Name: Abhishek Kumar Gupta
Regd. No.: 11804985
Email id.: abhishekkumargupta1207@gmail.com
GitHub: https://github.com/Abhikrgupta/Abhishek_OS-Project

- **Question Number 7:**

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.

- **Algorithms:**

  - **Round Robin Scheduling Algorithm:**

1: Create an array bursttime_remain[] to keep track of remaining burst time of the processes and should have an initial copy of bursttime[].

2: Create an array waitingtime[] that will store waiting times of processes and should be initially initialized as 0.

3: Initial time assigned to time, t should be zero i.e., t=0.

4: Traverse all the processes until all is completed and follow i process if its not done.

```
if bursttime_remain[] > quantum

i.      t = t + quantum;
ii.     bursttime_remain[i] - = quantum;

else

i.      t = t + bursttime_remain[i];
ii.     waitingtime[i] = t – bursttime[i];
iii.    bursttime_remain[i] = 0;
```

And hence the process gets over.

  - **Multilevel Queue Algorithm:**

1: As process starts the execution, it initially enters queue 1.

2: If the process in queue 1 executes for a fixed unit and completes in that fixed unit or it gives system for I/O operation in that fixed unit, that doesn’t change the priority. If it again comes in ready queue, then it needs to start execution again in queue 1.

3: If process of queue 1 is not possible in that fixed unit then the priority of that process reduces and is then shifted to queue 2.

4: The last 2 points are also valid for queue 2 but having the time quantum double of that fixed unit.

If the process fails to execute in that fixed time quantum then it is shifted to the lower priority queue.

5: Processes are scheduled in First Come First Serve (FCFS) manner in the last queue.

6: Whenever the higher priority queues are empty, that’s when a process in lower priority can execute.

7: Process arriving in the higher priority queue interrupts the process running in the lower priority queue.

- **Code Snippet:**

```
#include<stdio.h>

struct job

{

   int job_name, arrival, waiting, turntime, prio, burstcopy, bursttime;

}

queue_1[10], queue_2[10];

int main()

{          struct job temp;

           int i,time=0,x,z,b=0,largest,totaljob,c=0,k,pf2=0,totaljob2,n,position,j,f=0,y;

           float waitingtime=0,turnaround= 0,average_waiting_time,average_turnaround;

           printf("\n Please enter the total number of processes you want to have (enter a
number):\t");

           scanf("%d", &totaljob);

           n=totaljob;

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

   {

           printf("\nPlease name the process (enter a number):\t");

           scanf("%d",&queue_1[i].job_name);

           printf("\nPlease enter the details for processor %d:\n\t",queue_1[i].job_name);

           printf("Please enter the arrival time of this process:\t");

           scanf("%d",&queue_1[i].arrival);

           printf("Please enter the burst time of this process:\t");

           scanf("%d",&queue_1[i].bursttime);

           queue_1[i].burstcopy=queue_1[i].bursttime;

           printf("Please enter the priority of this process:\t");

           scanf("%d",&queue_1[i].prio);
```

```
}
        printf("\nPlease enter the Time Quantum for fixed priority queue processes:");
        scanf("%d",&x);
        printf("\nPlease enter the Time Quantum for round robin queue processes:");
        scanf("%d",&z);
        printf("\n\n\tThe Process\t|\tThe Turnaround Time\t|\tThe Waiting Time\n\n");
    for(i=0;i<totaljob;i++)
    {
        position=i;
        for(j=i+1;j<totaljob;j++)
        {
            if(queue_1[j].arrival<queue_1[position].arrival)
            position=j;
        }
        temp=queue_1[i];
        queue_1[i]=queue_1[position];
        queue_1[position]=temp;
    }
        time=queue_1[0].arrival;
    for(i=0;totaljob!=0;i++)
    {
        while(c!=x)
        {
            c++;
            if(queue_1[i].arrival<=time)
            {
                for(j=i+1;j<totaljob;j++)
```

```
                        {
                        if(queue_1[j].arrival==time && queue_1[j].prio<queue_1[i].prio)
                             {
                                  queue_2[pf2]=queue_1[i];
                                       pf2++;
                                  for(k=i; k<totaljob-1;k++)
                                       queue_1[k]=queue_1[k+1];
                                  totaljob--;
                                       c=0;
                                  i=j-1;
                                  j--;
                             }
                        }
                   }
                   time++;
                   queue_1[i].bursttime--;
                   if(queue_1[i].bursttime==0)
                   {
                        queue_1[i].turntime=time-queue_1[i].arrival;
                        queue_1[i].waiting=queue_1[i].turntime-queue_1[i].burstcopy;
printf("\t\t%d\t|\t\t%d\t\t|\t\t%d\n",queue_1[i].job_name,queue_1[i].turntime,queue_1[i].waiti
ng);
                        waitingtime+=time-queue_1[i].waiting;
                        turnaround+=time-queue_1[i].turntime;
                        for(k=i;k<totaljob-1;k++)
                             queue_1[k]=queue_1[k+1];i--;
                        totaljob--;
                        c=x;break;
                }
        }
        c=0;
        if(queue_1[i].bursttime!=0)
        {
                queue_2[pf2]=queue_1[i];
                pf2++;
                for(k=i;k<totaljob-1;k++)
                queue_1[k]=queue_1[k+1];
        totaljob--;
        }
                if(i==totaljob-1)
                        i=-1;
}

totaljob2 = pf2;
for(c=0;totaljob2!=0;)
{
        if(queue_2[c].bursttime<=z&&queue_2[c].bursttime>0)
        {
                time+=queue_2[c].bursttime;
                queue_2[c].bursttime=0;
                f=1;
        }
else if(queue_2[c].bursttime>0)
{
		queue_2[c].bursttime-=z;

		time+=z;

	}

	if(queue_2[c].bursttime==0 && f==1)

	{

		totaljob2--;

		queue_2[c].turntime=time-queue_2[c].arrival;

		queue_2[c].waiting=queue_2[c].turntime-queue_2[c].burstcopy;

	printf("%d\t|\t%d\t|\t%d\n",queue_2[c].job_name,queue_2[c].turntime,queue_2[c].wai
ting);

		turnaround+=time-queue_2[c].arrival;

		waitingtime+=time-queue_2[c].arrival-queue_2[c].burstcopy;

		for(k=c; k<totaljob2;k++)

			queue_2[k]=queue_2[k+1];c--;

		f=0;

	}


	if(c==totaljob2-1)

		c=0;

	else

		c++;

	}

	printf("\nThe Average Waiting Time is = %f\n", waitingtime/n);

	printf("The Average Turnaround Time is = %f" ,turnaround/n);

}
```

- **Complexity:** $O(n^3)$
- **Boundary Conditions:**
- Level 1: Fixed priority preemptive Scheduling
- Level 2: Round Robin Scheduling
- Consider: 1. Only after Queue 1 gets empty; Queue 2 will be processed.
- Consider 2. Queue 1 has higher priority than Queue 2.

- **Test Cases:**

Process	Arrival Time	Burst Time	Priority	Turnaround Time	Waiting Time
1	1	5	2	11	7
2	1	4	2	14	11
3	1	9	3	9	4
4	11	6	2	21	13

Process	Arrival Time	Burst Time	Priority	Turnaround Time	Waiting Time
1	0	4	1	9	5
2	1	3	2	15	9
3	3	7	2	18	15
4	4	6	1	15	9