CSA0496-OPERATING SYSTEM WITH TASK MIGRATION

DAY-01 ***NAME: P.PRANAVI***

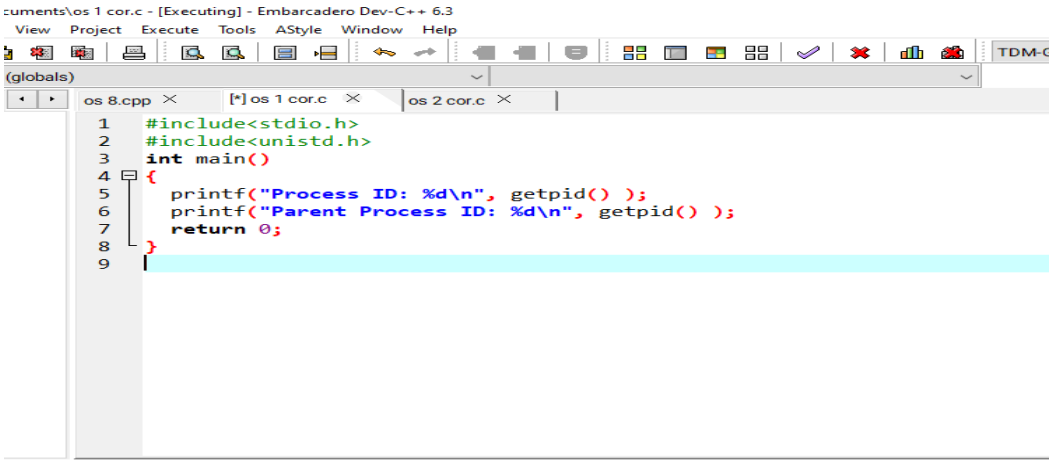
REG.NO:192110530

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.

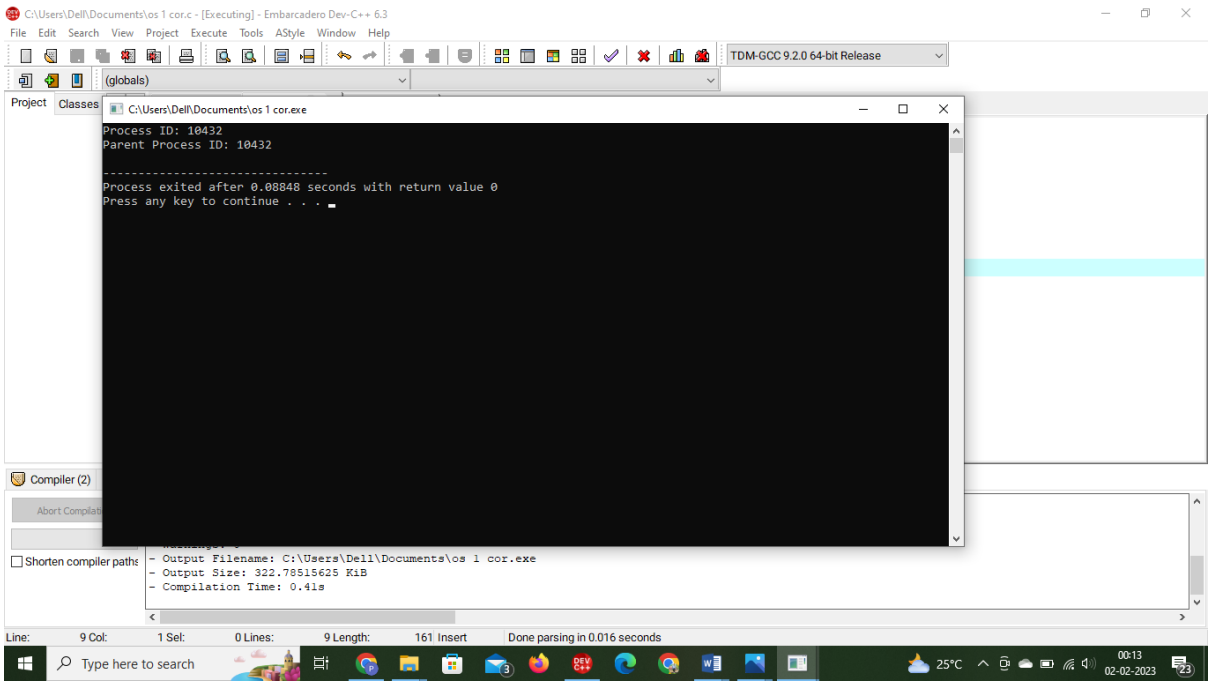
Program:

```
#include<stdio.h>
#include<unistd.h>
int main()
{
 printf("Process ID: %d\n", getpid() );
 printf("Parent Process ID: %d\n", getpid() );
 return 0;
}
```

Input:



Output:



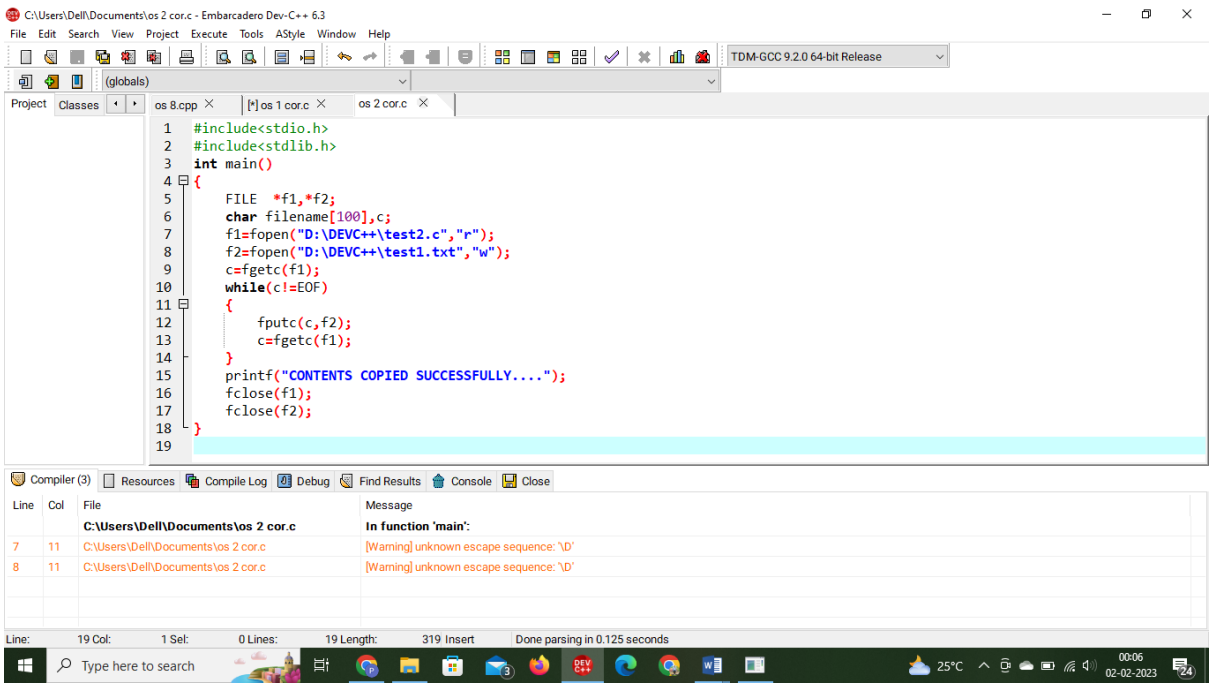
2.Identify the system calls to copy the content of one file to another and illustrate the same using a C program.

Program:

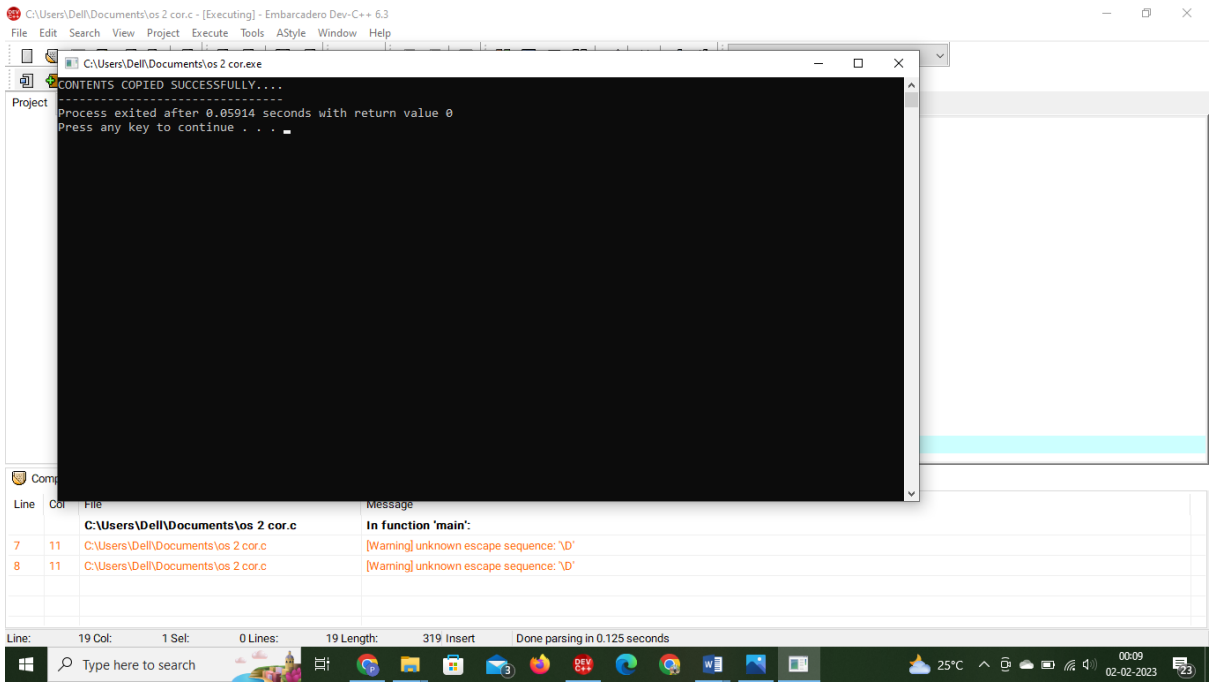
```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    FILE  *f1,*f2;
    char filename[100],c;
    f1=fopen("D:\DEVC++\test2.c","r");
    f2=fopen("D:\DEVC++\test1.txt","w");
    c=fgetc(f1);
    while(c!=EOF)
    {
            fputc(c,f2);
            c=fgetc(f1);
    }
    printf("CONTENTS COPIED SUCCESSFULLY....");
    fclose(f1);
    fclose(f2);
```

```
}
```

Input:



Output:



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.

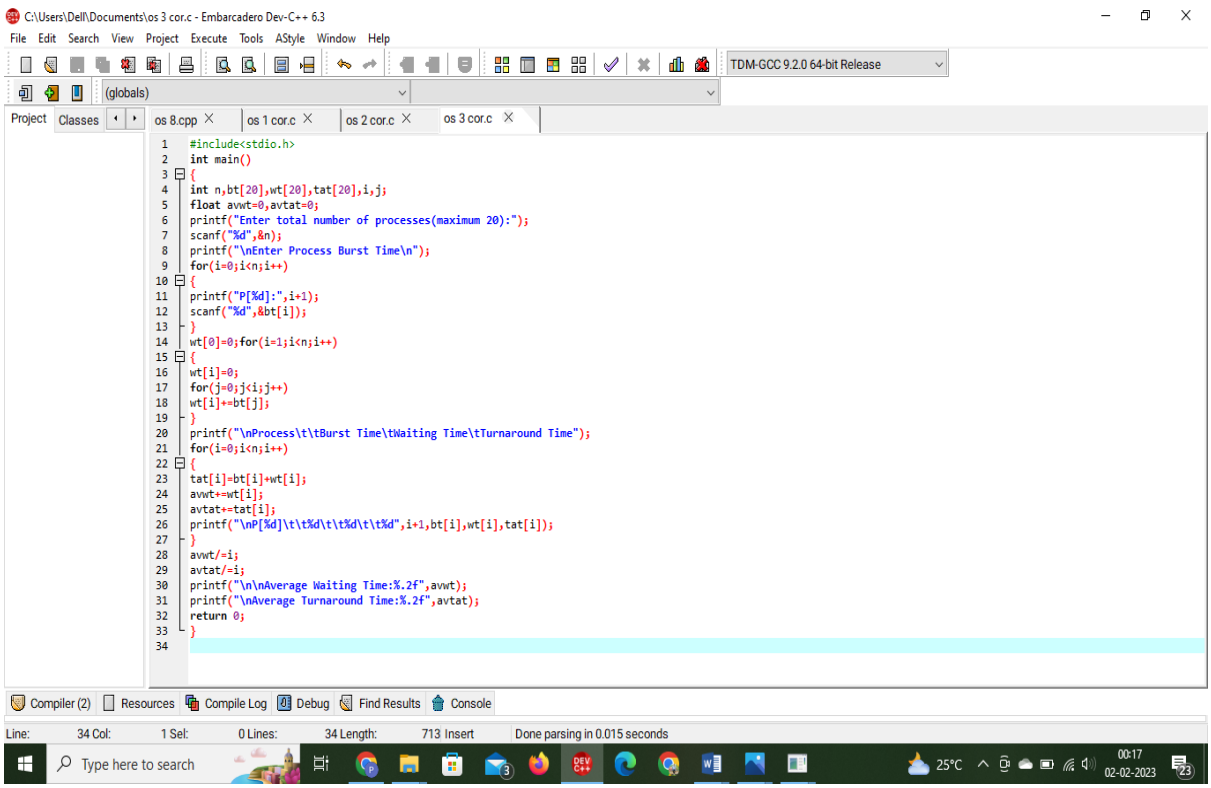
b. Assume that no process waits on I/O devices.

Program:

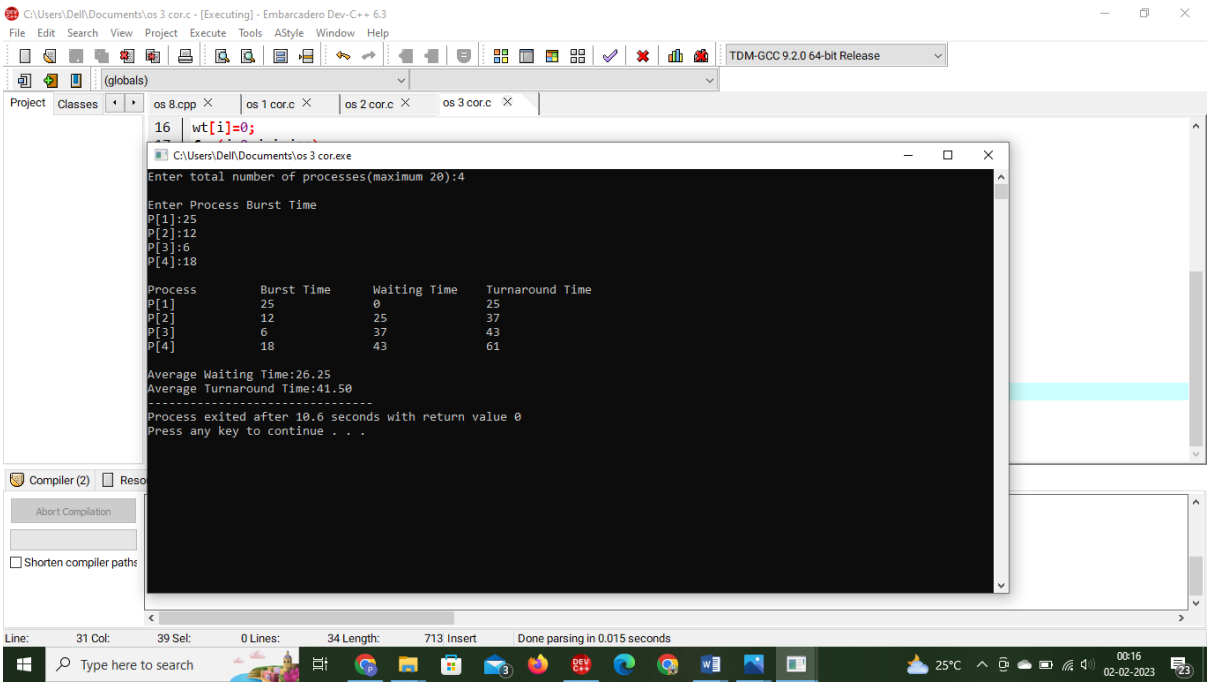
```
#include<stdio.h>
int main()
{
int n,bt[20],wt[20],tat[20],i,j;
float avwt=0,avtat=0;
printf("Enter total number of processes(maximum 20):");
scanf("%d",&n);
printf("\nEnter Process Burst Time\n");
for(i=0;i<n;i++)
{
printf("P[%d]:",i+1);
scanf("%d",&bt[i]);
}
wt[0]=0;for(i=1;i<n;i++)
{
wt[i]=0;
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;

}
```

Input:



Output:



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

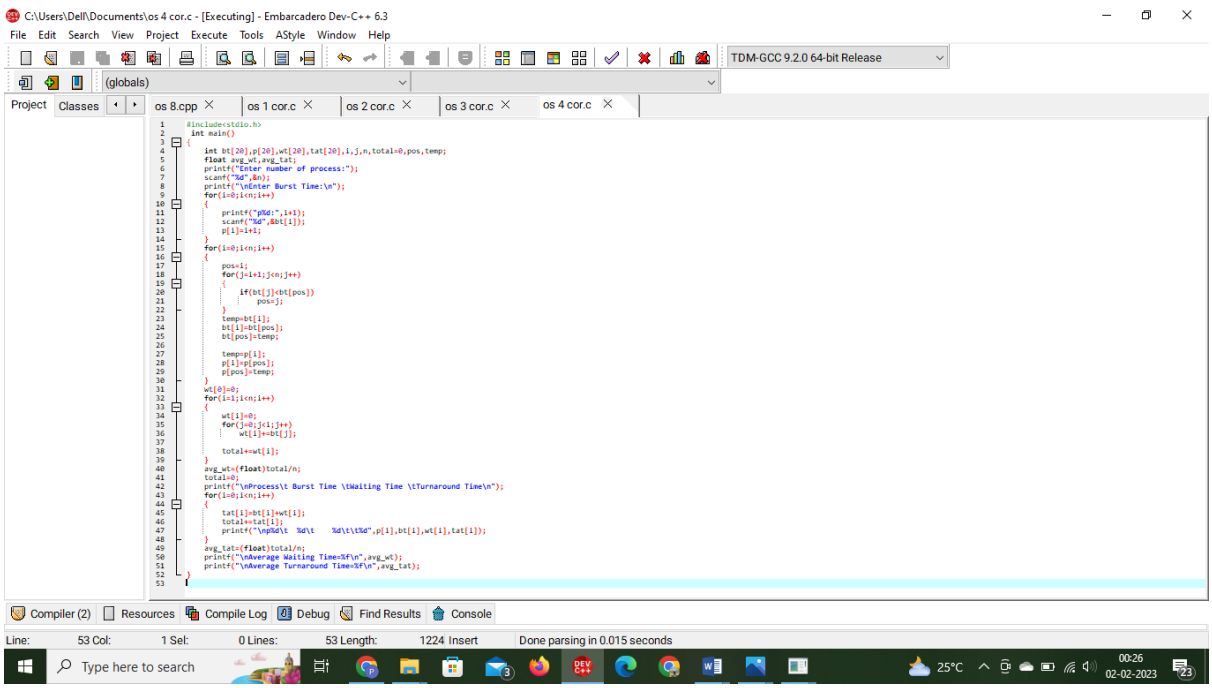
Program:

```
#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]);
```

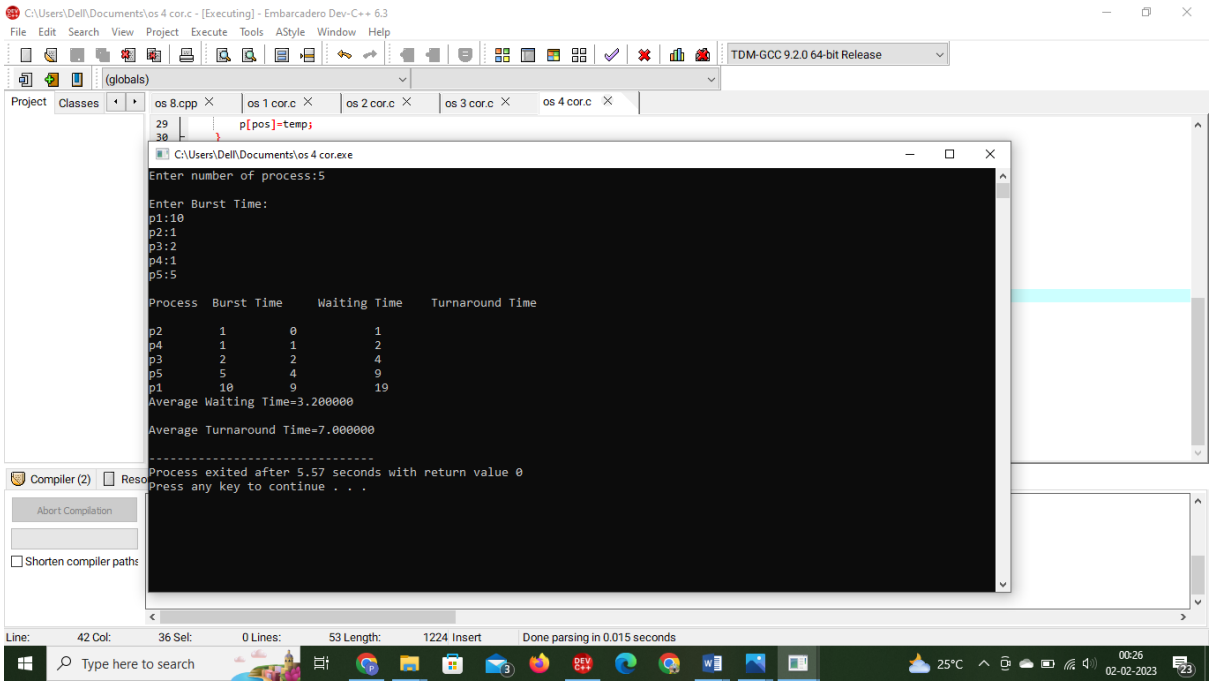
```
    p[i]=i+1;
}
for(i=0;i<n;i++)
{
    pos=i;
    for(j=i+1;j<n;j++)
    {
        if(bt[j]<bt[pos])
            pos=j;
    }
    temp=bt[i];
    bt[i]=bt[pos];
    bt[pos]=temp;

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

Input:



Output:



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

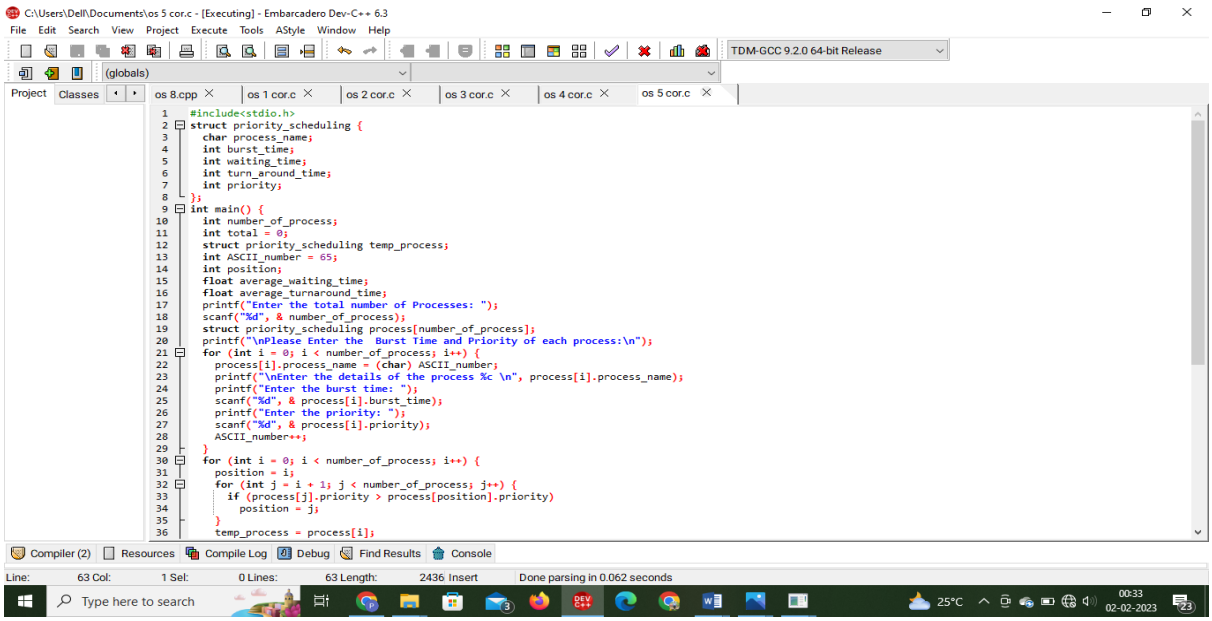
Program:

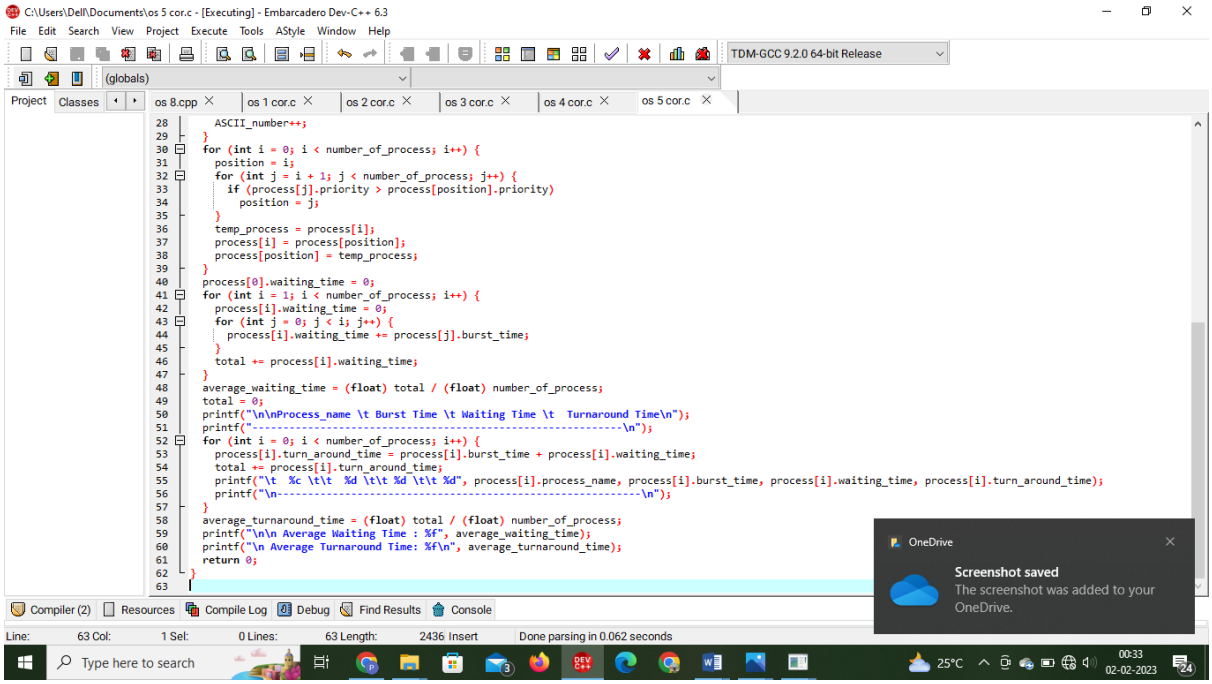
```
#include<stdio.h>

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;
```

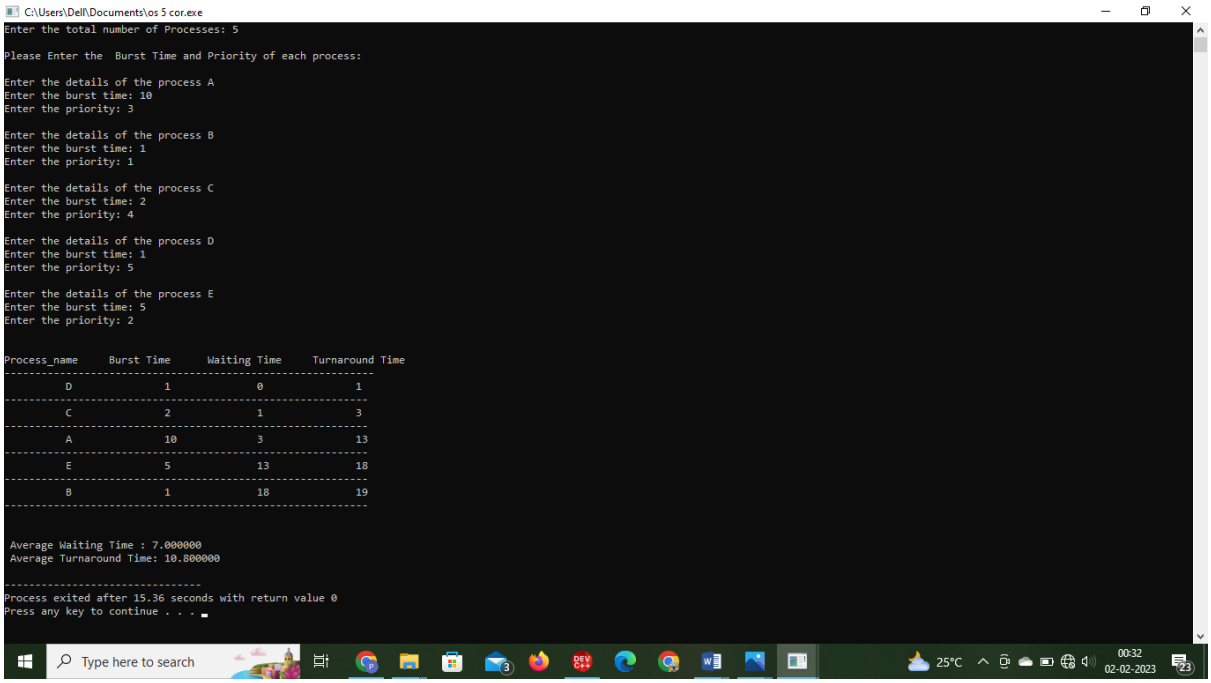
```
   for (int j = 0; j < i; j++) {
     process[i].waiting_time += process[j].burst_time;
   }
   total += process[i].waiting_time;
  }
  average_waiting_time = (float) total / (float) number_of_process;
  total = 0;
  printf("\n\nProcess_name \t Burst Time \t Waiting Time \t  Turnaround Time\n");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


}
```

Input:


C:\Users\Dell\Documents\os 5 cor.c - [Executing] - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 9.2.0 64-bit Release
(globals)
Project Classes
os 8.cpp
os 1 cor.c
os 2 cor.c
os 3 cor.c
os 4 cor.c
os 5 cor.c
#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];
Compiler (2) Resources Compile Log Debug Find Results Console
Line: 63 Col: 1 Sel: 0 Lines: 63 Length: 2436 Insert Done parsing in 0.062 seconds
Type here to search
25°C
00:33
02-02-2023


C:\Users\Dell\Documents\os 5 cor.c - [Executing] - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 9.2.0 64-bit Release
(globals)
Project Classes
os 8.cpp
os 1 cor.c
os 2 cor.c
os 3 cor.c
os 4 cor.c
os 5 cor.c
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");
printf("----------\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);
printf("\n----------\n");
}
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;
}
OneDrive
Screenshot saved
The screenshot was added to your OneDrive.
Compiler (2) Resources Compile Log Debug Find Results Console
Line: 63 Col: 1 Sel: 0 Lines: 63 Length: 2436 Insert Done parsing in 0.062 seconds
Type here to search
25°C
00:33
02-02-2023

Output:



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

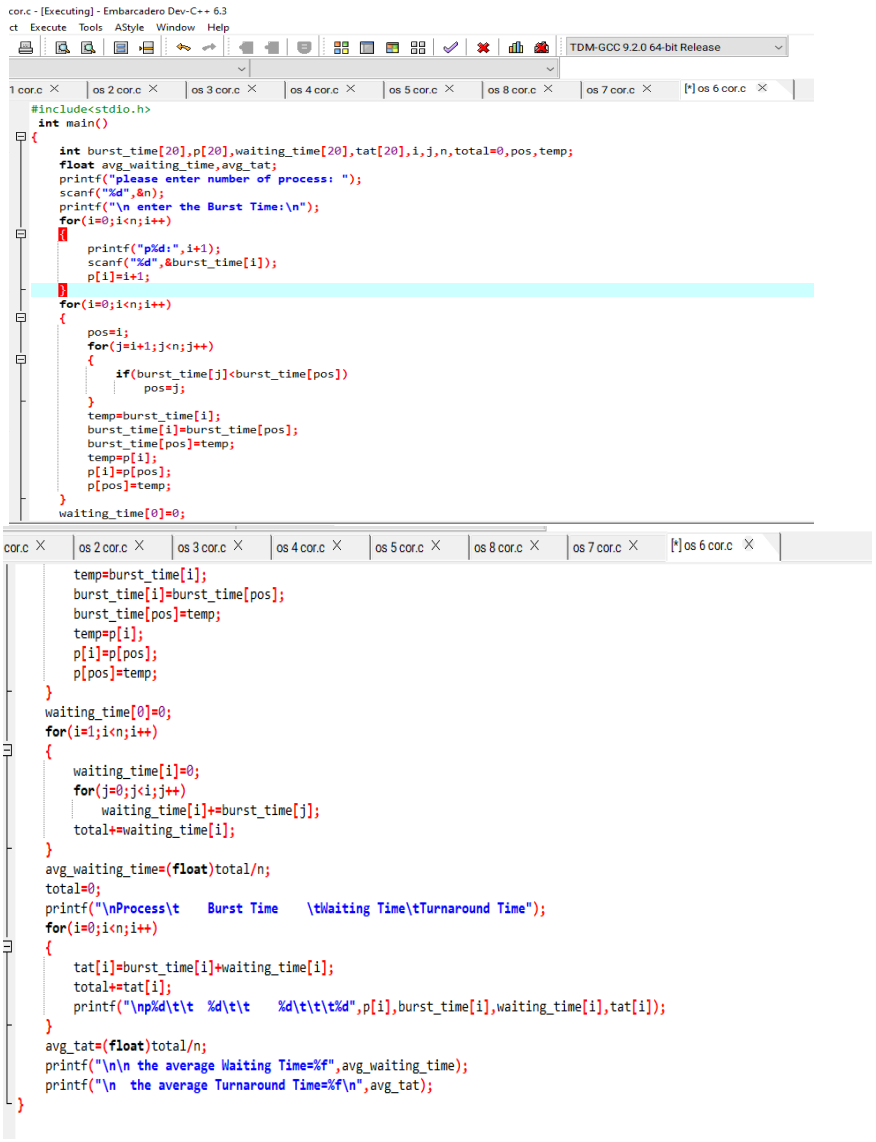
Program:

```
#include<stdio.h>
 int main()
{
   int burst_time[20],p[20],waiting_time[20],tat[20],i,j,n,total=0,pos,temp;
   float avg_waiting_time,avg_tat;
   printf("please enter number of process: ");
   scanf("%d",&n);
   printf("\n enter the Burst Time:\n");
   for(i=0;i<n;i++)
   {
      printf("p%d:",i+1);
      scanf("%d",&burst_time[i]);
      p[i]=i+1;
```

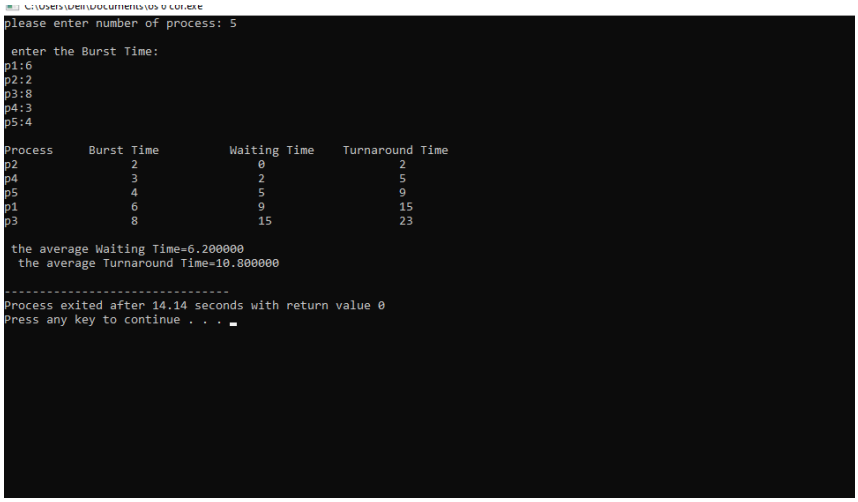
```
}
for(i=0;i<n;i++)
{
   pos=i;
   for(j=i+1;j<n;j++)
   {
      if(burst_time[j]<burst_time[pos])
         pos=j;
   }
   temp=burst_time[i];
   burst_time[i]=burst_time[pos];
   burst_time[pos]=temp;
   temp=p[i];
   p[i]=p[pos];
   p[pos]=temp;
}
waiting_time[0]=0;
for(i=1;i<n;i++)
{
   waiting_time[i]=0;
   for(j=0;j<i;j++)
      waiting_time[i]+=burst_time[j];
   total+=waiting_time[i];
}
avg_waiting_time=(float)total/n;
total=0;
printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
```

```
    tat[i]=burst_time[i]+waiting_time[i];
    total+=tat[i];
    printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],burst_time[i],waiting_time[i],tat[i]);
  }
  avg_tat=(float)total/n;
  printf("\n\n the average Waiting Time=%f",avg_waiting_time);
  printf("\n  the average Turnaround Time=%f\n",avg_tat);
}
```

Input:



Output:



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

Program:

```
#include<stdio.h>


int main() {
  int time, burst_time[10], at[10], sum_burst_time = 0, smallest, n, i;
  int sumt = 0, sumw = 0;
  printf("enter the no of processes : ");
  scanf("%d", & n);
  for (i = 0; i < n; i++) {
    printf("the arrival time for process P%d : ", i + 1);
    scanf("%d", & at[i]);
    printf("the burst time for process P%d : ", i + 1);
    scanf("%d", & burst_time[i]);
    sum_burst_time += burst_time[i];
  }
  burst_time[9] = 9999;
  for (time = 0; time < sum_burst_time;) {
    smallest = 9;
    for (i = 0; i < n; i++) {
```

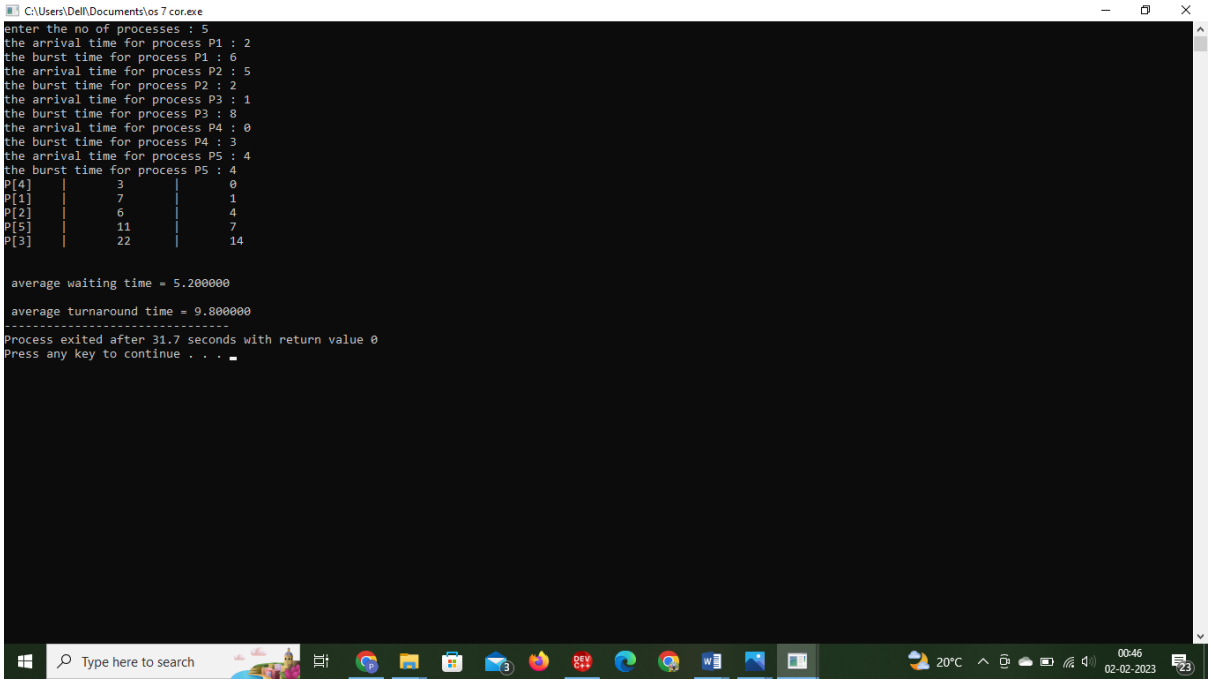
```
    if (at[i] <= time && burst_time[i] > 0 && burst_time[i] < burst_time[smallest])
      smallest = i;
   }
   printf("P[%d]\t|\t%d\t|\t%d\n", smallest + 1, time + burst_time[smallest] -
at[smallest], time - at[smallest]);
   sumt += time + burst_time[smallest] - at[smallest];
   sumw += time - at[smallest];
   time += burst_time[smallest];
   burst_time[smallest] = 0;
  }
printf("\n\n average waiting time = %f", sumw * 1.0 / n);
  printf("\n\n average turnaround time = %f", sumt * 1.0 / n);
return 0;
}
```

Input:

os 8.cpp | os 1 cor.c | os 2 cor.c | os 3 cor.c | os 4 cor.c | os 5 cor.c | os 8 cor.c | os 7 cor.c

```c
#include<stdio.h>

int main() {
  int time, burst_time[10], at[10], sum_burst_time = 0, smallest, n, i;
  int sumt = 0, sumw = 0;
  printf("enter the no of processes : ");
  scanf("%d", & n);
  for (i = 0; i < n; i++) {
    printf("the arrival time for process P%d : ", i + 1);
    scanf("%d", & at[i]);
    printf("the burst time for process P%d : ", i + 1);
    scanf("%d", & burst_time[i]);
    sum_burst_time += burst_time[i];
  }
  burst_time[9] = 9999;
  for (time = 0; time < sum_burst_time;) {
    smallest = 9;
    for (i = 0; i < n; i++) {
      if (at[i] <= time && burst_time[i] > 0 && burst_time[i] < burst_time[smallest])
        smallest = i;
    }
    printf("P[%d]\t|\t%d\t|\t%d\n", smallest + 1, time + burst_time[smallest] - at[smallest], time - at[smallest]);
    sumt += time + burst_time[smallest] - at[smallest];
    sumw += time - at[smallest];
    time += burst_time[smallest];
    burst_time[smallest] = 0;
  }
  printf("\n\n average waiting time = %f", sumw * 1.0 / n);
  printf("\n\n average turnaround time = %f", sumt * 1.0 / n);
  return 0;
}
```

Output:



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

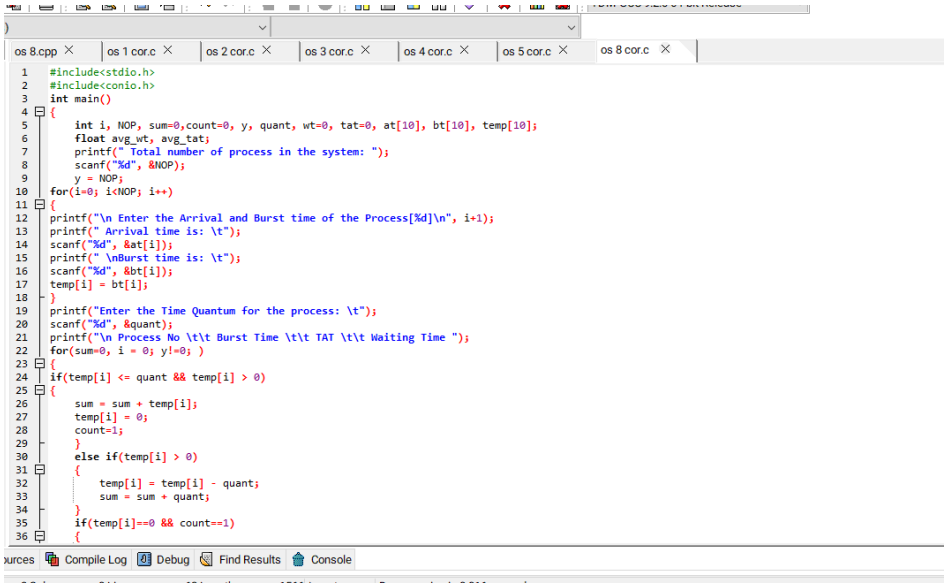
Program:

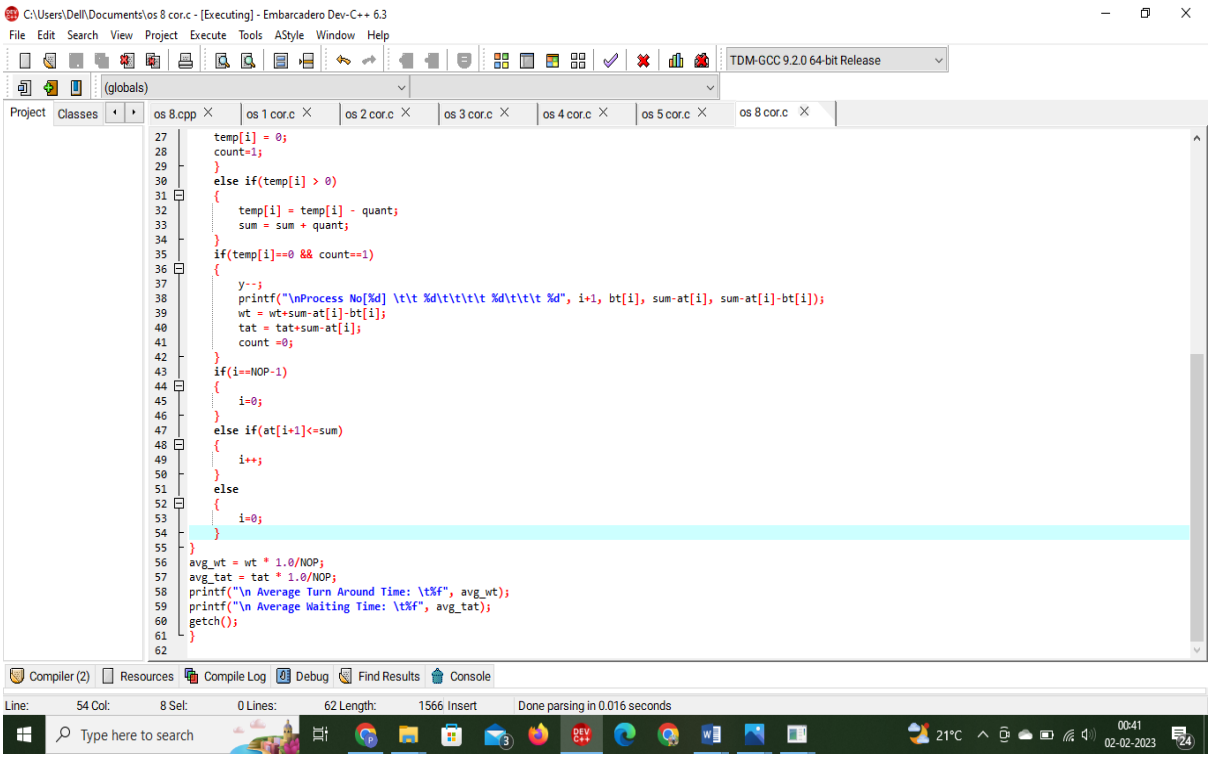
```
#include<stdio.h>
#include<conio.h>
int main()
{
   int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
   float avg_wt, avg_tat;
   printf(" Total number of process in the system: ");
   scanf("%d", &NOP);
   y = NOP;
for(i=0; i<NOP; i++)
{
printf("\n Enter the Arrival and 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;
   }
```

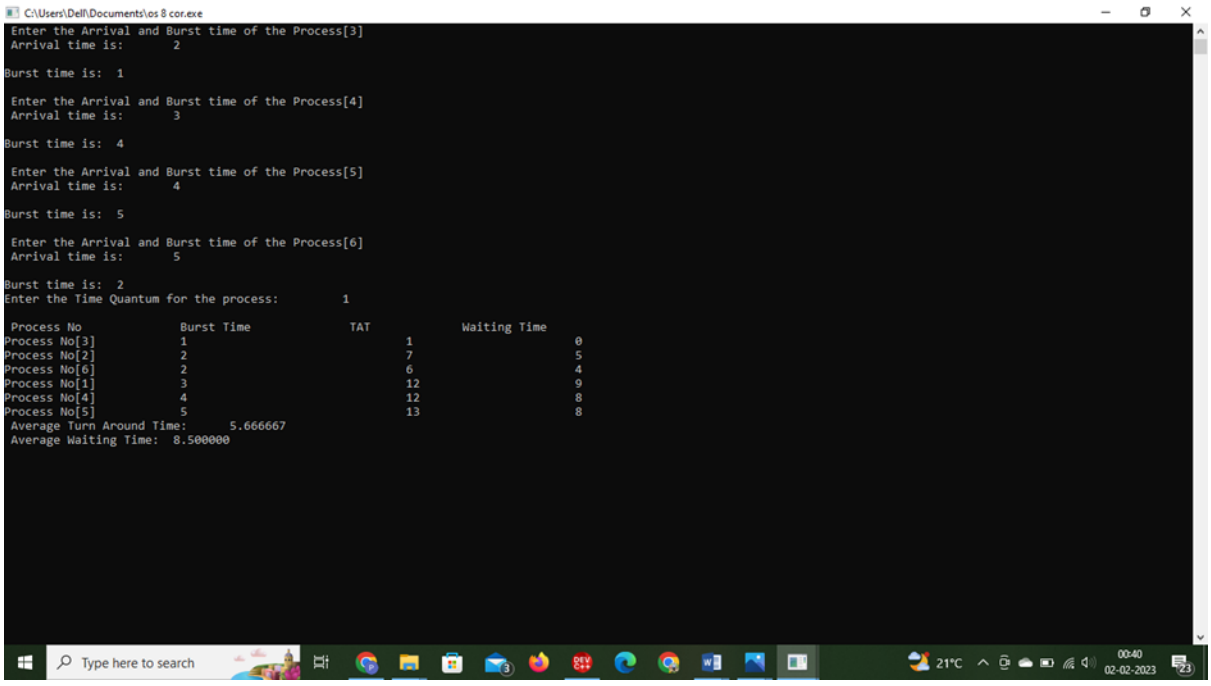
```
if(i==NOP-1)
{
   i=0;
}
else if(at[i+1]<=sum)
{
   i++;
}
else
{
   i=0;
}
}  avg_wt = wt * 1.0/NOP;
avg_tat = tat * 1.0/NOP;
printf("\n Average Turn Around Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
getch();}
```

Input:




C:\Users\Dell\Documents\os 8 cor.c - [Executing] - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 9.2.0 64-bit Release
(globals)
Project Classes
os 8.cpp
os 1 cor.c
os 2 cor.c
os 3 cor.c
os 4 cor.c
os 5 cor.c
os 8 cor.c
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();
}
Compiler (2) Resources Compile Log Debug Find Results Console
Line: 54 Col: 8 Sel: 0 Lines: 62 Length: 1566 Insert Done parsing in 0.016 seconds
Type here to search
21°C
00:41
02-02-2023

Output:


C:\Users\Dell\Documents\os 8 cor.exe
Enter the Arrival and Burst time of the Process[3]
Arrival time is: 2
Burst time is: 1
Enter the Arrival and Burst time of the Process[4]
Arrival time is: 3
Burst time is: 4
Enter the Arrival and Burst time of the Process[5]
Arrival time is: 4
Burst time is: 5
Enter the Arrival and Burst time of the Process[6]
Arrival time is: 5
Burst time is: 2
Enter the Time Quantum for the process: 1
Process No Burst Time TAT Waiting Time
Process No[3] 1 1 0
Process No[2] 2 7 5
Process No[6] 2 6 4
Process No[1] 3 12 9
Process No[4] 4 12 8
Process No[5] 5 13 8
Average Turn Around Time: 5.666667
Average Waiting Time: 8.500000
Type here to search
21°C
00:40
02-02-2023