Bradley Aiken

Professor Cesario

CS 438

5 May 2020

Final Exam

Question 1-A

The output for this program is not deterministic. This is because only the thread t1 is joined, so t2 and the main thread can continue running at unpredictable rates after t1 is finished, and t2 can run before t1 is joined.

The possible results are as follows:

```
X X Y Y MAIN

X X Y MAIN Y

X X MAIN Y Y

Y Y X X MAIN

Y X X Y MAIN

X Y Y X MAIN

X Y X Y MAIN

Y X Y X MAIN
```

Question 2-A

```java
package exam_final;

import java.util.concurrent.Semaphore;

public class Problem2 {

    private static Semaphore semaphoreA = new Semaphore(1);
    private static Semaphore semaphoreB = new Semaphore(0);
    private static Semaphore semaphoreC = new Semaphore(0);
    private static Semaphore semaphoreD = new Semaphore(0);

    private static int countB = 0;
    private static int countD = 0;
    public static void main(String[] args) throws
InterruptedException {

        while (true) {
            Thread a = new A();
            Thread b = new B();
            Thread c = new C();
            Thread d = new D();
            a.start();
            b.start();
            c.start();
            d.start();
            Thread.sleep(1000);
        }

    }

    static class A extends Thread {
        public void run() {
            try {
                semaphoreA.acquire();
            } catch (Exception e) { e.printStackTrace(); }
            System.out.print("A ");
            semaphoreB.release(1);
        }
    }

    static class B extends Thread {
        public void run() {
            try {
                semaphoreB.acquire();
            } catch (Exception e) { e.printStackTrace(); }
            System.out.print("B ");
            countB += 1;
            if (countB == 2) {
                countB = 0;
                semaphoreC.release(1);
            }
            else {
                semaphoreB.release(1);
            }
        }
    }

    static class C extends Thread {
        public void run() {
            try {
                semaphoreC.acquire();
            } catch (Exception e) { e.printStackTrace(); }
            System.out.print("C ");
            semaphoreD.release(1);
        }
    }

    static class D extends Thread {
        public void run() {
            try {
                semaphoreD.acquire();
            } catch (Exception e) { e.printStackTrace(); }
            System.out.print("D ");
            countD += 1;
            if (countD == 2) {
                countD = 0;
                System.out.print(" ");
                semaphoreA.release(1);
            }
            else {
                semaphoreD.release(1);
            }
        }
    }

}
```

Question 3-A

File class

```
package exam_final.problem3;

public class File {

    private static int count = 0;

    private int content;
    public File(boolean exists) {
        if (exists) {
            content = count;
            count += 1;
        }
        else {
            this.content = -1;
        }
    }

    public int getContent() {
        return content;
    }

}
```

Network class

```
package exam_final.problem3;

public abstract class Network {

    protected File[] buffer;
    protected int in = 0;
    protected int out = 0;

    public Network(int size) {
        buffer = new File[size];
        for (int i = 0; i < buffer.length; i++) {
            buffer[i] = new File(false);
        }
    }

    public abstract void send(File file) throws
InterruptedException;

    public abstract File receive() throws InterruptedException;

    public String getContent() {
        String string = "[ ";
        for (int i = 0; i < buffer.length; i++) {
            string += buffer[i].getContent() + " ";
        }
        string += "]";
        return string;
    }

}
```

NetworkSemaphore class

```
package exam_final.problem3;

import java.util.concurrent.Semaphore;

public class NetworkSemaphore extends Network {

    private Semaphore mutex;
    private Semaphore full;
    private Semaphore empty;

    public NetworkSemaphore(int size) {
        super(size);
        mutex = new Semaphore(1);
        full = new Semaphore(0);
        empty = new Semaphore(size);
    }

    public void send(File file) throws InterruptedException {
        empty.acquire();
        mutex.acquire();
        buffer[in] = file;
        in = (in + 1) % buffer.length;
        mutex.release();
        System.out.println("ServerA has sent file with content "
                + file.getContent() + ". " + getContent());
        full.release();
    }

    public File receive() throws InterruptedException {
        full.acquire();
        mutex.acquire();
        File file = buffer[out];
        buffer[out] = new File(false);
        out = (out + 1) % buffer.length;
        mutex.release();
        System.out.println("ServerB has received file with
content "
                + file.getContent() + ". " + getContent());
        empty.release();
        return file;
    }

}
```

ServerA class

```
package exam_final.problem3;

import java.util.Random;

public class ServerA extends Thread {

    private static int MAX_RANDOM = 100;
    private static int MIN_PRODUCING_TIME = 30;
    private static int MAX_PRODUCING_TIME = 50;

    private Random random = new Random();
    private Network boundedBuffer;

    public ServerA(String name, Network boundedBuffer) {
        super(name);
        this.boundedBuffer = boundedBuffer;
    }

    public void run() {
        try {
            while (true) {
                File file = produceFile();
                boundedBuffer.send(file);
            }
        } catch (Exception e) { e.printStackTrace(); }
    }

    private File produceFile() throws InterruptedException {
        randomWait(MIN_PRODUCING_TIME, MAX_PRODUCING_TIME);
        File file = new File(true);
        return file;
    }

    private void randomWait(int minTime, int maxTime) throws
InterruptedException {
        Thread.sleep(random.nextInt(maxTime - minTime + 1) +
minTime);
    }

}
```

ServerB class

```
package exam_final.problem3;

import java.util.Random;

public class ServerB extends Thread {

    private static int MIN_CONSUMING_TIME = 30;
    private static int MAX_CONSUMING_TIME = 50;

    private Random random = new Random();
    private Network network;

    public ServerB(String name, Network network) {
        super(name);
        this.network = network;
    }

    public void run() {
        try {
            while (true) {
                File file = network.receive();
                consumeFile(file);
            }
        } catch (Exception e) { e.printStackTrace(); }
    }

    private void consumeFile(File file) throws
InterruptedException {
        randomWait(MIN_CONSUMING_TIME, MAX_CONSUMING_TIME);
    }
    private void randomWait(int minTime, int maxTime) throws
InterruptedException {
        Thread.sleep(random.nextInt(maxTime - minTime + 1) +
minTime);
    }

}
```

MultiServerTest class

```
package exam_final.problem3;

public class MultiServerTest {

    public static void main(String[] args) {

        int size = 3;

        Network network = new NetworkSemaphore(size);

        ServerA serverA = new ServerA(Integer.toString(1),
network);
        ServerB serverB = new ServerB(Integer.toString(1),
network);

        serverA.start();
        serverB.start();

    }

}
```