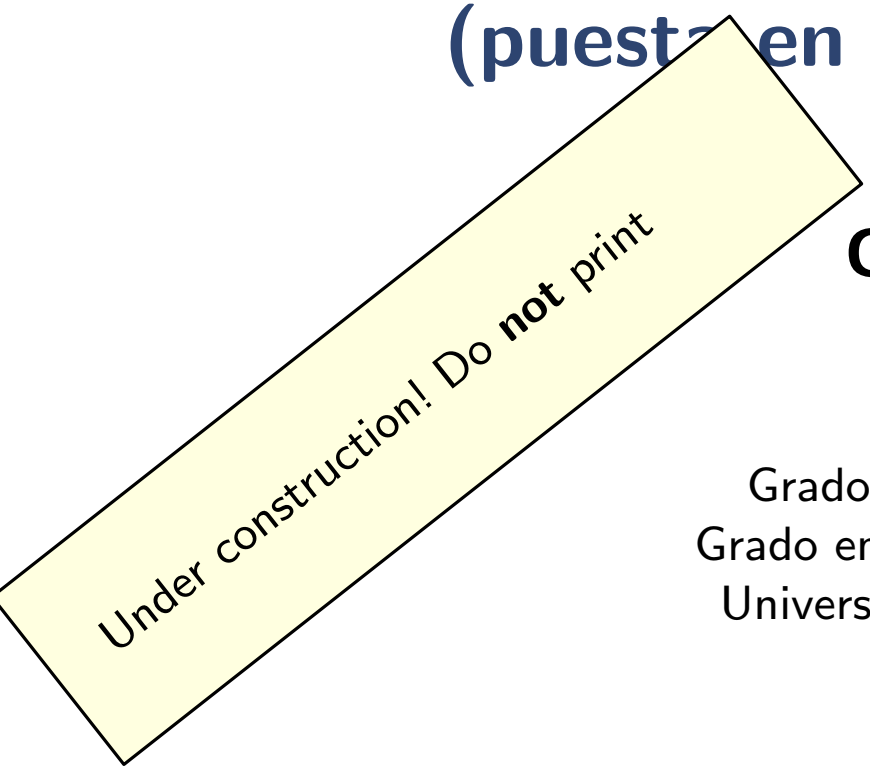
L13: Semaphores IV

(puesta en común HW5 y HW6)



César Sánchez

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

Wed, 25-March-2015

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`

Producers/Consumers with Unbounded Store

Use variable `products` to count the number of products:

`products` : *number of products*

- *Producers* increase `products` after producing
- *Consumer* decrease `products` before consuming
  (and block if 0)

Producers/Consumers with Unbounded Store

Use variable `products` to count the number of products:

`products` : *number of products*

- *Producers* increase `products` after producing
- *Consumer* decrease `products` before consuming (and block if 0)

How?

Producers/Consumers with Unbounded Store

Use variable `products` to count the number of products:

`products` : *number of products*

- *Producers* increase `products` after producing
- *Consumer* decrease `products` before consuming (and block if 0)

How?

Use a semaphore as counter!

Producers/Consumers with Unbounded Store

```
class AlmacenUnbounded implements Almacen {
//...
   class Cell {
      public Producto prod;
      public Cell next;
      Cell (Producto p, Cell n) {
         prod = p;
         next = n;
      }
   }
   private Cell head = null;
   private Cell tail = null;
//...
}
```

Producers/Consumers with Unbounded Store

```
class AlmacenUnbounded implements Almacen {
//...
   class Cell {/*... */}
```

Producers/Consumers with Unbounded Store

```
class AlmacenUnbounded implements Almacen {
//...
   class Cell {/*... */}
   public void almacenar(Producto producto) {

      Cell the_cell = new Cell(producto,null);
      if (head==null) { // INV: tail==null
         head = the_cell;
         tail = the_cell;
      } else {
         head.next = the_cell;
         head = head.next;
      }

   }
```

Producers/Consumers with Unbounded Store

```
class AlmacenUnbounded implements Almacen {
//...
   class Cell {/*... */}
   public void almacenar(Producto producto) {

      //inserts a new Cell(producto,null)


   }
```

Producers/Consumers with Unbounded Store

```
class AlmacenUnbounded implements Almacen {
//...
   class Cell {/*... */}
   public void almacenar(Producto producto) {

      //inserts a new Cell(producto,null)



   }
   public Producto extraer() {
      Producto result;

     result = tail.prod;
     if (tail == head) {
        tail = null; head = null;
     } else {
        tail = tail.next;
     }
```

Producers/Consumers with Unbounded Store

```
class AlmacenUnbounded implements Almacen {
//...
   class Cell {/*... */}
   public void almacenar(Producto producto) {

      //inserts a new Cell(producto,null)



   }
   public Producto extraer() {
      Producto result;



      // extract tail.prod

      return result;
   }
```

Producers/Consumers with Unbounded Store

```
class AlmacenUnbounded implements Almacen {
//...
   class Cell {/*... */}
   public void almacenar(Producto producto) {
      mutex.await();
      //inserts a new Cell(producto,null)
      mutex.signal();

   }
   public Producto extraer() {
      Producto result;

      mutex.await();
      // extract tail.prod
      mutex.signal();

      return result;
   }
```

Producers/Consumers with Unbounded Store

```
class AlmacenUnbounded implements Almacen {
//...
   class Cell {/*... */}
   public void almacenar(Producto producto) {
      mutex.await();
      //inserts a new Cell(producto,null)
      mutex.signal();
      products.signal();
   }
   public Producto extraer() {
      Producto result;
      products.await();
      mutex.await();
      // extract tail.prod
      mutex.signal();

      return result;
   }
```

Producers/Consumers with Almacen1

Producers/Consumers with Almacen1

Use variable `empty` to denote if store is empty

Use variable `full` to denote if store is full

- *Producers:*
  - decrease `empty` before producing (block if 0)
  - increase `full` after producing
- *Consumers*:
  - decrease `full` before consuming (block if 0)
  - increase `empty` after consuming

Producers/Consumers with Almacen1

Use variable `empty` to denote if store is empty

Use variable `full` to denote if store is full

- *Producers:*
  - decrease `empty` before producing (block if 0)
  - increase `full` after producing
- *Consumers*:
  - decrease `full` before consuming (block if 0)
  - increase `empty` after consuming

How?

Producers/Consumers with Almacen1

Use variable `empty` to denote if store is empty

Use variable `full` to denote if store is full

- *Producers:*
  - decrease `empty` before producing (block if 0)
  - increase `full` after producing
- *Consumers*:
  - decrease `full` before consuming (block if 0)
  - increase `empty` after consuming

How?

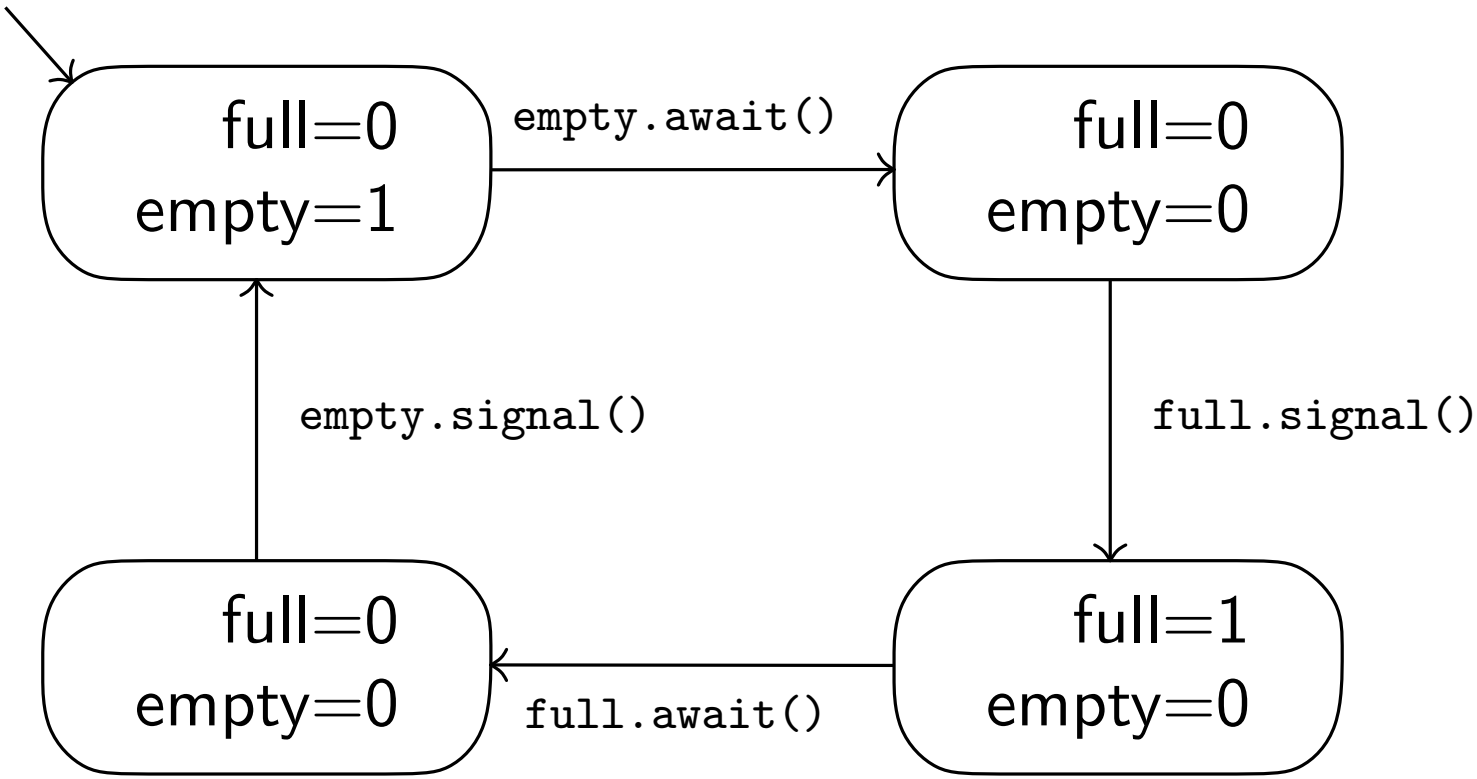
Use two semaphores!

Almacen1

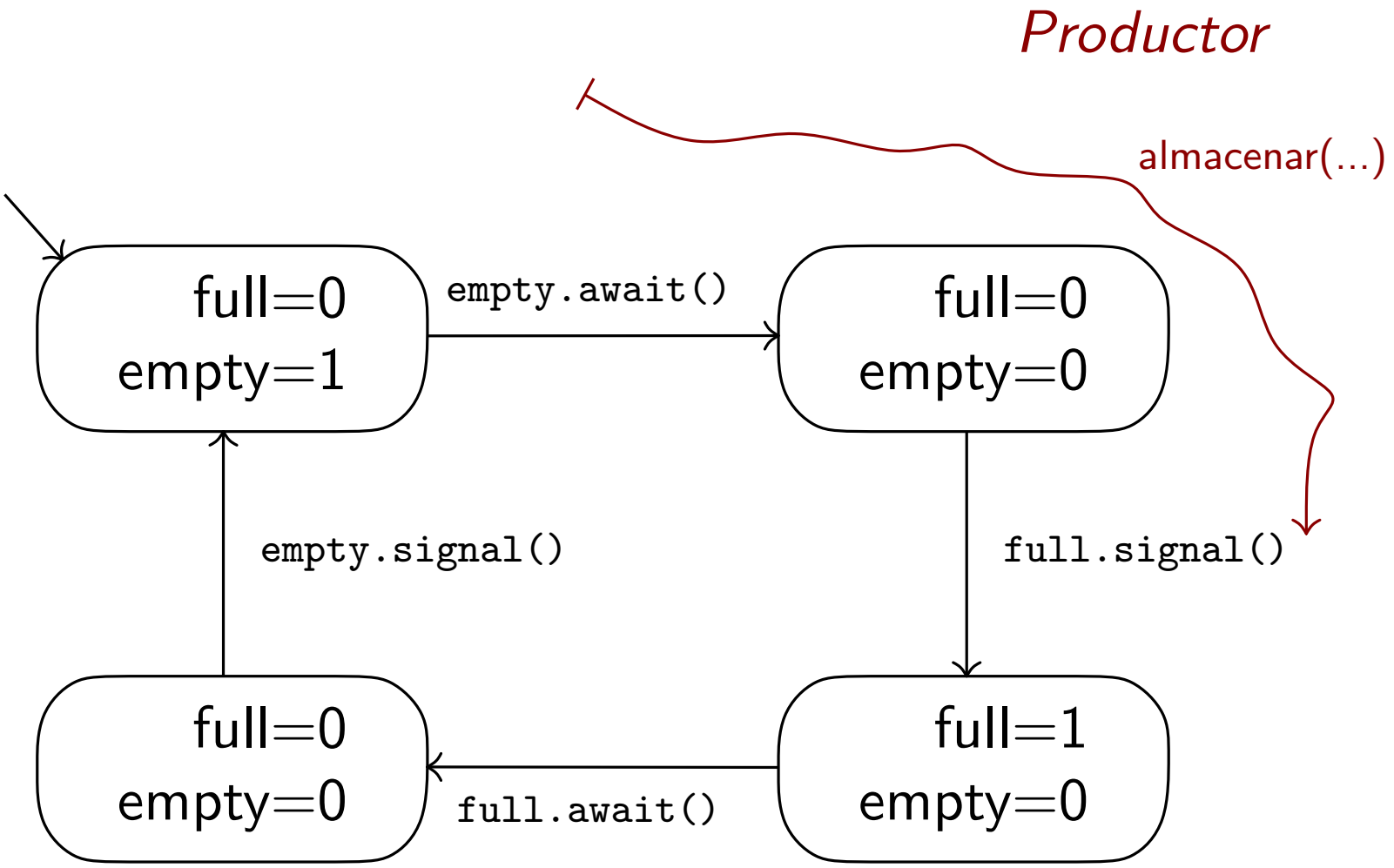
```
public void almacenar(Producto producto) {
   empty.await();
   mutex.await();
   almacenado = producto;
   mutex.signal();
   full.signal();
}
```

```
public Producto extraer() {
   Producto result;

   full.await();
   mutex.await();
   result = almacenado;
   almacenado = null;
   mutex.signal();
   empty.signal();

   return result;
}
```

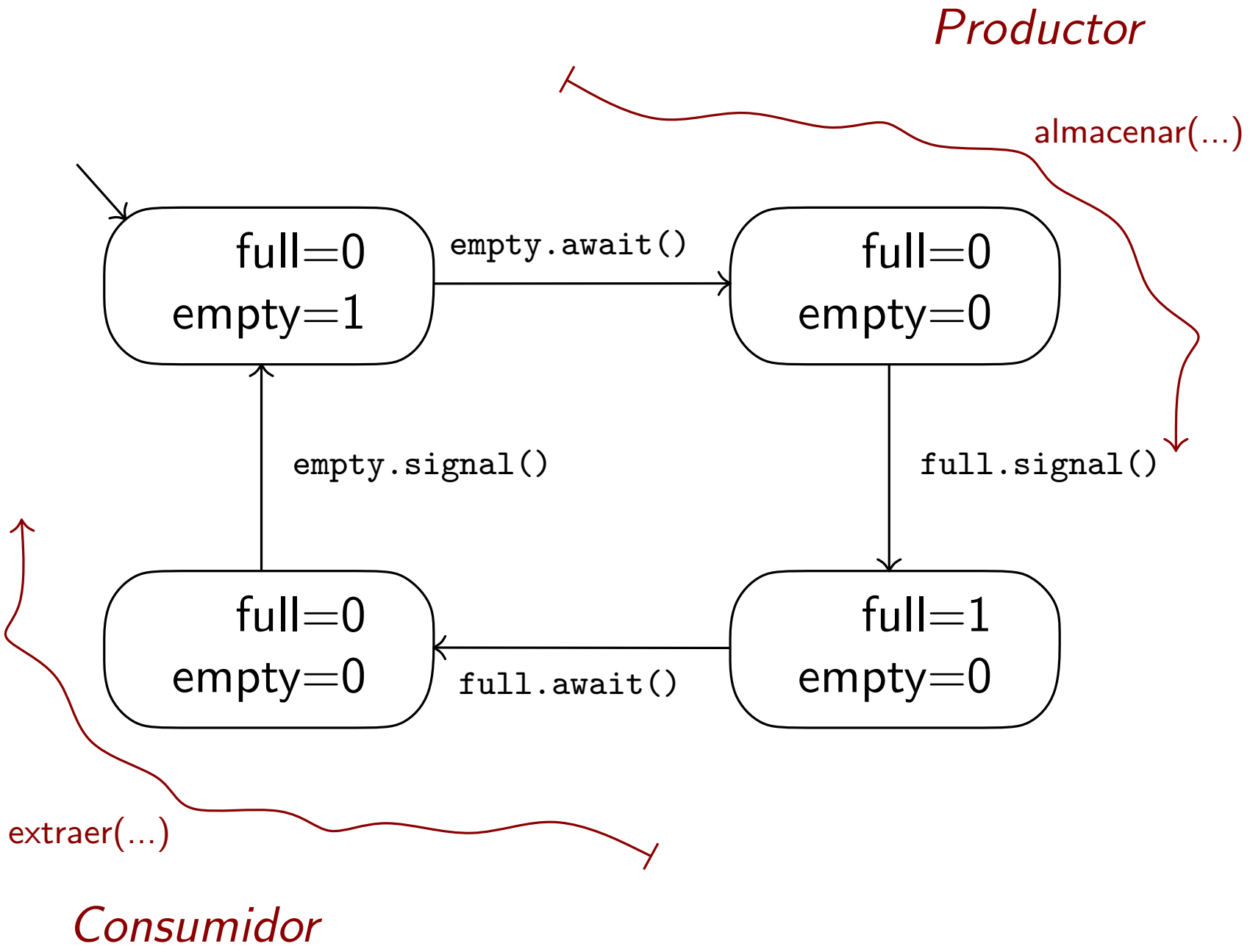
Almacen1: Abstracción



Almacen1: Abstracción



Almacen1: Abstracción



Almacen1 (sol 2)

```
public void almacenar(Producto producto) {
   empty.await();
   // mutex.await();
   almacenado = producto;
   // mutex.signal();
   full.signal();
}
```

```
public Producto extraer() {
   Producto result;

   full.await();
   // mutex.await();
   result = almacenado;
   almacenado = null;
   // mutex.signal();
   empty.signal();

   return result;
}
```

Producers/Consumers with AlmacenN

Use variable `holes` to denote available spaces

Use variable `products` to denote available products

- *Producers:*
  - decrease `holes` before producing (block if 0)
  - increase `products` after producing
- *Consumers*:
  - decrease `products` before consuming (block if 0)
  - increase `holes` after consuming

Producers/Consumers with AlmacenN

Use variable `holes` to denote available spaces

Use variable `products` to denote available products

- *Producers:*
  - decrease `holes` before producing (block if 0)
  - increase `products` after producing
- *Consumers*:
  - decrease `products` before consuming (block if 0)
  - increase `holes` after consuming

How?

Producers/Consumers with AlmacenN

Use variable `holes` to denote available spaces

Use variable `products` to denote available products

- *Producers:*
  - decrease `holes` before producing (block if 0)
  - increase `products` after producing
- *Consumers*:
  - decrease `products` before consuming (block if 0)
  - increase `holes` after consuming

How?

Use two counting semaphores!

AlmacenN

```
public void almacenar(Producto producto) {
    holes.await();

    mutex.await();
    almacenado[aInsertar] = producto;
    nDatos++;
    aInsertar++;
    aInsertar %= capacidad;
    mutex.signal();

    products.signal();
}
```

AlmacenN

```
public void almacenar(Producto producto) {
    holes.await();

    mutex.await();
    // actual product insertion
    mutex.signal();

    products.signal();
}
```

AlmacenN

```
public void almacenar(Producto producto) {
    holes.await();

    mutex.await();
    // actual product insertion
    mutex.signal();

    products.signal();
}
```

```
public Producto extraer() {
    Producto result;
    products.await();

    mutex.await();
    result = almacenado[aExtraer];
    almacenado[aExtraer] = null;
    nDatos--;
    aExtraer++;
    aExtraer %= capacidad;
    mutex.signal();

    holes.signal();
    return result;
}
```

AlmacenN

```
public void almacenar(Producto producto) {
    holes.await();

    mutex.await();
    // actual product insertion
    mutex.signal();

    products.signal();
}
```

```
public Producto extraer() {
    Producto result;
    products.await();

    mutex.await();
    // actual product consumption
    mutex.signal();
    return result;
}
```

Q: Can we safely remove `mutex`?