**OPERATING SYSTEM CASE BASED ASSIGNMENT**

**CA 03**

**QUESTION:**

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 provides processor and now 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     18

P2    2     23

P3     4     13

P4     13     10

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.

**CODE:**

**#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 3.\n");**

**time\_quan=3;**

**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=6;**

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

**}**

**Explanation to problem in terms of operating system**

In the given scenario the CPU iterates through 3 times and uses time quantum as 3 for first iteration and then uses 6 as time quantum to to finish the task.

It uses round robin scheduling algorithm in first two iteration with different time quantum and then it considers the third iteration and competes the process using shortest job first (SJF) algorithm.

Round robin scheduling algorithm :

In round robin scheduling , each ready task runs turn by turn only in a cyclic queue for a limited time slice.round robin uses premptive algorithm. The cpu is shifted to the next process after fixed interval time, which is called time quantum.

Since in the given scenario there is context switching after given time quantum in the both iterations and hence it provides sufficient conditions to support the problem statement that in the given case round robin cpu scheduling is used.

Shortest job first algorithm:

In this the process having the smallest execution time is chosen for the next execution .It significantly reduces the average wating time for the other process awaiting execution

This algorithm is helpful for batch-type processing ,where waiting for jobs to complete is not critical.

The scheduler after the second iteration checks for the number of processes waiting for the provides processor and now the processor to the process with the least time requirement to go in the terminated state. These statements supports that in the given scenario SJF cpu scheduling algorithm is used.

**Algorithm**

**Completion Time**: time at which process completes its execution.

**Turn Around Time**: Time difference between completion Time-Arrival time.

**Wating Time**: Time difference between turnaround time and burst time.

**Wating Time=Turn Around Time-Burst Time.**

Algorithm for Round Robin:

1. Iterate the cpu for finding wating time for time quantum as 3 and do the same for time quantum as 6.
   1. Create an array bti[] to keep track of remaing burst time of process. This array is initially a copy of b\_t[] (burst time array).
   2. Create another array wti[] to store wating times of processes . initialize this array as 0.
   3. Keep traversing the all processes while all processes are not done. Do following for i’th process if it is not done yet.
2. If bti[i] > quantum
3. t = t + quantum
4. bti[i] -=quantum
5. Else
6. t = t + bti[i]
7. wti[i] =t – b\_t[i]
8. bti[i] = 0

2.Sort all the process according to the arrival time.

2.1 Use **selection sort** to sort the burst time in ascending order

3.Then select that process which has minimum arrival time and minimum burst time.

4. After completion of process after which till the completion of previous process and select that process among the pool which is having minimum burst time.

**COMPLEXITY OF THE ALGORITHM**

1. Cpu iterates the loop for n times to get the arrival time and burst time .

hence this loop run for **O(n)** times.

1. To calculate the burst time the cpu iterates a nested loop for n^2 times.

Hence this loop run for **O(n^2)** times.

1. To calculate burst time in ascending order , selection sort is used which uses complexity of **O(n^2).**
2. The cpu iterates the loop for printing the values of burst time, arrival time, wating time and turnaround time.

Hence this loop runs for **O(n).**

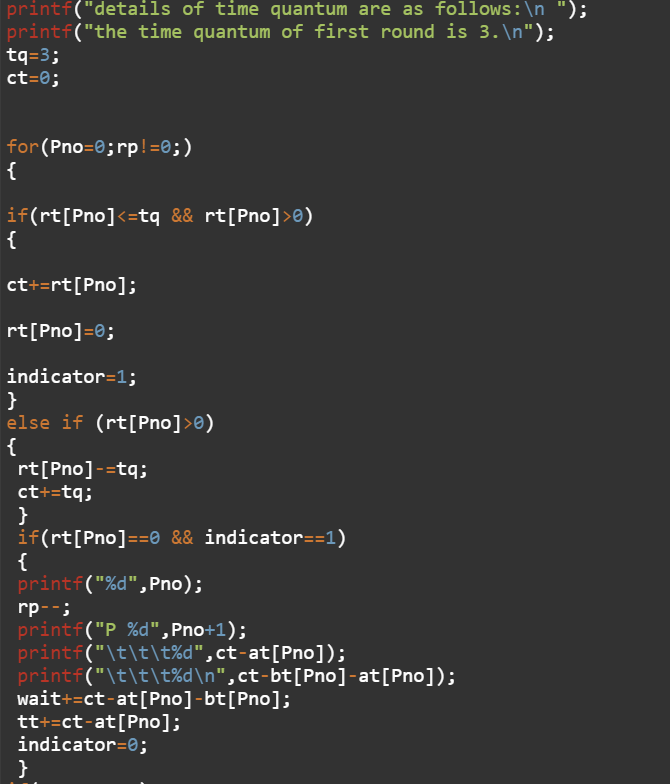
Therefore the total complexity for the code is **O(n^2).**

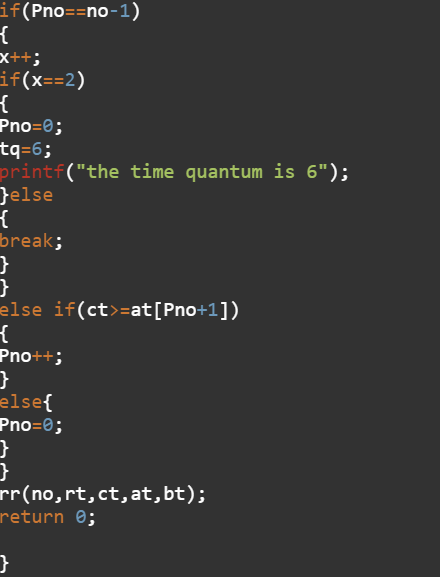
**CONSTRAINTS AND CODE SNIPPET**

**Constraints:**

1. If slicing time of os is low , the processor output will be reduced.
2. The performance heavily depends on time quantum .
3. It dones not give special priority to more important tasks.
4. Decreases comprehension.

**Code snippet of implemented constraints:**





**TEST CASES**

Considering the following units for refenece:

Process    Arrival time    Burst time

P1     0     18

P2    2     23

P3     4     13

P4     13     10

