**01-02-2023 L.SANTHOSH KUMAR**

**192125076**

**CSA0496-OPREATING SYSTEM**

**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>

int main()

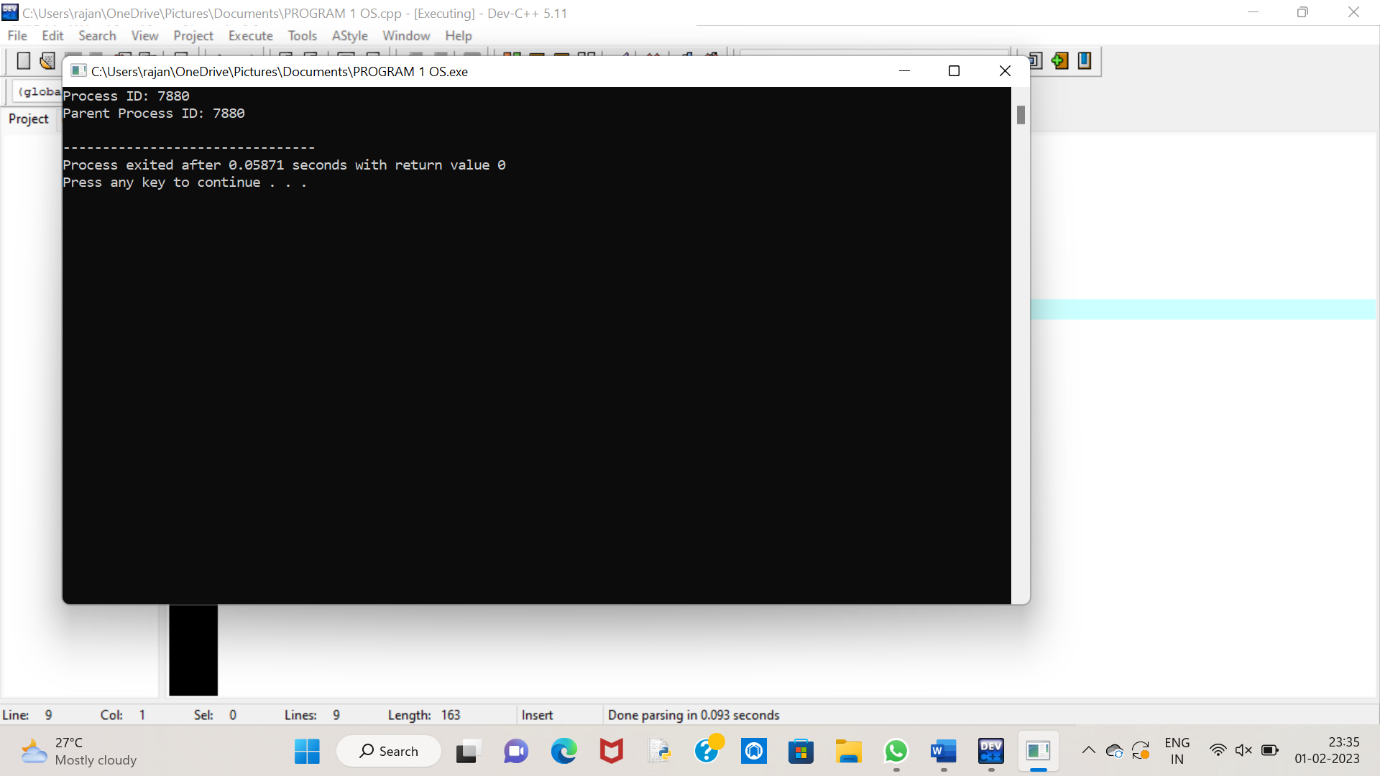
{

printf("Process ID: %d\n", getpid() );

printf("Parent Process ID: %d\n", getpid() );

return 0;

}



**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);

}

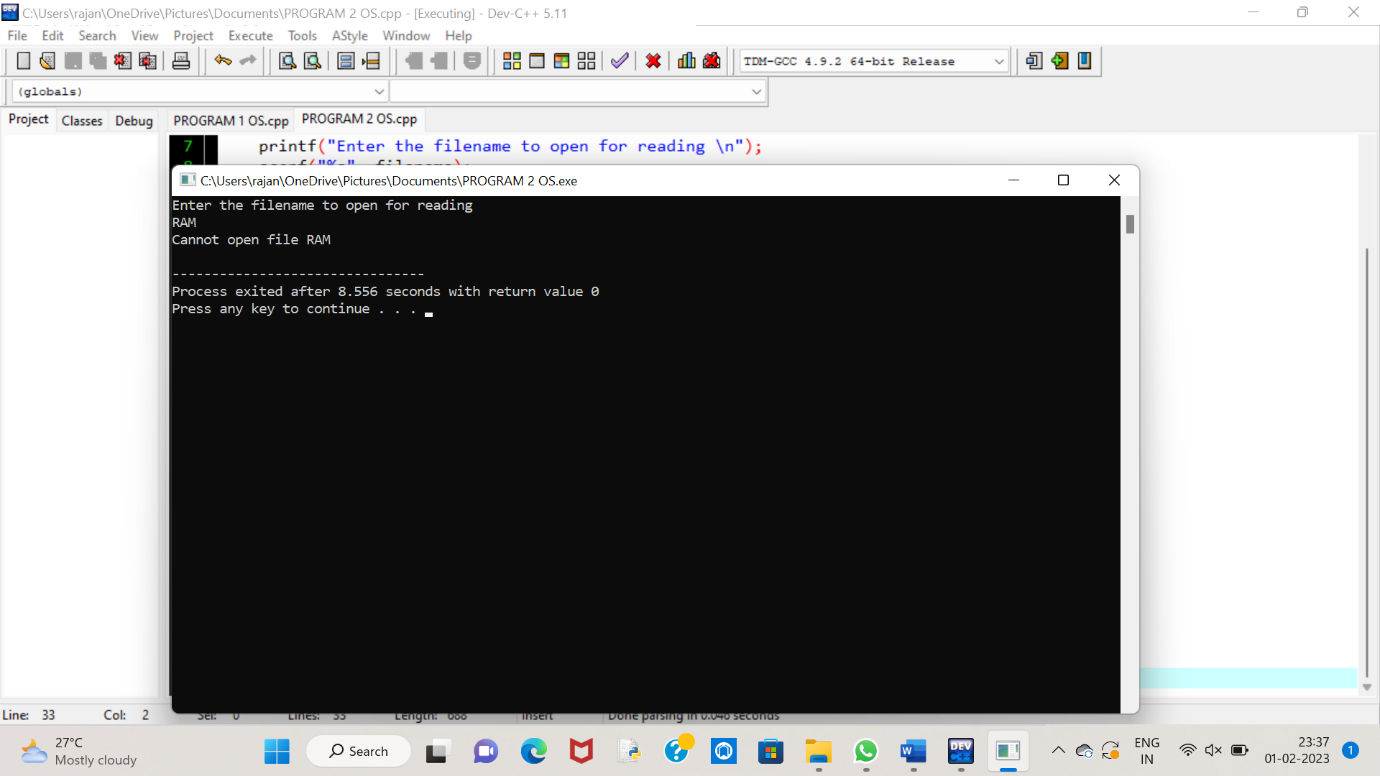
printf("\nContents copied to %s", filename);

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>

void 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++)

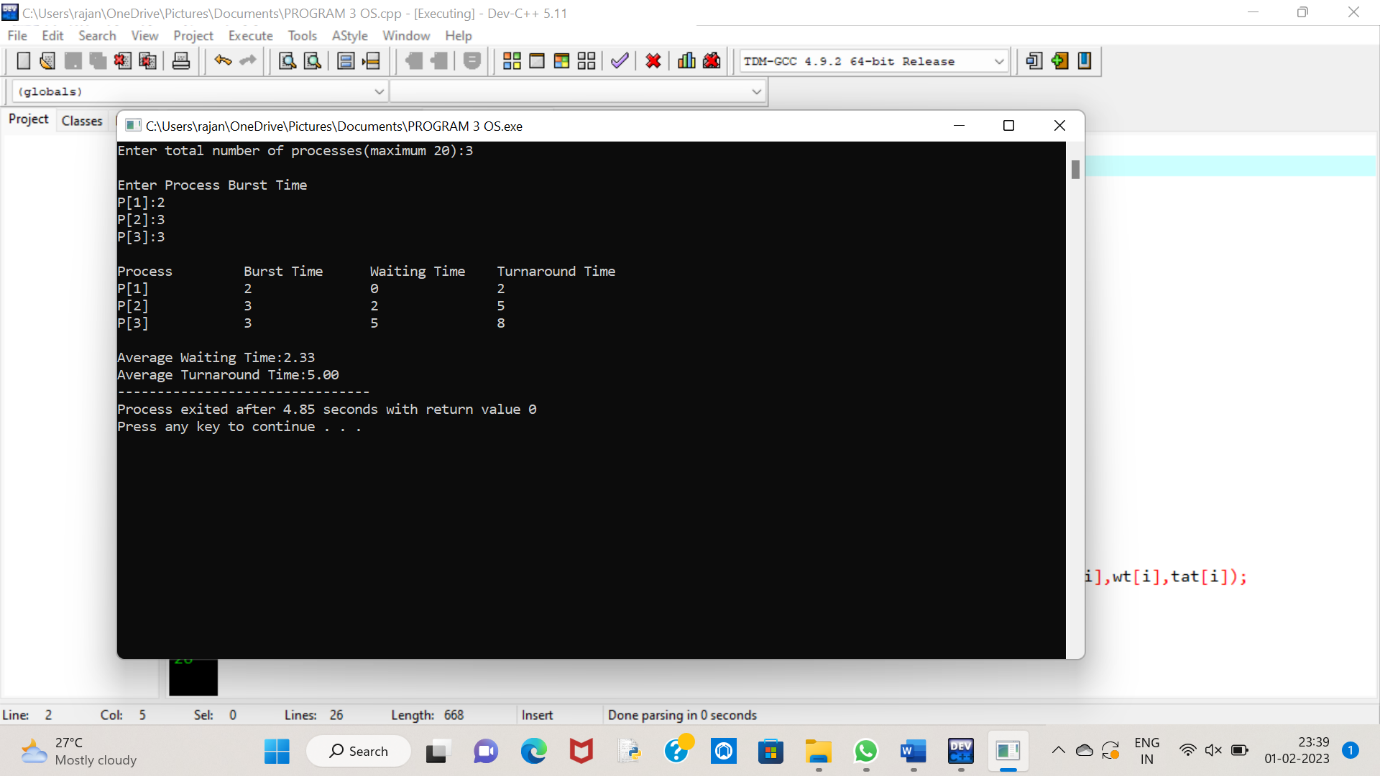
{

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);

}



**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 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("nProcesst Burst Time tWaiting TimetTurnaround Time");

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

{

tat[i]=bt[i]+wt[i];

total+=tat[i];

printf("np%dtt %dtt %dttt%d",p[i],bt[i],wt[i],tat[i]);

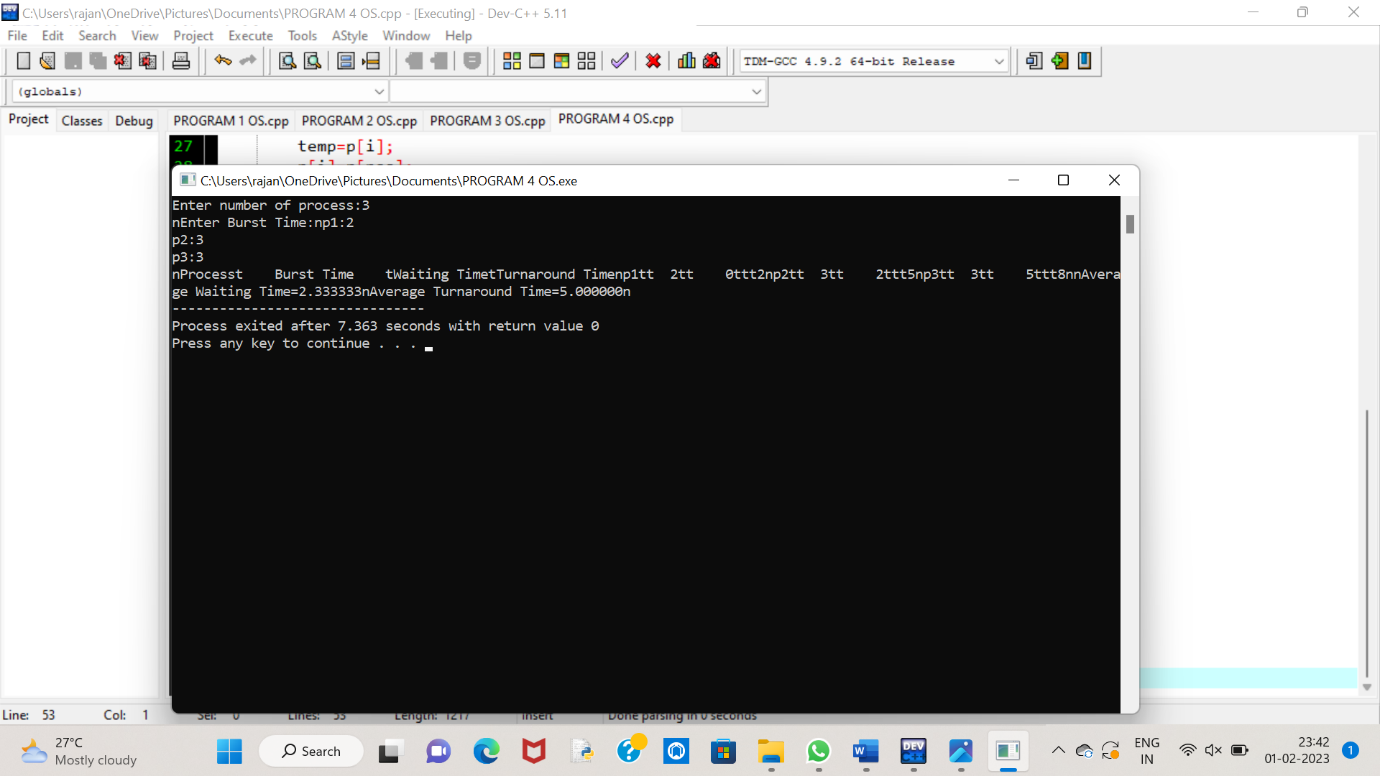
}

avg\_tat=(float)total/n;

printf("nnAverage Waiting Time=%f",avg\_wt);

printf("nAverage Turnaround Time=%fn",avg\_tat);

}



**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");

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");

}

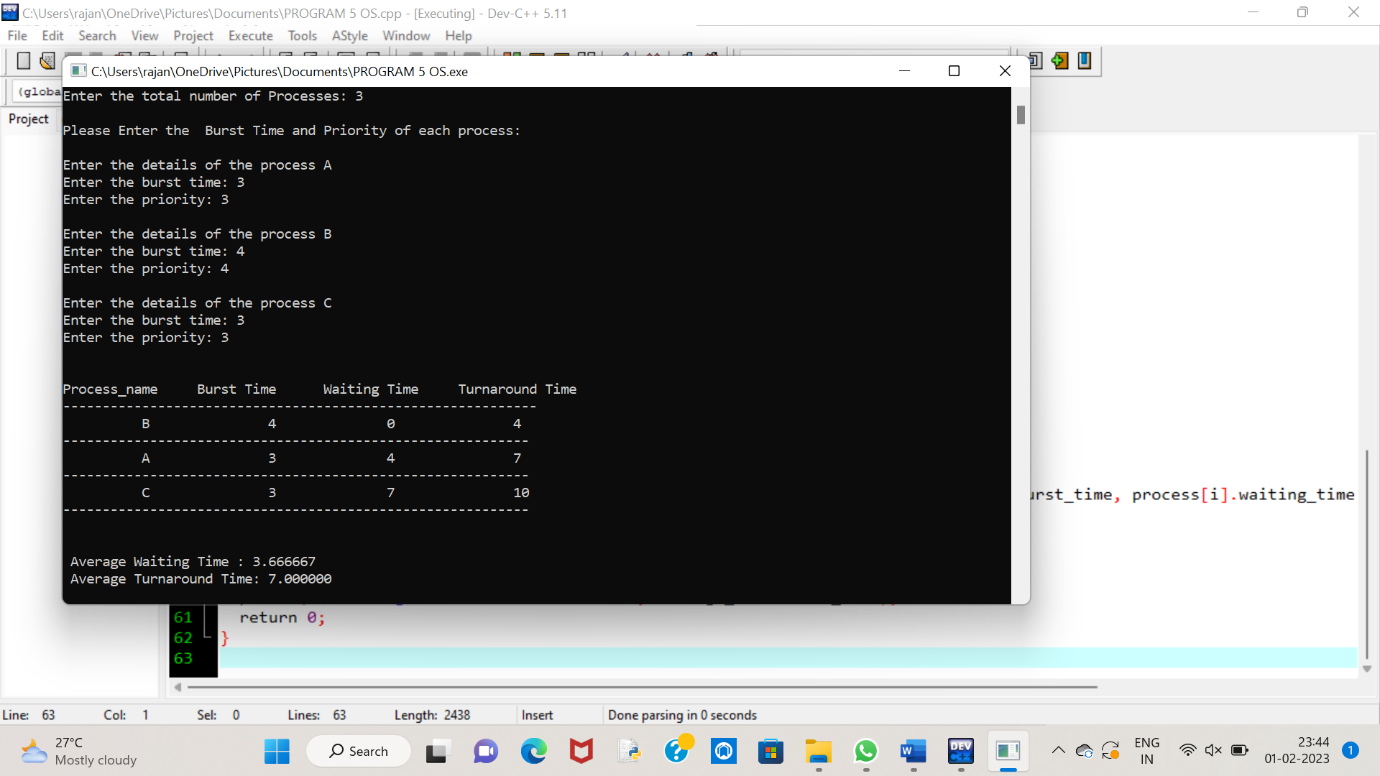
average\_turnaround\_time = (float) total / (float) number\_of\_process;

printf("\n\n Average Waiting Time : %f", average\_waiting\_time);

printf("\n Average Turnaround Time: %f\n", average\_turnaround\_time);

return 0;

}



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

#include <stdio.h>

#include <stdbool.h>

#define MAX\_PROCESSES 50

struct Process {

int id;

int arrival\_time;

int burst\_time;

int priority;

int waiting\_time;

int turnaround\_time;

};

int main() {

struct Process processes[MAX\_PROCESSES];

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

// Input the details of each process

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

printf("Enter the details of process %d\n", i + 1);

printf("Arrival time: ");

scanf("%d", &processes[i].arrival\_time);

printf("Burst time: ");

scanf("%d", &processes[i].burst\_time);

printf("Priority: ");

scanf("%d", &processes[i].priority);

processes[i].id = i + 1;

}

// Sort the processes in order of their priority

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

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

if (processes[j].priority < processes[i].priority) {

struct Process temp = processes[i];

processes[i] = processes[j];

processes[j] = temp;

}

}

}

// Calculate the waiting time and turnaround time for each process

int current\_time = 0;

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

struct Process current\_process = processes[i];

if (current\_time < current\_process.arrival\_time) {

current\_time = current\_process.arrival\_time;

}

current\_process.waiting\_time = current\_time - current\_process.arrival\_time;

current\_time += current\_process.burst\_time;

current\_process.turnaround\_time = current\_time - current\_process.arrival\_time;

}

// Display the details of each process

printf("\nProcess\tArrival Time\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");

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

struct Process current\_process = processes[i];

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", current\_process.id, current\_process.arrival\_time,

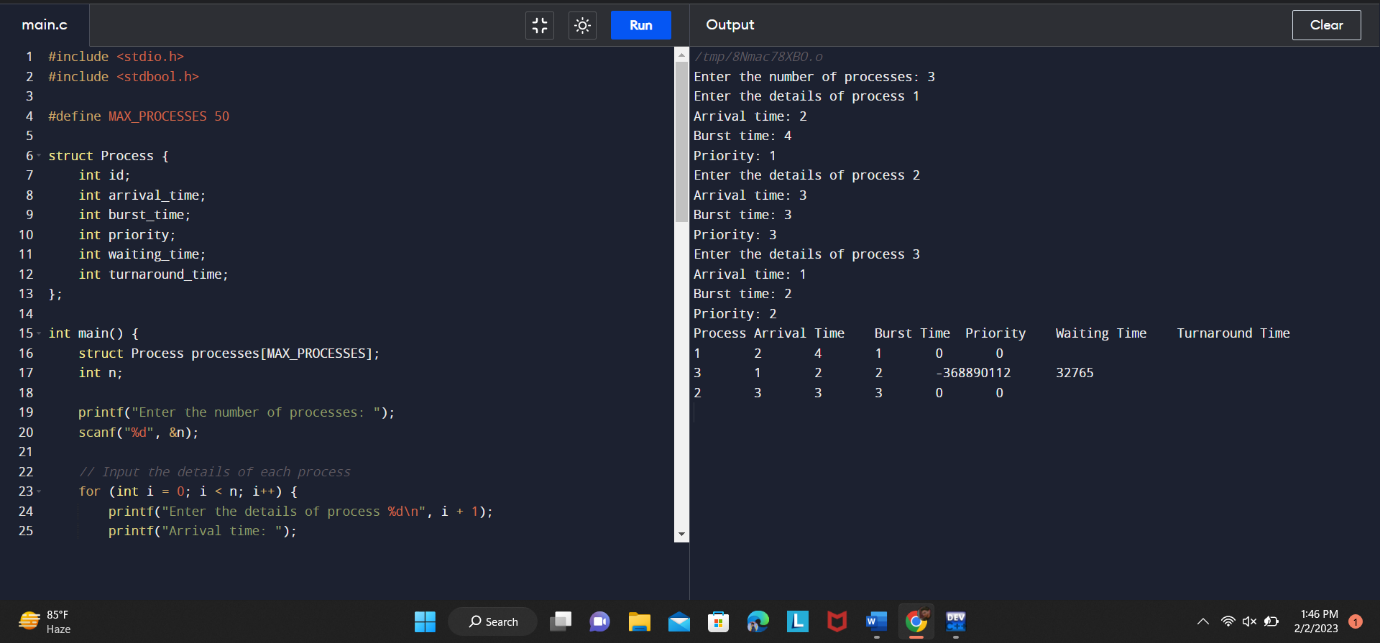
current\_process.burst\_time, current\_process.priority, current\_process.waiting\_time,

current\_process.turnaround\_time);

}

return 0;

}



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

#include <stdio.h>

#include <stdbool.h>

#define MAX\_PROCESSES 50

struct Process {

int id;

int arrival\_time;

int burst\_time;

int waiting\_time;

int turnaround\_time;

};

int main() {

struct Process processes[MAX\_PROCESSES];

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

// Input the details of each process

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

printf("Enter the details of process %d\n", i + 1);

printf("Arrival time: ");

scanf("%d", &processes[i].arrival\_time);

printf("Burst time: ");

scanf("%d", &processes[i].burst\_time);

processes[i].id = i + 1;

}

// Sort the processes in order of their burst time

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

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

if (processes[j].burst\_time < processes[i].burst\_time) {

struct Process temp = processes[i];

processes[i] = processes[j];

processes[j] = temp;

}

}

}

// Calculate the waiting time and turnaround time for each process

int current\_time = 0;

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

struct Process current\_process = processes[i];

if (current\_time < current\_process.arrival\_time) {

current\_time = current\_process.arrival\_time;

}

current\_process.waiting\_time = current\_time - current\_process.arrival\_time;

current\_time += current\_process.burst\_time;

current\_process.turnaround\_time = current\_time - current\_process.arrival\_time;

}

// Display the details of each process

printf("\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

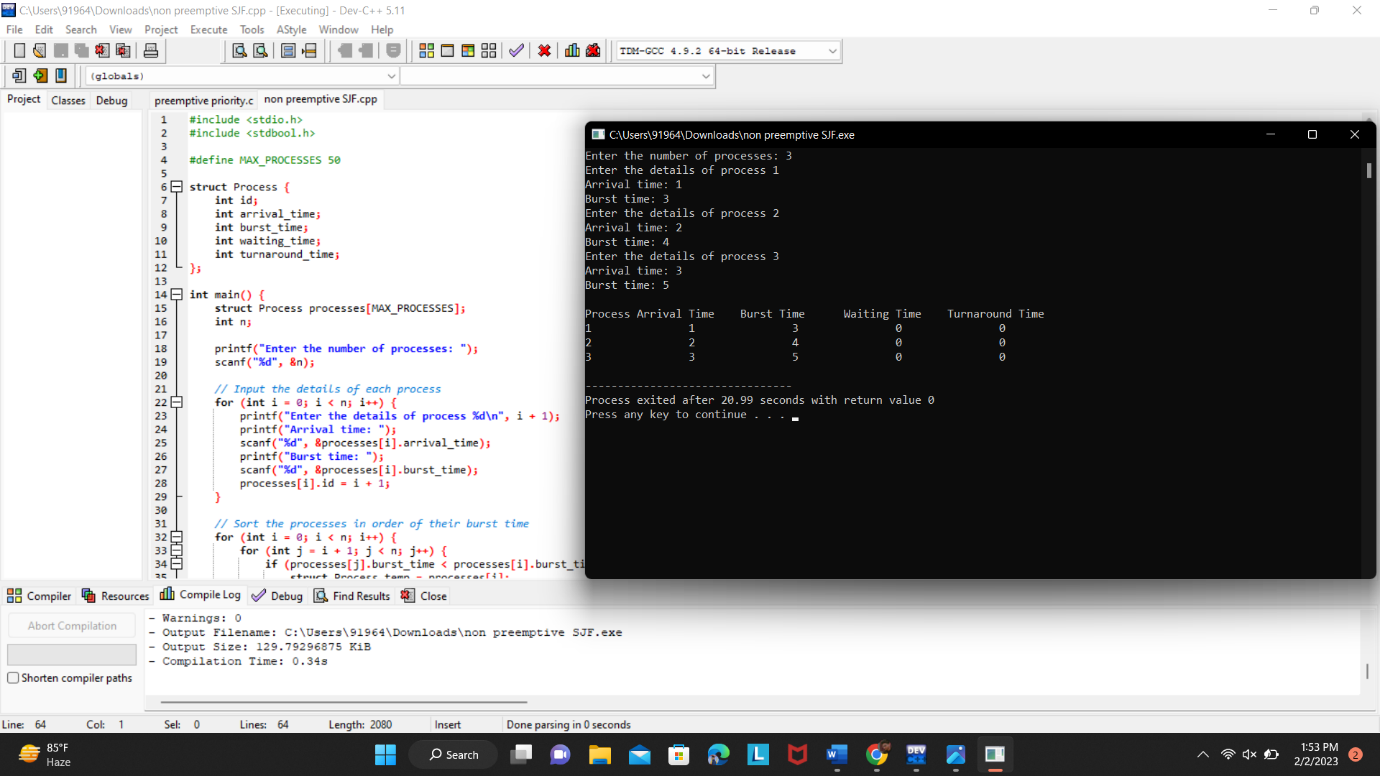
struct Process current\_process = processes[i];

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", current\_process.id, current\_process.arrival\_time,

current\_process.burst\_time, current\_process.waiting\_time, current\_process.turnaround\_time);

}

return 0;

}

**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();

}

