Registration Number: 19BCE2119

Name: Gaurav Kumar Singh

Course: CSE2005 Operating Systems (L7+L8)

OS Assignment 2

Process and Thread Management

1. Write a program to create a thread and perform the following (Easy) • Create a thread runner function • Set the thread attributes • Join the parent and thread • Wait for the thread to complete

**CODE**

#include <pthread.h>

#include <unistd.h>

#define NUM\_THREADS 5

void \*wait(void \*t) {

int i;

long tid;

tid = (long)t;

sleep(1);

printf("Sleeping \n");

printf("Thread id : %d\n" ,tid);

pthread\_exit(NULL);

}

int main () {

int rc;

int i;

pthread\_t threads[NUM\_THREADS];

pthread\_attr\_t attr;

void \*status;

pthread\_attr\_init(&attr);

pthread\_attr\_setdetachstate(&attr, PTHREAD\_CREATE\_JOINABLE);

for( i = 0; i < NUM\_THREADS; i++ ) {

printf("parent creating thread, %d\n",i);

rc = pthread\_create(&threads[i], &attr, wait, (void \*)i );

if (rc) {

printf("Error:unable to create thread, %d\n",rc);

exit(-1);

}

}

pthread\_attr\_destroy(&attr);

for( i = 0; i < NUM\_THREADS; i++ ) {

rc = pthread\_join(threads[i], &status);

if (rc) {

printf ("Error:unable to join, %d\n" ,rc);

exit(-1);

}

printf("Parent Process: completed thread id :%d",i) ;

printf(" exiting with status :%d\n",status);

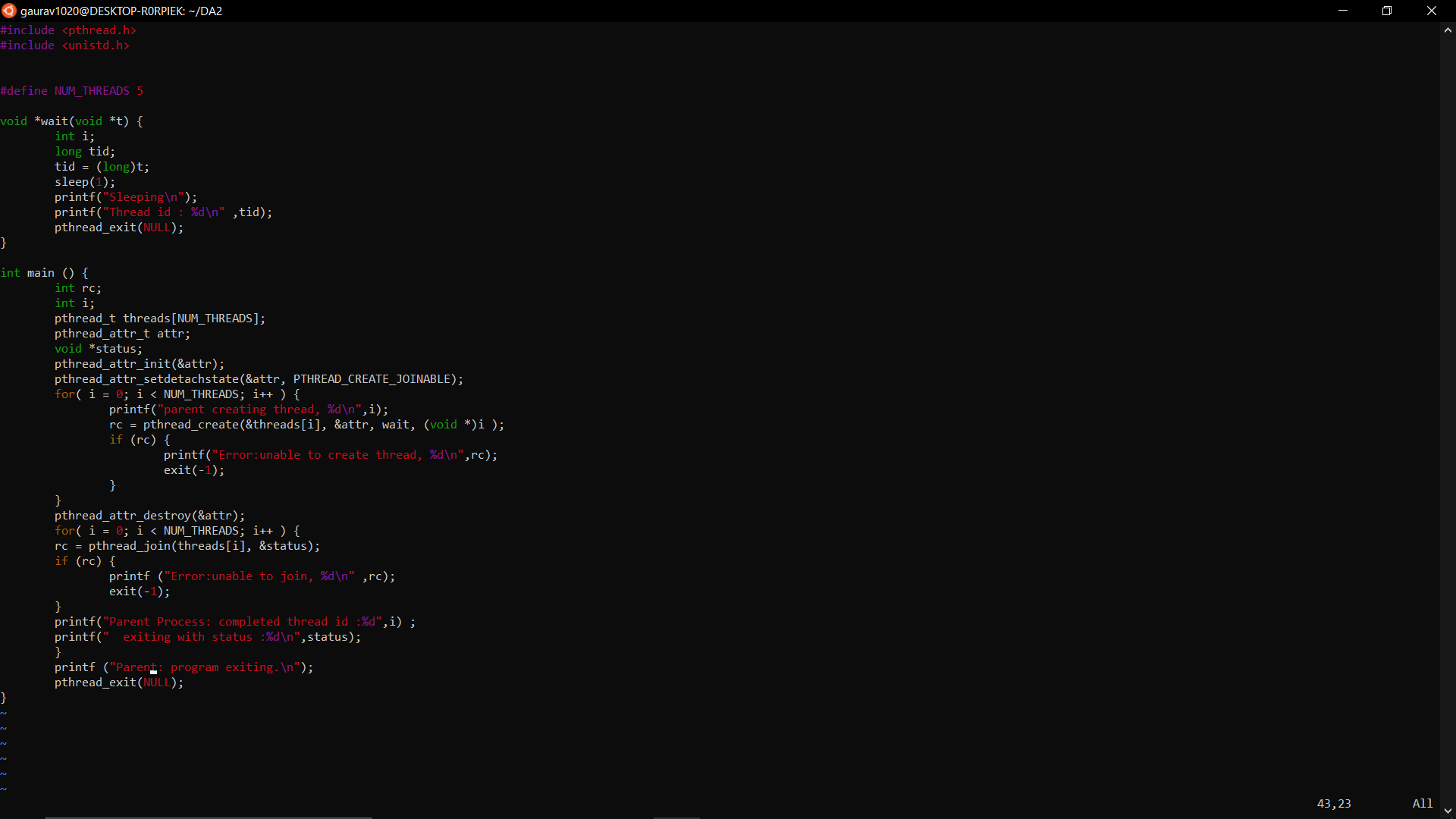
}

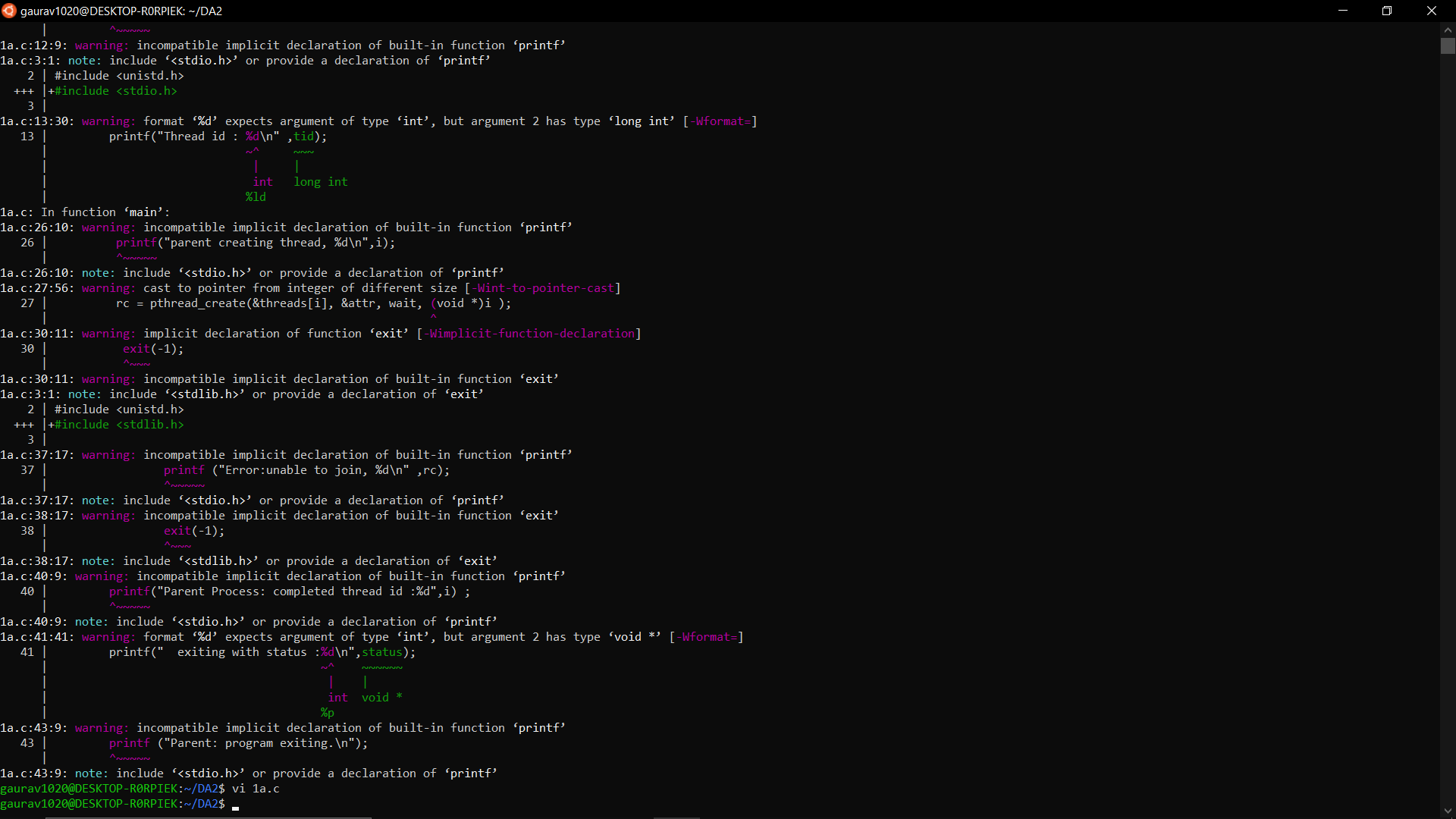
printf ("Parent: program exiting.\n");

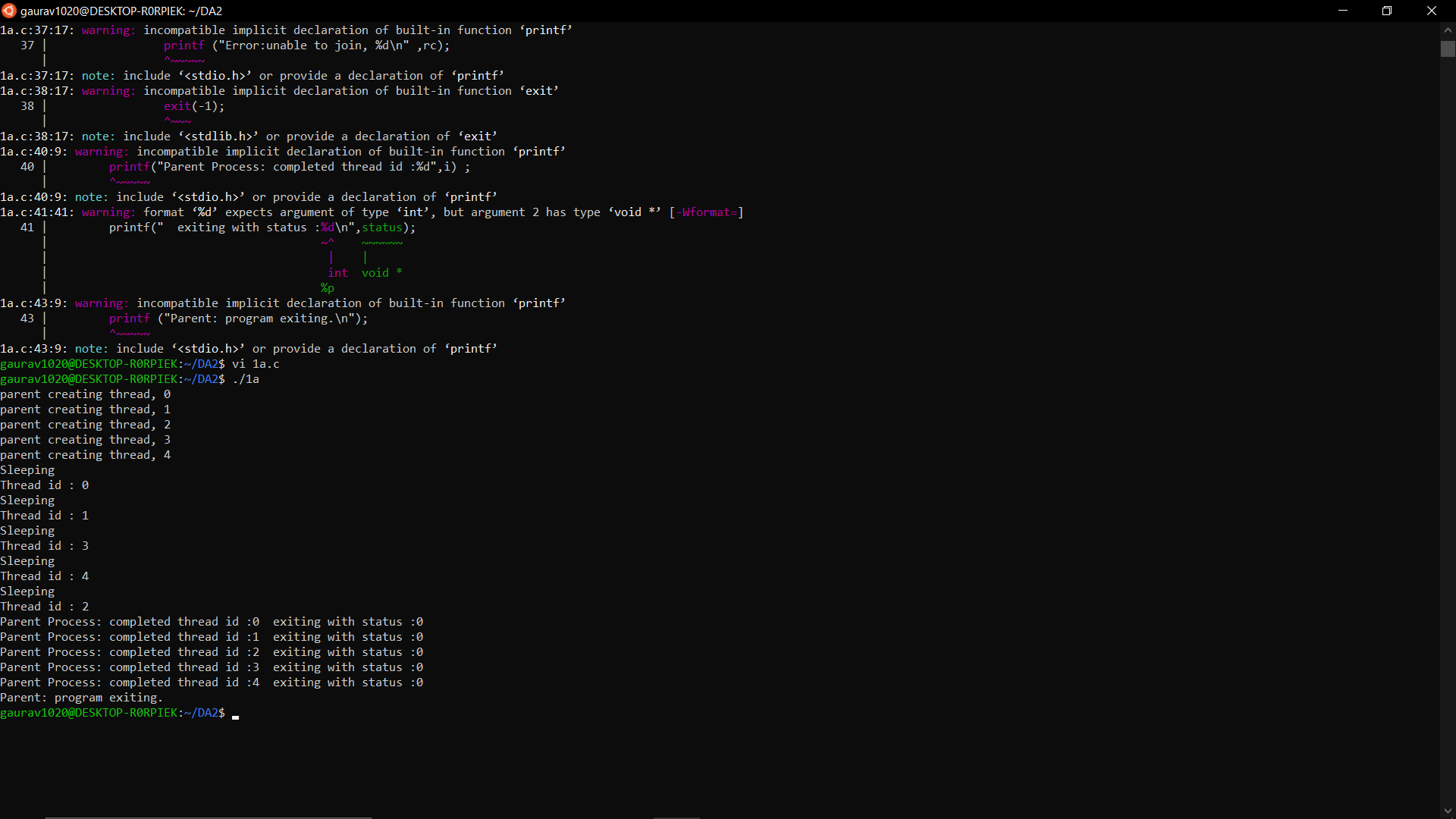
pthread\_exit(NULL);

}

**OUTPUT**







1. Write a program to create a thread to find the factorial of a natural number ‘n’. (Medium)

**CODE**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

void \*Factorial(void\* var){

printf("\nThis is thread process with pid:%d\n",getpid());

int n = (int \*)var;

int fact=1;

for (int i=n; i>0; i--){

fact=fact\*i;

}

printf("Factorial of %d is: %d\n", n, fact);

}

int main() {

int n; printf("\nThis is parent process with pid:%d",getpid());

printf("\nBefore Thread Creation.\nThread will now be Created...");

printf("\nEnter the number whose Factorial you want to calculate: ");

scanf("%d",&n);

pthread\_t tid;

pthread\_create (&tid,NULL,Factorial,(void\*)n);

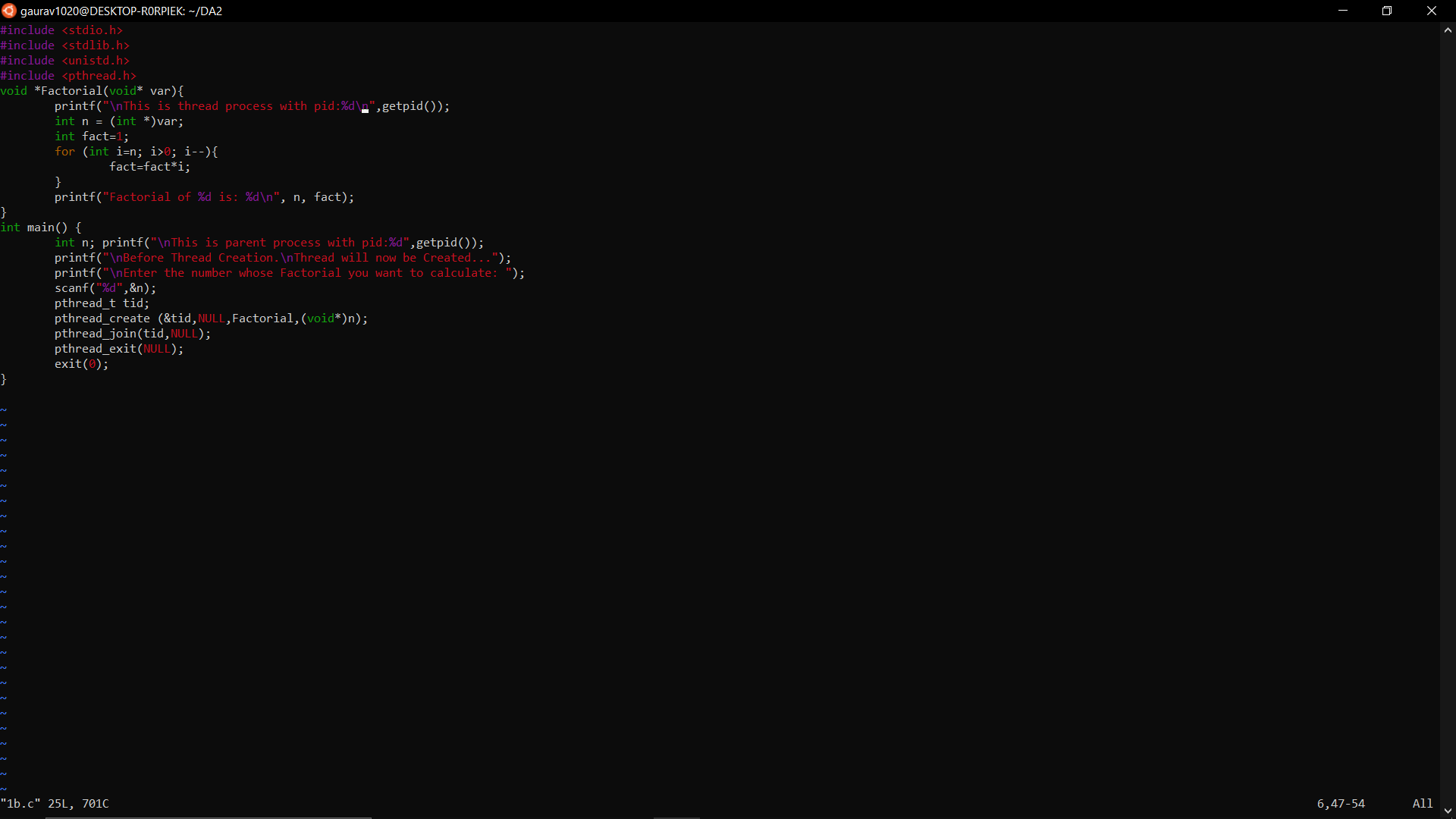
pthread\_join(tid,NULL);

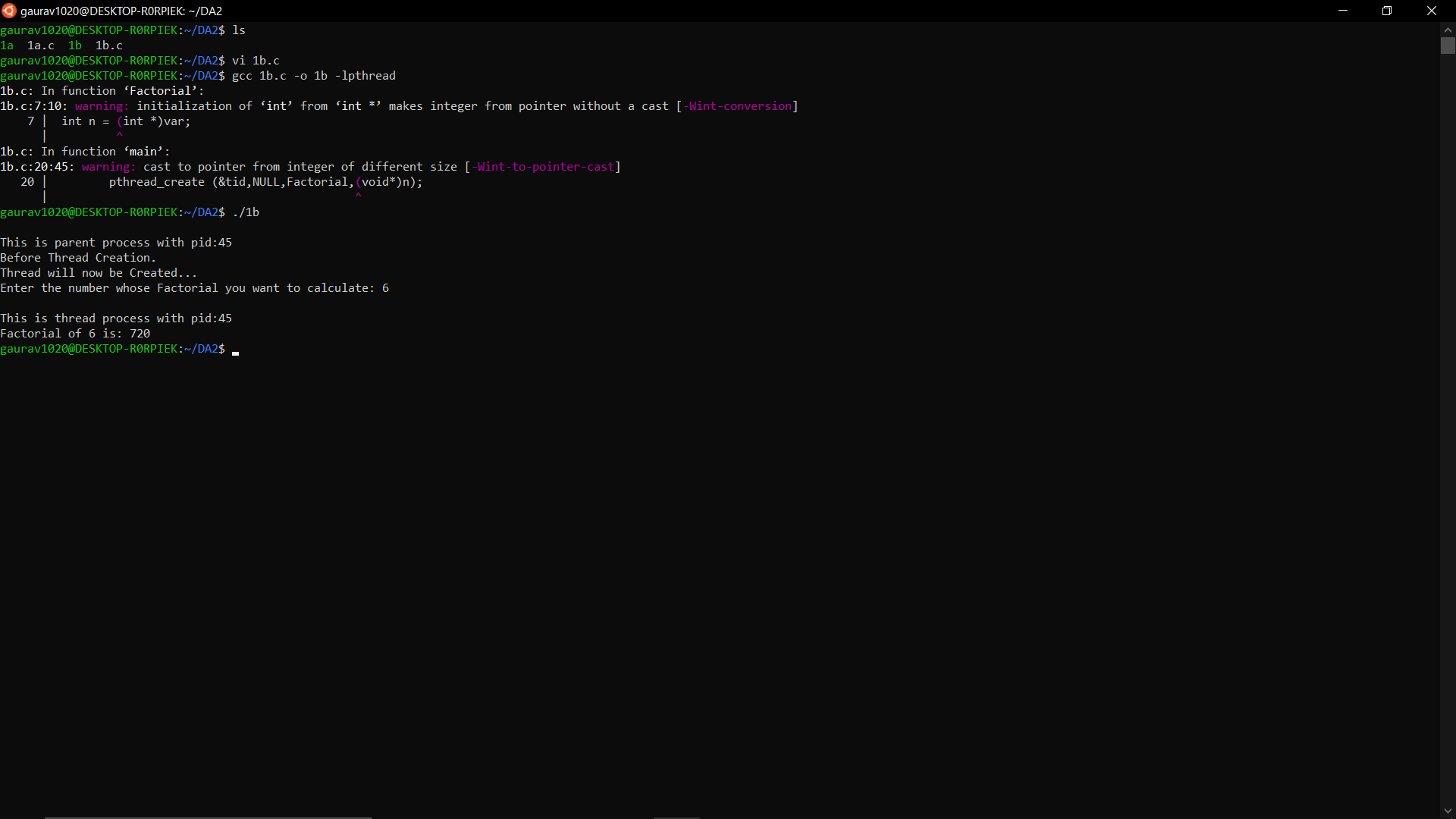
pthread\_exit(NULL);

exit(0);

}

**OUTPUT**





1. Assume that two processes named client and server running in the system. It is required that these two processes should communicate with each other using shared memory concept. The server writes alphabets from a..z to the shared memory .the client should read the alphabets from the shared memory and convert it to A…Z. Write a program to demonstrate the above mentioned scenario. (Medium)

**CODE SERVER**

#include<iostream>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

using namespace std;

int main()

{

key\_t my\_key = ftok("shmfile",65);

int shmid = shmget(my\_key,1024,0666|IPC\_CREAT);

char \*str = (char\*) shmat(shmid,(void\*)0,0);

cout<<"Write data:";

fgets(str, 50, stdin);

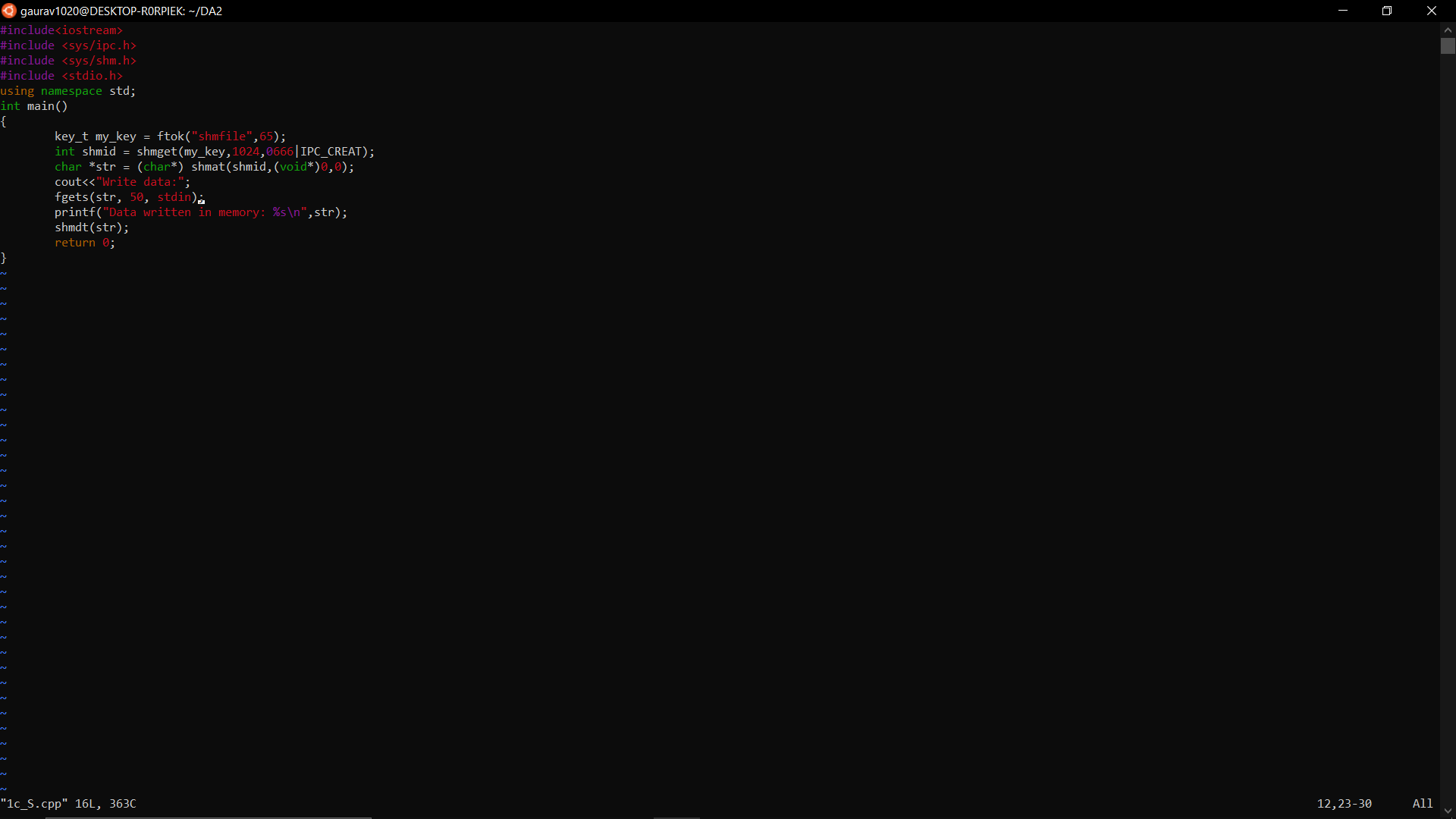
printf("Data written in memory: %s\n",str);

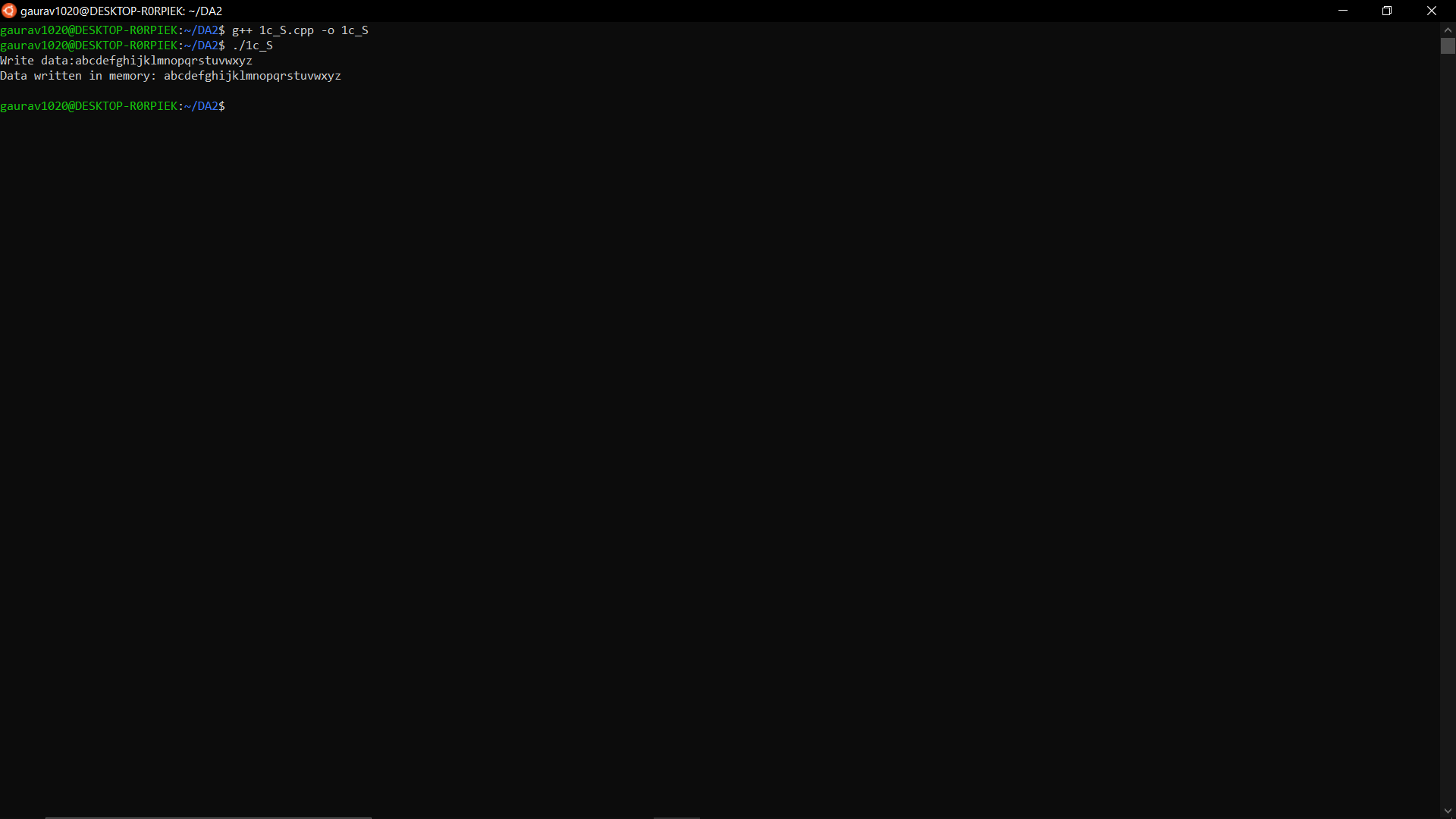
shmdt(str);

return 0;

}

**OUTPUT SERVER**





**CODE CLIENT**

#include<bits/stdc++.h>

#include <iostream>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <cstring>

using namespace std;

int main()

{

key\_t my\_key = ftok("shmfile",65);

int shmid = shmget(my\_key,1024,0666|IPC\_CREAT);

char\* str = (char\*) shmat(shmid,(void\*)0,0);

printf("Data read from memory:");

for (int x=0; x<strlen(str); x++)

putchar(toupper(str[x]));

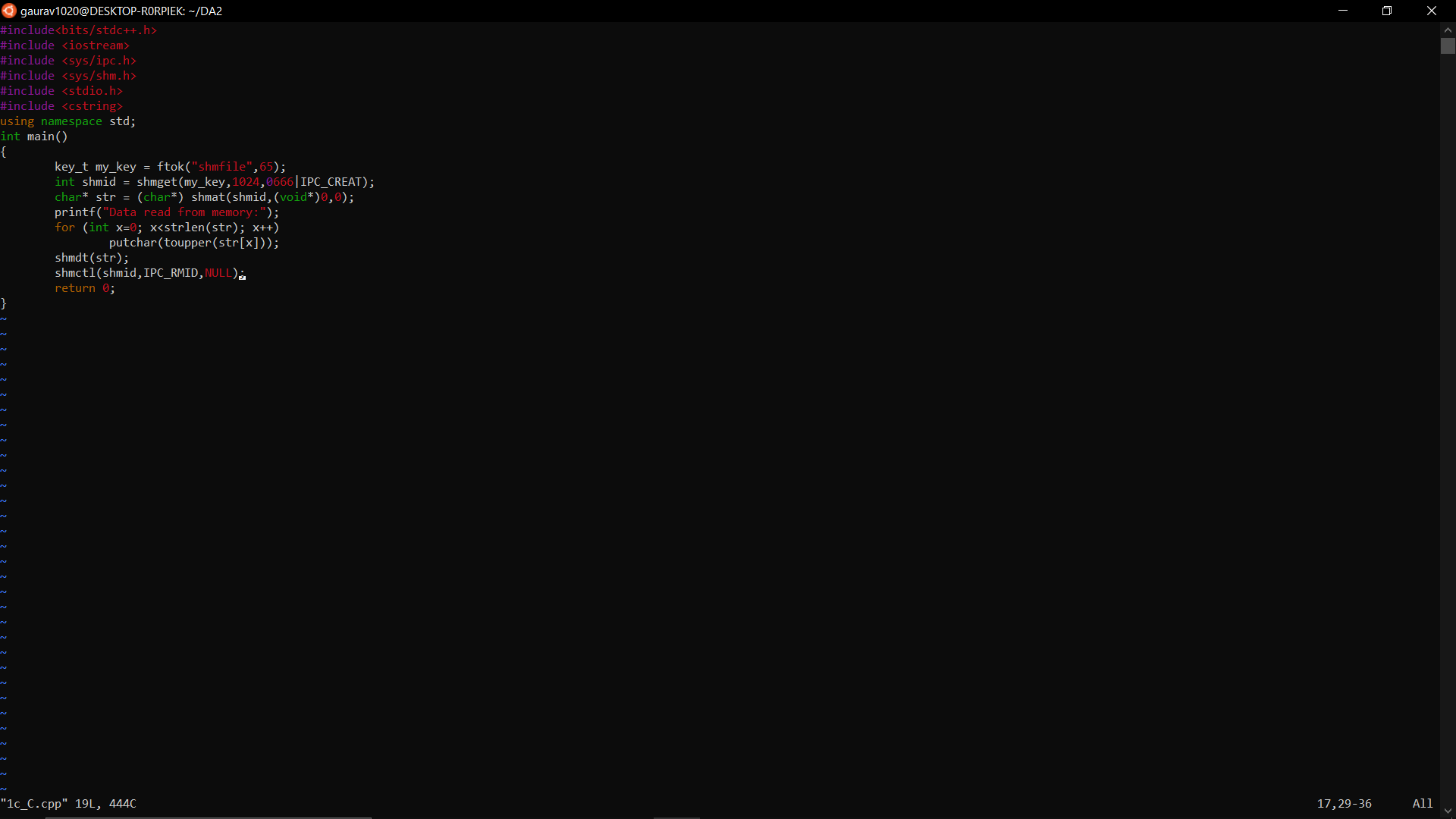
shmdt(str);

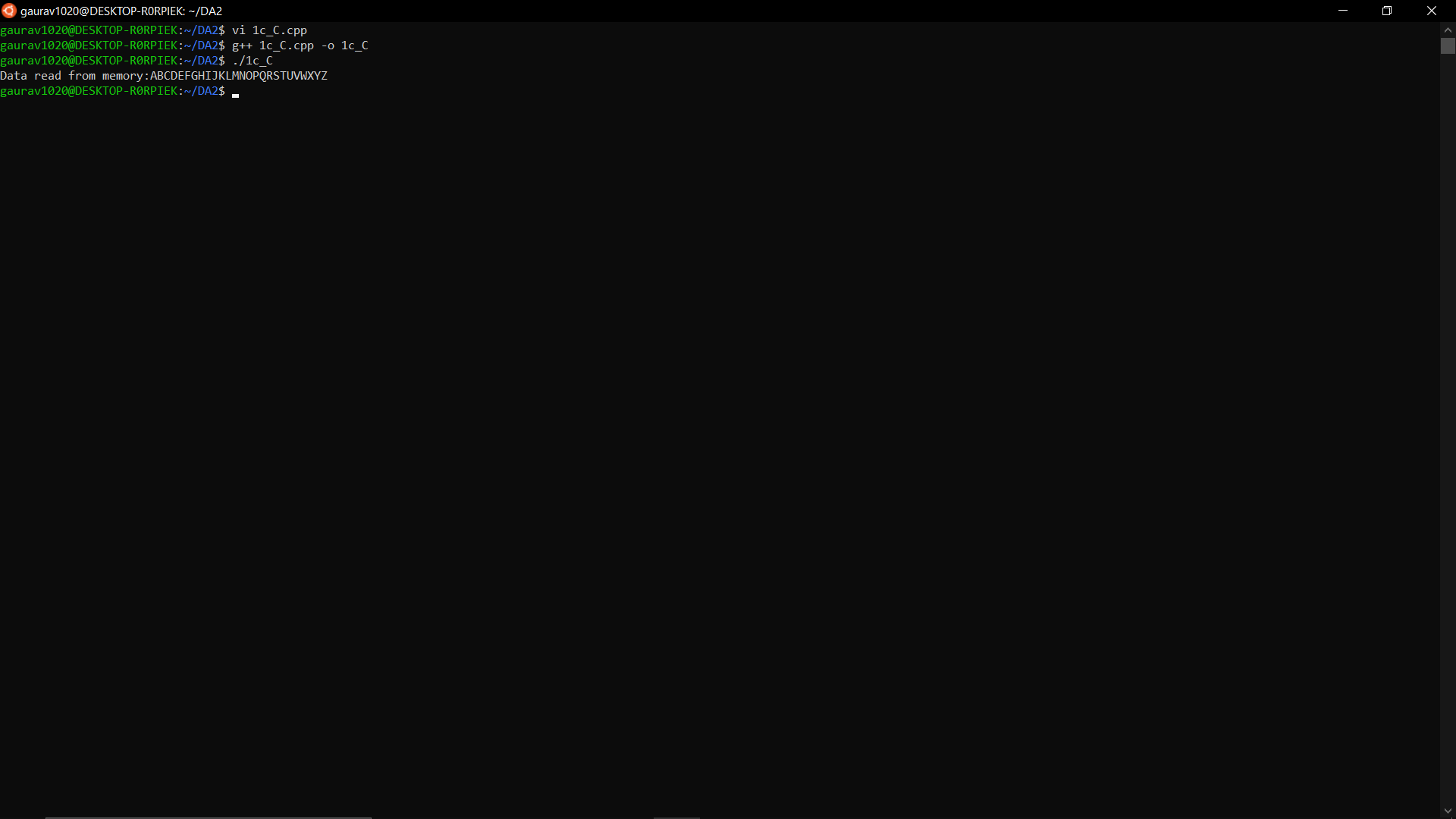
shmctl(shmid,IPC\_RMID,NULL);

return 0;

}

**OUTPUT CLIENT**





1. Write a multithreaded program that calculates various statistical values for a list of numbers. This program will be passed a series of numbers on the command line and will then create three separate worker threads. One thread will determine the average of the numbers, the second will determine the maximum value, and the third will determine the minimum value. For example, suppose your program is passed the integers 90 81 78 95 79 72 85 , the program will report the average value as 82. The minimum value as 72. The maximum value as 95. The variables representing the average, minimum, and maximum values will be stored globally. The worker threads will set these values, and the parent thread will output the values once the workers have exited. (High)

**CODE**

#include<stdio.h>

#include<pthread.h>

int arr[50],n,i;

void \*th()

{

float sum=0;

float average;

printf("enter your number :=");

scanf("%d",&n);

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

{

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

}

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

{

sum=sum+arr[i];

}

average=sum/n;

printf("The average value is:%f",average);

}

void \*th1()

{

int temp=arr[0];

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

{

if(temp>arr[i])

{

temp=arr[i];

}

}

printf("\nThe Minimum value is:=%d",temp);

}

void \*th2()

{

int temp=arr[0];

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

{

if(temp<arr[i])

{

temp=arr[i];

}

}

printf("\nThe Maximum value is:=%d",temp);

}

int main()

{

int n,i;

pthread\_t t1;

pthread\_t t2;

pthread\_t t3;

n=pthread\_create(&t1,NULL,&th,NULL);

pthread\_join(t1,NULL);

n=pthread\_create(&t2,NULL,&th1,NULL);

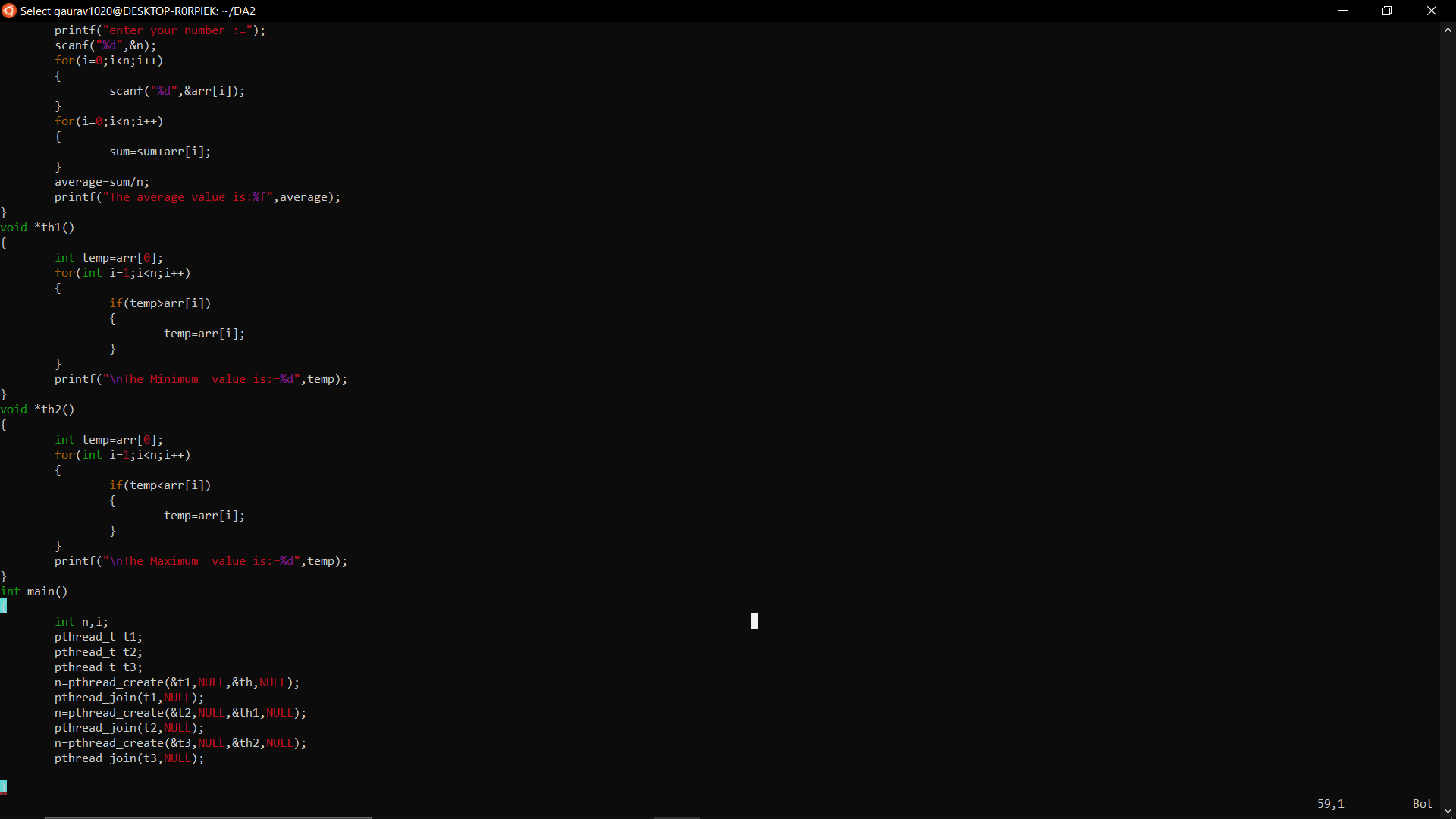
pthread\_join(t2,NULL);

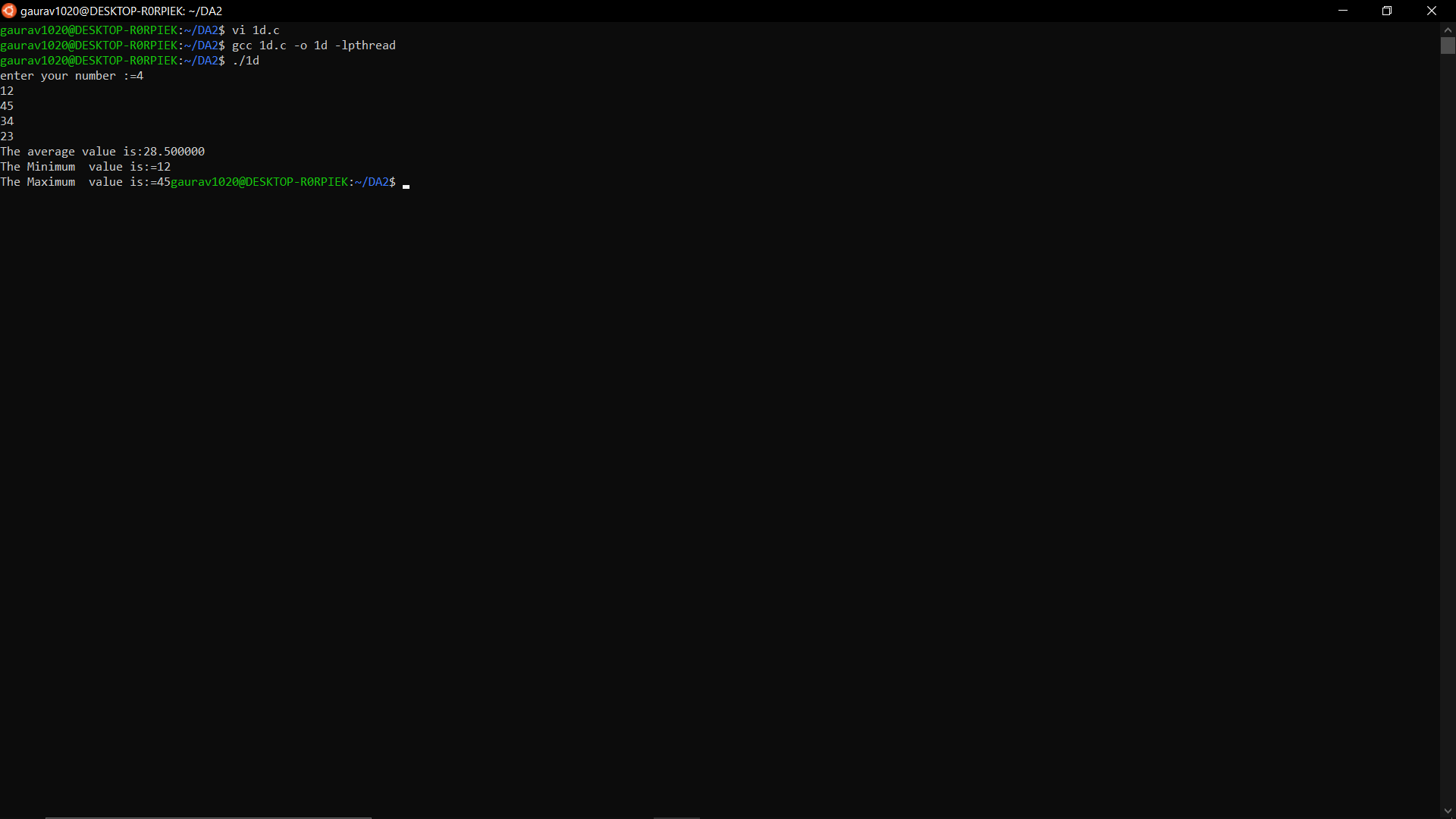
n=pthread\_create(&t3,NULL,&th2,NULL);

pthread\_join(t3,NULL);

}

**OUTPUT**





CPU Scheduling

1. Implement the various process scheduling algorithms such as FCFS, SJF, Priority (Non Preemptive). (Easy )

**CODE FCFS**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <stdlib.h>

#include <string.h>

struct Process {

int p\_id;

int AT;

int BT;

int CT;

int TAT;

int WT;

};

void display(struct Process Process\_Array[], int n){

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

int Pno=i+1;

printf("\n\nProcess %d\n",Pno);

printf("p\_id=%d\n",Process\_Array[i].p\_id);

printf("AT=%d\n",Process\_Array[i].AT);

printf("BT=%d\n",Process\_Array[i].BT);

printf("CT=%d\n",Process\_Array[i].CT);

printf("TAT=%d\n",Process\_Array[i].TAT);

printf("WT=%d\n",Process\_Array[i].WT);

}

}

void getStats(struct Process Process\_Array[], int n){

printf("\nEnter the details of every process in increasing order of their arrival times.\n");

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

printf("Enter PID of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].p\_id);

printf("Enter Arrival Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].AT);

printf("Enter Burst Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].BT);

}

}

void calcCT(struct Process Process\_Array[],int n){

int timeline=0;

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

if (timeline<Process\_Array[i].AT) {

timeline=Process\_Array[i].AT;

}

timeline = timeline + Process\_Array[i].BT;

Process\_Array[i].CT = timeline;

}

}

void calcTAT(struct Process Process\_Array[],int n){

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

Process\_Array[i].TAT = Process\_Array[i].CT - Process\_Array[i].AT;

}

}

void calcWT(struct Process Process\_Array[],int n){

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

Process\_Array[i].WT = Process\_Array[i].TAT - Process\_Array[i].BT;

}

}

void calcAvgTAT(struct Process Process\_Array[],int n){

float sumTAT=0;

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

sumTAT = sumTAT + Process\_Array[i].TAT;

}

printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));

}

void calcAvgWT(struct Process Process\_Array[],int n){

float sumWT=0;

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

sumWT = sumWT + Process\_Array[i].WT;

}

printf("\n\nAverage Waiting time= %.3f\n", (sumWT/n));

}

int main(){

int n;

printf("\nEnter the number of processes in the system:");

scanf ("%d",&n);

struct Process Process\_Array[100];

getStats(Process\_Array, n);

calcCT(Process\_Array, n);

calcTAT(Process\_Array, n);

calcWT(Process\_Array, n);

display(Process\_Array, n);

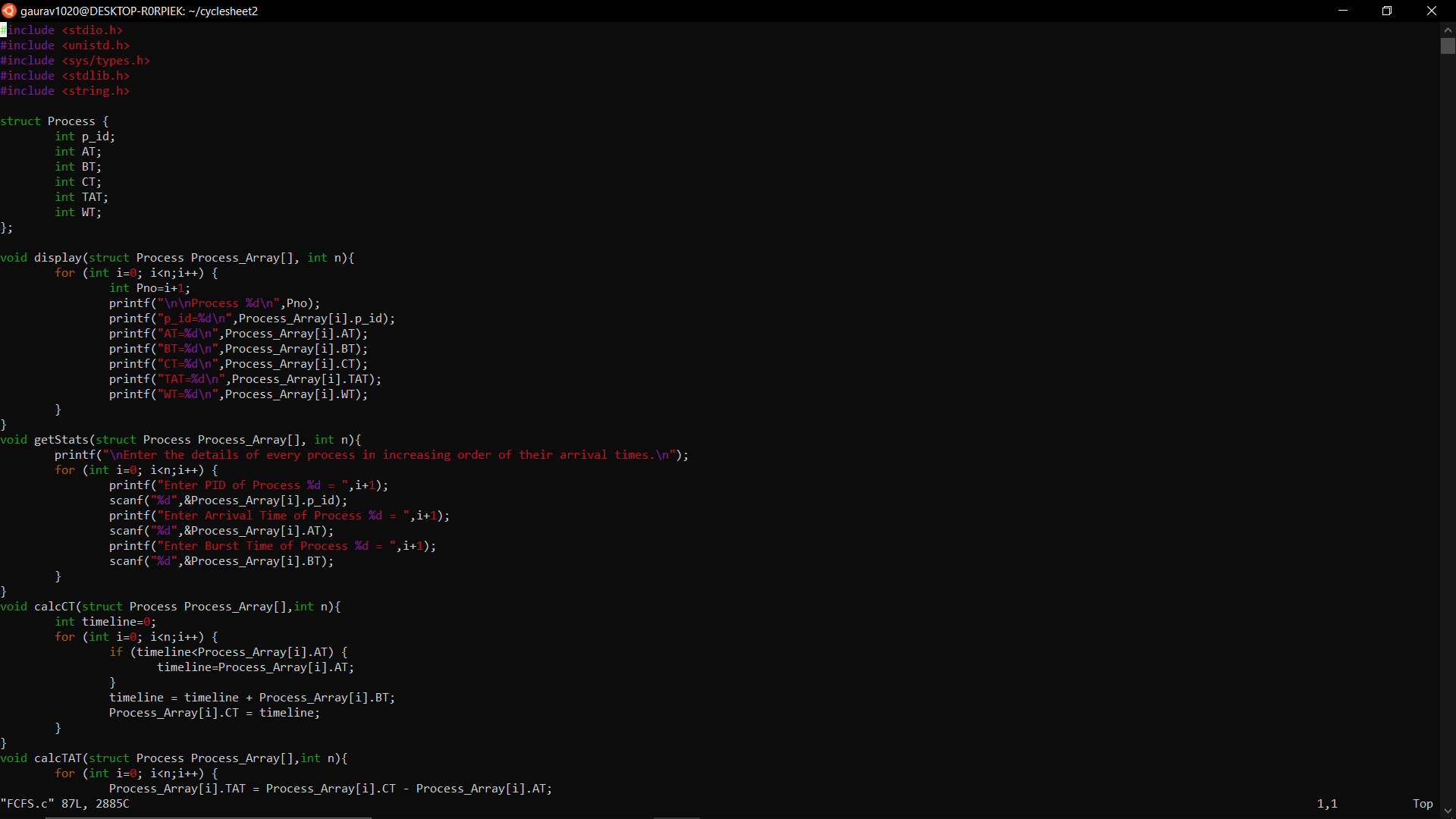
calcAvgTAT(Process\_Array, n);

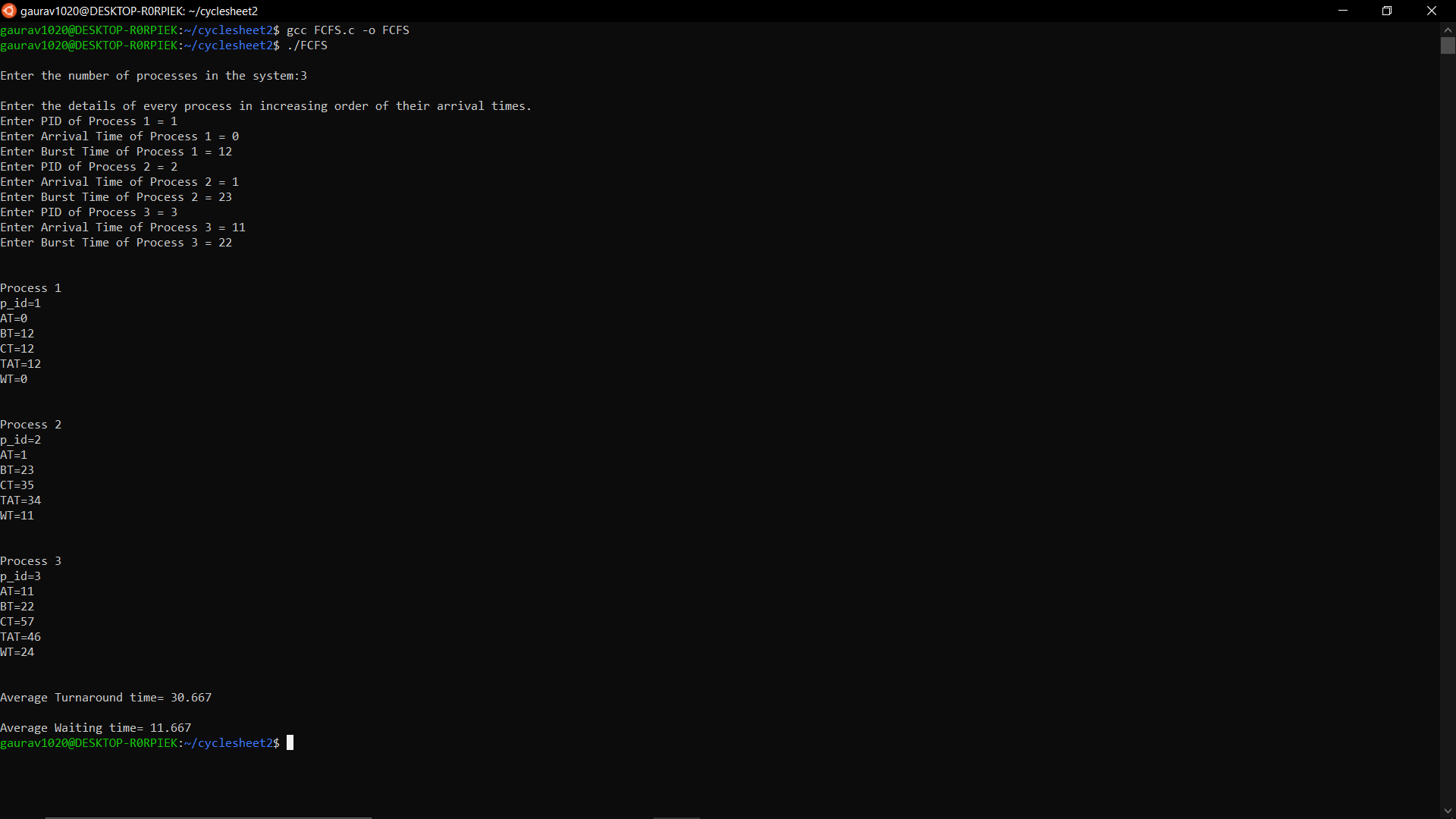
calcAvgWT(Process\_Array, n);

return 0;

}

**OUTPUT FCFS**





**CODE SJF**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <stdlib.h>

#include <string.h>

struct Process {

int p\_id;

int AT;

int BT;

int CT;

int TAT;

int WT;

int flag;

};

void display(struct Process Process\_Array[], int n){

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

int Pno=i+1;

printf("\n\nProcess %d\n",Pno);

printf("p\_id=%d\n",Process\_Array[i].p\_id);

printf("AT=%d\n",Process\_Array[i].AT);

printf("BT=%d\n",Process\_Array[i].BT);

printf("CT=%d\n",Process\_Array[i].CT);

printf("TAT=%d\n",Process\_Array[i].TAT);

printf("WT=%d\n",Process\_Array[i].WT);

}

}

void getStats(struct Process Process\_Array[], int n){

printf("\nEnter the details of every process in increasing order of their arrival times.\n");

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

printf("Enter PID of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].p\_id);

printf("Enter Arrival Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].AT);

printf("Enter Burst Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].BT);

Process\_Array[i].flag=0;

}

}

int MinBT(struct Process Process\_Array[],int n){

int min=0;

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

if (Process\_Array[i].BT<Process\_Array[min].BT && Process\_Array[i].flag==0){

min=i;

}

}

return min;

}

void calcCT(struct Process Process\_Array[], struct Process Ready[], int n){

int timeline=0;

int counter=0;

int min=0;

int j=0;

while (counter!=n){

j=0;

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

if(Process\_Array[i].AT<=timeline&& Process\_Array[i].flag==0){

Ready[j]=Process\_Array[i];

j++;

}

}

if (j==0){

timeline++;

}

else{

min=MinBT(Ready,j);

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

if (Process\_Array[i].p\_id==Ready[min].p\_id){

timeline=timeline+Process\_Array[i].BT;

Process\_Array[i].CT=timeline;

Process\_Array[i].flag=1;

counter++;

}

}

}

}

}

void calcTAT(struct Process Process\_Array[],int n){

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

Process\_Array[i].TAT = Process\_Array[i].CT - Process\_Array[i].AT;

}

}

void calcWT(struct Process Process\_Array[],int n){

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

Process\_Array[i].WT = Process\_Array[i].TAT - Process\_Array[i].BT;

}

}

void calcAvgTAT(struct Process Process\_Array[],int n){

float sumTAT=0;

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

sumTAT = sumTAT + Process\_Array[i].TAT;

}

printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));

}

void calcAvgWT(struct Process Process\_Array[],int n){

float sumWT=0;

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

sumWT = sumWT + Process\_Array[i].WT;

}

printf("\n\nAverage Waiting time= %.3f", (sumWT/n));

}

int main(){

int n;

printf("\nEnter the number of processes in the system");

scanf ("%d",&n);

struct Process Process\_Array[100];

struct Process Ready[100];

getStats(Process\_Array, n);

calcCT(Process\_Array, Ready, n);

calcTAT(Process\_Array, n);

calcWT(Process\_Array, n);

display(Process\_Array, n);

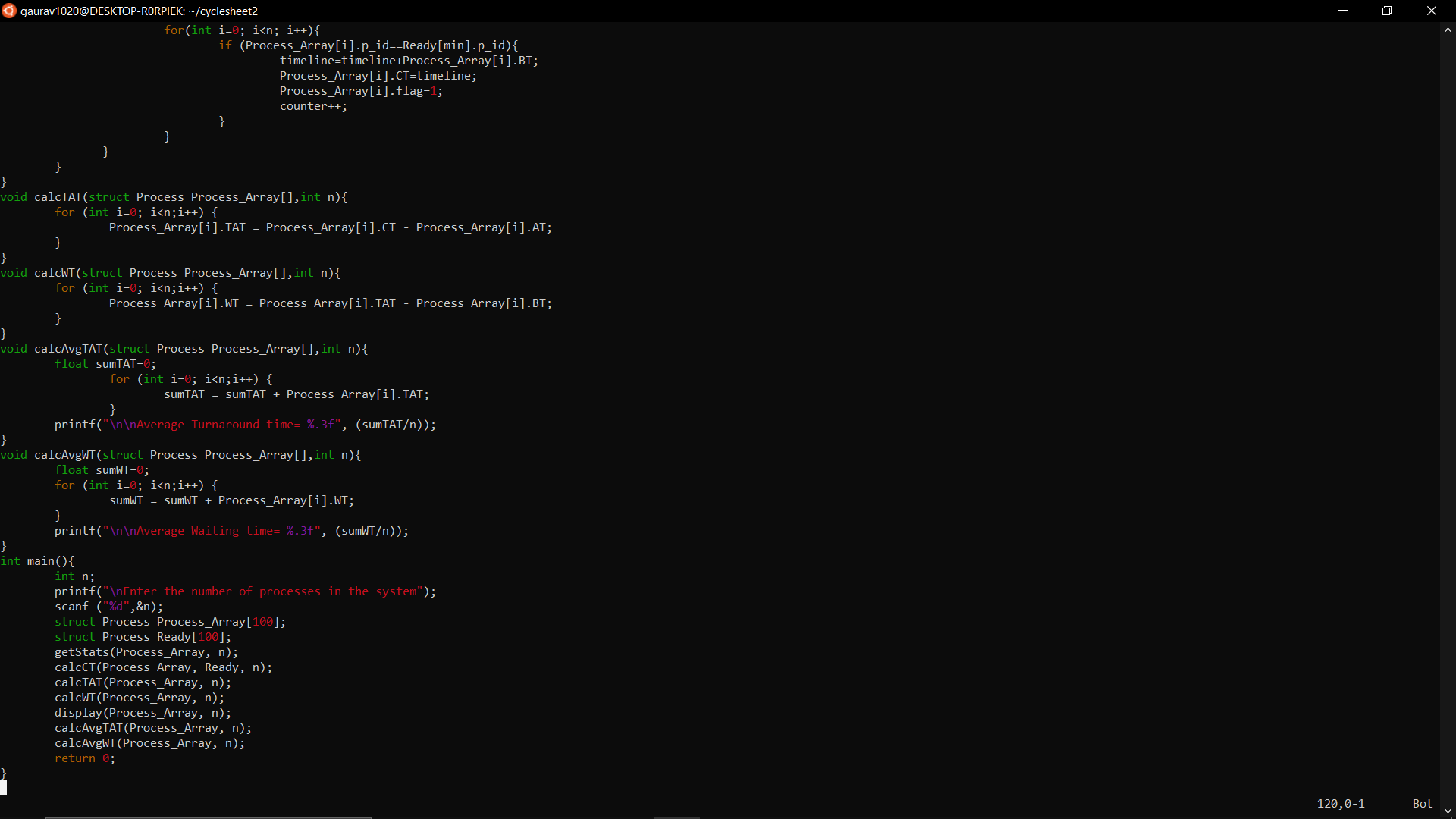
calcAvgTAT(Process\_Array, n);

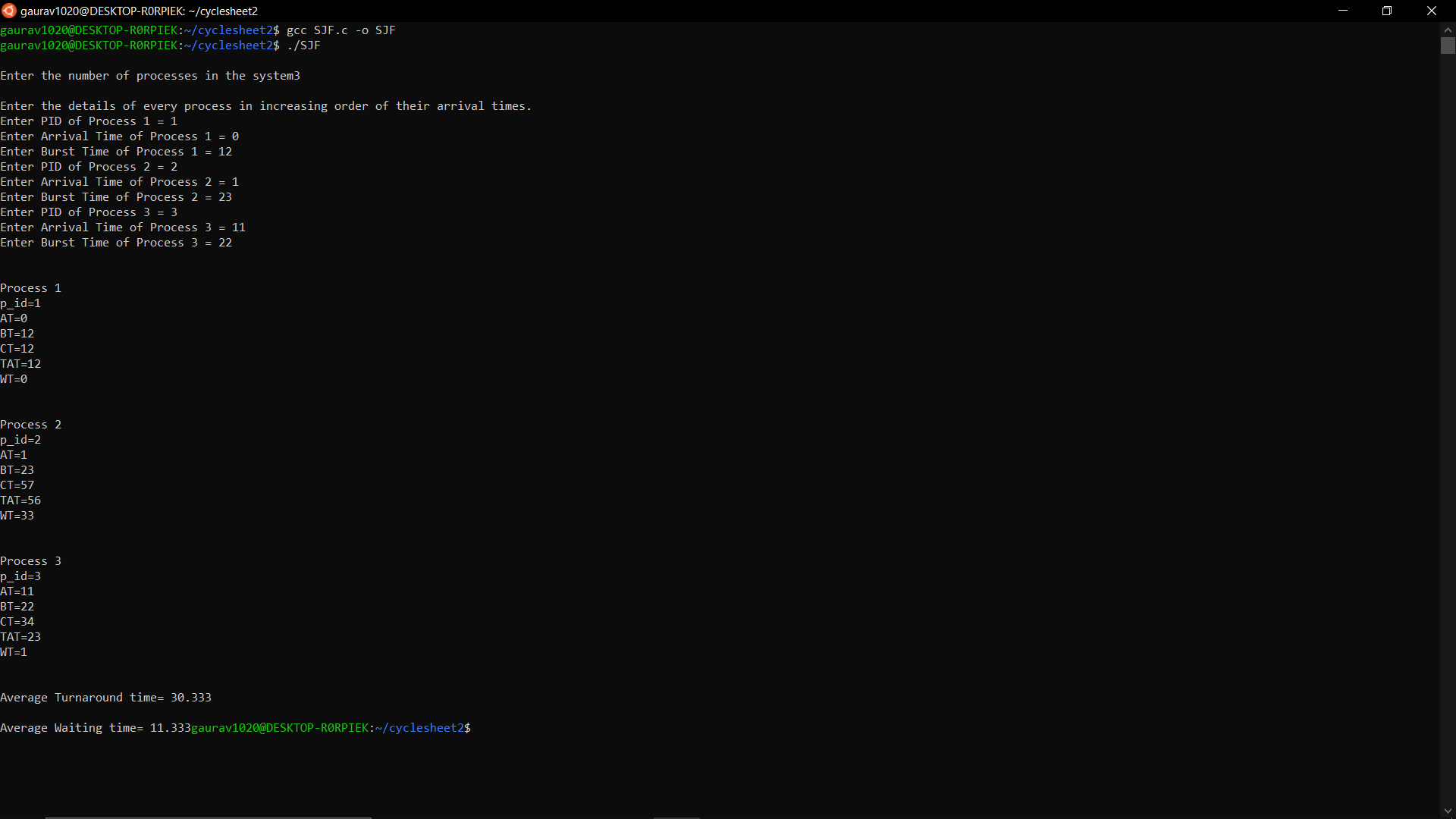
calcAvgWT(Process\_Array, n);

return 0;

}

**OUTPUT SJF**





**CODE PRIORITY**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <stdlib.h>

#include <string.h>

struct Process {

int p\_id;

int AT;

int BT;

int rem\_BT;

int CT;

int TAT;

int WT;

int Priority;

};

void display(struct Process Process\_Array[], int n){

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

int Pno=i+1;

printf("\n\nProcess %d\n",Pno);

printf("p\_id=%d\n",Process\_Array[i].p\_id);

printf("AT=%d\n",Process\_Array[i].AT);

printf("BT=%d\n",Process\_Array[i].BT);

printf("Priority=%d\n",Process\_Array[i].Priority);

printf("CT=%d\n",Process\_Array[i].CT);

printf("TAT=%d\n",Process\_Array[i].TAT);

printf("WT=%d\n",Process\_Array[i].WT);

}

}

void getStats(struct Process Process\_Array[], int n){

printf("\nEnter the details of every process in increasing order of their arrival times.\n");

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

printf("Enter PID of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].p\_id);

printf("Enter Arrival Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].AT);

printf("Enter Burst Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].BT);

Process\_Array[i].rem\_BT=Process\_Array[i].BT;

printf("Enter Priority of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].Priority);

}

}

int readyQueueManagement(struct Process Process\_Array[], struct Process Ready[],int timeline, int n){

int i=0, j=0;

while (Process\_Array[i].AT<=timeline && i<n){

if (Process\_Array[i].rem\_BT !=0){

Ready[j]=Process\_Array[i];

j++;

}

i++;

}

return j;

}

int MaxPriority(struct Process Ready[], int j){

int max=0;

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

if (Ready[i].Priority>Ready[max].Priority){

max=i;

}

}

return max;

}

void calcCT(struct Process Process\_Array[], struct Process Ready[],int n){

int timeline=0;

int counter = 0;

while (counter !=n) {

int j=readyQueueManagement(Process\_Array, Ready, timeline, n);

if (j==0){

timeline++;

}

else{

int max= MaxPriority(Ready, j);

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

if (Ready[max].p\_id==Process\_Array[i].p\_id){

Process\_Array[i].rem\_BT=0;

timeline=timeline+Process\_Array[i].BT;

Process\_Array[i].CT=timeline;

counter++;

}

}

}

}

}

void calcTAT(struct Process Process\_Array[],int n){

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

Process\_Array[i].TAT = Process\_Array[i].CT - Process\_Array[i].AT;

}

}

void calcWT(struct Process Process\_Array[],int n){

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

Process\_Array[i].WT = Process\_Array[i].TAT - Process\_Array[i].BT;

}

}

void calcAvgTAT(struct Process Process\_Array[],int n){

float sumTAT=0;

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

sumTAT = sumTAT + Process\_Array[i].TAT;

}

printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));

}

void calcAvgWT(struct Process Process\_Array[],int n){

float sumWT=0;

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

sumWT = sumWT + Process\_Array[i].WT;

}

printf("\n\nAverage Waiting time= %.3f", (sumWT/n));

}

int main(){

int n;

printf("\nEnter the number of Processes in the system:");

scanf ("%d",&n);

struct Process Process\_Array[100];

struct Process Ready[100];

getStats(Process\_Array, n);

calcCT(Process\_Array,Ready, n);

calcTAT(Process\_Array, n);

calcWT(Process\_Array, n);

display(Process\_Array, n);

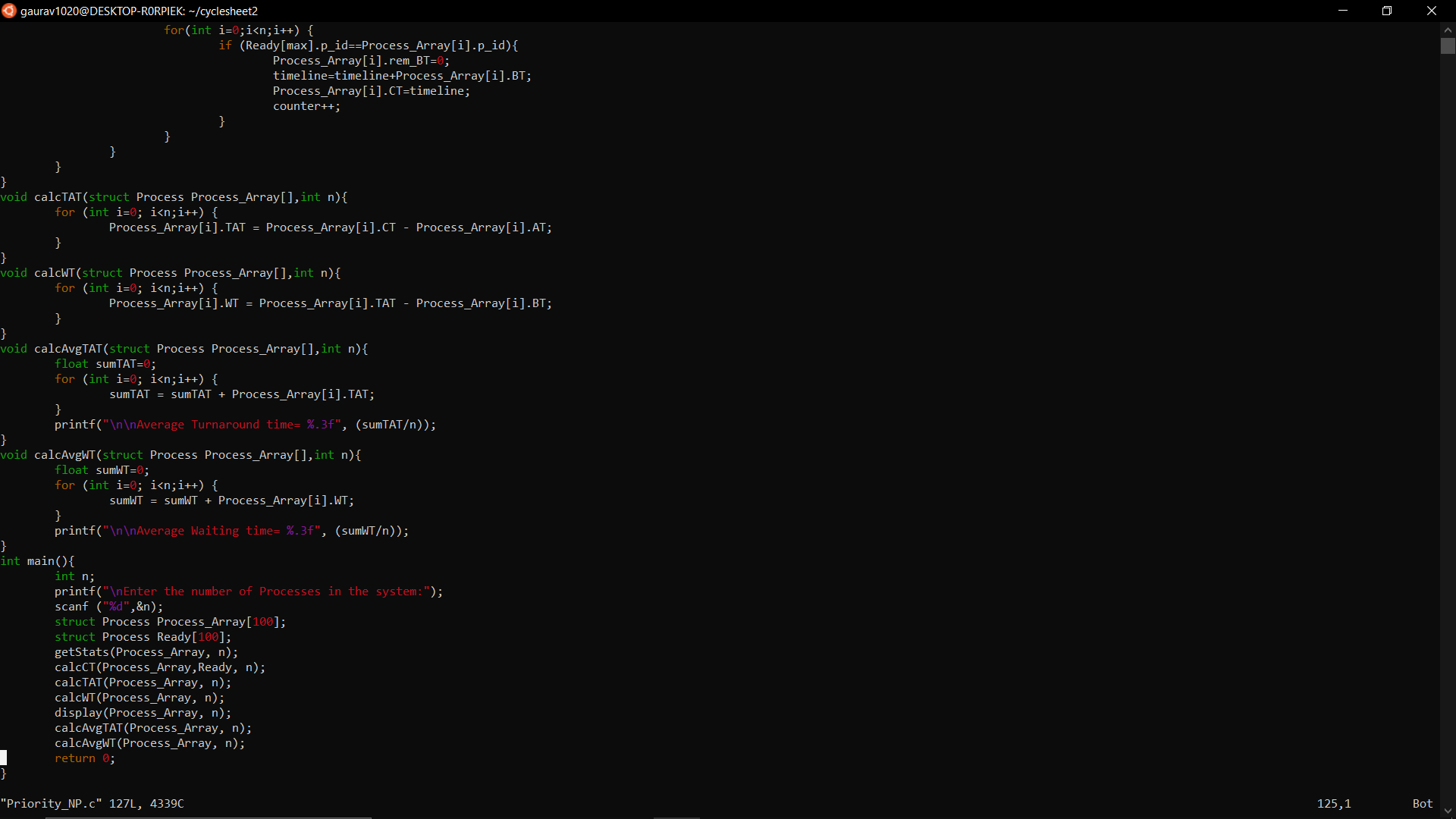
calcAvgTAT(Process\_Array, n);

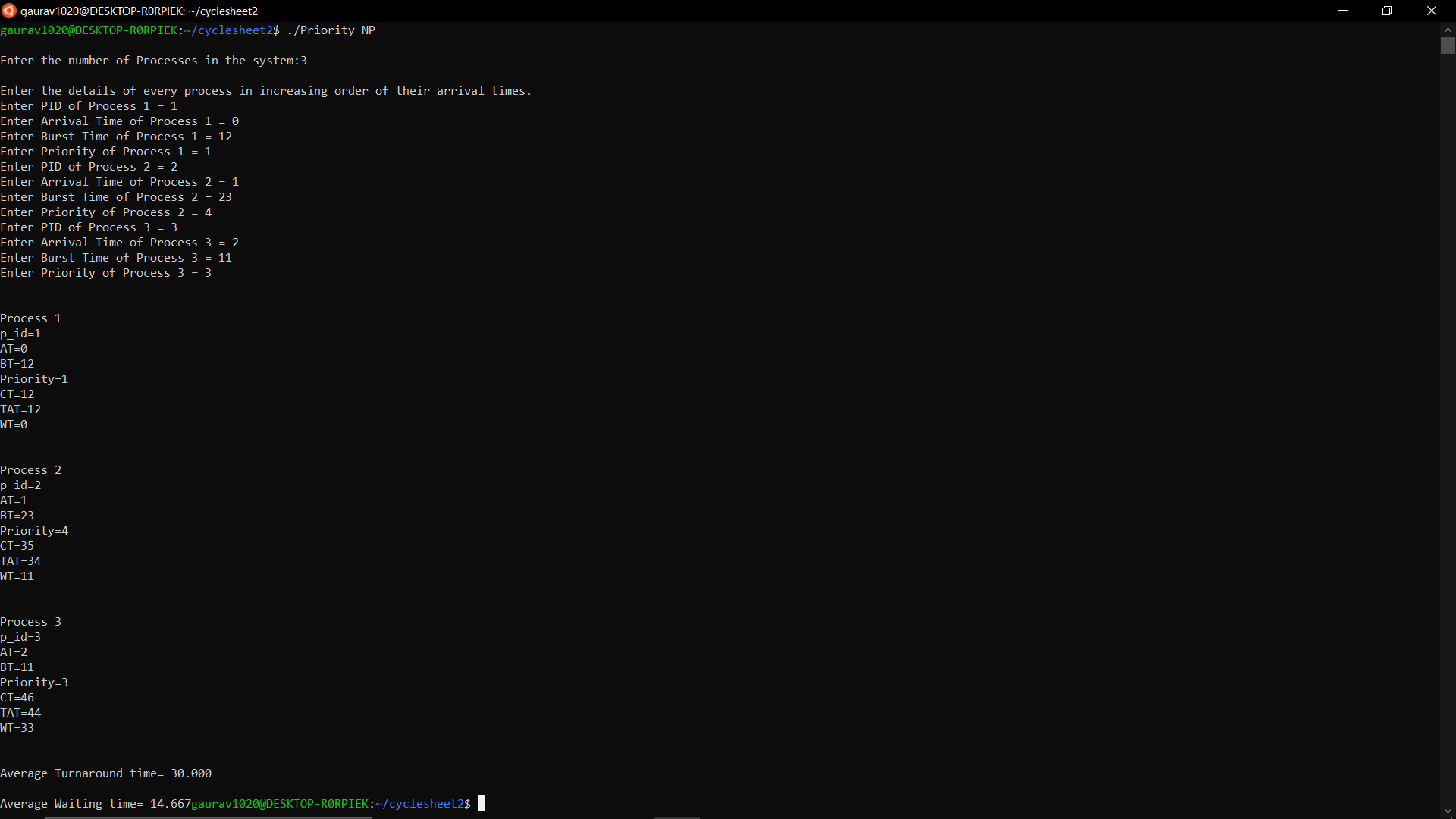
calcAvgWT(Process\_Array, n);

return 0;

}

**OUTPUT PRIORITY**





1. Implement the various process scheduling algorithms such as Priority, Round Robin (preemptive). (Medium)

**CODE PRIORITY**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <stdlib.h>

#include <string.h>

struct Process {

int p\_id;

int AT;

int BT;

int rem\_BT;

int CT;

int TAT;

int WT;

int Priority;

};

void display(struct Process Process\_Array[], int n){

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

int Pno=i+1;

printf("\n\nProcess %d\n",Pno);

printf("p\_id=%d\n",Process\_Array[i].p\_id);

printf("AT=%d\n",Process\_Array[i].AT);

printf("BT=%d\n",Process\_Array[i].BT);

printf("Priority=%d\n",Process\_Array[i].Priority);

printf("CT=%d\n",Process\_Array[i].CT);

printf("TAT=%d\n",Process\_Array[i].TAT);

printf("WT=%d\n",Process\_Array[i].WT);

}

}

void getStats(struct Process Process\_Array[], int n){

printf("\nEnter the details of every process in increasing order of their arrival times.\n");

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

printf("Enter PID of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].p\_id);

printf("Enter Arrival Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].AT);

printf("Enter Burst Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].BT);

Process\_Array[i].rem\_BT=Process\_Array[i].BT;

printf("Enter Priority of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].Priority);

}

}

int readyQueueManagement(struct Process Process\_Array[], struct Process Ready[],int timeline, int n){

int i=0, j=0;

while (Process\_Array[i].AT<=timeline && i<n){

if (Process\_Array[i].rem\_BT !=0){

Ready[j]=Process\_Array[i];

j++;

}

i++;

}

return j;

}

int MaxPriority(struct Process Ready[], int j){

int max=0;

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

if (Ready[i].Priority>Ready[max].Priority){

max=i;

}

}

return max;

}

void calcCT(struct Process Process\_Array[], struct Process Ready[],int n){

int timeline=0;

int counter = 0;

while (counter !=n) {

int j=readyQueueManagement(Process\_Array, Ready, timeline, n);

if (j==0){

timeline++;

}

else{

int max= MaxPriority(Ready, j);

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

if (Ready[max].p\_id==Process\_Array[i].p\_id){

Process\_Array[i].rem\_BT=Process\_Array[i].rem\_BT-1;

timeline=timeline+1;

if (Process\_Array[i].rem\_BT==0){

Process\_Array[i].CT=timeline;

counter++;

}

}

}

}

}

}

void calcTAT(struct Process Process\_Array[],int n){

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

Process\_Array[i].TAT = Process\_Array[i].CT - Process\_Array[i].AT;

}

}

void calcWT(struct Process Process\_Array[],int n){

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

Process\_Array[i].WT = Process\_Array[i].TAT - Process\_Array[i].BT;

}

}

void calcAvgTAT(struct Process Process\_Array[],int n){

float sumTAT=0;

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

sumTAT = sumTAT + Process\_Array[i].TAT;

}

printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));

}

void calcAvgWT(struct Process Process\_Array[],int n){

float sumWT=0;

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

sumWT = sumWT + Process\_Array[i].WT;

}

printf("\n\nAverage Waiting time= %.3f", (sumWT/n));

}

int main(){

int n;

printf("\nEnter the number of Processes in the system:");

scanf ("%d",&n);

struct Process Process\_Array[100];

struct Process Ready[100];

getStats(Process\_Array, n);

calcCT(Process\_Array,Ready, n);

calcTAT(Process\_Array, n);

calcWT(Process\_Array, n);

display(Process\_Array, n);

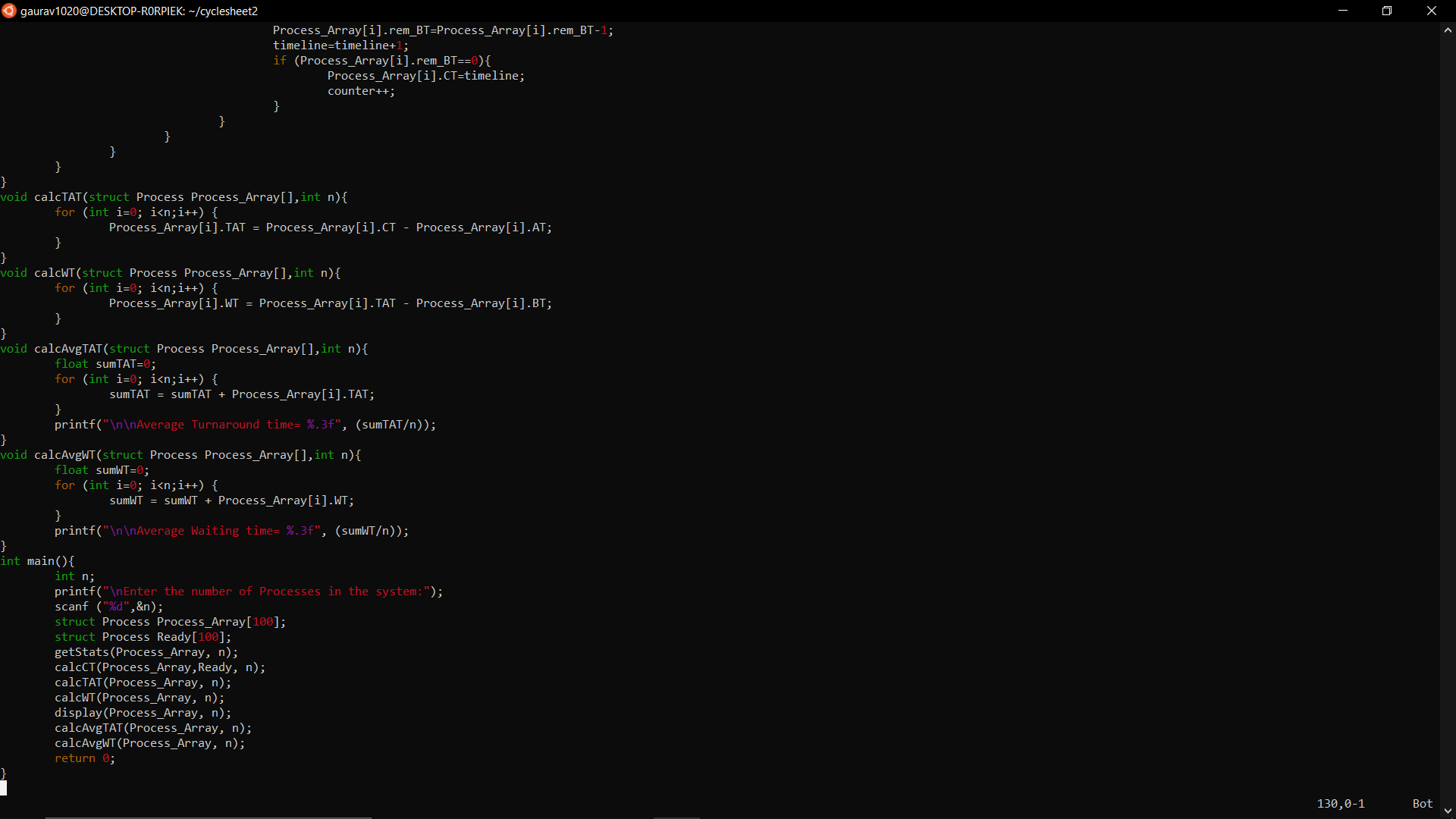
calcAvgTAT(Process\_Array, n);

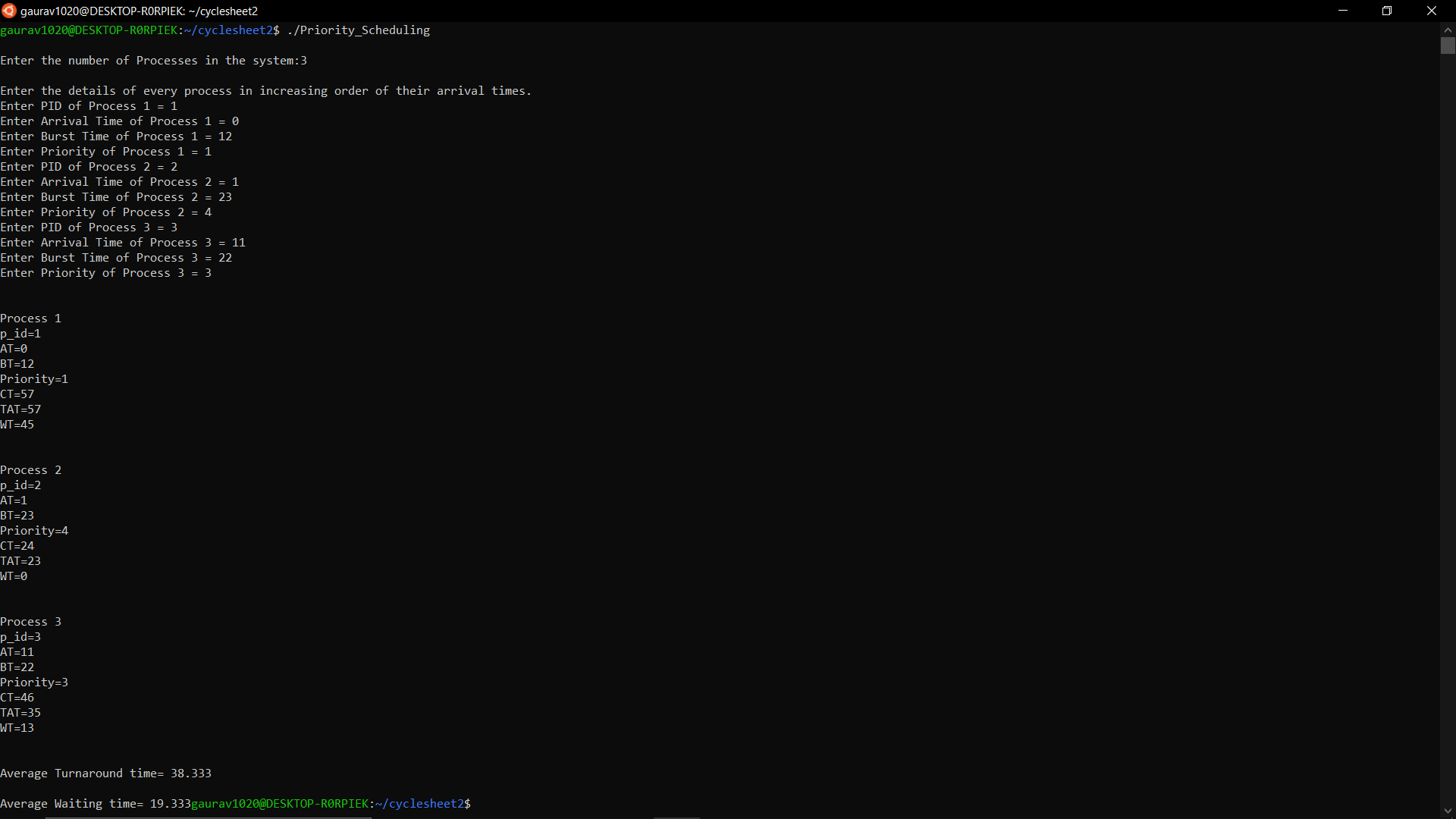
calcAvgWT(Process\_Array, n);

return 0;

}

**OUTPUT PRIORITY**





**CODE ROUND ROBIN**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <stdlib.h>

#include <string.h>

struct Process {

int p\_id;

int AT;

int BT;

int rem\_BT;

int CT;

int TAT;

int WT;

}buffer;

void SeT(struct Process Ready[], int n){

buffer.p\_id=-1;

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

Ready[i].p\_id=-1;

Ready[i].AT=-1;

Ready[i].BT=-1;

Ready[i].rem\_BT=-1;

Ready[i].CT=-1;

Ready[i].TAT=-1;

Ready[i].WT=-1;

}

}

void display(struct Process Process\_Array[], int n){

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

int Pno=i+1;

printf("\n\nProcess %d\n",Process\_Array[i].p\_id);

printf("p\_id=%d\n",Process\_Array[i].p\_id);

printf("AT=%d\n",Process\_Array[i].AT);

printf("BT=%d\n",Process\_Array[i].BT);

printf("CT=%d\n",Process\_Array[i].CT);

printf("TAT=%d\n",Process\_Array[i].TAT);

printf("WT=%d\n",Process\_Array[i].WT);

printf("Rem\_BT=%d\n",Process\_Array[i].rem\_BT);

}

}

void getStats(struct Process Process\_Array[], int n){

printf("\nEnter the details of every process in increasing order of their arrival times.\n");

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

printf("Enter PID of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].p\_id);

printf("Enter Arrival Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].AT);

printf("Enter Burst Time of Process %d = ",i+1);

scanf("%d",&Process\_Array[i].BT);

Process\_Array[i].rem\_BT=Process\_Array[i].BT;

}

}

void queue(struct Process Ready[], struct Process Process\_Array){

int i=0;

while(Ready[i].p\_id!=-1){

i++;

}

Ready[i]=Process\_Array;

}

struct Process dequeue(struct Process Ready[]){

struct Process temp=Ready[0];

int i=0;

while(Ready[i].p\_id!=-1){

i++;

}

int j=0;

for (j; j<i+1;j++){

Ready[j]=Ready[j+1];

}

return temp;

}

int Ready\_No(struct Process Ready[]){

int i=0;

while(Ready[i].p\_id!=-1){

i++;

}

return i;

}

void calcCT(struct Process Process\_Array[], struct Process Ready[], int Time\_Quantum, int n){

int timeline=0;

int counter=0;

int Process\_counter=0;

while(counter !=n){

for(int i=Process\_counter;i<n;i++){

if(Process\_Array[i].AT<=timeline){

queue(Ready, Process\_Array[i]);

Process\_counter++;

}

}

int ReadyQueueCheck=Ready\_No(Ready);

if(ReadyQueueCheck==0){

timeline++;

}

else{

struct Process temp=dequeue(Ready);

if(temp.rem\_BT<=Time\_Quantum){

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

if(Process\_Array[i].p\_id==temp.p\_id){

timeline=timeline+Process\_Array[i].rem\_BT;

Process\_Array[i].rem\_BT=0;

Process\_Array[i].CT=timeline;

counter++;

}

}

}

else{

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

if(Process\_Array[i].p\_id==temp.p\_id){

timeline=timeline+Time\_Quantum;

for(int i=Process\_counter;i<n;i++){

if(Process\_Array[i].AT<=timeline){

queue(Ready, Process\_Array[i]);

Process\_counter++;

}

}

Process\_Array[i].rem\_BT=Process\_Array[i].rem\_BT-Time\_Quantum;

queue(Ready, Process\_Array[i]);

}

}

}

}

}

}

void calcTAT(struct Process Process\_Array[],int n){

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

Process\_Array[i].TAT = Process\_Array[i].CT - Process\_Array[i].AT;

}

}

void calcWT(struct Process Process\_Array[],int n){

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

Process\_Array[i].WT = Process\_Array[i].TAT - Process\_Array[i].BT;

}

}

void calcAvgTAT(struct Process Process\_Array[],int n){

float sumTAT=0;

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

sumTAT = sumTAT + Process\_Array[i].TAT;

}

printf("\n\nAverage Turnaround time= %.3f", (sumTAT/n));

}

void calcAvgWT(struct Process Process\_Array[],int n){

float sumWT=0;

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

sumWT = sumWT + Process\_Array[i].WT;

}

printf("\n\nAverage Waiting time= %.3f", (sumWT/n));

}

int main(){

int n, Time\_Quantum;

printf("\nEnter Number of Processes in the system: ");

scanf ("%d",&n);

printf("Enter Time Quantum: ");

scanf("%d",&Time\_Quantum);

struct Process Process\_Array[100];

struct Process Ready[100];

SeT(Ready, 100);

struct Process temp=dequeue(Ready);

getStats(Process\_Array, n);

calcCT(Process\_Array,Ready, Time\_Quantum, n);

calcTAT(Process\_Array, n);

calcWT(Process\_Array, n);

display(Process\_Array, n);

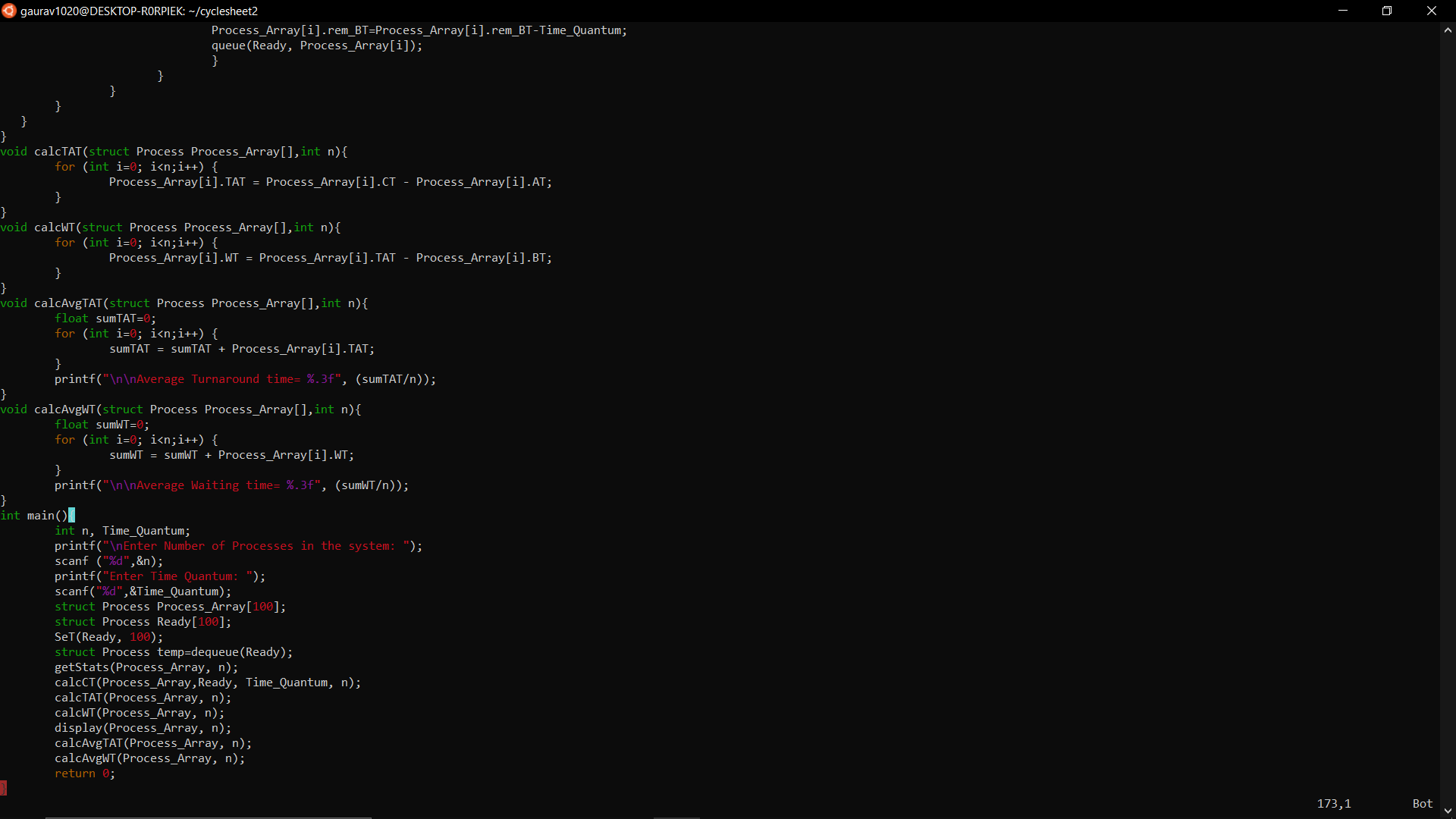
calcAvgTAT(Process\_Array, n);

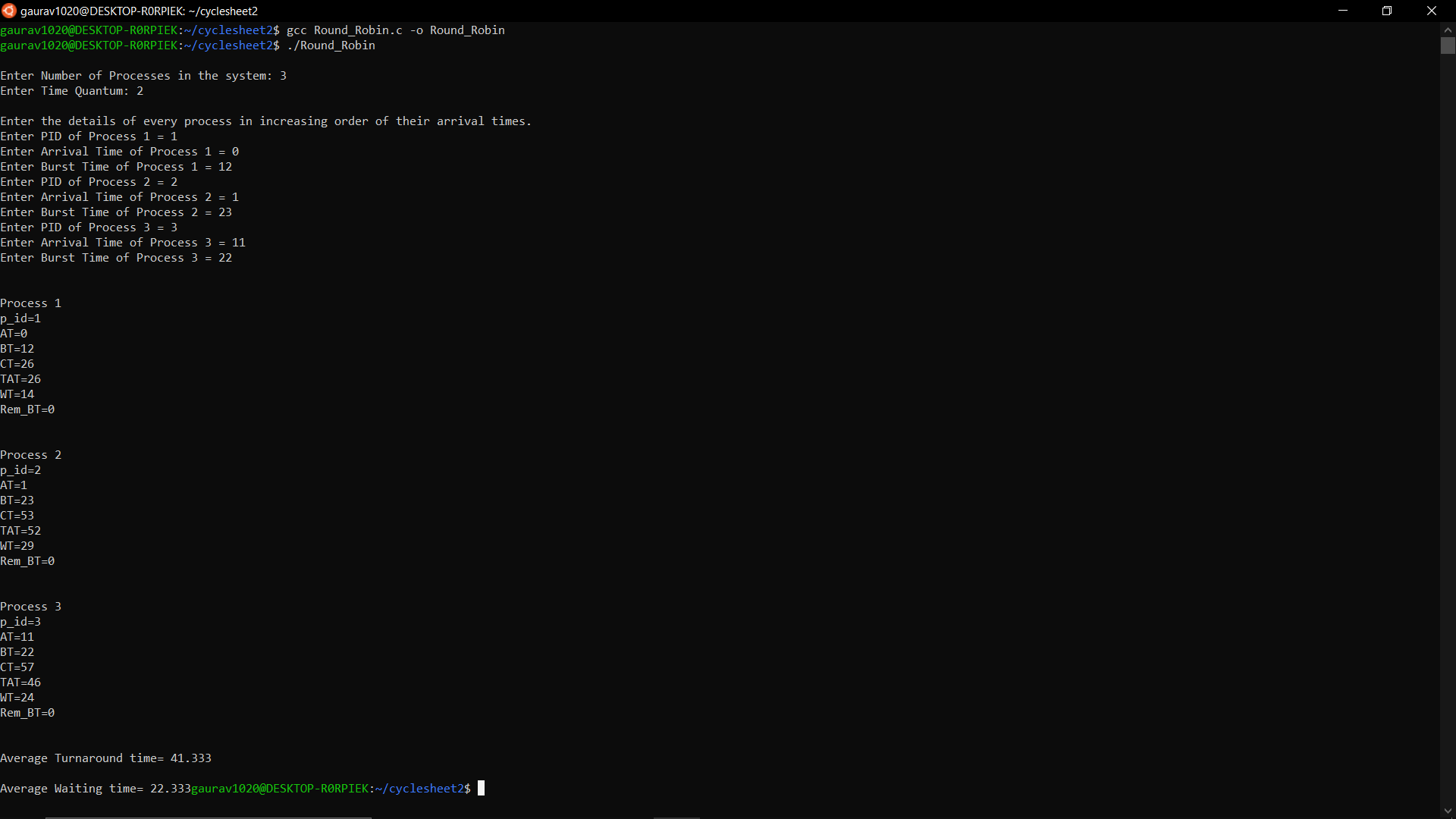
calcAvgWT(Process\_Array, n);

return 0;

}

**OUTPUT ROUND ROBIN**





1. Consider a corporate hospital where we have n number of patients waiting for consultation. The amount of time required to serve a patient may vary, say 10 to 30 minutes. If a patient arrives with an emergency, he /she should be attended immediately before other patients, which may increase the waiting time of other patients. If you are given this problem with the following algorithms how would you devise an effective scheduling so that it optimizes the overall performance such as minimizing the waiting time of all patients. [Single queue or multi-level queue can be used]. Consider the availability of single and multiple doctors • Assign top priority for patients with emergency case, women, children, elders, and youngsters. • Patients coming for review may take less time than others. This can be taken into account while using SJF. 1. FCFS 2. SJF (High)

**CODE FCFS**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <stdlib.h>

#include <string.h>

struct Process {

int p\_id;

int AT;

int BT;

int rem\_BT;

int CT;

int TAT;

int WT;

int Priority;

};

void display(struct Process Process\_Array[], int n) {

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

int Pno = i + 1;

printf("\n\nProcess %d\n", Pno);

printf("p\_id=%d\n", Process\_Array[i].p\_id);

printf("AT=%d\n", Process\_Array[i].AT);

printf("BT=%d\n", Process\_Array[i].BT);

printf("Priority=%d\n", Process\_Array[i].Priority);

printf("CT=%d\n", Process\_Array[i].CT);

printf("TAT=%d\n", Process\_Array[i].TAT);

printf("WT=%d\n", Process\_Array[i].WT);

}

}

void getStats(struct Process Process\_Array[], int n) {

printf("\nEnter the details of every process in increasing order of their arrival times.\n");

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

Process\_Array[i].p\_id= i+1;

printf("Enter Arrival Time of Patient %d = ", i + 1);

scanf("%d", & Process\_Array[i].AT);

printf("Enter Diagnosis Time of Patient %d = ", i + 1);

scanf("%d", & Process\_Array[i].BT);

Process\_Array[i].rem\_BT = Process\_Array[i].BT;

printf("Enter Priority of Patient (0 for normal and 1 for emergency) %d = ", i + 1);

scanf("%d", & Process\_Array[i].Priority);

}

}

int ATArray(struct Process Process\_Array[], int ATarray[], int n){

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

ATarray[i]= Process\_Array[i].AT;

}

}

int Ready\_to\_run(struct Process Process\_Array[], int n, int timeline){

int r=0;

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

if (Process\_Array[i].AT<=timeline && Process\_Array[i].rem\_BT!=0){

r++;

}

}

return r;

}

void calcCT(struct Process Process\_Array[], int n) {

int timeline = 0;

int counter = 0;

int ATarray[100];

ATArray(Process\_Array, ATarray, n);

while (counter != n) {

int runner=-1;

int j= Ready\_to\_run(Process\_Array, n, timeline);

if (j == 0) {

timeline++;

} else {

int flag=0;

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

if (Process\_Array[i].AT<=timeline && Process\_Array[i].rem\_BT!=0 && Process\_Array[i].Priority==1){

flag=1;

}

}

if (flag==1){

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

if (Process\_Array[i].AT<=timeline && Process\_Array[i].rem\_BT!=0 && Process\_Array[i].Priority==1){

runner=i;

break;

}

}

}

else {

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

if (Process\_Array[i].AT<=timeline && Process\_Array[i].rem\_BT!=0){

runner=i;

break;

}

}

}

Process\_Array[runner].rem\_BT=Process\_Array[runner].rem\_BT-1;

timeline++;

if (Process\_Array[runner].rem\_BT==0){

Process\_Array[runner].CT=timeline;

counter++;

}

}

}

}

void calcTAT(struct Process Process\_Array[], int n) {

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

Process\_Array[i].TAT = Process\_Array[i].CT - Process\_Array[i].AT;

}

}

void calcWT(struct Process Process\_Array[], int n) {

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

Process\_Array[i].WT = Process\_Array[i].TAT - Process\_Array[i].BT;

}

}

void calcAvgTAT(struct Process Process\_Array[], int n) {

float sumTAT = 0;

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

sumTAT = sumTAT + Process\_Array[i].TAT;

}

printf("\n\nAverage Turnaround time= %.3f", (sumTAT / n));

}

void calcAvgWT(struct Process Process\_Array[], int n) {

float sumWT = 0;

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

sumWT = sumWT + Process\_Array[i].WT;

}

printf("\n\nAverage Waiting time= %.3f", (sumWT / n));

}

int main() {

int n;

printf("\nEnter the number of Processes in the system:");

scanf("%d", & n);

struct Process Process\_Array[100];

getStats(Process\_Array, n);

calcCT(Process\_Array, n);

calcTAT(Process\_Array, n);

calcWT(Process\_Array, n);

display(Process\_Array, n);

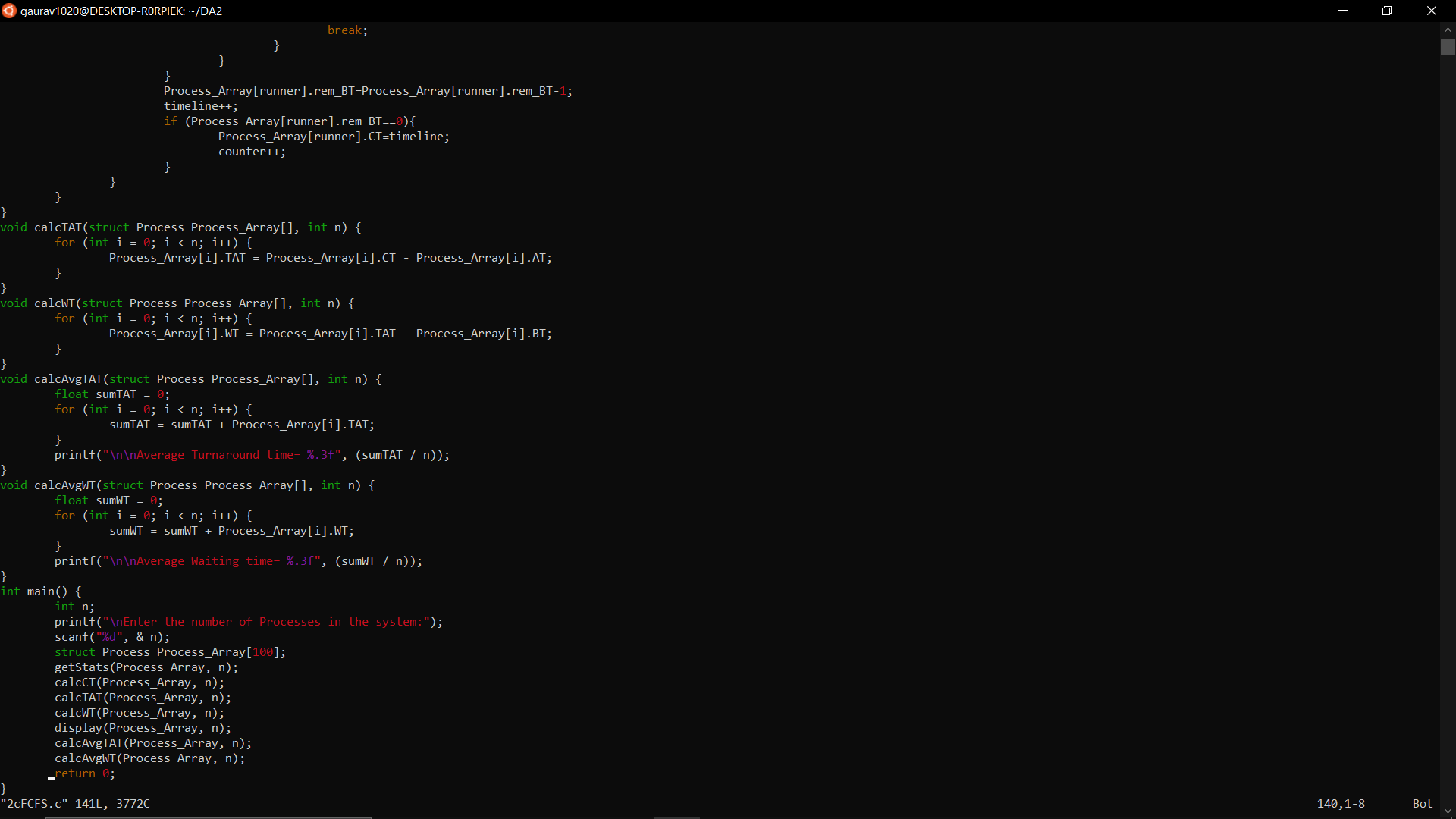
calcAvgTAT(Process\_Array, n);

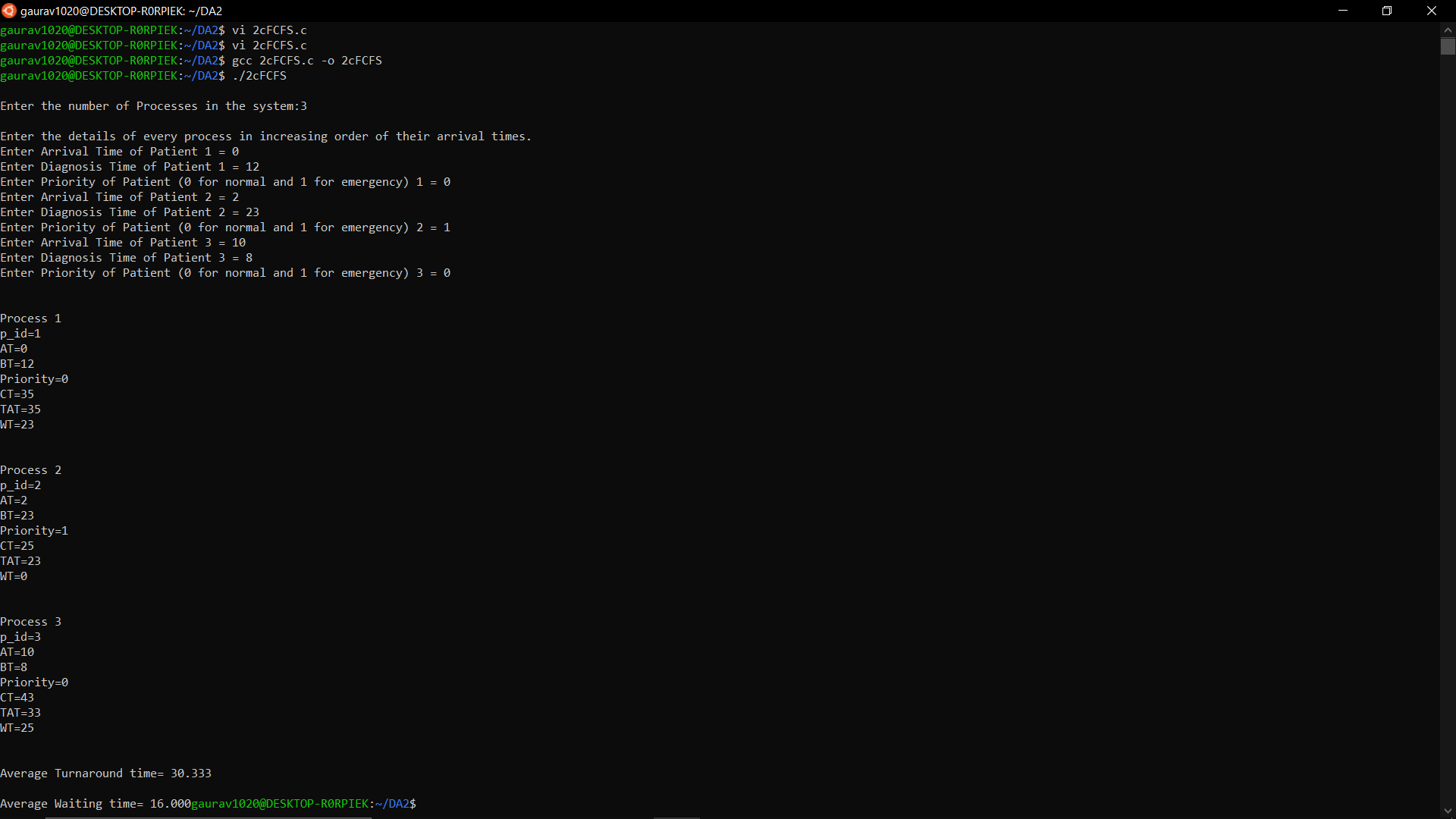
calcAvgWT(Process\_Array, n);

return 0;

}

**OUTPUT FCFS**





**CODE SJF**

**{Context: If the patients are for normal diagnosis then they are attended based on their diagnosis time. During that time if a patient with emergency condition arrives, treatment of ongoing patient with normal diagnosis is stopped immediately and the patient in emergency condition is diagnosed completely. If another patient with emergency condition arrives when treatment of an ongoing patient with emergency condition is going on, the treatment of ongoing patient is not halted and the next emergency case is dealt with after the current emergency is resolved.}**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <stdlib.h>

#include <string.h>

struct Process {

int p\_id;

int AT;

int BT;

int CT;

int TAT;

int WT;

int flag;

int rem\_BT;

int Priority;

};

void display(struct Process Process\_Array[], int n)

{

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

{

int Pno = i + 1;

printf("\n\nProcess %d\n", Pno);

printf("p\_id=%d\n", Process\_Array[i].p\_id);

printf("AT=%d\n", Process\_Array[i].AT);

printf("BT=%d\n", Process\_Array[i].BT);

printf("CT=%d\n", Process\_Array[i].CT);

printf("TAT=%d\n", Process\_Array[i].TAT);

printf("WT=%d\n", Process\_Array[i].WT);

}

}

void getStats(struct Process Process\_Array[], int n)

{

printf("\nEnter the details of every process in increasing order of their arrival times.\n");

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

{

Process\_Array[i].p\_id=i+1;

printf("Enter Arrival Time of Patient %d = ", i + 1);

scanf("%d", &Process\_Array[i].AT);

printf("Enter Diagnosis Time of Patient %d = ", i + 1);

scanf("%d", &Process\_Array[i].BT);

printf("Enter Priority of Patient %d (0 for normal, 1 for emergency)= ", i + 1);

scanf("%d", &Process\_Array[i].Priority);

Process\_Array[i].flag = 0;

Process\_Array[i].rem\_BT = Process\_Array[i].BT;

}

}

int MinBT(struct Process Process\_Array[], int n)

{

int min = 0;

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

{

if (Process\_Array[i].rem\_BT < Process\_Array[min].rem\_BT && Process\_Array[i].flag == 0)

{

min = i;

}

}

return min;

}

void calcCT(struct Process Process\_Array[], struct Process Ready[], int n)

{

int flag=0;

int timeline = 0;

int counter = 0;

int min = 0;

int j = 0;

while (counter != n)

{

int runner=-1;

int flag=0;

j = 0;

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

{

if (Process\_Array[i].AT <= timeline && Process\_Array[i].flag == 0)

{

Ready[j] = Process\_Array[i];

j++;

}

}

if (j == 0)

{

timeline++;

}

else

{

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

if (Ready[i].Priority==1){

flag=1;

runner=i;

break;

}

}

if (flag==1) {

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

if (Ready[runner].p\_id==Process\_Array[i].p\_id){

runner=i;

break;

}

}

Process\_Array[runner].flag=1;

Process\_Array[runner].rem\_BT=0;

timeline=timeline+Process\_Array[runner].BT;

counter++;

Process\_Array[runner].CT=timeline;

}

if (flag==0){

min = MinBT(Ready, j);

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

{

if (Process\_Array[i].p\_id == Ready[min].p\_id)

{

runner=i;

break;

}

}

Process\_Array[runner].rem\_BT=Process\_Array[runner].rem\_BT-1;

timeline++;

if (Process\_Array[runner].rem\_BT==0){

Process\_Array[runner].CT=timeline;

Process\_Array[runner].flag=1;

counter++;

}

}

}

}

}

void calcTAT(struct Process Process\_Array[], int n)

{

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

{

Process\_Array[i].TAT = Process\_Array[i].CT - Process\_Array[i].AT;

}

}

void calcWT(struct Process Process\_Array[], int n)

{

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

{

Process\_Array[i].WT = Process\_Array[i].TAT - Process\_Array[i].BT;

}

}

void calcAvgTAT(struct Process Process\_Array[], int n)

{

float sumTAT = 0;

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

{

sumTAT = sumTAT + Process\_Array[i].TAT;

}

printf("\n\nAverage Turnaround time= %.3f", (sumTAT / n));

}

void calcAvgWT(struct Process Process\_Array[], int n)

{

float sumWT = 0;

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

{

sumWT = sumWT + Process\_Array[i].WT;

}

printf("\n\nAverage Waiting time= %.3f", (sumWT / n));

}

int main()

{

int n;

printf("\nEnter the number of processes in the system:");

scanf("%d", &n);

struct Process Process\_Array[100];

struct Process Ready[100];

getStats(Process\_Array, n);

calcCT(Process\_Array, Ready, n);

calcTAT(Process\_Array, n);

calcWT(Process\_Array, n);

display(Process\_Array, n);

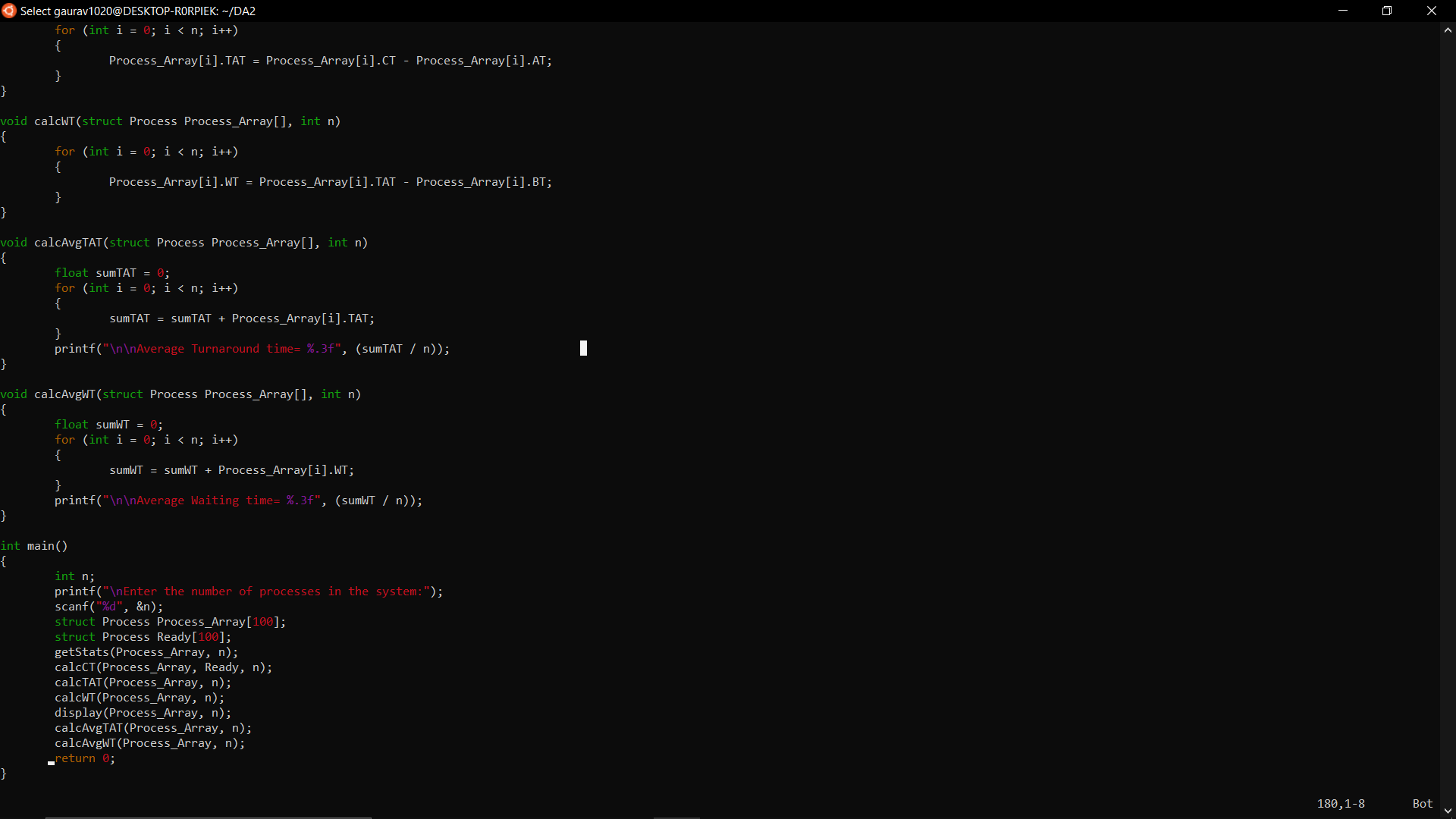
calcAvgTAT(Process\_Array, n);

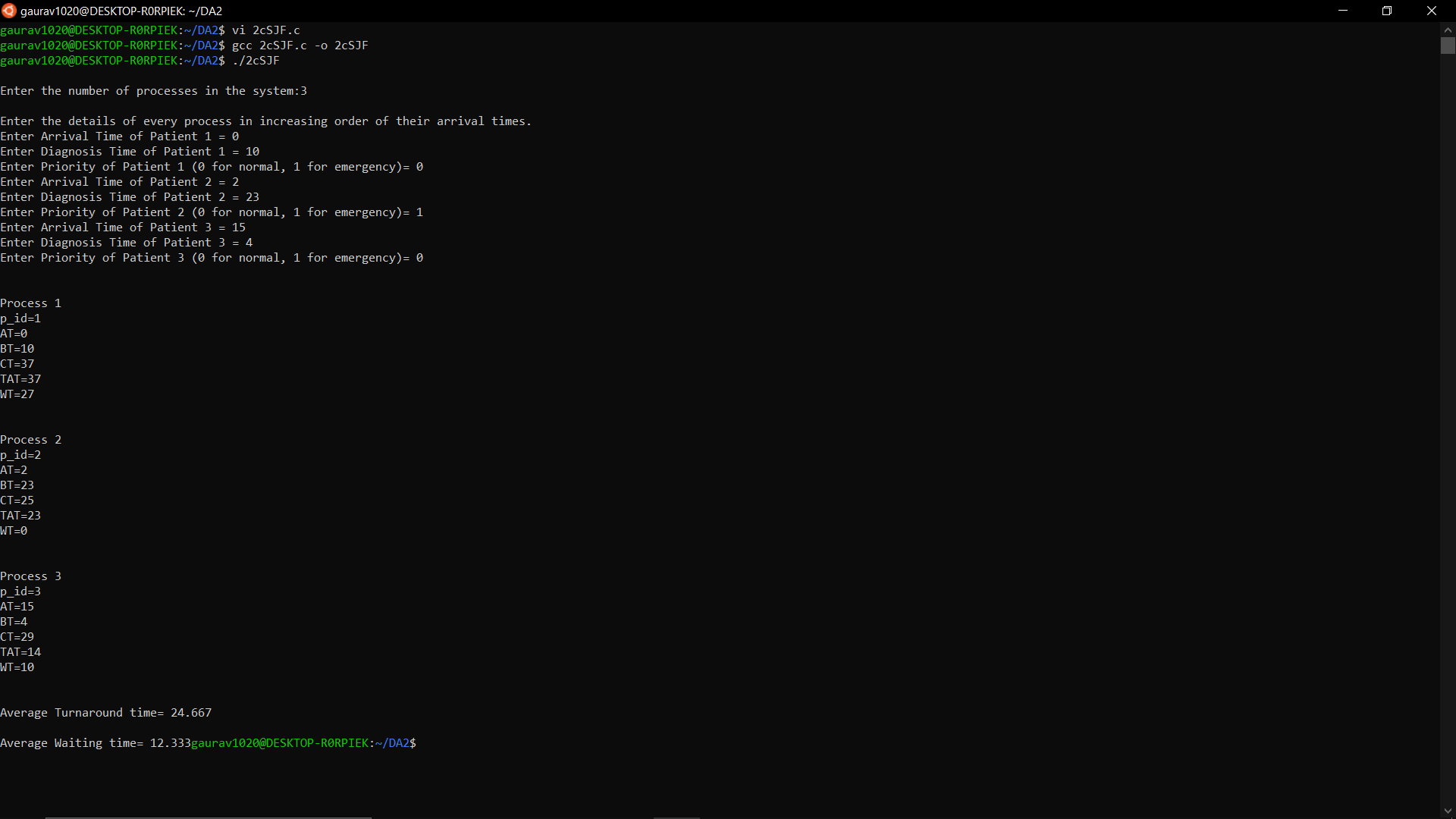
calcAvgWT(Process\_Array, n);

return 0;

}

**OUTPUT SJF**





1. Simulate with a program to provide deadlock avoidance of Banker’s Algorithm including Safe state and additional resource request (High).

**CODE**

int main()

{

int n, m, i, j, k;

n = 5;

m = 3;

int alloc[5][3] = { { 0, 1, 0 },

{ 2, 0, 0 },

{ 3, 0, 2 },

{ 2, 1, 1 },

{ 0, 0, 2 } };

int max[5][3] = { { 7, 5, 3 },

{ 3, 2, 2 },

{ 9, 0, 2 },

{ 2, 2, 2 },

{ 4, 3, 3 }};

int avail[3] = { 3, 3, 2 };

int f[n], ans[n], ind = 0;

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

f[k] = 0;

}

int need[n][m];

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

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

need[i][j] = max[i][j] - alloc[i][j];

}

int y = 0;

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

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

if (f[i] == 0) {

int flag = 0;

for (j = 0; j < m; j++) {

if (need[i][j] > avail[j]){

flag = 1;

break;

}

}

if (flag == 0) {

ans[ind++] = i;

for (y = 0; y < m; y++)

avail[y] += alloc[i][y];

f[i] = 1;

}

}

}

}

printf("Following is the SAFE Sequence\n");

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

printf(" P%d ->", ans[i]);

printf(" P%d", ans[n - 1]);

return (0);

}

**OUTPUT**

