**NAME: ANURAG SINGH PATEL**

**SECTION: K18KV G-2**

**ROLL NO: 51**

**EMAIL ID:** [**anuragsinghpatel98@gmail.com**](mailto:anuragsinghpatel98@gmail.com)

**Git hub link:**

**Question 12:** Implement the multi-level feedback queue scheduling algorithm by considering the following diagram: You can use the code of others to implement Roud-Robin, and FCFS but implement aging by your own self.

**Solution:**

**Description:**

**Multilevel Feedback Queue Scheduling**. In a **multilevel queue**-**scheduling** algorithm, processes are permanently assigned to a **queue** on entry to the system. ... The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority **queue**.

**Algorithm:**

1. When a process starts executing then it first enters queue 1.
2. In queue 1 process executes for 4 unit and if it completes in this 4 unit or it gives CPU for I/O operation in this 4 unit than the priority of this process does not change and if it again comes in the ready queue than it again starts its execution in Queue 1.
3. If a process in queue 1 does not complete in 4 unit then its priority gets reduced and it shifted to queue 2.
4. Above points 2 and 3 are also true for queue 2 processes but the time quantum is 8 unit. In a general case if a process does not complete in a time quantum than it is shifted to the lower priority queue.
5. In the last queue, processes are scheduled in FCFS manner.
6. A process in lower priority queue can only execute only when higher priority queues are empty.
7. A process running in the lower priority queue is interrupted by a process arriving in the higher priority queue.

**Uses:**

**Multilevel feedback queue scheduling**, however, allows a process to move between **queues**. The idea is to separate processes with different CPU-burst characteristics.

**Code:**

#include<stdio.h>

#include<stdlib.h>

struct process

{

char name;

int AT,BT,WT,TAT,RT,CT;

}

P1[10],P2[10],P3[10];

int n;

void sortByArrival()

{

struct process temp;

int i,j;

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

{

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

{

if(P1[i].AT>P1[j].AT)

{

temp=P1[i];

P1[i]=P1[j];

P1[j]=temp;

}

}

}

}

int main()

{

int i,j,k=0,r=0,time=0,tq1=8,tq2=16;

int flag=0;

char c;

printf("Enter no of processes:");

scanf("%d",&n);

for(i=0,c='A';i<n;i++,c++)

{

P1[i].name=c;

printf("\nEnter the arrival time and burst time of process %c: ",P1[i].name);

scanf("%d%d",&P1[i].AT,&P1[i].BT);

P1[i].RT=P1[i].BT;

}

sortByArrival();

time=P1[0].AT;

printf("Process in first queue following RR with qt=8");

printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t");

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

{

if(P1[i].RT<=tq1)

{

time+=P1[i].RT;

P1[i].RT=0;

P1[i].WT=time-P1[i].AT-P1[i].BT;

P1[i].TAT=time-P1[i].AT;

printf("\n%c\t\t%d\t\t%d\t\t%d",P1[i].name,P1[i].BT,P1[i].WT,P1[i].TAT);

}

else

{

P2[k].WT=time;

time+=tq1;

P1[i].RT-=tq1;

P2[k].BT=P1[i].RT;

P2[k].RT=P2[k].BT;

P2[k].name=P1[i].name;

k=k+1;

flag=1;

}

}

if(flag==1)

{

printf("\nProcess in second queue following RR with qt=16");

printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t");

}

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

{

if(P2[i].RT<=tq2)

{

time+=P2[i].RT;/from arrival time of first process +BT of this process/

P2[i].RT=0;

P2[i].WT=time-tq1-P2[i].BT;

P2[i].TAT=time-P2[i].AT;

printf("\n%c\t\t%d\t\t%d\t\t%d",P2[i].name,P2[i].BT,P2[i].WT,P2[i].TAT);

}

else

{

P3[r].AT=time;

time+=tq2;

P2[i].RT-=tq2;

P3[r].BT=P2[i].RT;

P3[r].RT=P3[r].BT;

P3[r].name=P2[i].name;

r=r+1;

flag=2;

}

}

{if(flag==2)

printf("\nProcess in third queue following FCFS ");

}

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

{

if(i==0)

P3[i].CT=P3[i].BT+time-tq1-tq2;

else

P3[i].CT=P3[i-1].CT+P3[i].BT;

}

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

{

P3[i].TAT=P3[i].CT;

P3[i].WT=P3[i].TAT-P3[i].BT;

printf("\n%c\t\t%d\t\t%d\t\t%d\t\t",P3[i].name,P3[i].BT,P3[i].WT,P3[i].TAT);

}

}

**Question 14:** CPU schedules N processes which arrive at different time intervals and each process is allocated the CPU for a specific user input time unit, processes are scheduled using a preemptive round robin scheduling algorithm. Each process must be assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes one task has priority 0. The length of a time quantum is T units, where T is the custom time considered as time quantum for processing. If a process is preempted by a higher priority process, the preempted process is placed at the end of the queue. Design a scheduler so that the task with priority 0 does not starve for resources and gets the CPU at some time unit to execute. Also compute waiting time, turn around.

**Answer:**

#include<stdio.h>

#include<conio.h>

int main()

{

int B\_T[10],Process\_no[10],W\_T[10],T\_A\_T[10];

int A\_T[10],Prior[10],i,j,Number\_of\_Process,tot=0,flag,temp,Avg\_W\_T,Avg\_T\_A\_T;

printf("Enter Number of Process:");

scanf("%d",&Number\_of\_Process);

printf("\nEnter Burst Time and Prior\n");

for(i=0;i<Number\_of\_Process;i++)

{

printf("\nP[%d]\n",(i+1));

printf("Enter Burst Time:");

scanf("%d",&B\_T[i]);

printf("Enter Arrival Time:");

scanf("%d",&A\_T[i]);

printf("Enter Prior:");

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

Process\_no[i]=i+1;

}

for(i=0;i<Number\_of\_Process;i++)

{

flag=i;

for(j=i+1;j<Number\_of\_Process;j++)

{

if(Prior[j]<Prior[flag])

flag=j;

}

temp=Prior[i];

Prior[i]=Prior[flag];

Prior[flag]=temp;

temp=B\_T[i];

B\_T[i]=B\_T[flag];

B\_T[flag]=temp;

temp=Process\_no[i];

Process\_no[i]=Process\_no[flag];

Process\_no[flag]=temp;

}

W\_T[0]=0;

for(i=1;i<Number\_of\_Process;i++)

{

W\_T[i]=0;

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

W\_T[i]+=B\_T[j];

tot+=W\_T[i];

}

Avg\_W\_T=tot/Number\_of\_Process;

tot=0;

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

printf("\nProcess\t B T W T T A T");

for(i=0;i<Number\_of\_Process;i++)

{

T\_A\_T[i]=B\_T[i]+W\_T[i];

tot+=T\_A\_T[i];

printf("\nProcess %d \t %d\t \t %d\t\t%d",Process\_no[i],B\_T[i],W\_T[i],T\_A\_T[i]);

}

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

Avg\_T\_A\_T=tot/Number\_of\_Process;

printf("\n\nAvg W T= %d",Avg\_W\_T);

printf("\nAvg T A T= %d",Avg\_T\_A\_T);

return 0;

}