Individual Assignment 3

**Due 5/8, 11:59 p.m.**

Modify the Java-based scheduler such that it has multiple queues representing different priorities. For example, have three separate queues, one each for priority 2, 3, and 4. Have the scheduler select a thread from the highest-priority queue, set the thread’s priority to 5, and allow the thread to run for a time quantum. When the time quantum expires, select the next thread from the highest queue and repeat the process. You should also modify the Scheduler class such that, when a thread is given to the scheduler, an initial priority is specified.

public class Scheduler extends Thread{

private CircularList queue;

private int timeSlice;

private static final int DEFAULT\_TIME\_SLICE = 1000; // 1 second

public Scheduler() {

timeSlice = DEFAULT\_TIME\_SLICE;

queue = new CircularList();

}

public Scheduler(int quantum) {

timeSlice = quantum;

queue = new CircularList();

}

/\*\*

\* adds a thread to the queue

\* @return void

\*/

public void addThread(Thread t) {

t.setPriority(2);

queue.addItem(t);

}

/\*\*

\* this method puts the scheduler to sleep for a time quantum

\* @return void

\*/

private void schedulerSleep() {

try {

Thread.sleep(timeSlice);

} catch (InterruptedException e) { };

}

public void run() {

Thread current;

// set the priority of the scheduler to the highest priority

this.setPriority(6);

while (true) {

try {

current = (Thread)queue.getNext();

if ( (current != null) && (current.isAlive()) ) {

current.setPriority(4);

schedulerSleep();

System.out.println("\* \* \* Context Switch \* \* \* ");

current.setPriority(2);

}

} catch (NullPointerException e3) { } ;

}

}

}

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

public class TestScheduler {

public static void main(String args[]) {

/\*\*

\* This must run at the highest priority to ensure that

\* it can create the scheduler and the example threads.

\* If it did not run at the highest priority, it is possible

\* that the scheduler could preempt this and not allow it to

\* create the example threads.

\*/

Thread.currentThread().setPriority(Thread.MAX\_PRIORITY);

Scheduler CPUScheduler = new Scheduler();

CPUScheduler.start();

TestThread t1 = new TestThread("Thread 1");

t1.start();

CPUScheduler.addThread(t1);

TestThread t2 = new TestThread("Thread 2");

t2.start();

CPUScheduler.addThread(t2);

TestThread t3 = new TestThread("Thread 3");

t3.start();

CPUScheduler.addThread(t3);

}

}

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

class TestThread extends Thread

{

private String name;

public TestThread(String id) {

name = id;

}

public void run() {

/\*

\* The thread does something

\*\*/

while (true) {

for (int i = 0; i < 500000; i++)

;

System.out.println("I am thread " + name);

}

} }

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

import java.util.\*;

public class CircularList

{

private Vector List;

private int index;

public CircularList() {

List = new Vector(10);

index = 0;

}

/\*\*

\* this method returns the next element in the list.

\* @return Object

\*/

public Object getNext() {

Object nextElement = null;

int lastElement;

if (!List.isEmpty() ) {

if (index == List.size() )

index = 0;

nextElement = List.elementAt(index);

++index;

}

return nextElement;

}

/\*\*

\* this method adds an item to the list

\* @return void

\*/

public void addItem(Object t) {

List.addElement(t);

}

}