L11: Semaphores II

Under construction! Do **not** print

César Sánchez

Grado en Ingeniería Informática
Grado en Matemáticas e Informática
Universidad Politécnica de Madrid

Wed, 11-March-2015

Este texto se distribuye bajo los términos de la Creative Commons License

Mapa Conceptual

$$\text{Concurrency} = \text{Simultaneous} + \text{Nondeterminism} + \text{Interaction}$$

$$\text{Interaction} = \text{Communication} \mid \text{Synchronization}$$

$$\text{Synchronization} = \text{Mutual Exclusion} \mid \text{Conditional Synchronization}$$

- Terminology:

atomic	race condition
interleaving	busy-wait
mutual exclusion	critical section
deadlock	livelock
liveness	semaphores

TODAY: HW4: Garantizar mutex con semáforos

Homework:

- HW1: Creación de threads en Java
- HW2: Provocar una condición de carrera
- HW3: Garanatizar la exclusión mutua con espera activa
- HW4: Garantizar la exlusión mutua con semáforos

Fecha de Cierre:

Miércoles 11-Marzo-2015 11am

Entrega online:

`http://lml.ls.fi.upm.es/~entrega`

TODAY: HW4: Garantizar mutex con semáforos

Homework:

- HW1: Creación de threads en Java
- HW2: Provocar una condición de carrera
- HW3: Garanatizar la exclusión mutua con espera activa
- HW4: Garantizar la exlusión mutua con semáforos

Fecha de Cierre:

Miércoles 11

HOY: PUESTA EN COMÚN

Entrega on

`http ml.ls.fi.upm.es/~entrega`

HW5 + HW6

Homework:

- HW1: Creación de threads en Java
- HW2: Provocar una condición de carrera
- HW3: Garanatizar la exclusión mutua con espera activa
- HW4: Garantizar la exlusión mutua con semáforos
- HW5: Almacén de un dato con semáforos
- HW6: Almacén de varios datos con semáforos

Fecha de Cierre:

Lunes 23-Marzo-2015 23:59

Entrega online:

`http://lml.ls.fi.upm.es/~entrega`

HW5: Almacén de un dato con semáforos

5. Almacén de un dato con semáforos

En este caso nos enfrentamos a un típico programa de concurrencia: productores-consumidores. Existen procesos de dos tipos diferentes:

- Productores: su hilo de ejecución consiste, repetidamente, en crear un producto (ver la clase `es.upm.babel.cclib.Producto`) y hacerlo llegar a uno de los consumidores.
- Consumidores: su hilo de ejecución consiste, repetidamente, recoger productos producidos por los productores y consumirlos.

Las clases que implementan ambos threads forman parte de la librería CCLib: `es.upm.babel.cclib.Productor` y `es.upm.babel.cclib.Consumidor`.

La comunicación entre productores y consumidores se realizará a través de un "almacén" compartido por todos los procesos. Dicho objeto respetará la interfaz `es.upm.babel.cclib.Almacen`:

Se pide implementar sólo con semáforos una clase que siga dicha interfaz. Sólo puede haber almacenado como máximo un producto, si un proceso quiere almacenar debe esperar hasta que no haya un producto y si un proceso quiera extraer espere hasta que haya un producto. Téngase en cuenta además los posible problemas de no asegurar la exclusión mutua en el acceso a los atributos compartidos.

Material a entregar

El fichero fuente a entregar debe llamarse `Almacen1.java`.

HW5: Almacén de un dato con semáforos (2)

Material de apoyo

Se ofrece un esqueleto de código que el alumno debe completar en el fichero `Almacen1.todo.java`:

El programa principal `CC_05_P1CSem.java` es el siguiente:

Para valorar si el problema está bien resuelto, os recordamos que el objetivo es asegurar

1. que todos los productos producidos acaban por ser consumidos,
2. que no se consume un producto dos veces y
3. que no se consume ningún producto no válido (null, por ejemplo).

Recomendación: jugar con los valores de número de productores y número de consumidores y observar con atención las trazas del programa.

HW6: Almacén de varios datos con semáforos

6. Almacén de varios datos con semáforos

Este ejercicio es una variación del problema anterior. En esta ocasión, el almacén a implementar tiene una capacidad de hasta N productos, lo que permite a los productores seguir trabajando aunque los consumidores se vuelvan, momentáneamente, lentos.

Material a entregar

El fichero fuente a entregar debe llamarse `AlmacenN.java`.

Material de apoyo

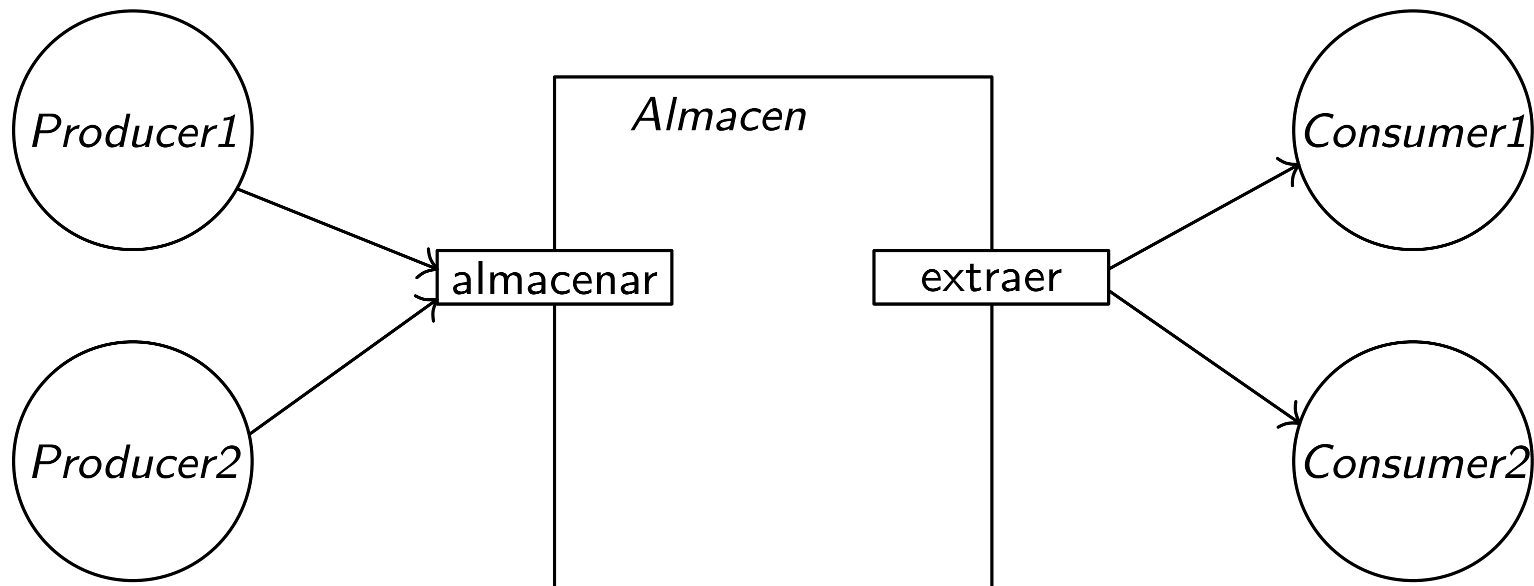
Se ofrece un esqueleto de código que el alumno debe completar en el fichero `AlmacenN.todo.java`:

Producers-Consumers Problem

- Producers: processes that create new data
- Consumers: threads that process and consume data
- Store: shared placeholder for data

Producers-Consumers Problem

- Producers: processes that create new data
- Consumers: threads that process and consume data
- Store: shared placeholder for data



CC_05_P1CSem.java

```java
class CC_05_P1CSem {
    public static final void main(final String[] args) throws
        InterruptedException {

        Almacen almac = new Almacen1();

        Productor[] productores = new Productor[N_PRODS];
        Consumidor[] consumidores = new Consumidor[N_CONSS];

        for (int i = 0; i < N_PRODS; i++) {
            productores[i] = new Productor(almac);
        }
        for (int i = 0; i < N_CONSS; i++) {
            consumidores[i] = new Consumidor(almac);
        }
        for (int i = 0; i < N_PRODS; i++) {
            productores[i].start();
        }
        for (int i = 0; i < N_CONSS; i++) {
            consumidores[i].start();
        }
        //...
```

Almacen1.java

```
class Almacen1 implements Almacen {
   public Almacen1() {}

   public void almacenar(Producto producto) {
      almacenado = producto;
  }

   public Producto extraer() {
      Producto result;
      result = almacenado;
      almacenado = null;
      return result;
   }
}
```

Productor.java

```
public class Productor extends Thread {

   private Almacen almacenCompartido;

   public Productor(Almacen a) {
      almacenCompartido = a;
   }

    public void run() {
       Producto p;
       while (true) {
          p = Fabrica.producir();
          almacenCompartido.almacenar(p);
      }
    }
}
```

Consumidor.java

```
public class Consumidor extends Thread {

   private Almacen almacenCompartido;

   public Consumidor(Almacen a) {
      almacenCompartido = a;
   }


    public void run() {
       Producto p;
       while (true) {
          p = almacenCompartido.extraer();
          Consumo.consumir(p);
       }
    }
}
```

Concurrency Problems

Recall **Almacen.java**:

```
class Almacen1 implements Almacen {
   public Almacen1() {}

   public void almacenar(Producto producto) {
      almacenado = producto;
  }

   public Producto extraer() {
      Producto result;
      result = almacenado;
      almacenado = null;
      return result;
   }
}
```

Concurrency Problems

Recall **Almacen.java**:

```
class Almacen1 implements Almacen {
   public Almacen1() {}

   public void almacenar(Producto producto) {
      almacenado = producto;
  }

   public Producto extraer() {
      Producto result;
      result = almacenado;
      almacenado = null;
      return result;
   }
}
```

Concurrency Problems

Recall **Almacen.java**:

```
class Almacen1 implements Almacen {
   public Almacen1() {}

   public void almacenar(Producto producto) {
      almacenado = producto;
  }

   public Producto extraer() {
      Producto result;
      result = almacenado;
      almacenado = null;
      return result;
   }
}
```

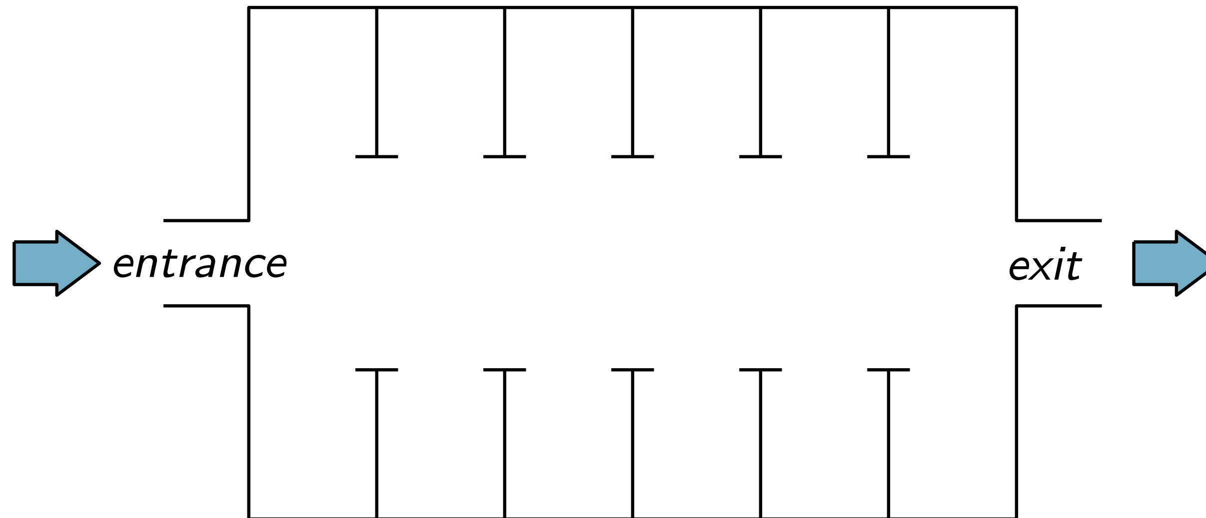
Block when full

2 conditions

Block when empty

Example: Parking Lot

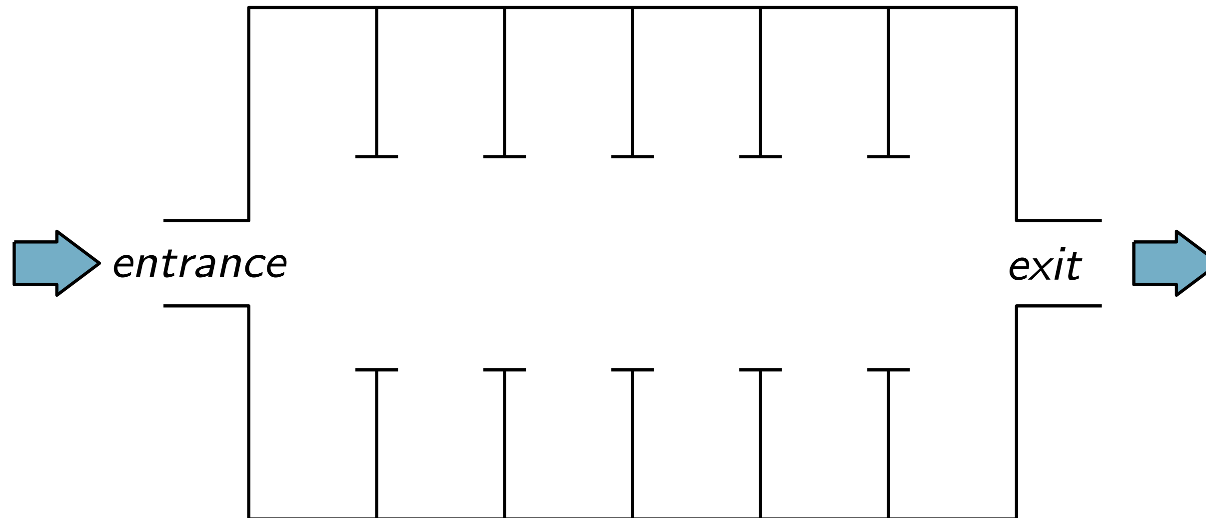
- One entry, one exit

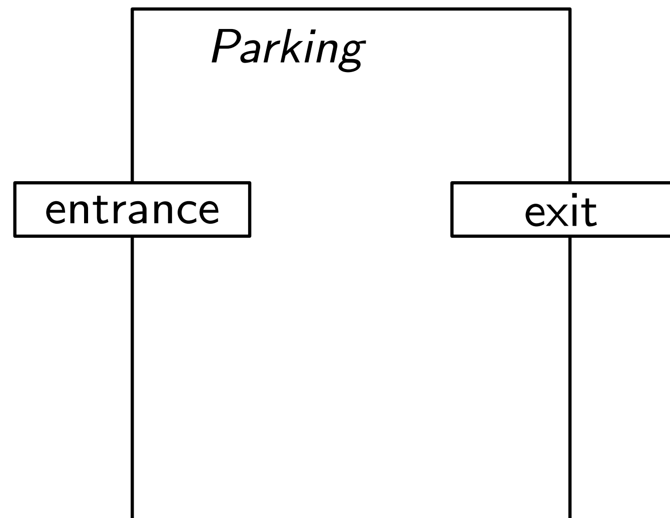


- Allow to enter if space available
- Discount upon exit

Example: Parking Lot

- One entry, one exit





Example: Parking Lot (Solution 0)

```
int num_cars;
```

```
public void exit() {
   num_cars--;
}
```

```
public void entrance() {
  bool space=false;
  while (!space) {
    if (num_cars < Max) {
       num_cars++;
       space = true;
    }
}
```

Example: Parking Lot (Solution 0)

```
int num_cars;
```

```
public void exit() {
   num_cars--;
}
```

```
public void entrance() {
  bool space=false;
  while (!space) {
    if (num_cars < Max) {
       num_cars++;
       space = true;
    }
}
```

Parking: Solution 1

```
int num_cars;
Semaphore sem(1);
```

```
public void exit() {
    sem.await();
   num_cars--;
   sem.signal();
}
```

```
public void entrance() {
  bool space=false;
  while (!space) {
    sem.await();
    if (num_cars < Max) {
       num_cars++;
       space = true;
    }
    sem.signal();
}
```

Parking: Solution 1

```
int num_cars;
Semaphore sem(1);
```

```
public void exit() {
    sem.await();
   num_cars--;
   sem.signal();
}
```

```
public void entrance() {
  bool space=false;
  while (!space) {
    sem.await();
    if (num_cars < Max) {
       num_cars++;
       space = true;
    }
    sem.signal();
}
```

Parking: Solution 2

```
int num_cars;
Semaphore sem(1);
Semaphore not_full(1)
```

```
public void exit() {
   sem.await();
   num_cars--;
   if (num_cars = Max -1) { not_full.signal(); }
   sem.signal();
}
```

```
public void entrance() {
  not_full.await();
  while (!space) {
    sem.await();
    num_cars++;
    if (num_cars < Max) {
        not_full.signal();
    }
    sem.signal();
}
```

Parking: Solution 2

```
int num_cars;
Semaphore sem(1);
Semaphore not_full(1)
```

```
public void exit() {
   sem.await();
   num_cars--;
   if (num_cars = Max -1) { not_full.signal(); }
   sem.signal();
}
```

```
public void entrance() {
  not_full.await();
  while (!space) {
    sem.await();
    num_cars++;
    if (num_cars < Max) {
        not_full.signal();
    }
    sem.signal();
}
```

Conditional Synchronization

Parking: Solution 2

```
int num_cars;
Semaphore sem(1);
Semaphore not_full(1)
```

```
public void exit() {
   sem.await();
   num_cars--;
   if (num_cars = Max -1) { not_full.signal(); }
   sem.signal();
}
```

```
public void entrance() {
  not_full.await();
  while (!space) {
    sem.await();
    num_cars++;
    if (num_cars < Max) {
        not_full.signal();
    }
    sem.signal();
}
```

Conditional Synchronization

`(not_full==1)`
if and only if
`num_cars<MAX`

Parking: Solution 3

```
int num_cars;
Semaphore spaces(Max);
```

```
public void exit() {
    spaces.signal();
}
```

Conditional Synchronization

```
public void entrance() {
   spaces.await();
}
```