

**ACADEMIC TASK-3(CSE 316)**

**Course Code: CSE 316**

**Course Title: Operating Systems**

|  |
| --- |
| **SUBMITTED BY**  **Name: Aakash Mishra**  **Registration no.:11717002**  **Roll no.: 01**  **Section: K17GA** |

**SUBMITTED TO : Manpreet Singh**

**DATE OF SUBMISSION: 19th April 2019**

**Table of Contents**

1. Introduction 1

2. Algorithm………………………………………………………………………………......2-4

3. Code 5-11

4. Boundary Condition 12-13

5. Example 14

**Name: Aakash Mishra**

**Registration Number:** 11717002

**Email Address: aakashmishra800@gmail.com**

**1) Description:**

**Q1**. 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.

Answer-

It is **round robin scheduling** as **time quanta** is given as 3-time units in first iteration and 6 unit time second iteration with **Shortest remaining Job First scheduling**.

Round robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way.

Shortest job first is scheduling policy that selects waiting process with a smallest execution time to execute next.

For this time interval the process gets executed and if the process’s burst time is greater than the time quantum then the process is preempted and given back to ready queue. For a series of process P1, P2, P3 and so on (in waiting queue) and for the time quantum of 3 seconds the processor executes the process P1 for 3 seconds and then process P2 for 3 seconds and then process P3 for 3 seconds and if the burst time of any of the process is more than 3 seconds then that process is again sent back to ready queue.

The other algorithm is simply Shortest Job First. In Shortest Job First, the processor is assigned to the processes according to the minimum of Burst Time. The Burst Time of a process is simply the time for which that process has to execute. In this context, the processor is assigned to the process considering the minimum burst time first. Here the processor does not leave any process in between that is it does not preempt the processes.

**2) Algorithm:**

1. Initialization: Complexity = 1

n-processes, a [n]-arrival time, b[n]-burst time, x[n]-duplicate of burst time

waiting [n]-waiting time, turnaround[n]-turnaround time, count:=0,avg:=0

smallest, I,j,tt,time:=0

1. User Inputs: Complexity = n

* Input for n
* From i=0 to i=n

{ Input for a[i],input for b[i],x[i]=b[i]; }

1. Method: Complexity=n\*n

From time=0 till count! =n do

{

* + Sorting to find minimum burst time Complexity = n\*n
  + If(time>3) then decrement b[smallest];
  + If(b[smallest]=0) then increment count, end=time+1, completion[smallest]=end;

Update waiting time and turnaround time

}

1. Showing Table: Complexity= n

* Print process id, burst, arrival, start, completion, waiting, turnaround time for each process
* Print average turnaround time, and average waiting time

**3) Complexity**:

1. Initialization 1
2. User Inputs n
3. Method n\*n
4. Showing table n

Overall Complexity = n\*n (Approx.)

1. **Code Snippet:**

**#include<stdio.h>**

**#include<conio.h>**

**void aakash(int no,int RemainTime[10],int CompTime,int ArivTime[10],int BurTime[10]);**

**int main()**

**{**

**int ProcessNo,j,no,CompTime,RemainProcess,indic,TimeQuant,Wait,TAT,ArivTime[10],BurTime[10],RemainTime[10],x=1;**

**indic = 0;**

**Wait=0;**

**TAT = 0;**

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

**scanf("%d",&no);**

**RemainProcess = no;**

**printf("\nEnter the Arrival Time and Burst Time of the Processes:\n");**

**for(ProcessNo = 0;ProcessNo < no;ProcessNo++)**

**{**

**printf("\nProcess P%d\n",ProcessNo+1);**

**printf("Arrival Time = ");**

**scanf("%d",&ArivTime[ProcessNo]);**

**printf("Burst time = ");**

**scanf("%d",&BurTime[ProcessNo]);**

**RemainTime[ProcessNo]=BurTime[ProcessNo];**

**}**

**printf("The details for Time Quantum are as follows:\n");**

**printf("The Time Quantum for First Round is 3.\n");**

**TimeQuant=3;**

**CompTime=0;**

**for(ProcessNo=0;RemainProcess!=0;)**

**{**

**if(RemainTime[ProcessNo]<=TimeQuant && RemainTime[ProcessNo]>0)**

**{**

**CompTime+=RemainTime[ProcessNo];**

**RemainTime[ProcessNo]=0;**

**indic=1;**

**}**

**else if(RemainTime[ProcessNo]>0)**

**{**

**RemainTime[ProcessNo]-=TimeQuant;**

**CompTime+=TimeQuant;**

**}**

**if(RemainTime[ProcessNo]==0 && indic==1)**

**{ printf("%d",ProcessNo);**

**RemainProcess--;**

**printf("P %d",ProcessNo+1);**

**printf("\t\t\t%d",CompTime-ArivTime[ProcessNo]);**

**printf("\t\t\t%d\n",CompTime-BurTime[ProcessNo]-ArivTime[ProcessNo]);**

**Wait+=CompTime-ArivTime[ProcessNo]-BurTime[ProcessNo];**

**TAT+=CompTime-ArivTime[ProcessNo];**

**indic=0;**

**}**

**if(ProcessNo==no-1)**

**{**

**x++;**

**if(x==2)**

**{**

**ProcessNo=0;**

**TimeQuant=6;**

**printf("The Time Quantum for the second round is 6.\n");**

**}**

**else**

**{**

**break;**

**}**

**}**

**else if(CompTime >= ArivTime[ProcessNo+1])**

**{**

**ProcessNo++;**

**}**

**else**

**{**

**ProcessNo=0;**

**}**

**}**

**aakash(no,RemainTime,CompTime,ArivTime,BurTime);**

**return 0;**

**}**

**void aakash(int no,int RemainTime[10],int CompTime,int ArivTime[10],int BurTime[10])**

**{**

**float AvgWait,AvgTAT;**

**int i,j,n=no,Temp,BTime[20],ProcessNo[20],WTime[20],TAT[20],Total=0,loc;**

**printf("Third round with least burst time.\n");**

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

**{**

**BTime[i]=RemainTime[i];**

**WTime[i]=CompTime-ArivTime[i]-BurTime[i];**

**ProcessNo[i]=i+1;**

**}**

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

**{**

**loc=i;**

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

**{**

**if(BTime[j]<BTime[loc])**

**{**

**loc=j;**

**}**

**}**

**Temp=BTime[i];**

**BTime[i]=BTime[loc];**

**BTime[loc]=Temp;**

**Temp=ProcessNo[i];**

**ProcessNo[i]=ProcessNo[loc];**

**ProcessNo[loc]=Temp;**

**}**

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

**{**

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

**{**

**WTime[i]+=BTime[j];**

**}**

**Total+=WTime[i];**

**}**

**AvgWait=(float)Total/n;**

**Total=0;**

**printf("\nProcess\t\tBurst Time\t\tWaiting Time\t\tTurnaround Time");**

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

**{**

**TAT[i]=BTime[i]+WTime[i];**

**Total=Total + TAT[i];**

**printf("\nP%d\t\t\t%d\t\t\t%d\t\t\t%d",ProcessNo[i],BTime[i],WTime[i],TAT[i]);**

**}**

**AvgTAT=(float)Total/n;**

**printf("\n\nAverage waiting time = %f",AvgWait);**

**printf("\n Average turnaround time = %f\n",AvgTAT);**

**}**

**4)Additional algorithms used :**

1. Round Robin Scheduling:It is CPU scheduling algorithm in which we assigned a fixed time slot of 3 units and 6 units in cyclic way.
2. Shortest Job First:It is a scheduling policy that selects the waiting processes with the smallest execution time.

**5) Boundary Conditions:**

1. Value of n(number of process):

Number of processes an be given to n size.

1. Value of Arrival Time:

Must be positive and should not be infinite.

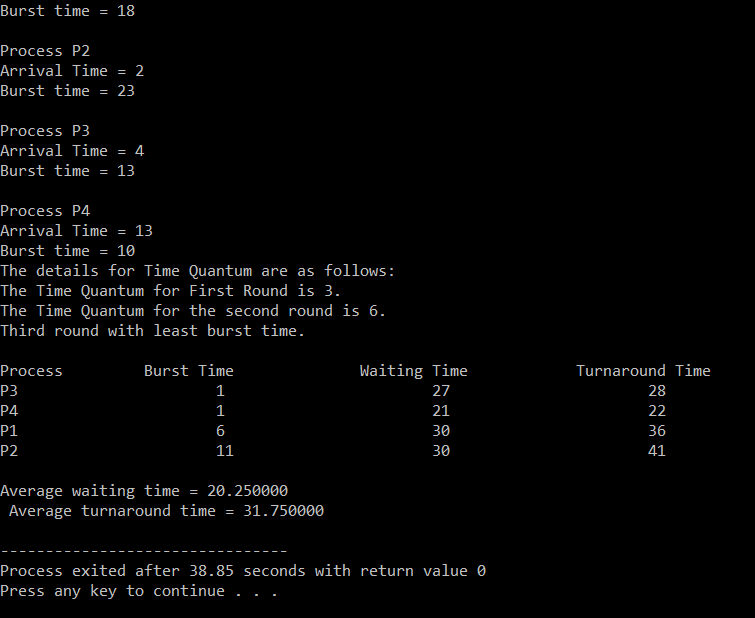
1. Value of Burst time:

Must be positive and should not approaches to infinite .

1. Integer inputs in all the inputs.

|  |  |  |
| --- | --- | --- |
| Process | Arrival Time | Burst Time |
| P1 | 0 | 18 |
| P2 | 2 | 23 |
| P3 | 4 | 13 |
| P4 | 13 | 10 |

**TEST CASES:**

****