**OS Project**

**Student Name** : Shakshi Sharma

**Student ID** : 11801940

**Email Address:** shakshiruchi26@gmail.com

**GitHub Link : https://github.com/ShakshiSharma/OS-Project**

=====================================================================

**Question Description**

• **Question No 2:**

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.

Consider the following units for reference.

Process Arrival time Burst time

P1 0 20

P2 5 36

P3 13 19

P4 26 42

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, turnaround time for process and

average waiting time and turnaround time.

**2. Algorithm:**

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- 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

(i) t = t + quantum

(ii) bt\_rem[i] -= quantum;

c- Else // Last cycle for this process

(i) t = t + bt\_rem[i];

(ii) wt[i] = t - bt[i]

(ii) bt\_rem[i] = 0; // This process is over

**3. Description (Purpose of Use):**

CPU scheduling algorithm is a process where each process is assigned a fixed time slot in

a cyclic way.

• It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.

• One of the most commonly used technique in CPU scheduling as a core.

• It is preemptive as processes are assigned CPU only for a fixed slice of time at most.

• The disadvantage of it is more overhead of context switching.

**4. Given Test Cases:**

• **Question No. 1:**

Process Arrival time Burst time

P1 0 20

P2 5 36

P3 13 19

P4 26 42

**5. Code snippet:**

**#include<stdio.h>**

**#include<conio.h>**

**void rr(int no,int remt[10],int Cur\_t,int arT[10], int bsT[10]);**

**main()**

**{**

**int Proc\_no,j,no,CurT,RemProc,indicator,time\_quan,wait,tut,arT[10],bsT[10],remt[10],x=1;**

**indicator = 0;**

**wait = 0;**

**tut = 0;**

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

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

**RemProc = no;**

**printf("\nEnter the arrival time and burst time of the processes\n");**

**for(Proc\_no = 0;Proc\_no < no;Proc\_no++)**

**{**

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

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

**scanf("%d",&arT[Proc\_no]);**

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

**scanf("%d",&bsT[Proc\_no]);**

**remt[Proc\_no]=bsT[Proc\_no];**

**}**

**printf("The details of time quantum are as follows:\n");**

**printf("The time quantum for first round is 6.\n");**

**time\_quan=6;**

**CurT=0;**

**for(Proc\_no=0;RemProc!=0;)**

**{**

**if(remt[Proc\_no]<=time\_quan && remt[Proc\_no]>0)**

**{**

**CurT+=remt[Proc\_no];**

**remt[Proc\_no]=0;**

**indicator=1;**

**}**

**else if(remt[Proc\_no]>0)**

**{**

**remt[Proc\_no]-=time\_quan;**

**CurT+=time\_quan;**

**}**

**if(remt[Proc\_no]==0 && indicator==1)**

**{ printf("%d",Proc\_no);**

**RemProc--;**

**printf("P %d",Proc\_no+1);**

**printf("\t\t\t%d",CurT-arT[Proc\_no]);**

**printf("\t\t\t%d\n",CurT-bsT[Proc\_no]-arT[Proc\_no]);**

**wait+=CurT-arT[Proc\_no]-bsT[Proc\_no];**

**tut+=CurT-arT[Proc\_no];**

**indicator=0;**

**}**

**if(Proc\_no==no-1){**

**x++;**

**if(x==2){**

**Proc\_no=0;**

**time\_quan=10;**

**printf("The time quantum for second round is 10. \n");**

**}**

**else{**

**break;**

**}**

**}**

**else if(CurT >= arT[Proc\_no+1]){**

**Proc\_no++;**

**}**

**else{**

**Proc\_no=0;**

**}**

**}**

**rr(no,remt,CurT,arT,bsT);**

**return 0;**

**}**

**void rr(int no,int remt[10],int Cur\_t,int arT[10], int bsT[10]){**

**float avg\_wait,avg\_tut;**

**int i,j,n=no,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<n;i++)**

**{**

**btime[i]=remt[i];**

**w\_time[i]=Cur\_t-arT[i]-btime[i];**

**Proc\_no[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=Proc\_no[i];**

**Proc\_no[i]=Proc\_no[loc];**

**Proc\_no[loc]=temp;**

**}**

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

**}**

**Github Link:https://github.com/ShakshiSharma/OS-Project/blob/master/OS%20Project.cpp**

**7. Test Case Applied:**

Process Arrival time Burst time Waiting Time TurnAround Time

P1 0 20 74 74

P2 5 36 55 58

P3 13 19 61 75

P4 26 42 39 65

Thank You