Memoria Bloc 1 AST

Daniel Vilardell

Índex

1 Practica 1

1.1 Exercici 1

Primer de tot hem hagut d'implementar la cua circular, i comprobar que funciona. Aquí nomes mostrarem el codi de la implementació de la cua, el codi per a comprovar que funciona esta dins del fitxer .zip adjuntat a la entrega. Cal apuntar que molts metodes estan posats de forma compacta per a que la memoria no s'estengues massa, el el fitxer .zip no estan en una linea, sinó en varies i es mes llegible.

Els comentaris per tal d'entendre la implementació estan posats dins del mateix codi mostrat a continuació.

Implementació de la cua circular

```
package practica1.CircularQ;

import java.util.Iterator;
import utils.Queue;

public class CircularQueue<E> implements Queue<E> {

    private final E[] queue;
    private final int N;
    private int elements;
    private int first;
    private int last;

    public CircularQueue(int N) {
      //Tot i que no es necessari assignem elements, first i last a 0 per claretat
        this.N = N;
        this.elements = 0;
        this.first = 0;
        this.last = 0;
        queue = (E[]) (new Object[N]);
    }

    @Override
    public int size() {return elements;}

    @Override
    public int free() {return N - elements;}

    @Override
    public boolean hasFree(int n) {return N != elements;}

    @Override
    public boolean empty() {return elements == 0;} //Si esta buida conte 0 elements

    @Override
    public boolean full() {return N == elements;}

    @Override
    public E peekFirst() {
      //Cal mirar primer que no estigui buida, al igual que al metode get() i peekLast()
        if(empty()) throw new IllegalStateException("cua buida");
        return queue[first];
    }

    @Override
    public E peekLast() {
        if(empty()) throw new IllegalStateException("cua buida");
        return queue[last];
    }
```

```
    @Override
    public E get() {
        if(empty()) throw new IllegalStateException("cua buida");
        E e = queue[first];
        first = (N + first + 1)%N;
        elements--;
        return e;
    }

    @Override
    public void put(E e) {
      //Cal mirar que no estigui plena
        if(full()) throw new IllegalStateException("cua plena");
        queue[last] = e;
        last = (N + last + 1)%N;
        elements++;
    }

    @Override
    public Iterator<E> iterator() {return new MyIterator();}

    class MyIterator implements Iterator {

        @Override
        public boolean hasNext() {return !empty();}

        @Override
        public E next() {
            E e = get();
            return e;
        }

        @Override
        public void remove() {throw new UnsupportedOperationException("Not supported yet.");}

    }
}
```

1.2 Exercici 2

Un cop hem implementat la cua circular sens proposa implementar una Linked queue, usant una classe auxiliar donada Node. Altre vegada nomes es mostrarà el codi de la implementació. Per a veure el main s'ha de descomprimir el fitxer .zip.

Implementació de la cua circular

```
package practica1.LinkedQ;

import java.util.Iterator;
import utils.Queue;

public class LinkedQueue<E> implements Queue<E> {

    private final int N;
    private Node<E> first;
    private Node<E> last;
    private int size;

    public LinkedQueue(int N) {
        //Inicialment posem first i last a null
        this.N = N;
        this.first = null;
        this.last = null;
        this.size = 0;
```

```
    }

    @Override
    public int size() {return size;}

    @Override
    public int free() {return N - size;}

    @Override
    public boolean hasFree(int n) {return size != N;}

    @Override
    public boolean empty() {return size == 0;}

    @Override
    public boolean full() {return size == N;}

    @Override
    public E peekFirst() {
        //Mirem que no estigui buida
        if(empty()) throw new IllegalStateException("cua buida");
        return first.getValue();
    }

    @Override
    public E peekLast() {
        //Mirem que no estigui buida
        if(empty()) throw new IllegalStateException("cua buida");
        return last.getValue();
    }

    @Override
    public E get() {
        //Mirem que no estigui buida
        if(empty()) throw new IllegalStateException("cua buida");
        E e = first.getValue();
        if(size != 1) { first = first.getNext();}
        size--;
        return e;
    }

    @Override
    public void put(E value) {
        //Mirem que no estigui plena
        if(full()) throw new IllegalStateException("cua plena");
        Node n = new Node(value, null);
        //Si esta buida fem assignacions pertinents
        if(empty()) {
            first = n; last = n; size++;
            return;
        }
        size++;
        last.setNext(n);
        last = n;
    }

    @Override
    public Iterator<E> iterator() {return new MyIterator();}

    class MyIterator implements Iterator {

        @Override
        public boolean hasNext() {return !empty();}

        @Override
        public E next() {
            E e = get();
            return e;
        }

        @Override
        public void remove() {
            throw new UnsupportedOperationException("Not supported yet.");
        }

    }
}
```

1.3 Exercici 3

La implementació de QueueChannel es simple ja que els seus metodes son basicament crides a metodes de la classe CircularQueue, a partir del seu atribut.

Implementació de QueueChannel

```
package practica1.Protocol;

import ast.protocols.tcp.TCPSegment;
import practica1.CircularQ.CircularQueue;
import utils.Channel;

public class QueueChannel implements Channel {

    CircularQueue<TCPSegment> Q;

    public QueueChannel(int N) {
        Q = new CircularQueue<TCPSegment>(N);
    }

    @Override
    public void send(TCPSegment s) throws IllegalStateException {
        //Si la cua esta plena tirara la excepcio
        Q.put(s);
    }

    @Override
    public TCPSegment receive() throws IllegalStateException {
        //Si la cua esta buida tirara la excepcio
        return Q.get();
    }

    @Override
    public int getMMS() {
        throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

1.4 Exercici 4

Finalment implementem les classes TSocketSend i TSocketRecv que tenen com a finalitat insertar i extreure elements del canal a partir dels metodes sendData i receiveData.

Implementació de TSocketSend

```
package practica1.Protocol;

import ast.protocols.tcp.TCPSegment;
import utils.Channel;

public class TSocketSend {

    private final Channel channel;

    public TSocketSend(Channel channel) {
        this.channel = channel;
    }

    public void sendData(byte[] data, int offset, int length) {
        byte[] valid_data = new byte[length];
```

```
        System.arraycopy(data, offset, valid_data, 0, length);
        TCPSegment seg = new TCPSegment();
        seg.setData(valid_data);
        channel.send(seg);
    }
}
```

Implementació de TSocketRecv

```
package practica1.Protocol;

import ast.protocols.tcp.TCPSegment;
import utils.Channel;

public class TSocketRecv {

    private final Channel channel;

    public TSocketRecv(Channel channel) {
        this.channel = channel;
    }

    //Retorna la quantitat de dades que s'ha pogut extreure, min(les que hi ha, les demanades)
    public int receiveData(byte[] data, int offset, int length) {
        TCPSegment receive = channel.receive();
        System.arraycopy(receive.getData(), receive.getDataOffset(), data, offset,
                Math.min(receive.getDataLength(), length));
        return Math.min(receive.getDataLength(), length);
    }
}
```

2 Practica 2

2.1 Exercici 1 i 2

En aquest primer exercici sens mostra un dels problemes de corre diferents threads alhora, a partir de cridar una funció incrementador de una variable static, cosa que comporta que si s'incrementa desde els 2 threads alhora, nomes s'incrementa en 1, i no en 2. Al exercici 3 sens proposa picar una solució al problema.

Implementació de CounterThread

```
package practica2.P0CZ;

public class CounterThread extends Thread {

    public static int x;
    private final int I = 100;

    @Override
    public void run() {
        int R;
        for (int i = 0; i < I; i++) {
            R = x;
            try{sleep(1);} catch(InterruptedException ex) {}
            R = R + 1;
            x = R;
        }
        /*for (int i = 0; i < I; i++) {
            x++;
        }*/
    }
}
```

2.2 Exercici 3

En aquest exercici busquem una solució al problema esmentat, a partir de usar una classe MonitorCZ que treballa amb metodes en exclusió mutua, gracies al atribut de tipus lock. Per aquesta part sens ha fet picar el metode inc().

Implementació de MonitorCZ

```
package practica2.P0CZ.Monitor;

import java.util.concurrent.locks.ReentrantLock;

public class MonitorCZ {

    private int x = 0;
    ReentrantLock lock = new ReentrantLock();

    public void inc() {
        lock.lock();
        try {
```

```
            x += 1;
        } finally {
            lock.unlock();
        }
    }

    public int getX() {
        lock.lock();
        try {
            return x;
        } finally {
            lock.unlock();
        }
    }

}
```

Un cop tenim aquesta classe, els increments no els farem ja desde CounterThreadCZ, sinó que cridarem la funció increment de dins de monitorCZ. El programa amb el main tindra un atribut de tipus MonitorCZ compartit entre els dos incrementadors.

2.3 Exercici 4

Aquest exercici ens demana que proposem una implementació de la classe Monitor amb el fí que les ordres s'executin amb un ordre en concret. Per això assignarem un ordre de torns d'execució de 0 fins a N on N es el nombre de fils que es corren alhora. Un cop s'arribi a N es tornara a començar a 0. Per a que dos fils amb possible igual identificació no entrin al seu torn alhora la espera es farà en exclusió mutua.

Implementació de MonitorSync

```
package practica2.P1Sync.Monitor;

import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.ReentrantLock;

public class MonitorSync {

    ReentrantLock lock = new ReentrantLock();
    private final int N;
    private static int turn = 0;

    public MonitorSync(int N) {
        this.N = N;
    }

    public void waitForTurn(int id) {
        lock.lock();
        while(this.turn != id){lock.unlock(); lock.lock();}
        lock.unlock();
    }

    public void transferTurn() {
        turn = (turn + 1)%N;
    }
}
```

2.4 Exercici 5 i 6

Finalment implementarem un sistema de comunicació emisor receptor. El que sens dona fet te un problema, que el receptor intenta treure dades de una cua buida o el receptor intenta emplenarne una de plena. Aixi que haurem d'afegir una espera mentres la cua sigui buida per al receptor i una espera mentres sigui plena per al emisor. Aixi doncs la implementació de la classe MonitorChannel fa el comentat amb un while.

Implementació de MonitorChannel

```
package practica2.Protocol;

import ast.protocols.tcp.TCPSegment;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import java.util.logging.Level;
import java.util.logging.Logger;
import practica1.CircularQ.CircularQueue;
import utils.Channel;

public class MonitorChannel implements Channel {

    CircularQueue<TCPSegment> Q;
    Lock lock = new ReentrantLock();
    Condition full  = lock.newCondition();
    Condition empty = lock.newCondition();

    public MonitorChannel(int N) {
        Q = new CircularQueue<TCPSegment>(N);
    }

    @Override
    public void send(TCPSegment seg) {
        lock.lock();
        try {
            while(Q.full()){
                try {
                    full.await();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
            empty.signal();
            Q.put(seg);
        }
        finally {
            lock.unlock();
        }


    }

    @Override
    public TCPSegment receive() {
        lock.lock();
        TCPSegment res = new TCPSegment();
        try {
            while(Q.empty()){
                try {
                    empty.await();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
            res  = Q.get();
            full.signal();
```

```
        }
        finally {
            lock.unlock();
        }

        return res;
    }

    @Override
    public int getMMS() {
        throw new UnsupportedOperationException("Not supported yet.");
    }

}
```