Write a program for multilevel queue scheduling algorithm. There must be three queues generated. There must be specific range of priority associated with every queue. Now prompt the user to enter number of processes along with their priority and burst time. Each process must occupy the respective queue with specific priority range according to its priority.

Apply Round robin algorithm with quantum time 4 on queue with highest priority range. Apply priority scheduling algorithm on the queue with medium range of priority and First come first

serve algorithm on the queue with lowest range of priority. Each and every queue should get a quantum time of 10 seconds. Cpu will keep on shifting between queues after every 10 seconds i.e. to apply round robin algorithm OF 10 seconds on over all structure.

Calculate Waiting time and turnaround time for every process. The input for number of processes should be given by the user.

#include<stdio.h>

#include<unistd.h>

struct process

{

    int priority;

    int burst\_time;

    bool executed = False;

};

struct queues

{

    int priority\_start;

    int priority\_end;

    int length = 0;

    process \*p;

    bool executed = False;

};

void notComplete(queues q[])

{

       for(int i=0;i<3;i++)

       {

            for(int j=0;j<q[i].length;j++)

             {

                if(q[i].p[j].burst\_time != 0)

                 {

                    return true;

                 }

              }

        }

        return false;

}

void shuffleQueuesOrder(queues q[])

  {

    queues temp;

    for(int i=0;i<2;i++)

     {

        temp = q[i];

        q[i] = q[i+1];

        q[i+1] = temp;

     }

}

void processNotComplete(queues q[])

 {

    if(q.p[0].burst\_time != 0)

     {

       return true;

     }

    return false;

}

void sort\_rr(process \*temp)

{

}

main()

{

    queues q[3];

    q[0].priority\_start = 1;

    q[0].priority\_end = 3;

    q[1].priority\_start = 4;

    q[1].priority\_end = 6;

    q[2].priority\_start = 7;

    q[2].priority\_end = 9;

   int no\_of\_processes,priority\_of\_process,burst\_time\_of\_process;

     printf("enter number of processes");

    scanf("%d",&no\_of\_processes");

    process p1[no\_of\_processes];

    for(int i=0;i<no\_of\_processes;i++)

     {

        printf("Enter the priority of the process\n");

        scanf("%d",&priority\_of\_process);

        printf("Enter the burst time of the process\n");

        scanf("%d",&burst\_time\_of\_process);

        p1[i].priority = priority\_of\_process;

        p1[i].burst\_time = burst\_time\_of\_process;

      for(int j=0;j<3;j++)

       {

if(q[j].priority\_start<=priority\_of\_process&&priority\_of\_process<=q[j].priority\_end){

            q[j].length++;

        }

      }

    }

    for(int i =0;i<3;i++)

     {

        int len = q[i].length;

        q[i].p = new process[len];

     }

    int a=0;

    int b=0;

    int c=0;

    for(int i =0;i<3;i++)

      {

        for(int j=0;j<no\_of\_processes;j++)

          {

            if((q[i].priority\_start<=p1[j].priority) && (p1[j].priority<=q[i].priority\_end))

              {

                if(i==0)

                  {

                    q[i].p[a++] = p1[j];

                    }

                else if(i==1)

                    {

                    q[i].p[b++] = p1[j];

                    }

                else

                    {

                    q[i].p[c++] = p1[j];

                    }

            }

        }

    }

    a--;

    b--;

    c--;

    for(int i=0;i<3;i++)

       {

        printf("Queue%d",i+1);

        for(int j=0;j<q[i].length;j++)

         {

            printf(q[i].p[j].priority<<"->";

         }

        printf("NULL\n");

    }

    int timer = 0;

    int l =0;

    while(notComplete(q))

      {

         if(timer == 10)

          {

            l+=1;

             if(l>=3)

              {

                l=l%3;

               }

        }

        if(q[l].executed == true){

            l+=1;

            if(l>=3)

             {

                l=l%3;

             }

            continue;

        }

                if(l==0)

          {

            int rr\_timer = 4;

            while(processNotComplete(q[0])){

              while(rr\_timer!=0 || timer!= 10||processNotComplete(q))

                  {

                    q[0].p[0].burst\_time--;

                    rr\_timer--;

                    timer++;

                    }

                if(timer==10)

                 {

                    break;

                 }

                if(rr\_timer!=0)

                  {

                    q[0].p[0].executed = true;   
                    sort\_rr(q[0].p); }

                    else if(rr\_timer==0 && processNotComplete(q)){

                     sort\_rr(q[0].p);

                }

            }

        }

        else if(l==1)

         {

          //Priority Scheduling

        }

        else

         {

            //FCFS

        }

    }

}