**Homework 1**

**CSE 461 – Yousef Jarrar**

**Dr. Tong Yu**

1. ( 10 points ) \*\*Extra Credit for 461 Students\*\*   
We have discussed in the class the implementation of the readers-writers problem in Java. However, the read and write tasks of the **reader** thread and the **writer** thread are not given. Implement these tasks in Java as reading and writing of a file named *counter.txt*, which contains an integer counter.   
A **reader** thread

reads the counter from the file, and

prints out its thread name and the value of the counter.

A **writer** thread

increments the value of the counter in the file,

prints out its thread name and the new value of the counter.

Each thread repeats its task indefinitely in a random amount of time between 0 and 3000 ms. Your **main** program should create 20 **reader** threads and 3 **writer** threads.

Besides the source code, turn in scripts showing that you compile and run the program successfully. Turn in also some sample outputs.

/\*

\* Class created to hold lock and condition

\* \*/

import java.io.File;

import java.io.FileInputStream;

import java.io.FileWriter;

import java.io.IOException;

import java.util.Scanner;

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

public class ReadWrites {

private final String file = "counter.txt";

private final Lock \_mutex = new ReentrantLock();// create mutex instance

private final Condition readerQueue = \_mutex.newCondition();// Returns a new condition for

// reader that is bound to this Lock instance.

private final Condition writerQueue = \_mutex.newCondition();// Returns a new condition for writer that

// is bound to this Lock instance.

private int readers\_count = 0;// to store current readers count

private int writers\_count = 0;// to store current writers count

public ReadWrites() {

try {

FileWriter filewriter = new FileWriter(new File(file));

filewriter.write(new Integer(0).toString());

filewriter.close();

} catch (IOException e) {

e.printStackTrace();

}

}

void readerrun() throws InterruptedException {

\_mutex.lock(); // acquire lock

while (!(writers\_count == 0)) {

readerQueue.await();

}

readers\_count++;

\_mutex.unlock();

read(file);//read file

\_mutex.lock();

if (--readers\_count == 0) {

writerQueue.signal();//signal writers

}

\_mutex.unlock();//remove lock

}

void writerrun() throws InterruptedException {

\_mutex.lock();

while (!((readers\_count == 0) && (writers\_count == 0))) {

writerQueue.await();// when reader and writer is zero wait in

// writerQueue

}

writers\_count++; // increment writer

\_mutex.unlock();// remove lock

write(file);// write to file

\_mutex.lock(); // acquire lock

writers\_count--; // only one writer at a time

writerQueue.signal(); // signal writers

readerQueue.signalAll(); // signal all readers

\_mutex.unlock();// remove lock

}

void read(String path) {

try {

Scanner reader = new Scanner(new FileInputStream(path));

int x = reader.nextInt();

System.out.printf(Thread.currentThread().getName() + " is reading ...");

System.out.printf(" Counter value : %d\n", x);

reader.close();

} catch (IOException ex) {

ex.printStackTrace();

}

}

void write(String path) {

int countw;

try {

Scanner reader = new Scanner(new FileInputStream(path));

countw = (int) reader.nextInt();

countw++;

FileWriter f = new FileWriter(new File(path));

f.write(new Integer(countw).toString());

f.close();

System.out.printf(Thread.currentThread().getName() + " Writing... ");

System.out.printf(" Counter value : %d\n", countw);

reader.close();

} catch (IOException ex) {

ex.printStackTrace();

}

}

}

**OUTPUT:**

**Program Start**

**Reader Thread-21: Started**

**Reader Thread-10: Started**

**Reader Thread-4: Started**

**Reader Thread-14: Started**

**Reader Thread-20: Started**

**Reader Thread-16: Started**

**Reader Thread-5: Started**

**Reader Thread-8: Started**

**Reader Thread-9: Started**

**Reader Thread-7: Started**

**Reader Thread-19: Started**

**Writer Thread-1: Started**

**Reader Thread-15: Started**

**Reader Thread-11: Started**

**Writer Thread-0: Started**

**Reader Thread-13: Started**

**Reader Thread-17: Started**

**Reader Thread-6: Started**

**Writer Thread-2: Started**

**Reader Thread-22: Started**

**Reader Thread-12: Started**

**Reader Thread-18: Started**

**Reader Thread-3: Started**

**Thread-7 is reading ...Thread-15 is reading ... Counter value : 0**

**Thread-5 is reading ... Counter value : 0**

**Thread-13 is reading ... Counter value : 0**

**Thread-10 is reading ... Counter value : 0**

**Thread-20 is reading ... Counter value : 0**

**Thread-8 is reading ... Counter value : 0**

**Thread-18 is reading ... Counter value : 0**

**Thread-17 is reading ... Counter value : 0**

**Thread-22 is reading ... Counter value : 0**

**Thread-6 is reading ... Counter value : 0**

**Thread-19 is reading ... Counter value : 0**

**Thread-3 is reading ... Counter value : 0**

**Thread-14 is reading ... Counter value : 0**

**Thread-9 is reading ... Counter value : 0**

**Thread-16 is reading ... Counter value : 0**

**Thread-21 is reading ... Counter value : 0**

**Thread-4 is reading ... Counter value : 0**

**Thread-12 is reading ... Counter value : 0**

**Thread-11 is reading ... Counter value : 0**

**Counter value : 0**

**Thread-1 Writing... Counter value : 1**

**Thread-0 Writing... Counter value : 2**

**Thread-2 Writing... Counter value : 3**

**Thread-4 is reading ... Counter value : 3**

**Thread-0 Writing... Counter value : 4**

**Thread-5 is reading ... Counter value : 4**

**Thread-19 is reading ... Counter value : 4**

**Thread-8 is reading ... Counter value : 4**

**Thread-9 is reading ... Counter value : 4**

**Thread-14 is reading ... Counter value : 4**

**Thread-9 is reading ... Counter value : 4**

**Thread-13 is reading ... Counter value : 4**

**Thread-2 Writing... Counter value : 5**

**Thread-19 is reading ... Counter value : 5**

**Thread-18 is reading ... Counter value : 5**

**Thread-17 is reading ... Counter value : 5**

**Thread-3 is reading ... Counter value : 5**

**Thread-20 is reading ... Counter value : 5**

**Thread-11 is reading ... Counter value : 5**

**Thread-15 is reading ... Counter value : 5**

**Thread-6 is reading ... Counter value : 5**

**Thread-1 Writing... Counter value : 6**

**Thread-13 is reading ... Counter value : 6**

**Thread-19 is reading ... Counter value : 6**

**Thread-17 is reading ... Counter value : 6**

**Thread-8 is reading ... Counter value : 6**

**Thread-16 is reading ... Counter value : 6**

**Thread-10 is reading ... Counter value : 6**

**Thread-14 is reading ... Counter value : 6**

**Thread-3 is reading ... Counter value : 6**

**Thread-16 is reading ... Counter value : 6**

**Thread-22 is reading ... Counter value : 6**

**Thread-3 is reading ... Counter value : 6**

**Thread-2 Writing... Counter value : 7**

**Thread-12 is reading ... Counter value : 7**

**Thread-21 is reading ... Counter value : 7**

**Thread-7 is reading ... Counter value : 7**

**Thread-17 is reading ... Counter value : 7**

**Thread-4 is reading ... Counter value : 7**

**Thread-9 is reading ... Counter value : 7**

**Thread-7 is reading ... Counter value : 7**

**Thread-5 is reading ... Counter value : 7**

**Thread-11 is reading ... Counter value : 7**

**Thread-2 Writing... Counter value : 8**

**Thread-1 Writing... Counter value : 9**

**Thread-7 is reading ... Counter value : 9**

**Thread-16 is reading ... Counter value : 9**

**Thread-0 Writing... Counter value : 10**

**Thread-11 is reading ... Counter value : 10**

2. We discussed in class the readers-writer’s problem with **writers priority**, which can be solved in guarded commands:

|  |  |  |
| --- | --- | --- |
| **void reader()**  **{**  **when ( writers == 0 ) [**  **readers++;**  **]**  **//read**  **[readers--;]**  **}** |  | **void writer()**  **{**  **[writers++;]**  **when ( (readers == 0) && (active\_writers == 0) )[**  **active\_writers++;**  **]**  **//write**  **[writers--; active\_writers--;]**  **}** |

Here *writers* represents the number of threads that are either writing or waiting to write. The variable *active\_writers* represents the number of threads ( 0 or 1 ) that are currently writing.

Implement the solution using Java threads. Again simulate the tasks by reading from and writing to a file named *counter.txt* as in the previous problem.

Besides the source code, turn in scripts showing that you compile and run the program successfully. Turn in also some sample outputs.

*/\*\*  
 \* read and writer problem for threads  
 \*/*

import java.io.File;  
import java.io.FileInputStream;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.util.Random;  
import java.util.Scanner;  
import java.util.concurrent.locks.Condition;  
import java.util.concurrent.locks.Lock;  
import java.util.concurrent.locks.ReentrantLock;  
  
public class question6 {  
 final Lock mutex = new ReentrantLock();  
 final Condition readerQueue = mutex.newCondition();  
 final Condition writerQueue = mutex.newCondition();  
 int reader\_count = 0;  
 int writer\_count = 0;  
 int activeWriter\_count = 0;  
 String INP\_FILE = "counter.txt";  
 public static Random *random* = new Random();  
  
 */\*\*  
 \* constructor  
 \*/* public question6() {  
 FileWriter f;  
 try {  
 f = new FileWriter(new File(INP\_FILE));  
 f.write(new Integer(0).toString());  
 f.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Reader implementation  
 \*/* void reader() throws InterruptedException {  
 mutex.lock();  
 while (writer\_count != 0) {  
 readerQueue.await();  
 }  
 reader\_count++;  
 mutex.unlock();  
 readFile(INP\_FILE);  
 mutex.lock();  
 if (--reader\_count == 0) {  
 writerQueue.signal();  
 }  
 mutex.unlock();  
 }  
  
 */\*\*  
 \* writer implementation  
 \*/* void writer() throws InterruptedException {  
 mutex.lock();  
 writer\_count++;  
 while (!((reader\_count == 0) && (activeWriter\_count == 0))) {  
 writerQueue.await();  
 }  
 activeWriter\_count++;  
 mutex.unlock();  
 writeFile(INP\_FILE);  
 mutex.lock();  
 activeWriter\_count--;  
 if (--writer\_count == 0) {  
 readerQueue.signalAll();  
 } else {  
 writerQueue.signal();  
 }  
 mutex.unlock();  
 }  
  
 */\*\*  
 \* increment read count  
 \*/* public void readFile(String path) {  
 try {  
 int inp = new Scanner(new FileInputStream(path)).nextInt();  
 System.*out*.printf(Thread.*currentThread*().getName() + " Reader reading");  
 System.*out*.printf(" Counter value: %d\n", inp);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* increment writer count  
 \*/* public void writeFile(String path) {  
 int writerCount;  
 try {  
 writerCount = (int) new Scanner(new FileInputStream(path)).nextInt();  
 writerCount++;  
 FileWriter writr = new FileWriter(new File(path));  
 writr.write(new Integer(writerCount).toString());  
 writr.close();  
 System.*out*.printf(Thread.*currentThread*().getName() + " Writer writing");  
 System.*out*.printf(" Counter value: %d\n", writerCount);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

import java.util.Random;  
  
public class sample {  
  
 public final static int *READ\_MAX* = 20;//maximum number of readers  
 public final static int *WRITE\_MAX* = 3;//maximum number of writers  
 public static question6 *readerWriterthread* = new question6(); //instantiate reader writer thread  
 public static Random *random* = new Random();//create random variable  
  
 static class readerThread extends Thread {  
 public void run() {  
 System.*out*.println("Reader :" + getName() + ": Start");  
 while (true) {  
 try {  
 *readerWriterthread*.reader();  
 int time = *random*.nextInt(3000);  
 Thread.*sleep*(time);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
 static class writerThread extends Thread {  
 public void run() {  
 System.*out*.print("Writer :" + getName() + ": Start");  
 while (true) {  
 try {  
 *readerWriterthread*.writer();  
 Thread.*sleep*(*random*.nextInt(3000));  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 readerThread readerThreads[] = new readerThread[*READ\_MAX*];  
 writerThread writerThreads[] = new writerThread[*WRITE\_MAX*];  
 System.*out*.println("start");  
 for (int i = 0; i < *WRITE\_MAX*; ++i) {  
 writerThreads[i] = new writerThread();  
 writerThreads[i].start();  
 }  
 for (int i = 0; i < *READ\_MAX*; ++i) {  
 readerThreads[i] = new readerThread();  
 readerThreads[i].start();  
 }  
 }  
}

start

Reader :Thread-8: Start

Writer :Thread-2: StartReader :Thread-7: Start

Reader :Thread-12: Start

Reader :Thread-6: Start

Reader :Thread-15: Start

Reader :Thread-13: Start

Reader :Thread-5: Start

Writer :Thread-1: StartReader :Thread-16: Start

Reader :Thread-21: Start

Reader :Thread-22: Start

Reader :Thread-3: Start

Reader :Thread-20: Start

Reader :Thread-9: Start

Reader :Thread-17: Start

Reader :Thread-11: Start

Reader :Thread-14: Start

Reader :Thread-18: Start

Reader :Thread-10: Start

Reader :Thread-19: Start

Reader :Thread-4: Start

Writer :Thread-0: StartThread-8 Reader reading Counter value: 0

Thread-2 Writer writing Counter value: 1

Thread-1 Writer writing Counter value: 2

Thread-0 Writer writing Counter value: 3

Thread-5 Reader reading Counter value: 3

Thread-13 Reader reading Counter value: 3

Thread-6 Reader reading Counter value: 3

Thread-15 Reader reading Counter value: 3

Thread-16 Reader reading Counter value: 3

Thread-21 Reader reading Counter value: 3

Thread-22 Reader reading Counter value: 3

Thread-3 Reader reading Counter value: 3

Thread-9 Reader reading Counter value: 3

Thread-11 Reader reading Counter value: 3

Thread-12 Reader reading Counter value: 3

Thread-14 Reader reading Counter value: 3

Thread-17 Reader readingThread-18 Reader reading Counter value: 3

Thread-7 Reader reading Counter value: 3

Thread-10 Reader reading Counter value: 3

Thread-19 Reader reading Counter value: 3

Counter value: 3

Thread-20 Reader reading Counter value: 3

Thread-4 Reader reading Counter value: 3

Thread-10 Reader reading Counter value: 3

Thread-5 Reader reading Counter value: 3

Thread-0 Writer writing Counter value: 4

Thread-18 Reader reading Counter value: 4

Thread-16 Reader reading Counter value: 4

Thread-9 Reader reading Counter value: 4

Thread-4 Reader reading Counter value: 4

Thread-22 Reader reading Counter value: 4

Thread-10 Reader reading Counter value: 4

Thread-15 Reader reading Counter value: 4

Thread-19 Reader reading Counter value: 4

Thread-11 Reader reading Counter value: 4

Thread-20 Reader reading Counter value: 4

Thread-9 Reader reading Counter value: 4

Thread-20 Reader reading Counter value: 4

Thread-21 Reader reading Counter value: 4

Thread-13 Reader reading Counter value: 4

Thread-18 Reader reading Counter value: 4

Thread-7 Reader reading Counter value: 4

Thread-9 Reader reading Counter value: 4

Thread-4 Reader reading Counter value: 4

Thread-18 Reader reading Counter value: 4

Thread-5 Reader reading Counter value: 4

Thread-1 Writer writing Counter value: 5

Thread-19 Reader reading Counter value: 5

Thread-8 Reader reading Counter value: 5

Thread-10 Reader reading Counter value: 5

Thread-14 Reader reading Counter value: 5

Thread-5 Reader reading Counter value: 5

Thread-0 Writer writing Counter value: 6

Thread-2 Writer writing Counter value: 7

Thread-10 Reader reading Counter value: 7

Thread-6 Reader reading Counter value: 7

Thread-3 Reader reading Counter value: 7

Thread-12 Reader reading Counter value: 7

Thread-3 Reader reading Counter value: 7

Thread-7 Reader reading Counter value: 7

Thread-14 Reader reading Counter value: 7

Thread-15 Reader reading Counter value: 7

Thread-21 Reader reading Counter value: 7

Thread-14 Reader reading Counter value: 7

Thread-17 Reader reading Counter value: 7

Thread-8 Reader reading Counter value: 7

Thread-2 Writer writing Counter value: 8

Thread-17 Reader reading Counter value: 8

Thread-7 Reader reading Counter value: 8

Thread-18 Reader reading Counter value: 8

Thread-16 Reader reading Counter value: 8

Thread-22 Reader reading Counter value: 8

Thread-11 Reader reading Counter value: 8

Thread-3 Reader reading Counter value: 8

Thread-14 Reader reading Counter value: 8

Thread-4 Reader reading Counter value: 8

Thread-1 Writer writing Counter value: 9

Thread-15 Reader reading Counter value: 9

Thread-17 Reader reading Counter value: 9

Thread-8 Reader reading Counter value: 9

Thread-6 Reader reading Counter value: 9

Thread-13 Reader reading Counter value: 9

Thread-0 Writer writing Counter value: 10

Thread-19 Reader reading Counter value: 10

Thread-13 Reader reading Counter value: 10

Thread-20 Reader reading Counter value: 10

Thread-7 Reader reading Counter value: 10

3. Consider a chain of processes P1, P2, ..., Pn implementing a multitiered client-server architecture. Process Pi is client of process Pi+1, and Pi will return a reply to Pi-1 only after receiving a reply from Pi+1. What are the main problems with this organization when taking a look at the request-reply performance at process P1?

* P1 needs to wait on pi to finish all its processes (waiting until the last one). It then waits for all processes to give a reply in the reverse order. If it so happens that a process gets stuck. Then the entire process then fails.
* Also note that when one machine in the chain performs badly, the whole entire organization is affected with bad performance. If N is “Large” then the performance will not be reliable.

4. Show the B-trees of order four resulted from loading the following sets of keys (each letter is a key) in order:

1. C G J X
2. C G J X N S U O A E B H I
3. C G J X N S U O A E B H I F
4. C G J X N S U O A E B H I F K L Q R T V U W Z

5. Given a B-tree of order 256,

1. what is the maximum number of children from a node?
2. excluding the root and the leaves, what is the minimum number of children from a node?
3. what is the minimum number of children from the root?
4. What is the maximum depth of the tree if it contains 100 000 keys?

6. Construct a general resource graph for the following scenario and determine if the graph is completely reducible: R1, R2, and R3 are reusable resources with a total of two, two, and three units. Process P1 is allocated one unit each of R2 and R3 and is requesting one unit of R1. Process P2 is allocated one unit of R1 and is requesting two units of R3. Process P3 is allocated one unit each of R1 and R2 and is requesting one unit of R3.

