



Departamento de Programación
Facultad de Informática
Universidad Nacional del Comahue



Programación Concurrente



Sincronización – Cerrojos

Sincronización competencia y cooperación

Mecanismos

```
graph TD; A[Mecanismos] --> B[Monitores y esperas guardadas]; A --> C[Semáforos binarios y generales]; A --> D[Cerros + Variables de condición]; A --> E[Otras posibilidades ...];
```

Monitores y esperas guardadas

Semáforos binarios y generales

Cerros + Variables de condición

Otras posibilidades ...



Interfaz Condition

También conocido como
colas de condición o variables de condición

- Hacen que un hilo suspenda la ejecución hasta que otro hilo avise alguna condición de estado como cierta.
- El acceso al objeto protegido se produce en diferentes lugares
- El bloqueo está asociado a la Condición
- El hilo que debe esperar por una condición libera atómicamente el bloqueo asociado y se suspende


Variables de condición

asociadas a un reentrantLock

Operaciones

(throws InterruptedException)

- void ***await()***
- boolean ***await(long time, TimeUnit unit)***
- void ***signal()*** *//Despierta un hilo*
- void ***signalAll()*** *//Despierta todos los hilos*




Problema Prod/Cons -- objetos *Condition*

- Cuando se utiliza un Lock explícito para definir una **región crítica**, dentro de ella se utilizan los objetos **Condition** como mecanismo de sincronización de cooperación entre hilos.

```
class BufferDatos {  
    Lock lock = new ReentrantLock(true);  
    Condition noLleno = lock.newCondition();  
    Condition noVacio = lock.newCondition();
```

- Un objeto Condition está estructuralmente ligado a un objeto Lock
 - Se crea mediante **newCondition()** sobre un objeto Lock.
- El objeto Condition solo puede ser invocado por un hilo que previamente haya tomado el Lock al que pertenece.



Problema Prod/Cons -- objetos *Condition*

```
class BufferDatos {  
    Lock lock = new ReentrantLock(true);  
    Condition noLleno = lock.newCondition();  
    Condition noVacio = lock.newCondition();  
}
```

Los hilos productores quedarán bloqueados en espera de lugar en la cola de espera de la variable de condición “noLleno”

Los hilos consumidores quedarán bloqueados en espera de items para consumir en la cola de espera de la variable de condición “noVacio”



```
import java.util.concurrent.locks.*;

class buffer_limitado {

    final Lock cerrojo = new ReentrantLock(); //declara y crea un cerrojo

    // variables de condicion asociadas a "cerrojo" para control de buffer lleno y vacio
    final Condition noLleno = cerrojo.newCondition();
    final Condition noVacio = cerrojo.newCondition();

    //estructura elegida para mantener los items de datos
    final Object[] items = new Object[100];

    ... //otras variables necesarias

    public void poner(Object x) throws InterruptedException {
        cerrojo.lock();

        ...

    }
    ...
}
```

```
import java.util.concurrent.locks.*;
```

```
class buffer_limitado {
```

```
    final Lock cerrojo = new ReentrantLock();
```

```
    final Condition noLlena = cerrojo.newCondition();
```

```
    final Condition noVacia = cerrojo.newCondition();
```

```
    final Object[] items = new Object[10];
```

```
    public void poner (Object x) throws InterruptedException {
```

```
        cerrojo.lock();
```

```
        .....
```

```
        cerrojo.lock();
```

```
        //duerme al hilo productor si buffer lleno hasta  
        //ser notificado
```

```
        try {while (lleno(items))
```

```
            noLlena.await();
```

```
        }
```

```
        // se agrega el item x a la estructura
```

```
        //se despierta a un hilo consumidor esperando
```

```
            noVacia.signal();
```

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

```
    }
```




Clase ReentrantLock

(implementa interfaz Lock)

- Librerías que utiliza
 - `import java.util.concurrent.locks.Condition;`
 - `import java.util.concurrent.locks.Lock;`
 - `import java.util.concurrent.locks.ReentrantLock;`
- Creación de objetos ReentrantLock
 - `l = new ReentrantLock(true);` //con política equitativa
 - `l = new ReentrantLock(false);` //con política aleatoria