```
#include <iostream>
#include <queue>
#include <fstream>
#include <time.h>

/*
Adam Novoa
CS 372
1/20/2015
*/

using namespace std;

struct process
{
	int totalTime;
	int remainingTime;
	int priorty;
	int level;
};
int totalWait = 0;
int totalNumProcess = 0;
class scheduler
{
private:
	process temp;//holds the process being run;
	int nextCheck;//time remaing until current quantum ends
	class qLevel
	{
	private:
		queue<process> Q;// Temp name for Queue
		int quantum; // Time Quantum for this queue
	public:
		qLevel(int time = 1)
		{
			if (time >= 1)
			{
				quantum = time;
			}
		}
		void setQuantum(int val)//REMOVE
		{
			if (val >= 1)
			{
				quantum = val;
			}
			return;
		}
		int getQuantum()
		{
			return quantum;
		}
		void addProcess(int time, int priorty, int level)
		{
			process temp;
			temp.totalTime = temp.remainingTime = time;
			temp.priorty = priorty;
			temp.level = level;
			Q.push(temp);
			return;
```

```
                }
                void addProcess(process temp)
                {
                        Q.push(temp);
                }
                bool empty()
                {
                        return Q.empty();
                }
                process getprocess()
                {
                        process temp = Q.front();
                        Q.pop();
                        return temp;
                }
                int getPriorty()
                {
                        process temp = Q.front();
                        return temp.priorty;
                }
                int getSize()
                {
                        int i = Q.size();
                        return i;
                }
        };
        vector<qLevel> feedbackQueue;
        int timeQuantum;// multiple for queue time quantum
        int numQueue;// number of queues
public:

        scheduler(int time = 2, int num = 4)
        {
                temp.totalTime = 0;
                temp.remainingTime = 0;
                temp.priorty = 51;
                temp.level = 0;
                timeQuantum = time;
                numQueue = num;
                if (time >= 1 && num >= 1)
                {
                        for (int i = 1; i <= num; i++)
                        {
                                feedbackQueue.emplace_back(pow(time,i));//should set time
quantum equal to time raised to pos
                        }
                }
        }

        void addProcess(int time, int priorty)
        {
                if (time >= 1 && time <= 500 && priorty >= 1 && priorty <= 50)
                {
                        for (int i = 0; i < numQueue; i++)
                        {
                                if (feedbackQueue[i].getQuantum() >= priorty)
                                {
                                        feedbackQueue[i].addProcess(time, priorty, i);
                                        break;
                                }else if (i == numQueue - 1)
                                {
```

```
                                        feedbackQueue[numQueue - 1].addProcess(time,
priorty,i);
                                }
                        }
                }
                return;
        }

        void runTick()
        {
                if (temp.remainingTime == 0)
                {
                        temp.priorty = 51;
                        int tempIndex = 0;
                        int index = 0;
                        while (tempIndex < numQueue)
                        {
                                if (feedbackQueue[tempIndex].empty())//Finds first non empty
queue
                                {
                                        tempIndex++;
                                }
                                else
                                {
                                        if (feedbackQueue[tempIndex].getPriorty() <
temp.priorty)//Check for higher priorty processes
                                        {
                                                temp.priorty =
feedbackQueue[tempIndex].getPriorty();
                                                index = tempIndex;
                                        }
                                        tempIndex++;
                                }
                        }
                        if (temp.priorty <= 50)//moves process to temp to run
                        {
                                temp = feedbackQueue[index].getprocess();
                                nextCheck = pow(timeQuantum, temp.level + 1);
                        }
                }
                if (temp.priorty <= 50)// && temp.remainingTime > timeQuantum)
                {
                        temp.remainingTime -= 1;
                        nextCheck -= 1;
                        if (temp.level < numQueue - 1 && nextCheck == 0 && temp.remainingTime
!= 0)//Process not from last queue
                        {
                                temp.level++;
                                feedbackQueue[temp.level].addProcess(temp);
                                temp.remainingTime = 0;
                        }
                        else if (nextCheck == 0 && temp.remainingTime != 0)//Process from last
queue
                        {
                                feedbackQueue[temp.level].addProcess(temp);
                                temp.remainingTime = 0;
                        }
                        if (temp.remainingTime == 0)
                        {
                                nextCheck = 0;
                                temp.priorty = 51;
```

```
                        }
                }
                return;
        }
        int getSize(int index)
        {
                return feedbackQueue[index].getSize();
        }
};
int main()
{
        srand(time(NULL));
        ofstream outfile("result.txt");
        int quantum, numQueues;
        cout << "Please enter the time quantum and number of queues." << endl;
        cin >> quantum;
        cin >> numQueues;
        scheduler myScheduler(quantum, numQueues);
        int numSec;
        cout << "Please enter the number of seconds to simulate." << endl;
        cin >> numSec;
        int time, priorty;
        int newProcess = 0;
        for (int i = 1; i < numSec+1; i++)
        {
                if (i%quantum == 0)
                {
                        for (int x = 0; x < numQueues; x++)
                        {
                                totalWait += myScheduler.getSize(x);
                        }
                        for (int x = 1; x <= numQueues; x++)
                        {
                                cout << "The number of processes in Q" << x << " is:" <<
myScheduler.getSize(x - 1) << endl;
                                outfile << "The number of processes in Q" << x << " is:" <<
myScheduler.getSize(x - 1) << endl;
                        }
                        cout << endl;
                        outfile << endl;
                }
                if (newProcess == 0)
                {
                        time = rand() % 500 + 1;
                        priorty = rand() % 50 + 1;
                        myScheduler.addProcess(time, priorty);
                        newProcess = rand() % 5 + 1;
                        totalNumProcess++;
                }
                myScheduler.runTick();
                newProcess--;
        }
        cout << "Average wait time per process:" << totalWait / (double)totalNumProcess <<
endl;
        cout << "The average length of the queues is:" << totalNumProcess / (double)numQueues
<< endl;

        system("pause");
        return 0;
}
```