; i < limit; i++)
        {
            if(arrival_time[i]<=time&&burst_time[i]<burst_time[smallest]&&burst_time[i]>0)
            {
                smallest = i;
            }
        }
        burst_time[smallest]--;
        if(burst_time[smallest] == 0)
        {
            count++;
            end = time + 1;
            wait_time = wait_time + end - arrival_time[smallest] - temp[smallest];
            turnaround_time = turnaround_time + end - arrival_time[smallest];
        }
    }
    average_waiting_time = wait_time / limit;
    average_turnaround_time = turnaround_time / limit;
    printf("\n\nAverage Waiting Time:\t%lf\n", average_waiting_time);
    printf("Average Turnaround Time:\t%lf\n", average_turnaround_time);
    return 0;
}
```

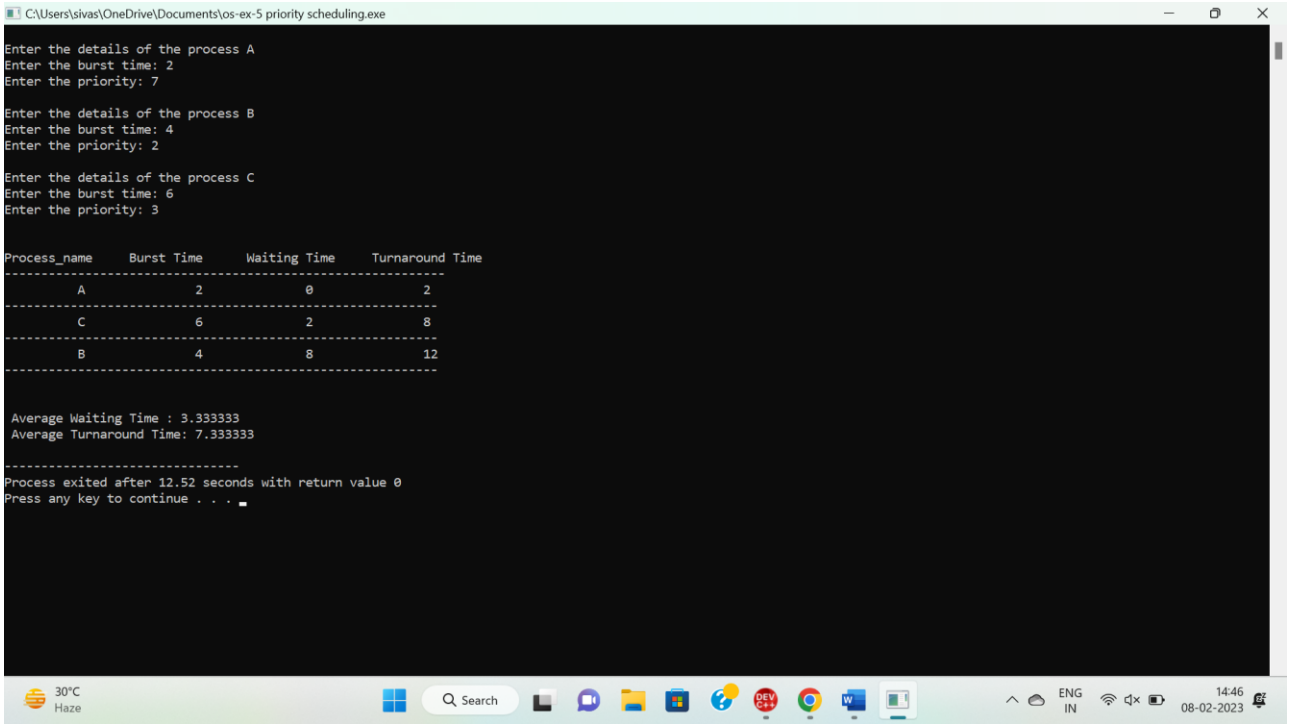



**5.Construct a scheduling program with C that selects the waiting process with the highest priority to execute next**.

```
#include<stdio.h>
struct priority_scheduling
 {
  char process_name;
  int burst_time;
  int waiting_time;
  int turn_around_time;
  int priority;
};
int main()
{
  int number_of_process;
  int total = 0;
  struct priority_scheduling temp_process;
  int ASCII_number = 65;
```

```
int position;
float average_waiting_time;
float average_turnaround_time;
printf("Enter the total number of Processes: ");
scanf("%d", & number_of_process);
struct priority_scheduling process[number_of_process];
printf("\nPlease Enter the  Burst Time and Priority of each process:\n");
for (int i = 0; i < number_of_process; i++)
{
  process[i].process_name = (char) ASCII_number;
  printf("\nEnter the details of the process %c \n", process[i].process_name);
  printf("Enter the burst time: ");
  scanf("%d", & process[i].burst_time);
  printf("Enter the priority: ");
  scanf("%d", & process[i].priority);
  ASCII_number++;
}
for (int i = 0; i < number_of_process; i++)
 {
  position = i;
  for (int j = i + 1; j < number_of_process; j++)
          {
    if (process[j].priority > process[position].priority)
      position = j;
  }
  temp_process = process[i];
  process[i] = process[position];
  process[position] = temp_process;
}
```

```
 process[0].waiting_time = 0;
 for (int i = 1; i < number_of_process; i++)
 {
  process[i].waiting_time = 0;
  for (int j = 0; j < i; j++)
        {
    process[i].waiting_time += process[j].burst_time;
  }
  total += process[i].waiting_time;
 }
 average_waiting_time = (float) total / (float) number_of_process;
 total = 0;
 printf("\n\nProcess_name \t Burst Time \t Waiting Time \t  Turnaround Time\n");
 for (int i = 0; i < number_of_process; i++)
 {
  process[i].turn_around_time = process[i].burst_time + process[i].waiting_time;
  total += process[i].turn_around_time;
  printf("\t  %c \t\t  %d \t\t %d \t\t %d", process[i].process_name, process[i].burst_time,
process[i].waiting_time, process[i].turn_around_time);

 }
 average_turnaround_time = (float) total / (float) number_of_process;
 printf("\n\n Average Waiting Time : %f", average_waiting_time);
 printf("\n Average Turnaround Time: %f\n", average_turnaround_time);
 return 0;
}
```

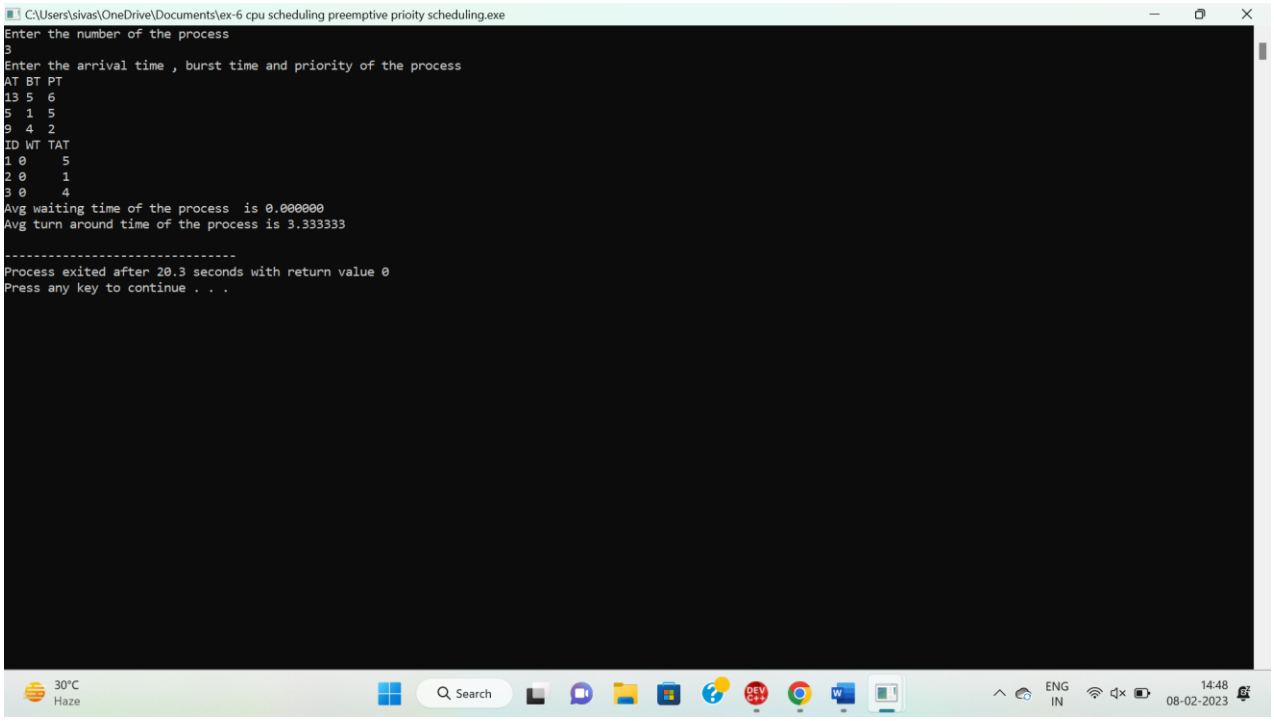



**6.Construct a C program to implement pre-emptive priority scheduling algorithm.**

```
#include<stdio.h>
struct process
{
   int WT,AT,BT,TAT,PT;
};
struct process a[10];
int main()
{
   int n,temp[10],t,count=0,short_p;
   float total_WT=0,total_TAT=0,Avg_WT,Avg_TAT;
   printf("Enter the number of the process\n");
   scanf("%d",&n);
   printf("Enter the arrival time , burst time and priority of the process\n");
   printf("AT BT PT\n");
   for(int i=0;i<n;i++)
```

```
{
   scanf("%d%d%d",&a[i].AT,&a[i].BT,&a[i].PT);
   temp[i]=a[i].BT;
}
a[9].PT=10000;
for(t=0;count!=n;t++)
{
   short_p=9;
   for(int i=0;i<n;i++)
   {
      if(a[short_p].PT>a[i].PT && a[i].AT<=t && a[i].BT>0)
      {
         short_p=i;
      }
   }
   a[short_p].BT=a[short_p].BT-1;
   if(a[short_p].BT==0)
   {
      count++;
      a[short_p].WT=t+1-a[short_p].AT-temp[short_p];
      a[short_p].TAT=t+1-a[short_p].AT;
      total_WT=total_WT+a[short_p].WT;
      total_TAT=total_TAT+a[short_p].TAT;
   }
}
Avg_WT=total_WT/n;
Avg_TAT=total_TAT/n;
printf("ID WT TAT\n");
for(int i=0;i<n;i++)
```

```
    {
        printf("%d %d\t%d\n",i+1,a[i].WT,a[i].TAT);
    }
    printf("Avg waiting time of the process  is %f\n",Avg_WT);
    printf("Avg turn around time of the process is %f\n",Avg_TAT);
    return 0;
}
```

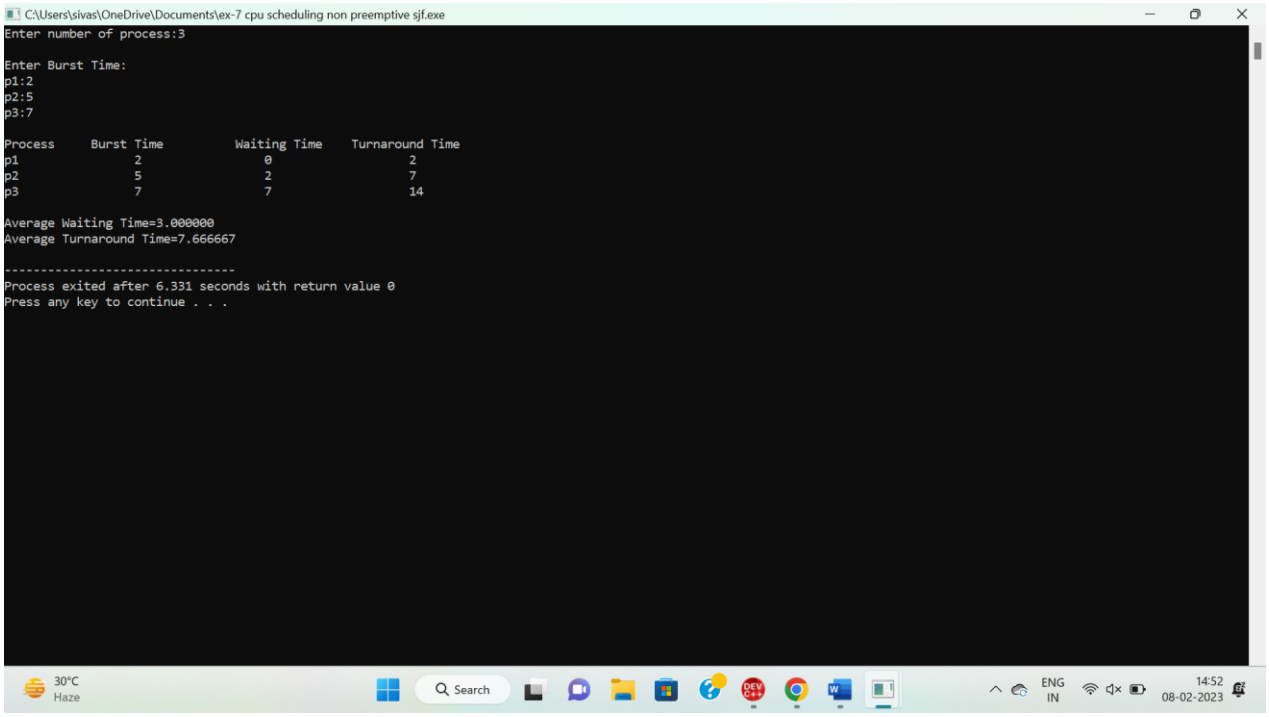



**7. Construct a C program to implement non-preemptive SJF algorithm.**

```
#include<stdio.h>
 int main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("Enter number of process:");
    scanf("%d",&n);
    printf("\nEnter Burst Time:\n");
    for(i=0;i<n;i++)
```

```
{
   printf("p%d:",i+1);
   scanf("%d",&bt[i]);
   p[i]=i+1;
}
for(i=0;i<n;i++)
{
   pos=i;
   for(j=i+1;j<n;j++)
   {
      if(bt[j]<bt[pos])
         pos=j;
   }
   temp=bt[i];
   bt[i]=bt[pos];
   bt[pos]=temp;

   temp=p[i];
   p[i]=p[pos];
   p[pos]=temp;
}
wt[0]=0;
for(i=1;i<n;i++)
{
   wt[i]=0;
   for(j=0;j<i;j++)
      wt[i]+=bt[j];

   total+=wt[i];
```

```
    }
    avg_wt=(float)total/n;
    total=0;
    printf("\nProcess\t    Burst Time    \tWaiting Time \tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        total+=tat[i];
        printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }
    avg_tat=(float)total/n;
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

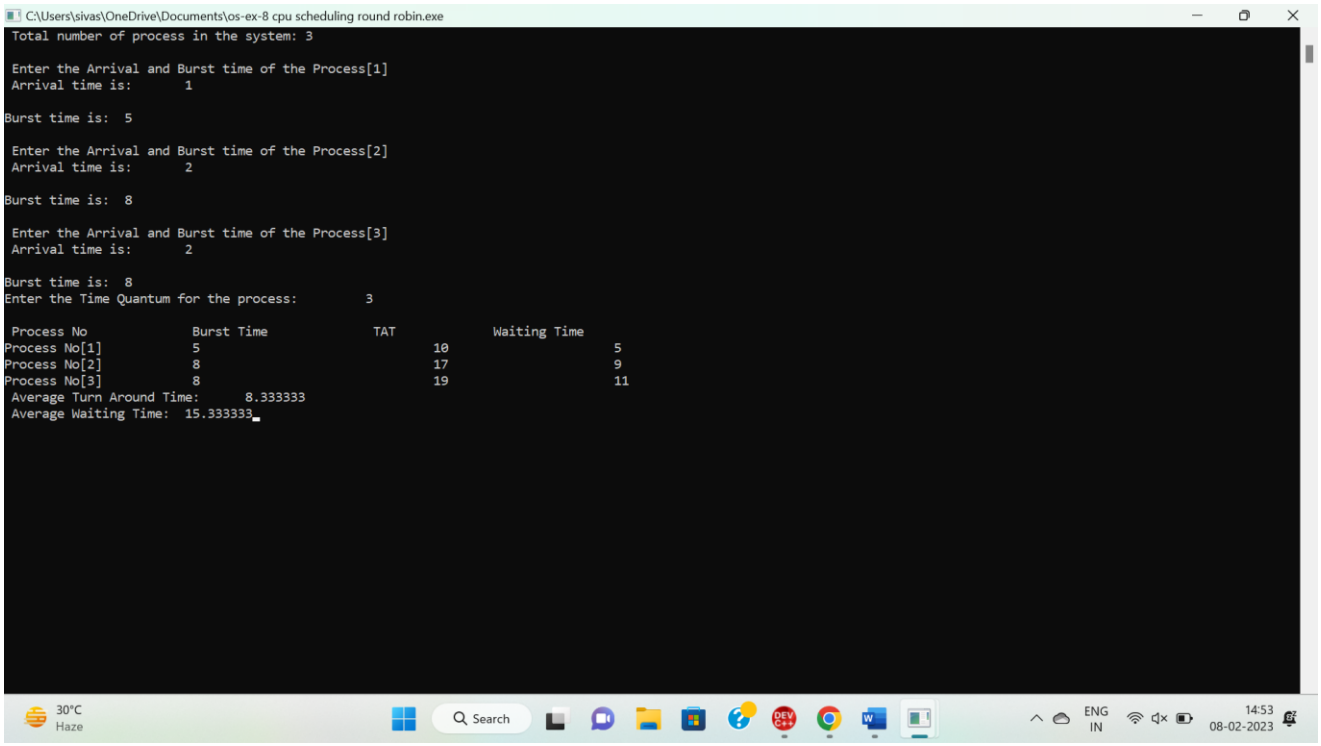



**8. Construct a C program to simulate Round Robin scheduling algorithm with C.**

```
#include<stdio.h>
#include<conio.h>
int main()
```

```
{
   int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
   float avg_wt, avg_tat;
   printf(" Total number of process in the system: ");
   scanf("%d", &NOP);
   y = NOP;
for(i=0; i<NOP; i++)
{
printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
printf(" Arrival time is: \t");
scanf("%d", &at[i]);
printf(" \nBurst time is: \t");
scanf("%d", &bt[i]);
temp[i] = bt[i];
}
printf("Enter the Time Quantum for the process: \t");
scanf("%d", &quant);
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
for(sum=0, i = 0; y!=0; )
{
if(temp[i] <= quant && temp[i] > 0)
{
   sum = sum + temp[i];
   temp[i] = 0;
   count=1;
   }
   else if(temp[i] > 0)
   {
      temp[i] = temp[i] - quant;
```

```
        sum = sum + quant;
    }
    if(temp[i]==0 && count==1)
    {
        y--;
        printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-
bt[i]);
        wt = wt+sum-at[i]-bt[i];
        tat = tat+sum-at[i];
        count =0;
    }
    if(i==NOP-1)
    {
        i=0;
    }
    else if(at[i+1]<=sum)
    {
        i++;
    }
    else
    {
        i=0;
    }
}
avg_wt = wt * 1.0/NOP;
avg_tat = tat * 1.0/NOP;
printf("\n Average Turn Around Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
getch();
}
```


C:\Users\sivas\OneDrive\Documents\os-ex-8 cpu scheduling round robin.exe
Total number of process in the system: 3
Enter the Arrival and Burst time of the Process[1]
Arrival time is: 1
Burst time is: 5
Enter the Arrival and Burst time of the Process[2]
Arrival time is: 2
Burst time is: 8
Enter the Arrival and Burst time of the Process[3]
Arrival time is: 2
Burst time is: 8
Enter the Time Quantum for the process: 3
Process No Burst Time TAT Waiting Time
Process No[1] 5 10 5
Process No[2] 8 17 9
Process No[3] 8 19 11
Average Turn Around Time: 8.333333
Average Waiting Time: 15.333333
30°C
Haze
Search
ENG
IN
14:53
08-02-2023