NAME : Navdeep Upadhyay
REGNO:11802595
MAILID: navdeep2nov99@gmail.com
GitHub Link: https://github.com/navdeepupadhyay/OSProject

Description:

For problem 6:

- •This is a scheduling program to implement a Queue with two levels:

- •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 the multiples of 2.

- •All the processes 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:

➢ In this problem algorithm for round robin scheduling and multilevel queue scheduling is used.

Algorithm for round robin scheduling:

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

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.

Code Snippet :

```c
#include<stdio.h>

struct process

{

    int pro_name;

    int arrT, waiturnT, turnT, priority, burT,burTcopy;

}queue1[10],queue2[10];

void main()

{

        struct process temp;

    int
i,time=0,t1,t2,bu_t=0,largest,totalProcess,count=0,k,pf2=0,totalProcess2,n,pos,j,flag=0,y;

    float wait_time=0,turnaround_time= 0,average_waiting_time,average_turnaround_time;

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

    scanf("%d", &totalProcess);

    n=totalProcess;

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

    {

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

        //fflush(stdin);

      scanf("%d",&queue1[i].pro_name);

      printf("\nEnter Details For processor %d:\n",queue1[i].pro_name);
```

```c
        printf("Enter Arrival Time:-");

        scanf("%d",&queue1[i].arrT);

        printf("Enter Burst Time:-");

        scanf("%d",&queue1[i].burT);

        queue1[i].burTcopy=queue1[i].burT;

        printf("Enter Priority:\t");

        scanf("%d",&queue1[i].priority);

    }

    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<totalProcess;i++)

    {

        pos=i;

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

        {

            if(queue1[j].arrT<queue1[pos].arrT)

                pos=j;

        }

        temp=queue1[i];

        queue1[i]=queue1[pos];
```

```
        queue1[pos]=temp;

    }

    time=queue1[0].arrT;

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

    {

        while(count!=t1)

        {

                count++;

                if(queue1[i].arrT<=time)

                {

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

                        {

                                if(queue1[j].arrT==time &&
queue1[j].priority<queue1[i].priority)//pr<

                                {

                                        queue2[pf2]=queue1[i];

                                                pf2++;

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

                                                queue1[k]=queue1[k+1];

                                        totalProcess--;

                                                count=0;

                                        i=j-1;

                                        j--;
```

```
                    }

                }

            }

            time++;

            queue1[i].burT--;

            if(queue1[i].burT==0)

            {

                    queue1[i].turnT=time-queue1[i].arrT;

                    queue1[i].waiturnT=queue1[i].turnT-queue1[i].burTcopy;


printf("%d\t|\t%d\t|\t%d\n",queue1[i].pro_name,queue1[i].turnT,queue1[i].waiturnT);

                    wait_time+=time-queue1[i].waiturnT;

            turnaround_time+=time-queue1[i].turnT;

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

                    queue1[k]=queue1[k+1];i--;

            totalProcess--;

                    count=t1;break;

            }

        }

        count=0;

        if(queue1[i].burT!=0)

        {

                queue2[pf2]=queue1[i];
```

6

```
                pf2++;

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

                queue1[k]=queue1[k+1];

        totalProcess--;

        }

                if(i==totalProcess-1)

                        i=-1;

}


totalProcess2=pf2;

for(count=0;totalProcess2!=0;)

{

        if(queue2[count].burT<=t2&&queue2[count].burT>0)

{

        time+=queue2[count].burT;

        queue2[count].burT=0;

        flag=1;

}

else if(queue2[count].burT>0)

{

        queue2[count].burT-=t2;

        time+=t2;

}
```

```c
        if(queue2[count].burT==0&&flag==1)

        {

                totalProcess2--;

                queue2[count].turnT=time-queue2[count].arrT;

                        queue2[count].waiturnT=queue2[count].turnT-
queue2[count].burTcopy;


        printf("%d\t|\t%d\t|\t%d\n",queue2[count].pro_name,queue2[count].turnT,queue2[count].waiturnT);

                turnaround_time+=time-queue2[count].arrT;

                wait_time+=time-queue2[count].arrT-queue2[count].burTcopy;

                for(k=count; k<totalProcess2;k++)

                        queue2[k]=queue2[k+1];count--;

                flag=0;

        }


        if(count==totalProcess2-1)

                count=0;

        else

                count++;

    }

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

    printf("Avg Turnaround Time = %f" ,turnaround_time/n);

}
```

Complexity: $O(n^3)$

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.

Test Cases:

Time Quantum for fixed priority queue = 2

Time Quantum for Round Robin queue = 2

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