Student Name: Amit Singh

Registration No.: 11806977

Email Address: [amit28amical@gmail.com](mailto:amit28amical@gmail.com)

GitHub Link:

Question Code: 12 and 15

Question No. 12

**Multi-Level Feedback Queue Scheduling Algorithm**

Multilevel Feedback Queue Scheduling –

* In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. This setup has the advantage of low scheduling overhead, but the disadvantage of being inflexible.
* Multilevel feedback queue scheduling, however, allows a process to move between queues.
* 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.
* Similarly, a process that waits too long in a lower-priority queue may be moved to a higher priority queue. This form of aging prevents starvation.
* Multilevel Feedback Queue Scheduling (MLFQ) keep analyzing the behavior (time of execution) of processes and according to which it changes its priority.

In general, a multilevel feedback queue scheduler is defined by the following parameters:

* The number of queues.
* The scheduling algorithm for each queue.
* The method used to determine when to upgrade a process to a higher-priority queue.
* The method used to determine when to demote a process to a lower-priority queue.
* The method used to determine which queue a process will enter when that process needs service.

**C program to simulate multilevel feedback queue scheduler**

Multi-level feedback queue scheduler Q consists of 3 linear queues, i.e., Q1, Q2, and Q3.

Q1 is round robin with time quantum 8 (RR8)

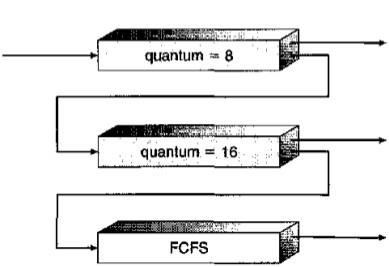
Q2 is round robin with time quantum 16 (RR16), and

Q3 follows first come first serve (FCFS)

The process cannot be executed in the lower queue if there are any jobs in all higher queues. For example, Q1 has 5 processes, Q2 has 1 process, and Q3 has 1 process. Then, first the process in Q1 should be executed (and completed), and then a process in Q2 is executed. Finally, Q3 will get CPU resource.

A new process enters queue Q1 which is served RR5. • When it gains CPU, a process receives 5 milliseconds. • If it does not finish in 5 milliseconds, the process is moved to queue Q2. • At Q2 process is again served RR8 and receives 8 additional milliseconds. • If it still does not complete, it is preempted and moved to queue Q3. • At Q3 process is executed by first come first serve. • If it still does not complete, it is processed at Q2 until completed.

OUTPUT: The remaining time of processes in each queue level, total waiting time and total turnaround time are displayed.



#include<stdio.h>

struct process

{

char name;

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

}Q1[8],Q2[16],Q3[100];/\*Three queues\*/

int n;

void sortByArrival()

{

struct process temp;

int i,j;

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

{

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

{

if(Q1[i].AT>Q1[j].AT)

{

temp=Q1[i];

Q1[i]=Q1[j];

Q1[j]=temp;

}

}

}

}

int main()

{

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

char c;

printf("Enter no of processes:");

scanf("%d",&n);

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

{

Q1[i].name=c;

printf("\nEnter the arrival time and burst time of process %c: ",Q1[i].name);

scanf("%d%d",&Q1[i].AT,&Q1[i].BT);

Q1[i].RT=Q1[i].BT;/\*save burst time in remaining time for each process\*/

}

sortByArrival();

time=Q1[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(Q1[i].RT<=tq1)

{

time+=Q1[i].RT;/\*from arrival time of first process to completion of this process\*/

Q1[i].RT=0;

Q1[i].WT=time-Q1[i].AT-Q1[i].BT;/\*amount of time process has been waiting in the first queue\*/

Q1[i].TAT=time-Q1[i].AT;/\*amount of time to execute the process\*/

printf("\n%c\t\t%d\t\t%d\t\t%d",Q1[i].name,Q1[i].BT,Q1[i].WT,Q1[i].TAT);

}

else/\*process moves to queue 2 with qt=8\*/

{

Q2[k].WT=time;

time+=tq1;

Q1[i].RT-=tq1;

Q2[k].BT=Q1[i].RT;

Q2[k].RT=Q2[k].BT;

Q2[k].name=Q1[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(Q2[i].RT<=tq2)

{

time+=Q2[i].RT;/\*from arrival time of first process +BT of this process\*/

Q2[i].RT=0;

Q2[i].WT=time-tq1-Q2[i].BT;/\*amount of time process has been waiting in the ready queue\*/

Q2[i].TAT=time-Q2[i].AT;/\*amount of time to execute the process\*/

printf("\n%c\t\t%d\t\t%d\t\t%d",Q2[i].name,Q2[i].BT,Q2[i].WT,Q2[i].TAT);

}

else/\*process moves to queue 3 with FCFS\*/

{

Q3[r].AT=time;

time+=tq2;

Q2[i].RT-=tq2;

Q3[r].BT=Q2[i].RT;

Q3[r].RT=Q3[r].BT;

Q3[r].name=Q2[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)

Q3[i].CT=Q3[i].BT+time-tq1-tq2;

else

Q3[i].CT=Q3[i-1].CT+Q3[i].BT;

}

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

{

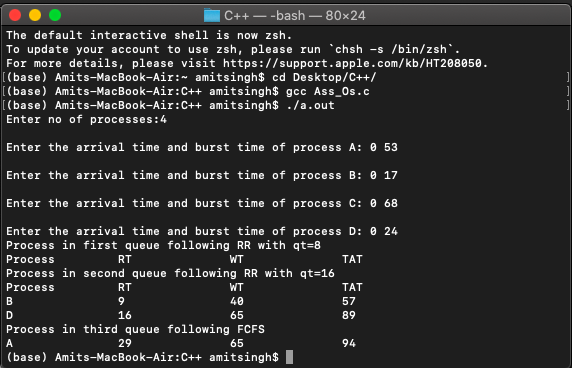
Q3[i].TAT=Q3[i].CT;

Q3[i].WT=Q3[i].TAT-Q3[i].BT;

printf("\n%c\t\t%d\t\t%d\t\t%d\t\t",Q3[i].name,Q3[i].BT,Q3[i].WT,Q3[i].TAT);

}

}

**Output**

**Question No. 15**

**Round Robin Scheduling Algorithm**

* The queue structure in ready queue is of First In First Out (FIFO) type.
* A fixed time is allotted to every process that arrives in the queue. This fixed time is known as time slice or time quantum.
* The first process that arrives is selected and sent to the processor for execution. If it is not able to complete its execution within the time quantum provided, then an interrupt is generated using an automated timer.
* The process is then stopped and is sent back at the end of the queue. However, the state is saved and context is thereby stored in memory. This helps the process to resume from the point where it was interrupted.
* The scheduler selects another process from the ready queue and dispatches it to the processor for its execution. It is executed until the time Quantum does not exceed.
* The same steps are repeated until all the process are finished.

The round robin algorithm is simple and the overhead in decision making is very low. It is the best scheduling algorithm for achieving better and evenly distributed response time.

**Priority Scheduling**

* Priority scheduling is a more general case of SJF, in which each job is assigned a priority and the job with the highest priority gets scheduled first. ( SJF uses the inverse of the next expected burst time as its priority - The smaller the expected burst, the higher the priority. )
* Note that in practice, priorities are implemented using integers within a fixed range, but there is no agreed-upon convention as to whether "high" priorities use large numbers or small numbers. This book uses low number for high priorities, with 0 being the highest possible priority.
* For example, the following Gantt chart is based upon these process burst times and priorities

|  |
| --- |
| #include<stdio.h>  int main()  {    int count,j,n,time,remain,flag=0,time\_quantum;    int wait\_time=0,turnaround\_time=0,at[10],bt[10],rt[10];    printf("Enter Total Process:\t ");    scanf("%d",&n);    remain=n;    for(count=0;count<n;count++)    {      printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);      scanf("%d",&at[count]);      scanf("%d",&bt[count]);      rt[count]=bt[count];    }    printf("Enter Time Quantum:\t");    scanf("%d",&time\_quantum);    printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");    for(time=0,count=0;remain!=0;)    {      if(rt[count]<=time\_quantum && rt[count]>0)      {        time+=rt[count];        rt[count]=0;        flag=1;      }      else if(rt[count]>0)      {        rt[count]-=time\_quantum;        time+=time\_quantum;      }      if(rt[count]==0 && flag==1)      {        remain--;        printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);        wait\_time+=time-at[count]-bt[count];        turnaround\_time+=time-at[count];        flag=0;      }      if(count==n-1)        count=0;      else if(at[count+1]<=time)        count++;      else        count=0;    }    printf("\nAverage Waiting Time= %f\n",wait\_time\*1.0/n);    printf("Avg Turnaround Time = %f",turnaround\_time\*1.0/n);      return 0;  } |

A screenshot of a cell phone

Description automatically generated