**STUDENT NAME: P. HASINI**

**REG NUMBER : 192111535**

**SUBJECT NAME : OPERATING SYSTEM WITH DESIGN PRINCIPLES**

**SUBJECT CODE : CSA0470**

**1.SYSTEM CALLS**

#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);

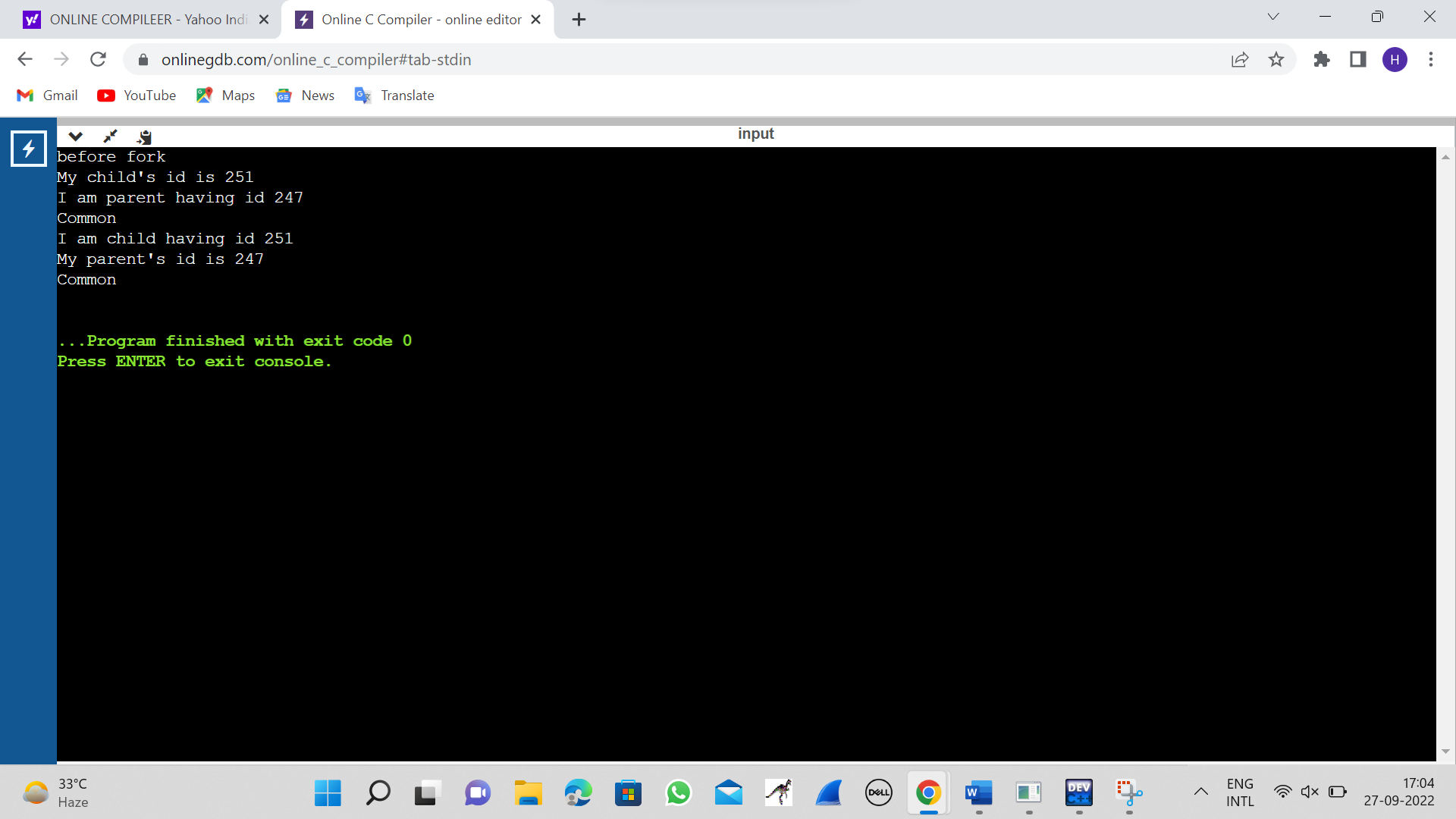
printf("I am parent having id %d\n",getpid());

}

printf("Common\n");

}

**OUTPUT**

****

**2.FILE COPYING**

#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);

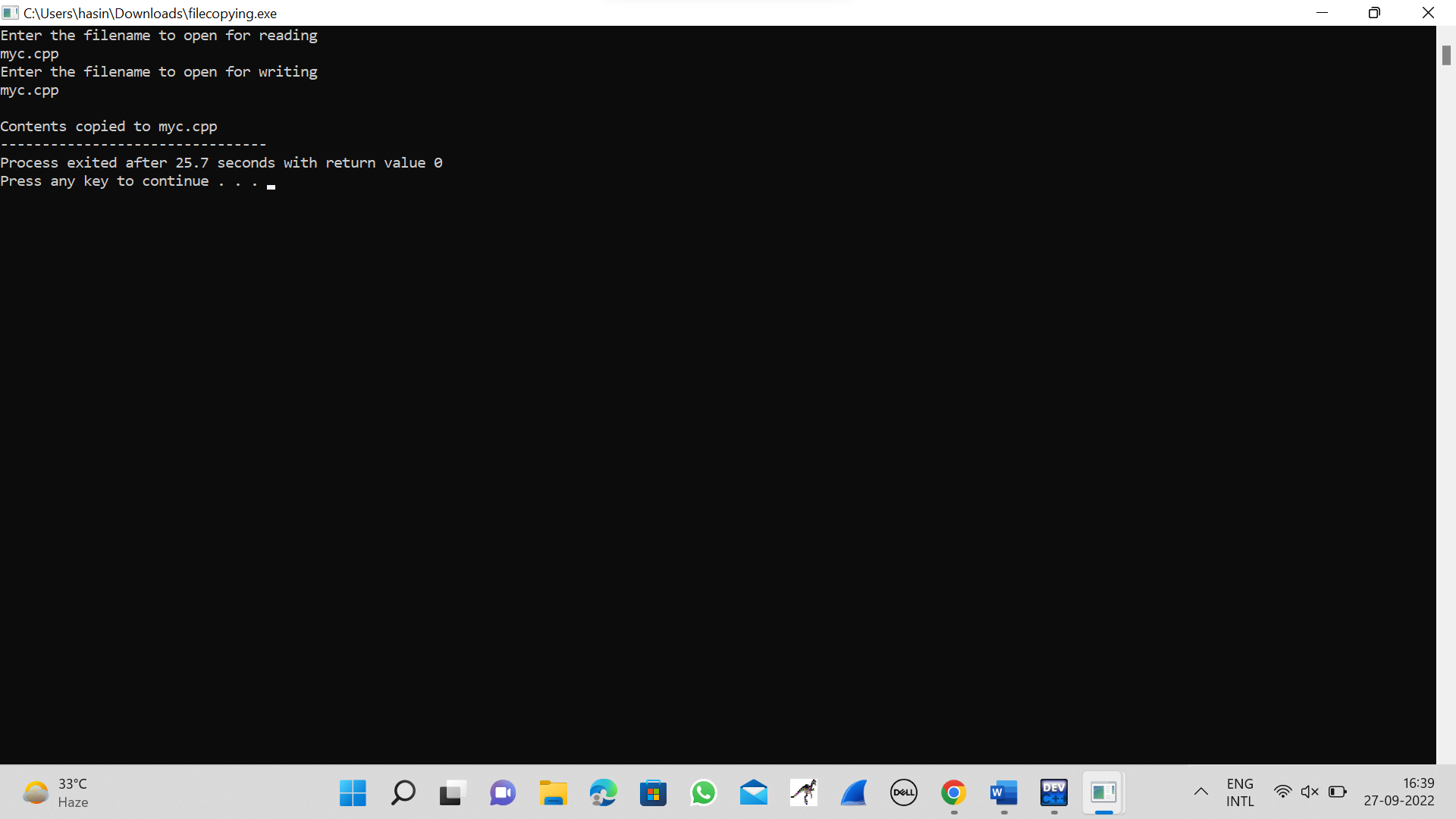
fclose(fptr1);

fclose(fptr2);

return 0;

}

**OUTPUT**

****

**3.FIRST COME FIRST SERVE(FCFS)**

#include<stdio.h>

int main()

{

int bt[10]={0},at[10]={0},tat[10]={0},wt[10]={0},ct[10]={0};

int n,sum=0;

float totalTAT=0,totalWT=0;

printf("Enter number of processes ");

scanf("%d",&n);

printf("Enter arrival time and burst time for each process\n\n");

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

{

printf("Arrival time of process[%d] ",i+1);

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

printf("Burst time of process[%d] ",i+1);

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

printf("\n");

}

for(int j=0;j<n;j++)

{

sum+=bt[j];

ct[j]+=sum;

}

for(int k=0;k<n;k++)

{

tat[k]=ct[k]-at[k];

totalTAT+=tat[k];

}

for(int k=0;k<n;k++)

{

wt[k]=tat[k]-bt[k];

totalWT+=wt[k];

}

printf("Solution: \n\n");

printf("P#\t AT\t BT\t CT\t TAT\t WT\t\n\n");

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

{

printf("P%d\t %d\t %d\t %d\t %d\t %d\n",i+1,at[i],bt[i],ct[i],tat[i],wt[i]);

}

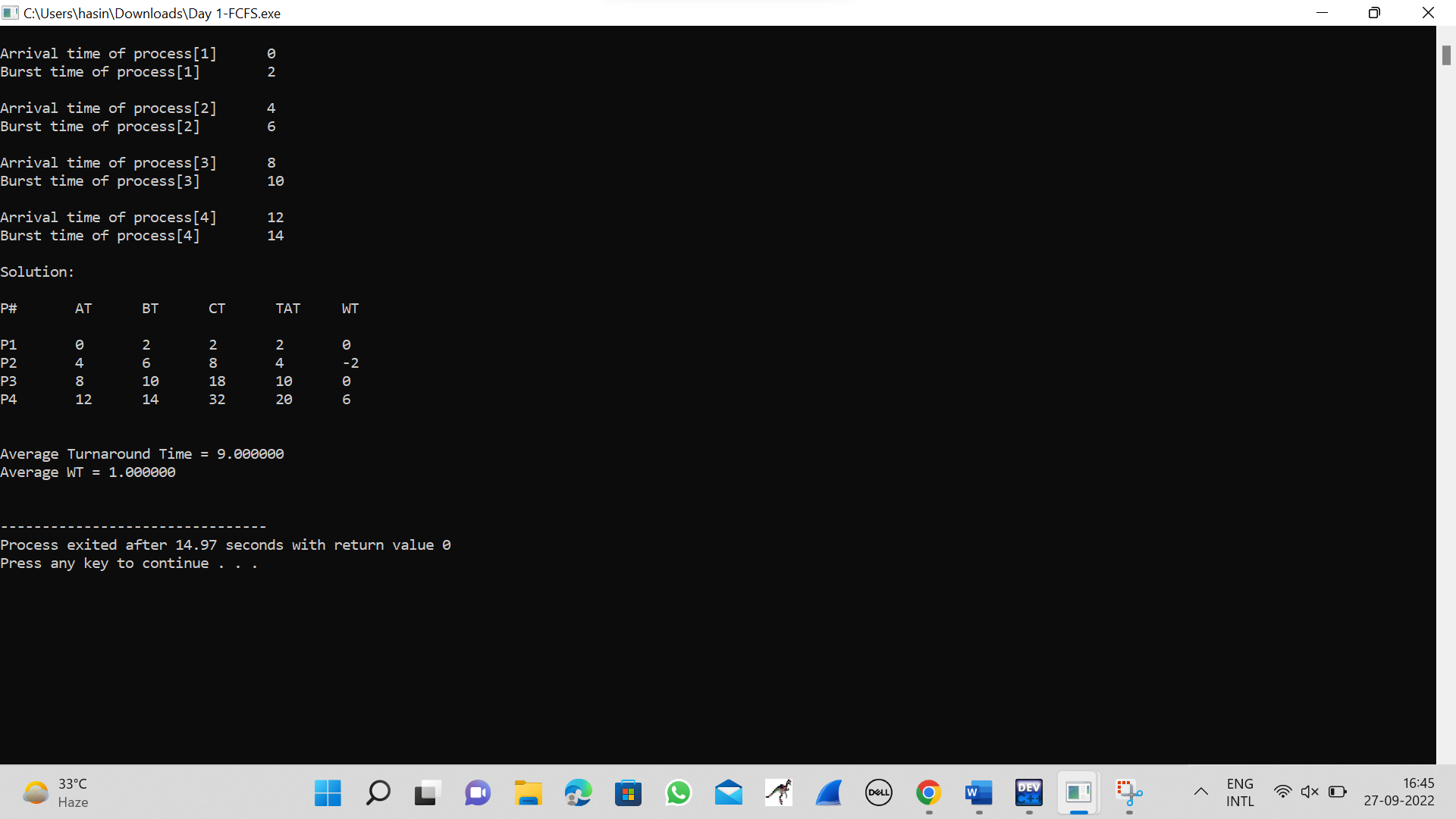
printf("\n\nAverage Turnaround Time = %f\n",totalTAT/n);

printf("Average WT = %f\n\n",totalWT/n);

return 0;

}

**OUTPUT**

****

**4.SHORTEST JOB FIRST(SJF)**

#include <stdio.h>

int main()

{

int A[100][4];

int i, j, n, total = 0, index, temp; float avg\_wt, avg\_tat;

printf("Enter number of process: "); scanf("%d", &n);

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

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

printf("P%d: ", i + 1); scanf("%d", &A[i][1]); A[i][0] = i + 1;

}

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

index = i;

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

if (A[j][1] < A[index][1]) index = j;

temp = A[i][1]; A[i][1] = A[index][1]; A[index][1] = temp;

temp = A[i][0];

A[i][0] = A[index][0]; A[index][0] = temp;

}

A[0][2] = 0;

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

A[i][2] = 0;

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

A[i][2] += A[j][1];

total += A[i][2];

}

avg\_wt = (float)total / n; total = 0;

printf("P BT WT TAT\n"); for (i = 0; i < n; i++) {

A[i][3] = A[i][1] + A[i][2];

total += A[i][3];

printf("P%d %d %d %d\n", A[i][0], A[i][1], A[i][2], A[i][3]);

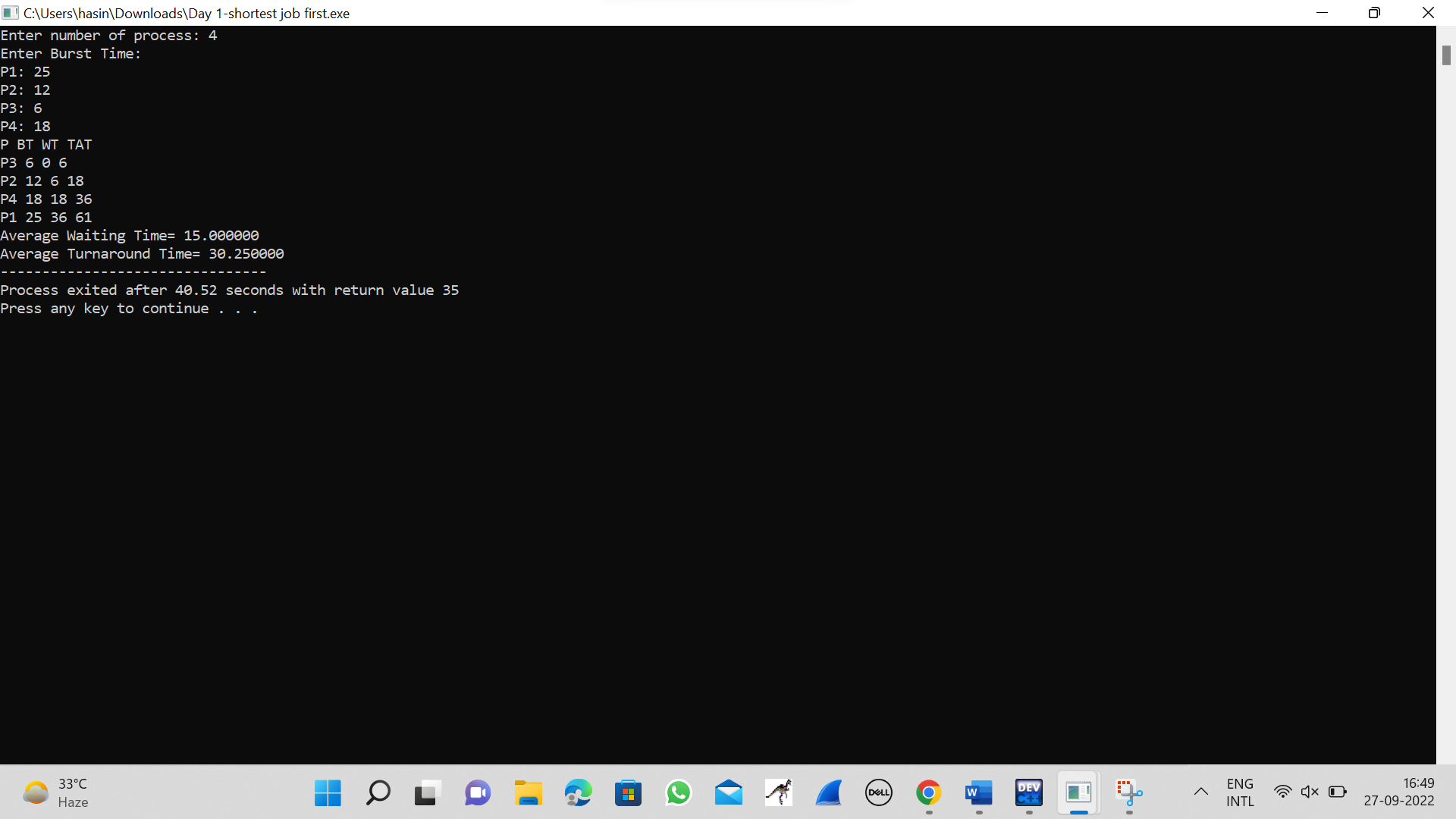
}

avg\_tat = (float)total / n;

printf("Average Waiting Time= %f", avg\_wt); printf("\nAverage Turnaround Time= %f", avg\_tat);

}

**OUTPUT**



**5.PRIORITY SCHEDULLING**

#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;

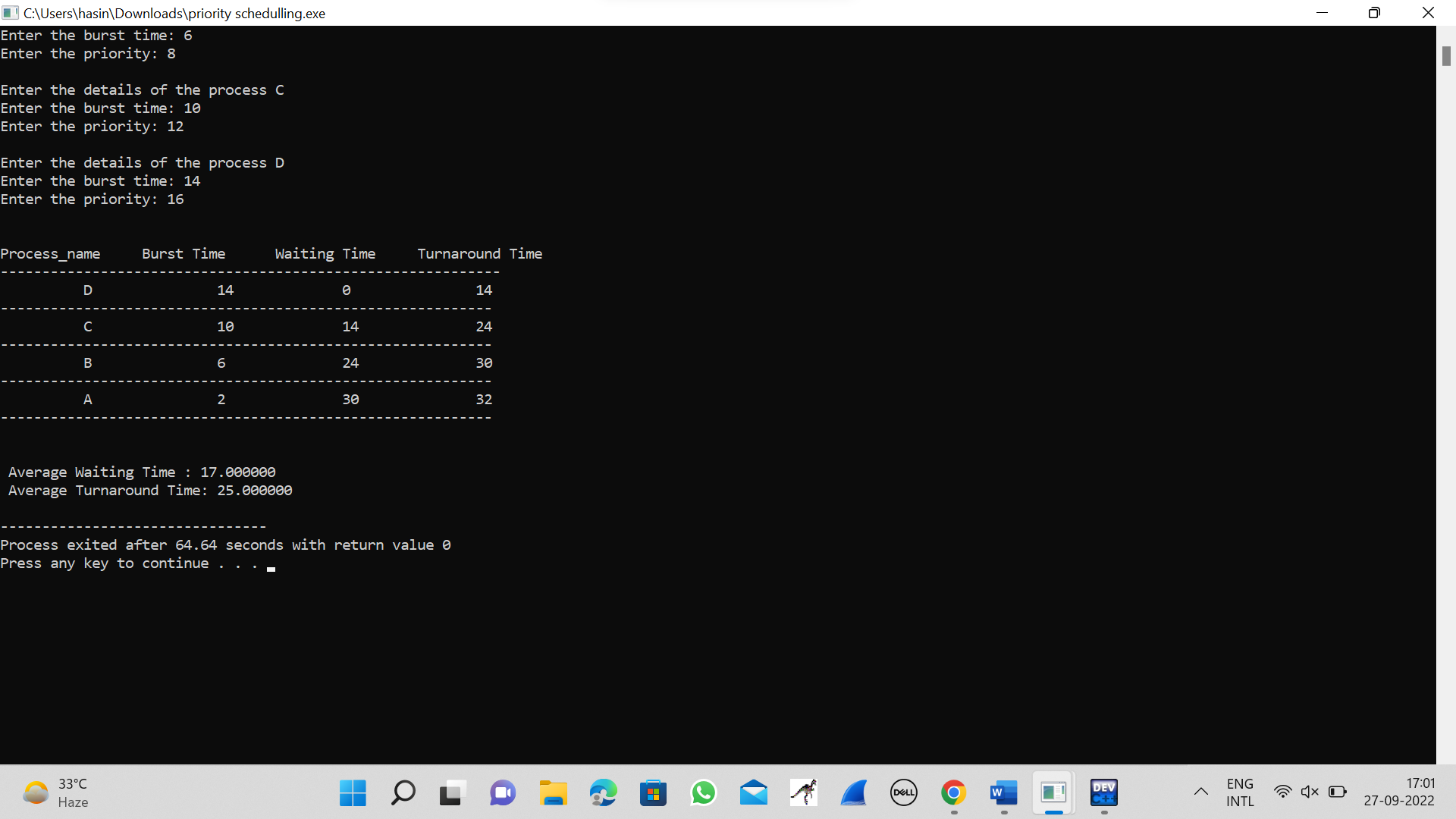
printf("\n\n Average Waiting Time : %f", average\_waiting\_time);

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

return 0;

}

**OUTPUT**

****

**6.ROUND ROBIN SCHEDULLING**

#include<stdio.h>

int main()

{

int count,j,n,time,remain,flag=0,time\_quantum;

int wait\_time=0,turnaround\_time=0,at[10],bt[10],rt[10];

printf("Enter Total Process:\t ");

scanf("%d",&n);

remain=n;

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

{

printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);

scanf("%d",&at[count]);

scanf("%d",&bt[count]);

rt[count]=bt[count];

}

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

scanf("%d",&time\_quantum);

printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");

for(time=0,count=0;remain!=0;)

{

if(rt[count]<=time\_quantum && rt[count]>0)

{

time+=rt[count];

rt[count]=0;

flag=1;

}

else if(rt[count]>0)

{

rt[count]-=time\_quantum;

time+=time\_quantum;

}

if(rt[count]==0 && flag==1)

{

remain--;

printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);

wait\_time+=time-at[count]-bt[count];

turnaround\_time+=time-at[count];

flag=0;

}

if(count==n-1)

count=0;

else if(at[count+1]<=time)

count++;

else

count=0;

}

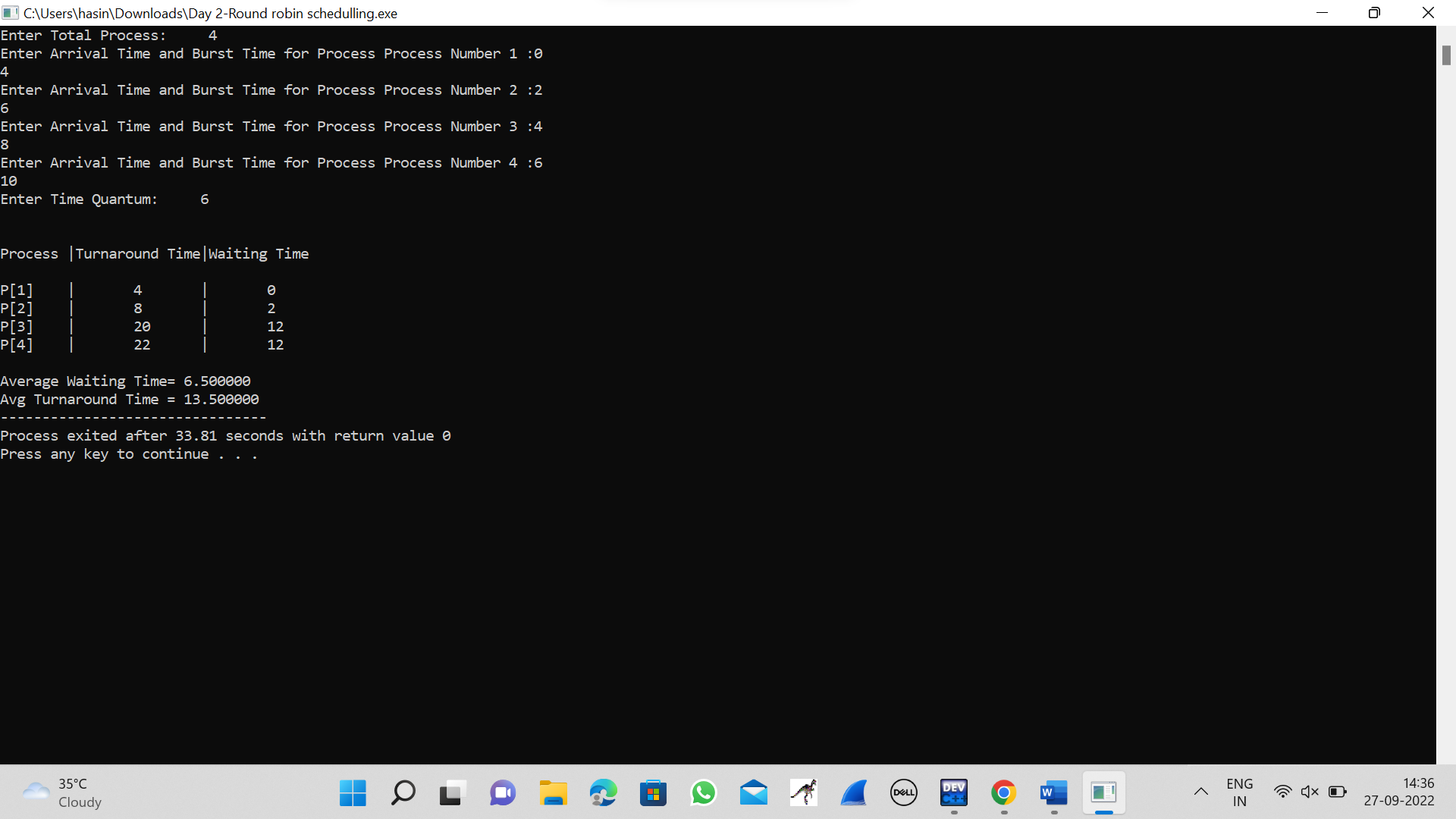
printf("\nAverage Waiting Time= %f\n",wait\_time\*1.0/n);

printf("Avg Turnaround Time = %f",turnaround\_time\*1.0/n);

return 0;

}

**OUTPUT**

****

**7.INTER-PROCESS COMMUNICATION**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/shm.h>

#include<string.h>

int main()

{

int i;

void \*shared\_memory;

char buff[100];

int shmid;

shmid=shmget((key\_t)2345, 1024, 0666|IPC\_CREAT);

printf("Key of shared memory is %d\n",shmid);

shared\_memory=shmat(shmid,NULL,0);

printf("Process attached at %p\n",shared\_memory);

printf("Enter some data to write to shared memory\n");

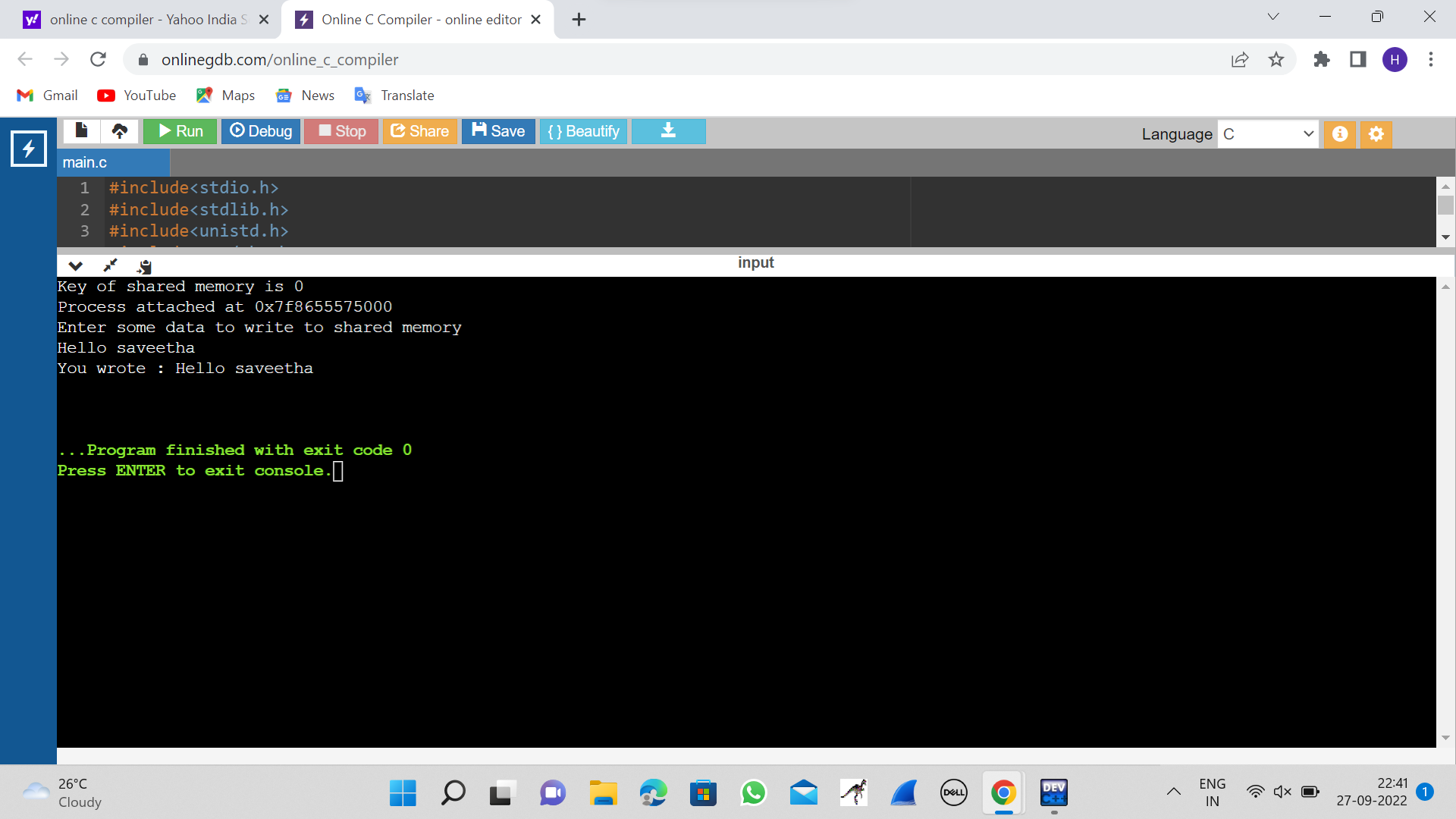
read(0,buff,100);

strcpy(shared\_memory,buff);

printf("You wrote : %s\n",(char \*)shared\_memory);

}

**OUTPUT**

****

**8.MULTITHREADING**

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

void \*myThreadFun(void \*vargp)

{

sleep(1);

printf("Printing GeeksQuiz from Thread \n");

return NULL;

}

int main()

{

pthread\_t thread\_id;

printf("Before Thread\n");

pthread\_create(&thread\_id, NULL, myThreadFun, NULL);

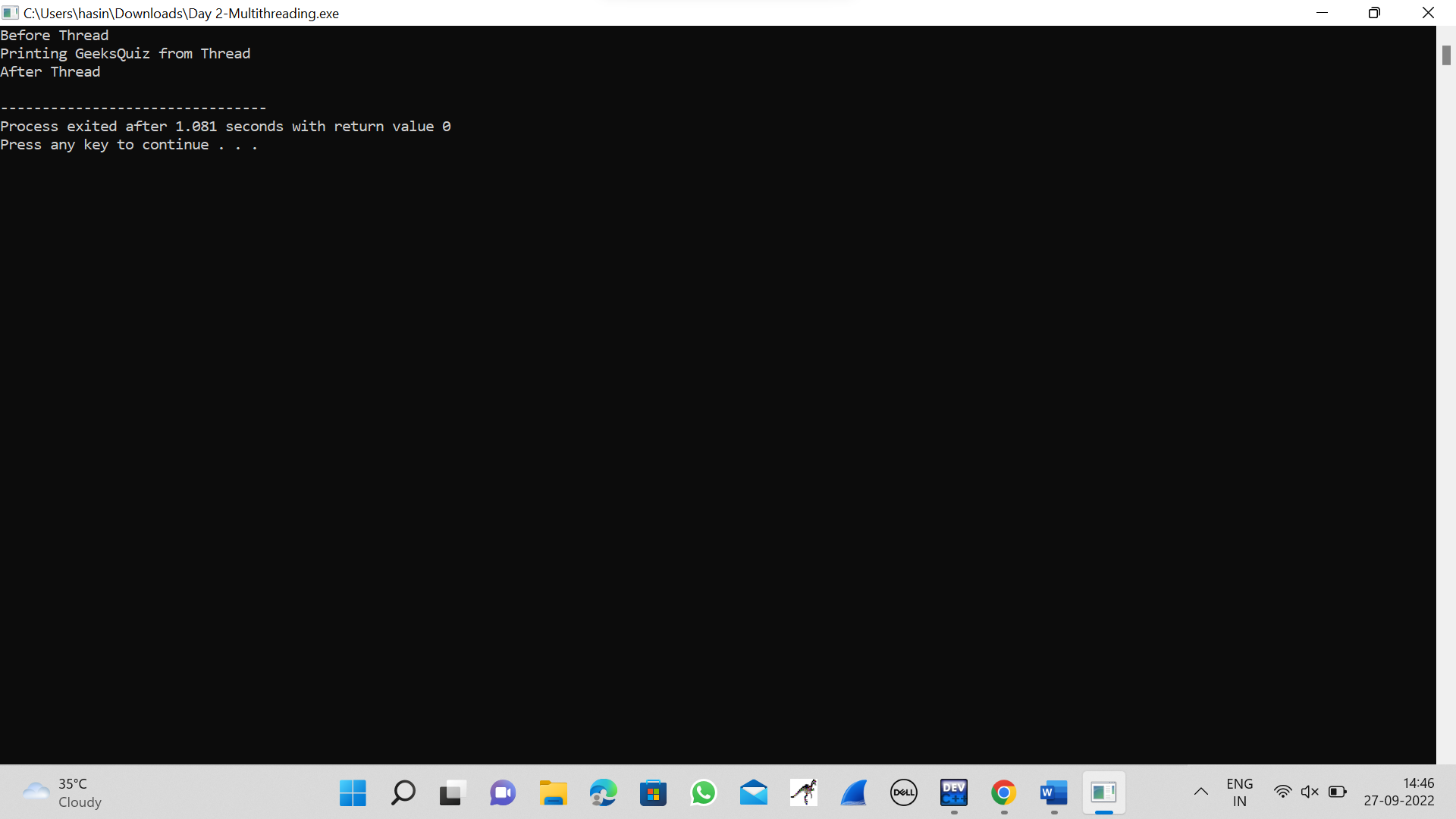
pthread\_join(thread\_id, NULL);

printf("After Thread\n");

exit(0);

}

**OUTPUT**

****

**9.DINING-PHILOSOPHERS**

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<semaphore.h>

#include<unistd.h>

sem\_t room;

sem\_t chopstick[5];

void \* philosopher(void \*);

void eat(int);

int main()

{

int i,a[5];

pthread\_t tid[5];

sem\_init(&room,0,4);

for(i=0;i<5;i++)

sem\_init(&chopstick[i],0,1);

for(i=0;i<5;i++){

a[i]=i;

pthread\_create(&tid[i],NULL,philosopher,(void \*)&a[i]);

}

for(i=0;i<5;i++)

pthread\_join(tid[i],NULL);

}

void \* philosopher(void \* num)

{

int phil=\*(int \*)num;

sem\_wait(&room);

printf("\nPhilosopher %d has entered room",phil);

sem\_wait(&chopstick[phil]);

sem\_wait(&chopstick[(phil+1)%5]);

eat(phil);

sleep(2);

printf("\nPhilosopher %d has finished eating",phil);

sem\_post(&chopstick[(phil+1)%5]);

sem\_post(&chopstick[phil]);

sem\_post(&room);

}

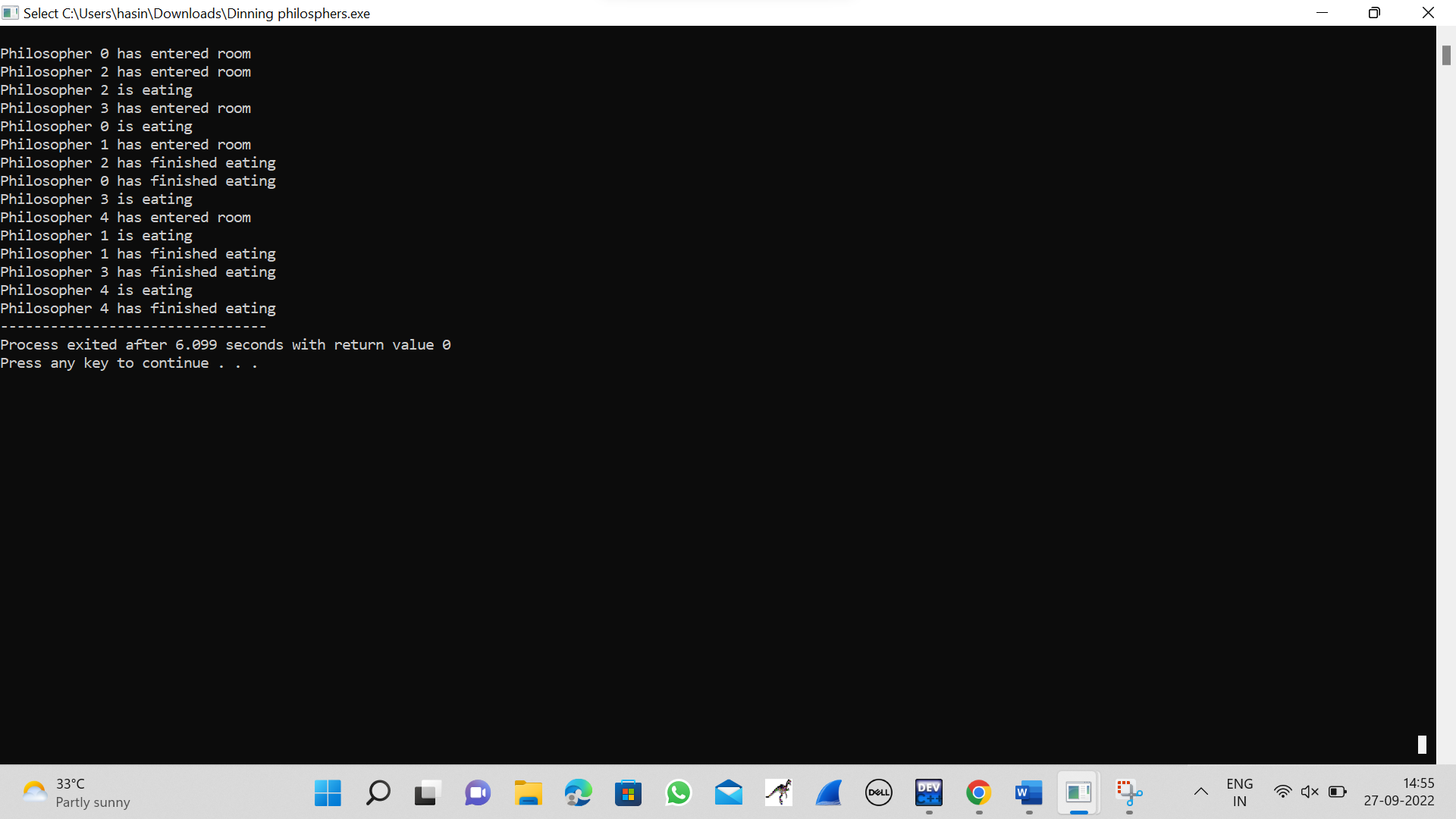
void eat(int phil)

{

printf("\nPhilosopher %d is eating",phil);

}

**OUTPUT**

****

**10.MEMORY ALLOCATION USING FIRST FIT STRATEGY**

#include<stdio.h>

int main( )

{

int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;

for(i = 0; i < 10; i++)

{

flags[i] = 0;

allocation[i] = -1;

}

printf("Enter no. of blocks: ");

scanf("%d", &bno);

printf("\nEnter size of each block: ");

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

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

printf("\nEnter no. of processes: ");

scanf("%d", &pno);

printf("\nEnter size of each process: ");

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

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

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

for(j = 0; j < bno; j++)

if(flags[j] == 0 && bsize[j] >= psize[i])

{

allocation[j] = i;

flags[j] = 1;

break;

}

printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");

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

{

printf("\n%d\t\t%d\t\t", i+1, bsize[i]);

if(flags[i] == 1)

printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);

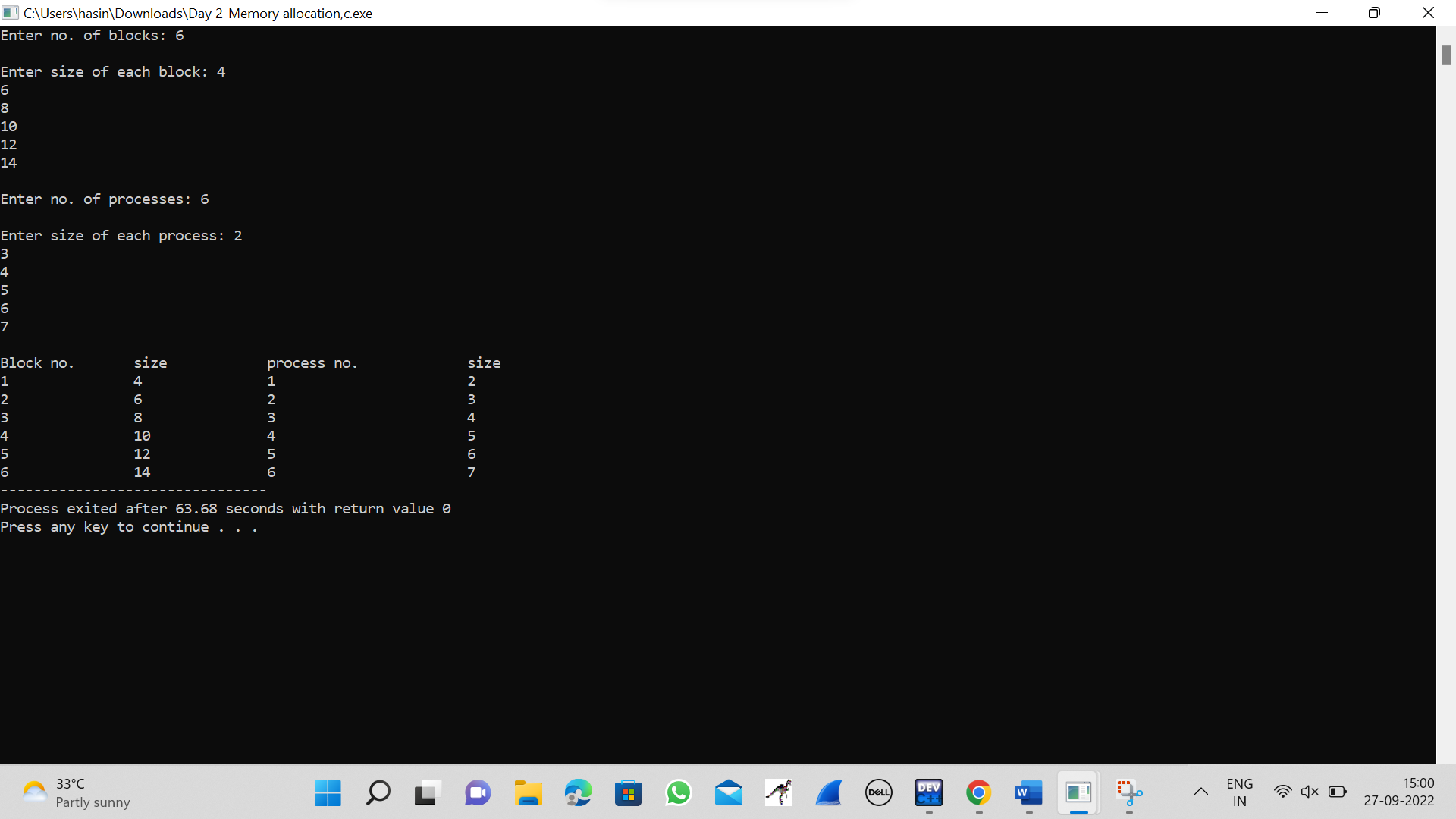
else

printf("Not allocated");

}

}

**OUTPUT**

****