**FINAL REPORT**

**Of CSE-316**



School of Computer Science and Engineering

Lovely Professional University.

Phagwara, Punjab (India).

|  |  |
| --- | --- |
| Name | PEDDAPURAM SHIVA KUMAR |
| Registration No | 11811598 |
| Section | K18NS |
| Roll\_No | RK18NSA25 |
| Course Code | CSE-316 |
| Submitted to | Ms. Suruchi Talwani |
| Github Link | <https://github.com/shiva9596/osprojectca3/blob/master/code.txt> |
| E-mail | [shivakumarpeddapuram05@gmail.com](mailto:shivakumarpeddapuram05@gmail.com)  gouds5788@gmail.com |

**Contents:-**

|  |  |  |
| --- | --- | --- |
| **s.no** | **topic** | **Page.no** |
| **1.)** | **Assigned Problem** | **3** |
| **2.)** | **Description of problem** | **3-4** |
| **3.)** | **Definitions of different terminologies** | **4** |
| **4.)** | **Objective,advantages,disadvantages** | **4-5** |
| **5.)** | **Explanation of algorithm detailly and complexity of algorithm** | **5-7** |
| **6.)** | **Code Snippet** | **8-10** |
| **7.)** | **Boundary Conditions** | **11** |
| **8.)** | **Test Cases** | **11-13** |
| **9.)** | **Code compilation proof** | **13-14** |
| **10.)** | **Output of the project** | **15** |
| **11.)** | **Git Hub link** | **16** |
| **12.)** | **Constraints** | **16** |

**1.)Question Assigned for me :**

Considering 4 processes with 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 then 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. The inputs for the number of requirements, arrival time and burst time should be provided by the user.

**Reference table and values Assigned for me :**

|  |  |  |
| --- | --- | --- |
| PROCESS | Arrival \_time | Burst \_time |
| P1 | 0 | 18 |
| P2 | 2 | 23 |
| P3 | 4 | 13 |
| P4 | 13 | 10 |

**To solve the following question :**

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

**2.)Description:**

For the above assigned question we can solve it by using the scheduling approach called **ROUND ROBIN** algorithm which is pre-emptive in nature.

**Working steps :**

Here once a process is executed for a given time period , it is preempted and other process executes for a given time period.The newly created process is added to end of ready queue.By using this there will be no starvation because all the processes are given the same priority

**Why do we need these scheduling?**  
scheduling is a typical process which involves both I/O time and CPU time. In a uni programming system time spent waiting for I/O is wasted and CPU is free during this time

but In multi programming systems, one process can use CPU while another is waiting for I/O. This is possible only with process scheduling.

**3.)Some Definitions of Terms which are used during these project:**

1. **pre-emptive scheduling** :In pre-emptive scheduling, once the CPU cycle is allocated to process, the process holds it till it reaches the Quantum time and then pre-empt to the process which has least burst time .
2. **response time** : it is defined as the time when a process produces its first response
3. **completion time** : it is defined as the time at which a process completes its execution
4. **Burst time:** it is the total time taken by the process for execution on the CPU.
5. **Arrival time**: it is the time at which the process arrives in the ready queue .
6. **Waiting time:** it is defined as the time at which a process waits in ready queue

|  |
| --- |
| **Waiting time = “Turnaround time - Burst time”** |

1. **Turnaround time:** it is the time difference between completion time and arrival time

|  |
| --- |
| **Turnaround time(TAT) = completion time – arrival time** |

**Objectives of scheduling programs:**

To increase Maximum CPU utilization.

To increase Maximum throughput  
Minimum turnaround time   
Minimum waiting time

Minimums response time

**4.)Some Advantages of ROUND ROBIN SCHEDULER :-**

1. Each process is served by the cpu for a fixed time quantum , so all the processes are given the same priority
2. Starvation doesn't occur because, for each round robin cycle, every process is given a fixed time to execute. No process is left behind.
3. Every Thread / Process gets a chance to run.
4. CPU is shared between all processes.
5. Threads with the same priority are handled perfectly with Round Robin.

Disadvantages:

**4.)Some Disadvantages of round robin ::-**

1. Low Priority tasks may wait for more time if the many tasks are given high priority.
2. High Priority tasks may not execute the full instruction given stipulated amount of time.
3. The throughput in RR largely depends on the choice of the length of the time quantum. If time quantum is too large it behaves as FCFS. If time quantum is too short much of the time is spent in process switching and hence low throughput.
4. Also here one cannot assign priority to any process which can be a drawback.

**5.)ALGORITHM explanation :**

1. First declare the variables as per requirements :

int p\_no,j,no,CurT,RemProc,Time\_Q;

int AT[10],BT[10],remt[10];

int x=1,indicator = 0,wait = 0,TAT = 0;

1. In order to perform these project we need some inputs which need to be taken from the user as per the requirements in the question .take inputs like :arrival time burst time for each process .

for(p\_no = 0;p\_no < no;p\_no++)

{

printf("\nProcess P%d :-\n",p\_no+1);

printf("Arrival time : ");

scanf("%d",&AT[p\_no]);

printf("Burst time : ");

scanf("%d",&BT[p\_no]);

}

1. For the processes entered as 0 (or) Arrival time,Burst Time is less than 0 then the program will be terminated . as -ve values of burst time and arrival time does not exists .

if(AT[p\_no]<0)

{

printf("\nArrival Time Should not be less than 0 Please Try Again\n");

return 0;}

if(BT[p\_no]<=0)

{

printf("Burst Time Should not be less than 0 Please Try Again");

return 0;

}

1. Assigning the time quantum as 6 in first iteration.(as per the question requirements)
2. By using the below formulas let us find the following thing :

|  |
| --- |
| **TAT(turnaround time) = CT (completion time) – AT(arrival time)** |
| **WT(w aiting time) = TAT (turnaround time) - BT(Burst time).** |

Now by Using loop till the processes comes to 0, we should decrease the burst time by subtracting the time quantum from it and storing the remining thing in remt[] (remaining time) ,adding the time quantum to the current time represented as CT

WT+=CT-AT[p\_no]-BT[p\_no];

TAT+=CT-AT[p\_no];

1. If the process comes to last like

eg :if there are 5 processes and p\_no is equal to 5 then time quantum is shifted to 10 ,moves to 2nd iteration and same thing will be repeated.

1. Storing the remaining burst time into b\_time and to waiting time from the formula.

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

{

b\_time[i]=remt[i];

w\_time[i]=Cur\_t-AT[i]-b\_time[i];

p\_no[i]=i+1;

}

1. In third iteration it checks the highest value of burst time and assign the values in that order to the burst time.

if(b\_time[j]>b\_time[loc]){

loc=j;

}

temp=b\_time[i];

b\_time[i]=b\_time[loc];

b\_time[loc]=temp;

1. By using the formulas of TAT ,WT I found the values of TAT and WT and the avg of those values are stored in var named avg\_TAT and avg\_WT and the output is displayed.

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

{

TAT\_t[i]=b\_time[i]+w\_time[i];

total=total + TAT\_t[i];

printf("\nP%d\t\t\t%d\t\t\t%d\t\t\t%d",p\_no[i],b\_time[i],w\_time[i],TAT\_t[i]);

}

avg\_TAT=(float)total/n;

printf("\n\nAverage Waiting Time = %f",avg\_WT);

printf("\nAverage Turnaround Time = %f\n",avg\_TAT);

**5.)algorithm for better understanding :**

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue along with arrival time and burst time.

Step 3: For each process in the Ready Queue, assign the CPU to the process at time 0 till 3 units of time

Step 4: Assign the CPU to the process with low burst time if there are more than one process is arrived at the particular instant of time ,if there is only one process is there for execution then assign that process for execution and since it is pre-emptive in nature the process will preempt for every 3 units of time.

Step 4: For each process again assign the process repeating the step-3 until all the processes in the queue gets executed

Step 5: Calculate Waiting time for process(n) = waiting time of process(n-1)+ burst time of process(n-1 )+ the time difference in getting the CPU from process(n-1)

1. Turnaround time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).

Step 6: Calculate

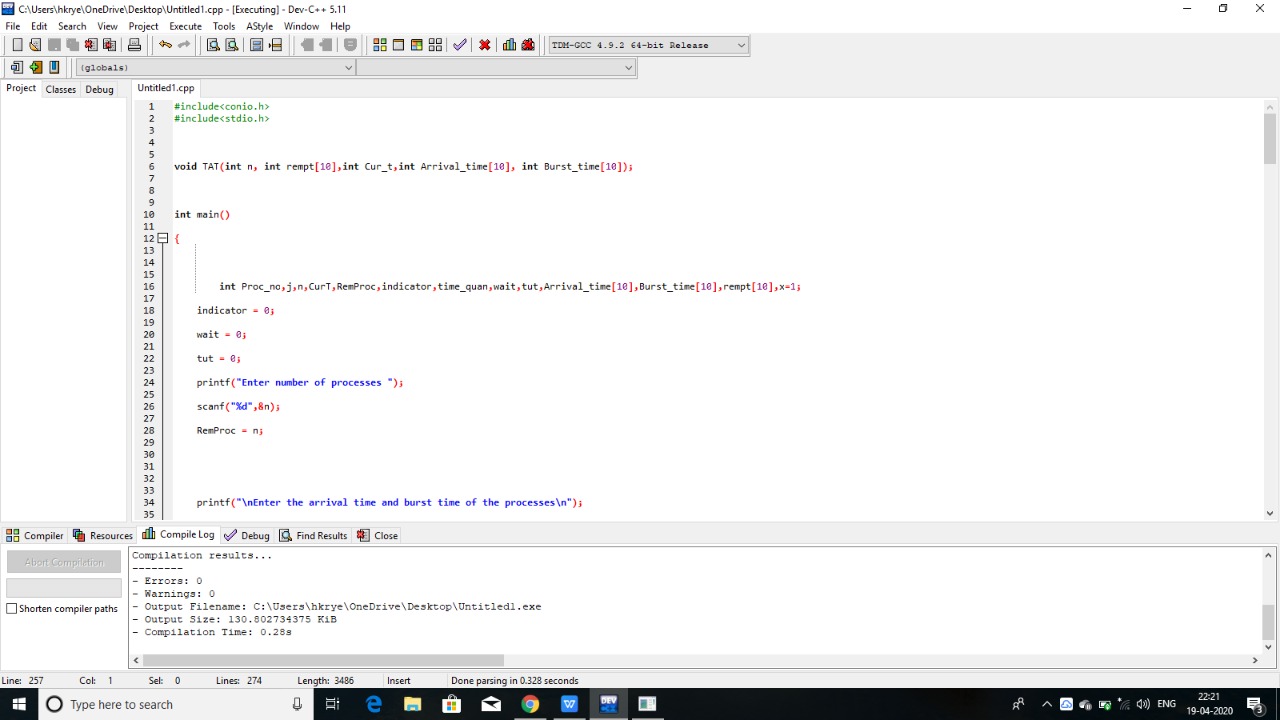
1. Average waiting time = Total waiting Time / Number of process
2. Average Turnaround time = Total Turnaround Time / Number of process Step 8: Stop the process

**Complexity:**

**Time complexity :O(n^2)**

**6.)Code for assigned question:**

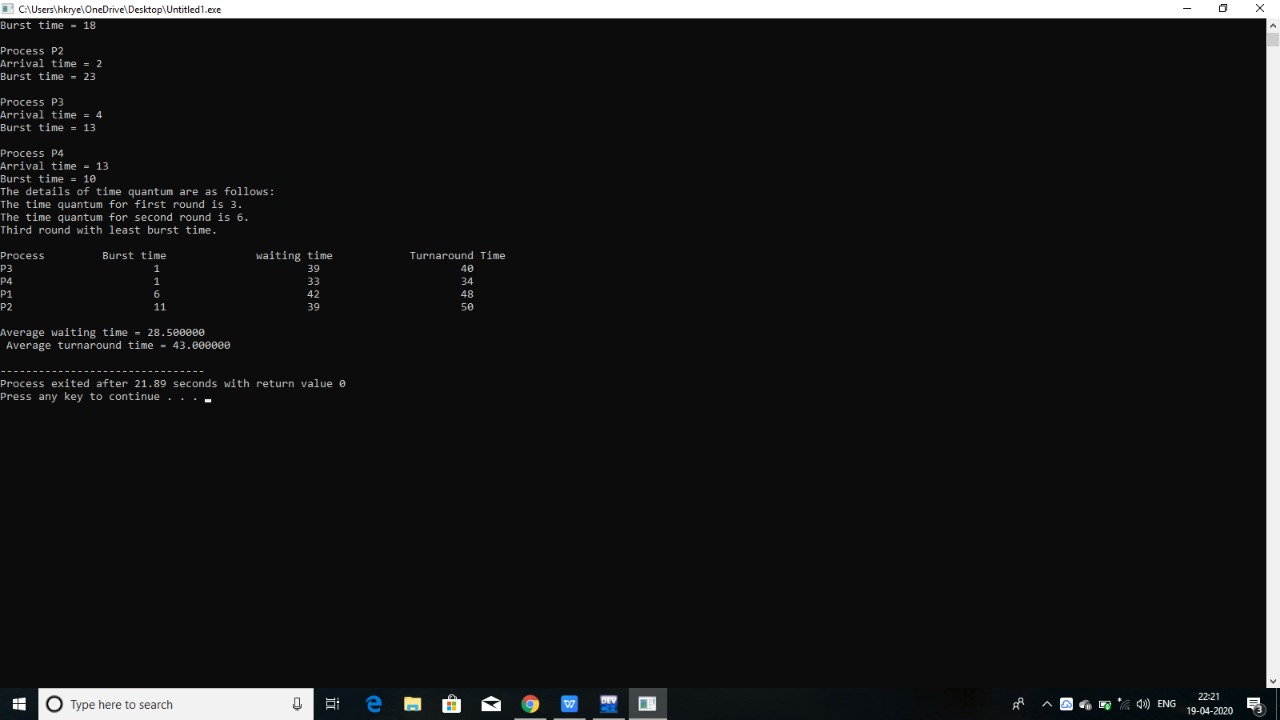
|  |
| --- |
|  |
| #include<conio.h> //declaring all necessary libraries for the code  #include<stdio.h>  void TAT(int n, int rempt[10],int Cur\_t,int Arrival\_time[10], int Burst\_time[10]);  int main()  {  intProc\_no,j,n,CurT,RemProc,indicator,time\_quan,wait,tut,Arrival\_time[10],Burst\_time[10],rempt[10],x=1;  indicator = 0;  wait = 0;  tut = 0;  printf("Enter number of processes ");  scanf("%d",&n);  RemProc = n;  printf("\nEnter the arrival time and burst time of the processes\n");  for(Proc\_no = 0;Proc\_no < n;Proc\_no++)  {  printf("\nProcess P%d\n",Proc\_no+1);  printf("Arrival time = ");  scanf("%d",&Arrival\_time[Proc\_no]);  printf("Burst time = ");  scanf("%d",&Burst\_time[Proc\_no]);  rempt[Proc\_no]=Burst\_time[Proc\_no];  }  printf("The details of time quantum are as follows:\n");  printf("The time quantum for first round is 3.\n");  time\_quan=3;  CurT=0;  for(Proc\_no=0;RemProc!=0;)  {  if(rempt[Proc\_no]<=time\_quan && rempt[Proc\_no]>0)  {  CurT+=rempt[Proc\_no];  rempt[Proc\_no]=0;  indicator=1;  }  else if(rempt[Proc\_no]>0)  {  rempt[Proc\_no]-=time\_quan;  CurT+=time\_quan;  }  if(rempt[Proc\_no]==0 && indicator==1)  {  printf("%d",Proc\_no);  RemProc--;  printf("P %d",Proc\_no+1);  printf("\t\t\t%d",CurT-Arrival\_time[Proc\_no]);  printf("\t\t\t%d\n",CurT-Burst\_time[Proc\_no]-Arrival\_time[Proc\_no]);  wait+=CurT-Arrival\_time[Proc\_no]-Burst\_time[Proc\_no];  tut+=CurT-Arrival\_time[Proc\_no];  indicator=0;  }  if(Proc\_no==n-1){  x++;  if(x==2)  {  Proc\_no=0;  time\_quan=6;  printf("The time quantum for second round is 6. \n");  }  else{  break;  }  }  else if(CurT >= Arrival\_time[Proc\_no+1]){  Proc\_no++;  }  else{  Proc\_no=0;  }  }  TAT(n, rempt, CurT, Arrival\_time, Burst\_time);  return 0;  }  void TAT(int n,int rempt[10],int Cur\_t,int Arrival\_time[10], int Burst\_time[10]){  float avg\_wait,avg\_tut;  int i,j,an=n,temp,btime[20],Proc\_no[20],w\_time[20],tut\_t[20],total=0,loc;printf("Third round with least burst time.\n");  for(i=0;i<an;i++)  {  btime[i]=rempt[i];  w\_time[i]=Cur\_t-Arrival\_time[i]-btime[i];  Proc\_no[i]=i+1;  }  for(i=0;i<an;i++)  {  loc=i;  for(j=i+1;j<an;j++)  {  if(btime[j]<btime[loc])  {  loc=j;  }  }  temp=btime[i];  btime[i]=btime[loc];  btime[loc]=temp;  temp=Proc\_no[i];  Proc\_no[i]=Proc\_no[loc];  Proc\_no[loc]=temp;  }  for(i=1;i<an;i++)  {  for(j=0;j<i;j++){  w\_time[i]+=btime[j];  }  total+=w\_time[i];  }  avg\_wait=(float)total/n;  total=0;  printf("\nProcess\t\tBurst time\t\twaiting time\t\tTurnaround Time");  for(i=0;i<an;i++)  {  tut\_t[i]=btime[i]+w\_time[i];  total=total + tut\_t[i];  printf("\nP%d\t\t\t%d\t\t\t%d\t\t\t%d",Proc\_no[i],btime[i],w\_time[i],tut\_t[i]);  }  avg\_tut=(float)total/n;  printf("\n\nAverage waiting time = %f",avg\_wait);  printf("\n Average turnaround time = %f\n",avg\_tut);  }  **7. Boundary Conditions :**   * Arrival time should not be negative. * Burst Time should be greater than 0. * Number of Processes should be greater than 1.   **8 .) some Test Cases to check our code whether working good or throwing any error: :**    **Test case 1 :**  In this test case let us consider the number of Processes entered is 0 it will exit from program.  A screenshot of a computer screen  Description automatically generated    **Test case-2:-** in this test case let us consider If arrival time is less than 0 the compiler will ask to enter different value which should be greater than 0. In the same way if we enter the burst time value less than 0 or equal to 0 it will show invalid message.  A screenshot of a computer  Description automatically generated  **Test case-3 :**  **Some data to be used during the evaluation :**  **Process Arrival time Burst time**  **P1 0 18**  **P2 2 23**  **P3 4 13**  **P4 13 10**  A screenshot of a computer  Description automatically generated  **9.) The following code got got successfully compiled without an error :** |
|  |
|  |
|  |
|  |
|  |



Assigned table to enter the values after execution of code

|  |  |  |
| --- | --- | --- |
| PROCESS | Arrival \_time | Burst \_time |
| P1 | 0 | 18 |
| P2 | 2 | 23 |
| P3 | 4 | 13 |
| P4 | 13 | 10 |

**10.)OUTPUT:**



**11.)GitHub Repository Link:** <https://github.com/shiva9596/osprojectca3/blob/master/code.txt>

**12.)Constraints:**

1. Number of Processes should be 1 or more than one.
2. Any Processes Should have same process numbers.
3. Any value of arrival time or Burst Time or priority value should not be negative.

The end