Operating Systems

CSE-316

Report -Assignment Simulation Based



Name: Pavuluri Sairam

Reg no: 11704286

Section: EE032

Roll no: A22

Email: pavulurisairam99@gmail.com

Phone no: 9182117482

Github link-

Question-1:

Develop a scheduler which submits the processes to the processor in the following scenario, and compute the scheduler performance by providing the waiting time for process, turnaround time for process and average waiting time and turnaround time. Considering the arrival time and the burst time requirement of the processes the scheduler schedules the processes by interrupting the processor after every 3 units of time and does consider the completion of the process in this iteration. The schedulers than checks for the number of processes waiting for the processor and allots the processor to the process but interrupting the processor after every 6 units of time and considers the completion of the process in this iteration. The scheduler after the second iteration checks for the number of processes waiting for the processor and now provides the processor to the process with the least time requirement to go in the terminated state.

Answer:-

Description:-

Round-Robin Scheduling:-

Round-robin (RR) is one of the algorithms employed by process and network schedulers in computing. As the term is generally used, time slices (also known as time quanta) are assigned to each process in equal portions and in circular order, handling all processes without priority (also known as cyclic executive). Round-robin scheduling is simple, easy to implement, and starvation -free. Round-robin scheduling can be applied to other scheduling problems, such as data packet scheduling in computer networks. It is an operating system concept. The name of the algorithm comes from the round-robin principle known from other fields, where each person takes an equal share of something in turn.

**Burst time**

Every process in a computer system requires some amount of time for its execution. This time is both the CPU time and the I/O time. The CPU time is the time taken by CPU to execute the process. While the I/O time is the time taken by the process to perform some I/O operation. In general, we ignore the I/O time and we consider only the CPU time for a process. So, **Burst time is the total time taken by the process for its execution on the CPU.**

#### Arrival time

Arrival time is the time when a process enters into the ready state and is ready for its execution

#### Exit time

Exit time is the time when a process completes its execution and exit from the system.

#### Response time

Response time is the time spent when the process is in the ready state and gets the CPU for the first time. For example, here we are using the First Come First Serve

**Response time = Time at which the process gets the CPU for the first time - Arrival time**

#### Waiting time

Waiting time is the total time spent by the process in the ready state waiting for CPU.

**Waiting time = Turnaround time - Burst time**

#### Turnaround time

Turnaround time is the total amount of time spent by the process from coming in the ready state for the first time to its completion.

**Turnaround time = Burst time + Waiting time**

Algorithm: -

Steps to find waiting times of all processes:

1. Create an array rem\_bt[] to keep track of remaining burst time of processes. This array is initially a copy of bt[] (burst times array)
2. 2- Create another array wt[] to store waiting times of processes. Initialize this array as 0
3. Initialize time : t = 0
4. Keep traversing the all processes while all processes are not done. Do following for i'th process if it is not done yet.

a- If rem\_bt[i] > quantum

1. t = t + quantum
2. bt\_rem[i] -= quantum;

c- Else // Last cycle for this process

1. t = t + bt\_rem[i];
2. wt[i] = t - bt[i]
3. bt\_rem[i] = 0; // This process is over

Code:- **The Following code is implemented on c++ and for the waitimg time,average waiting time,turnaround time by taking burst time and arrival time and no of processers.**

**Consider a scheduler which schedules the job by considering the arrival time of the processes . The scheduler implements the shortest job first scheduling policy, but checks the queue of the processes after the every process terminates and time taken for checking and arranging the process according to the shortest job is 3 time unit. Compute the waiting time, turnaround time and average waiting time and turnaround time of the processes. Also compute the total time taken by the processor to compute all the jobs.**

#include<iostream>

using namespace std;

void findWaitingTime(int processes[], int n, int bt[],

int wt[], int at[])

{

int service\_time[n];

service\_time[0] = 0;

wt[0] = 0;

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

{

service\_time[i] = service\_time[i-1] + bt[i-1];

wt[i] = service\_time[i] - at[i];

if (wt[i] < 0)

wt[i] = 0;

}

}

void findTurnAroundTime(int processes[], int n, int bt[],

int wt[], int tat[])

{

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

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

}

void findavgTime(int processes[], int n, int bt[], int at[])

{

int wt[n], tat[n];

findWaitingTime(processes, n, bt, wt, at);

findTurnAroundTime(processes, n, bt, wt, tat);

cout << "Processes " << " Burst Time " << " Arrival Time "

<< " Waiting Time " << " Turn-Around Time "

<< " Completion Time \n";

int total\_wt = 0, total\_tat = 0;

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

int compl\_time = tat[i] + at[i];

cout << " " << i+1 << "\t\t" << bt[i] << "\t\t"

<< at[i] << "\t\t" << wt[i] << "\t\t "

<< tat[i] << "\t\t " << compl\_time << endl;

}

cout << "Average waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

int main()

{

int n,i;

cout<<"Enter the no.of Processes"<<endl;

cin>>n;

int processes[n],burst\_time[n],arrival\_time[n];

cout<<"Enter the processes numbers"<<endl;

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

cin>>processes[i];

}

cout<<"Enter the Burst time"<<endl;

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

cin>>burst\_time[i];

}

cout<<"Enter the Arrival time"<<endl;

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

cin>>arrival\_time[i];

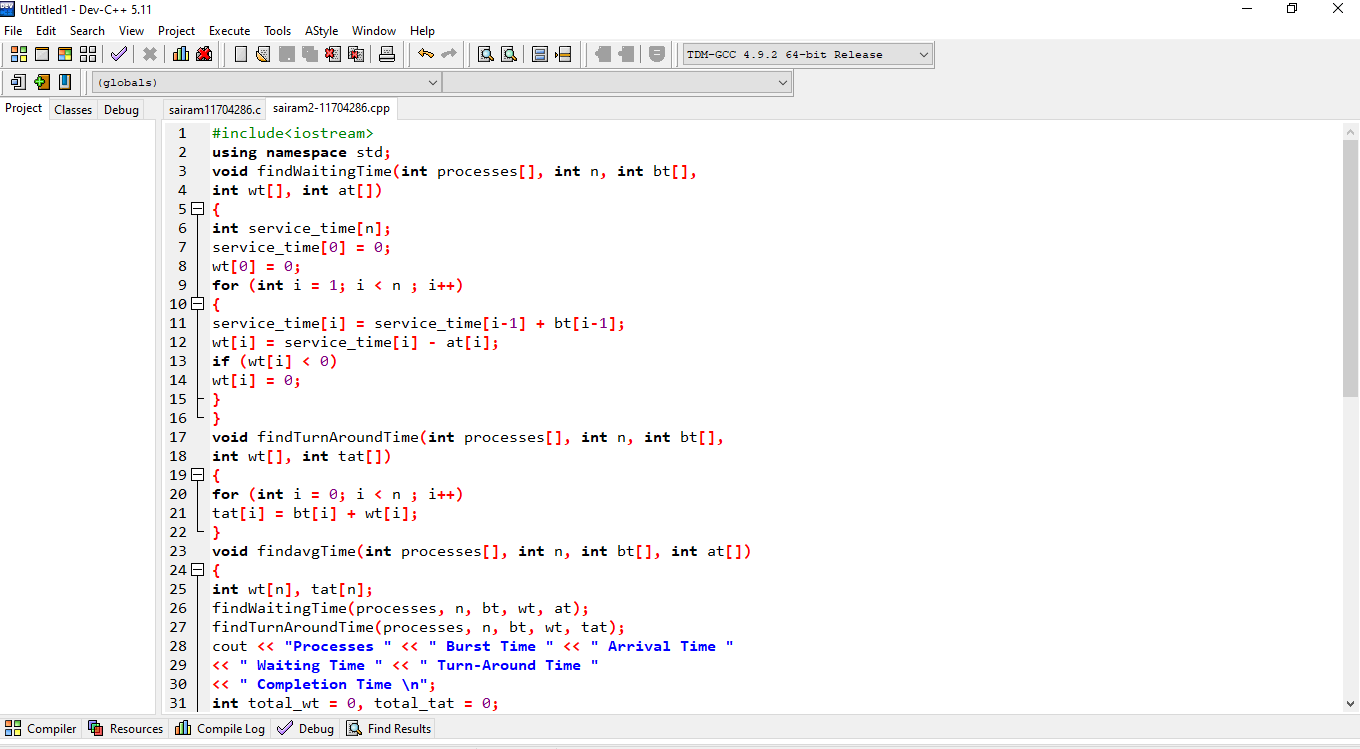
}

findavgTime(processes, n, burst\_time, arrival\_time);

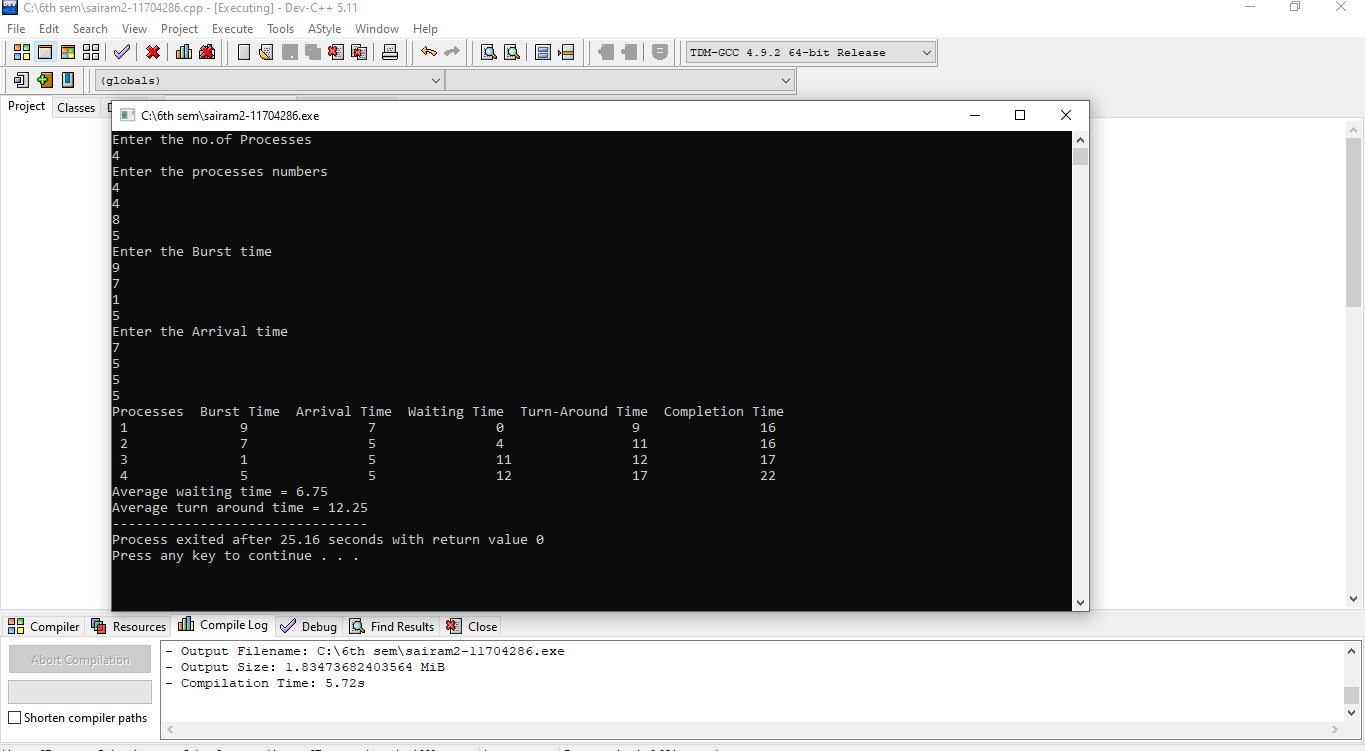
return 0;

}

**After Compilation:-**

****

Code Output



**Consider a scheduler which schedules the job by considering the arrival time of the processer. The scheduler implements the shortest job first scheduling policy, but checks the queue of the processes after the every process terminates and time taken for checking and arranging the process according to the shortest job is 3 time unit. Compute the waiting time, turnaround time and average waiting time and turnaround time of the processes. Also compute the total time taken by the processor to compute all the jobs. The inputs for the number of requirements, arrival time and burst time should be provided by the user.**

**Consider the following units for reference.**

**Process Arrival time Burst Time**

**1 0 6**

**2 3 2**

**3 5 1**

**4 9 7**

**5 10 5**

**Develop a scheduler which submits the processes to the processor in the defined scenario, and compute the scheduler performance by providing the waiting time for process, turnaround time for process and average waiting time and turnaround time.**

Code In C language:-

#include<stdio.h>

#include<conio.h>

int main()

{

int bt[10],p[10],n,temp,i,j,wt[10],sum=0;

float avg;

printf("Enter total no of proces:");

scanf("%d",&n);

printf("\n Enter burst time for each process:-");

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

{

printf("\nBurst time of process P%d:",i);

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

p[i]=i;

}

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

{

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

{

if(bt[i]>bt[j])

{

temp=bt[i];

bt[i]=bt[j];

bt[j]=temp;

temp=p[i];

p[i]=p[j];

p[j]=temp;

}

}

}

wt[0]=0;

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

{

wt[i]=wt[i-1]+bt[i-1];

}

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

{

sum+=wt[i];

}

avg=(float)sum/n;

printf("\n Waiting time for each process:-");

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

{

printf("\n Waiting time for process P%d is %d sec.",p[i],wt[i]);

}

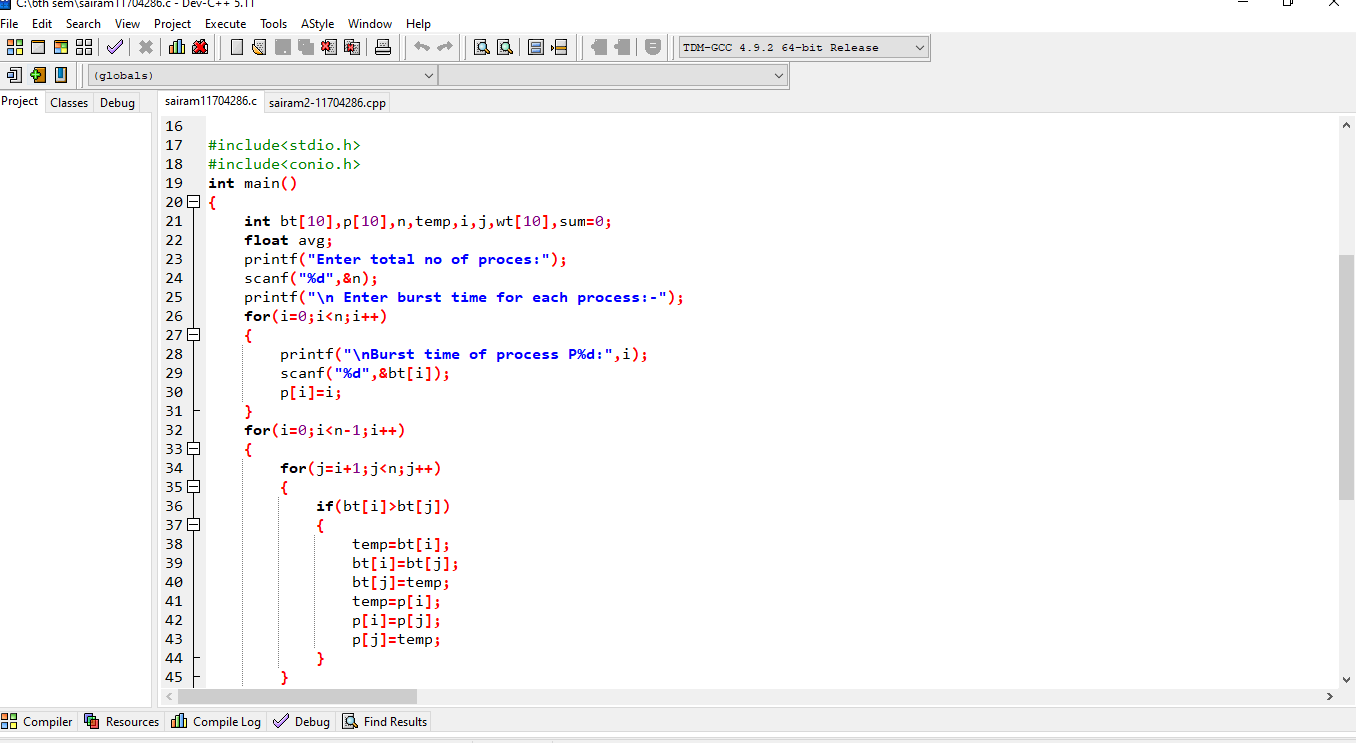
printf("\n Average waiting time is %f sec.",avg);

getch();

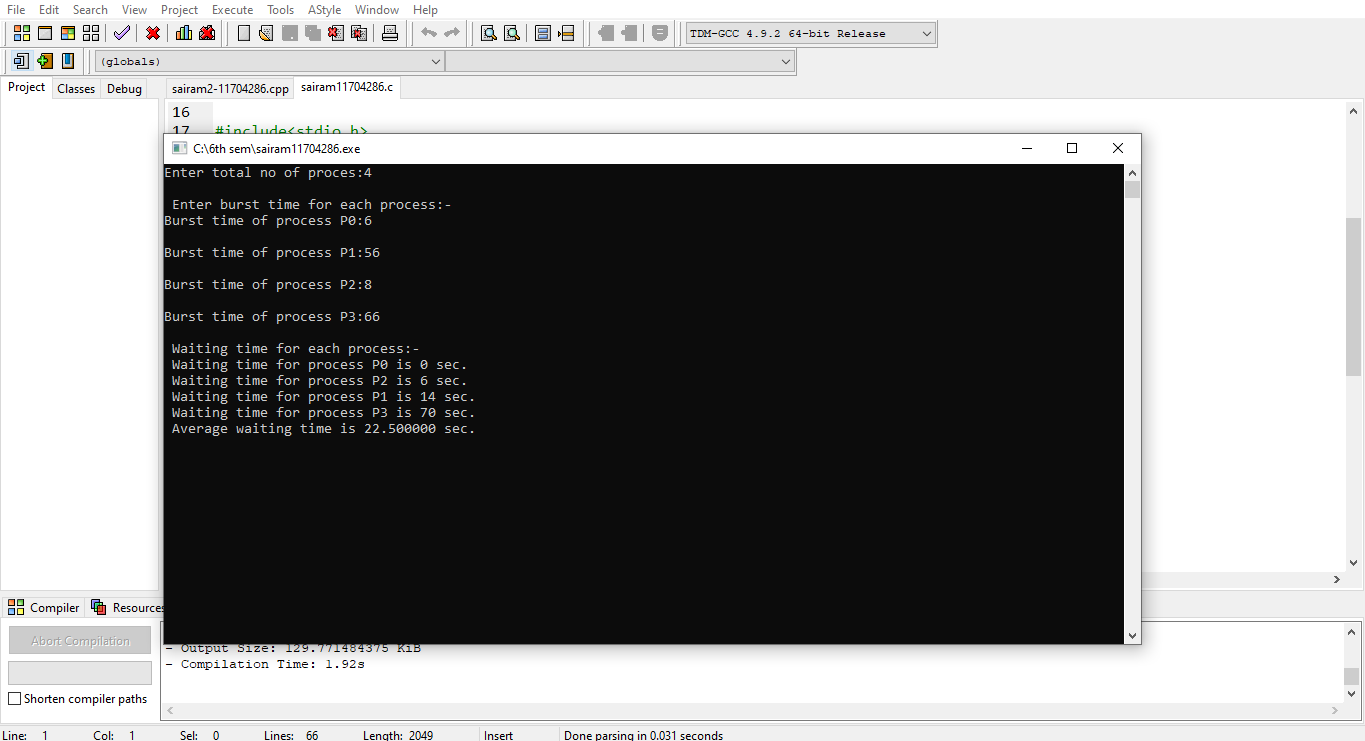
return 0;

}

**After compilation:-**

****

**Code Output:-**

****