***Illustration:***

**For Question no. 24:**

* This is a program implementing Two level scheduling Queue:
* Level 1 : Fixed priority preemptive Scheduling
* Level 2 : Round Robin Scheduling
* For a Fixed priority pre-emptive scheduling if one process P1 is scheduled and running and another process P2 with higher priority comes. The New process with high priority process P2 preempts currently running process P1 and process P1 will go to second level queue. Time for which process will strictly execute must be considered in themultiplesof 2.
* All theprocesses in second level queue will complete their execution according to round robin scheduling.
* In this program Queue 2 will be processed after Queue 1 becomes empty and Priority of Queue 2 has lower priority than in Queue 1.

**Algorithm:**

* This program consists of two algorithms:- round robin scheduling and multilevel queue scheduling

**Algorithm For Multilevel Queue:**

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.

**Algorithm for round robin scheduling:**

1- Design an array **remain\_bur[]** to keep an eye on the remaining

burst time of processes. This array is initially a

copy of bur[] (burst times array)

2- Design another array **wait\_t[]** 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.

* 1. If remain\_bur[i] > quantum

(i) t = t + quantum

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

c- Else // Last cycle for this process

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

(ii) wait\_t[i] = t - bur[i]

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

**Boundary Conditions:**

* Level 1 : Fixed priority preemptive Scheduling
* Level 2 : Round Robin Scheduling
* Consider: 1. Queue 2 will be processed after Queue 1 becomes empty.
* Consider 2. Priority of Queue 2 has lower priority than in Queue 1.

**Code Snippet :**

#include<stdio.h>

struct pp

{

int pname;

int atime, wtime, ttime, p, btimec,burT;

}q1[10],q2[10];

void main()

{

struct pp tp;

int i,time=0,t1,t2,b\_t=0,greatest,totalp,count=0,k,pf2=0,totalp2,n,pos,j,f=0,y;

float waitT=0,tTime= 0,avg\_waitT,average\_tTime;

printf("\n Enter Total Number of Processes:\t");

scanf("%d", &totalp);

n=totalp;

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

{

printf("\nEnter name of the Process:-");

scanf("%d",&q1[i].pname);

printf("\nEnter Characteristics For processor %d:\n",q1[i].pname);

printf("Enter Arrival Time:-");

scanf("%d",&q1[i].atime);

printf("Enter Burst Time:-");

scanf("%d",&q1[i].burT);

q1[i].btimec=q1[i].burT;

printf("Enter Priority:\t");

scanf("%d",&q1[i].p);

}

printf("\nEnter Time Quantum for Fixed priority queue:-");

scanf("%d",&t1);

printf("\nEnter Time Quantum for Round Robin queue:-");

scanf("%d",&t2);

printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");

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

{

pos=i;

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

{

if(q1[j].atime<q1[pos].atime)

pos=j;

}

tp=q1[i];

q1[i]=q1[pos];

q1[pos]=tp;

}

time=q1[0].atime;

for(i=0;totalp!=0;i++)

{

while(count!=t1)

{

count++;

if(q1[i].atime<=time)

{

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

{

if(q1[j].atime==time && q1[j].p<q1[i].p)//pr<

{

q2[pf2]=q1[i];

pf2++;

for(k=i; k<totalp-1;k++)

q1[k]=q1[k+1];

totalp--;

count=0;

i=j-1;

j--;

}

}

}

time++;

q1[i].burT--;

if(q1[i].burT==0)

{

q1[i].ttime=time-q1[i].atime;

q1[i].wtime=q1[i].ttime-q1[i].btimec;

printf("%d\t|\t%d\t|\t%d\n",q1[i].pname,q1[i].ttime,q1[i].wtime);

waitT+=time-q1[i].wtime;

tTime+=time-q1[i].ttime;

for(k=i;k<totalp-1;k++)

q1[k]=q1[k+1];i--;

totalp--;

count=t1;break;

}

}

count=0;

if(q1[i].burT!=0)

{

q2[pf2]=q1[i];

pf2++;

for(k=i;k<totalp-1;k++)

q1[k]=q1[k+1];

totalp--;

}

if(i==totalp-1)

i=-1;

}

totalp2=pf2;

for(count=0;totalp2!=0;)

{

if(q2[count].burT<=t2&&q2[count].burT>0)

{

time+=q2[count].burT;

q2[count].burT=0;

f=1;

}

else if(q2[count].burT>0)

{

q2[count].burT-=t2;

time+=t2;

}

if(q2[count].burT==0&&f==1)

{

totalp2--;

q2[count].ttime=time-q2[count].atime;

q2[count].wtime=q2[count].ttime-q2[count].btimec;

printf("%d\t|\t%d\t|\t%d\n",q2[count].pname,q2[count].ttime,q2[count].wtime);

tTime+=time-q2[count].atime;

waitT+=time-q2[count].atime-q2[count].btimec;

for(k=count; k<totalp2;k++)

q2[k]=q2[k+1];count--;

f=0;

}

if(count==totalp2-1)

count=0;

else

count++;

}

printf("\n Average Waiting Time= %f\n", waitT/n);

printf("Average Turnaround Time = %f" ,tTime/n);

}

**Complexity:** O(n3)

**Test Cases:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Process | Arrival Time | Burst Time | Priority | Turnaround Time | Waiting Time |
| 1 | 0 | 4 | 1 | 10 | 6 |
| 2 | 0 | 3 | 1 | 13 | 10 |
| 3 | 0 | 8 | 2 | 8 | 3 |
| 4 | 10 | 5 | 1 | 20 | 12 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Process | Arrival Time | Burst Time | Priority | Turnaround Time | Waiting Time |
| 1 | 0 | 4 | 1 | 9 | 5 |
| 2 | 1 | 3 | 2 | 15 | 9 |
| 3 | 2 | 6 | 1 | 17 | 14 |
| 4 | 4 | 6 | 1 | 15 | 9 |