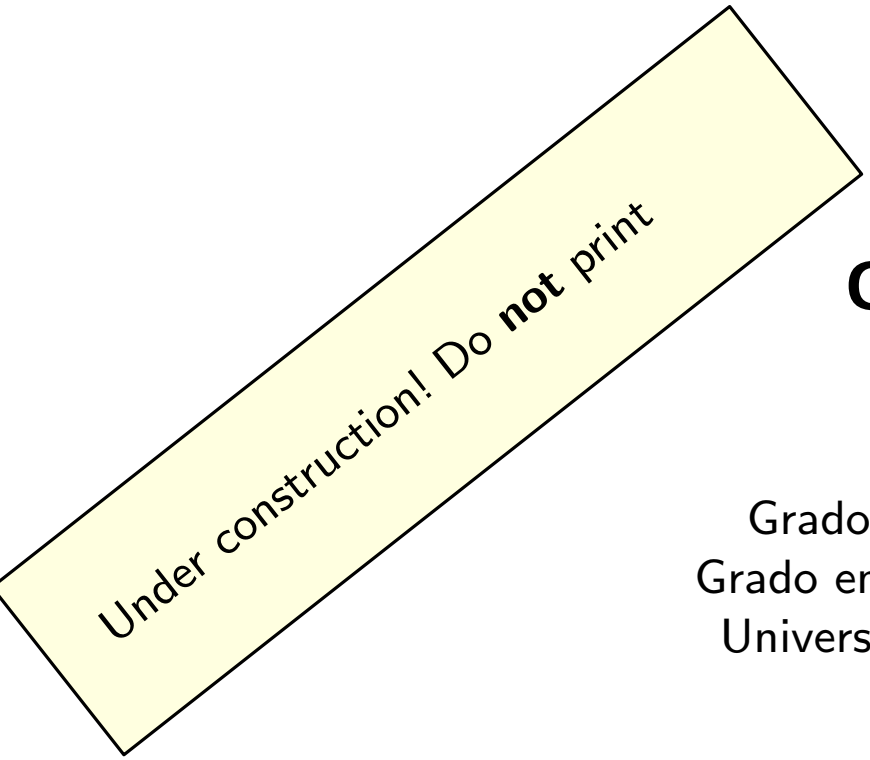
L12: Semaphores III



César Sánchez

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

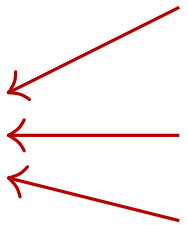
Fri, 13-March-2015

Mapa Conceptual

Concurrency = Simultaneous + Nondeterminism + Interaction

Interaction = Communication | Synchronization

Synchronization = Mutual Exclusion | Conditional Synchronization



- Terminology:

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

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

Readers/Writers: the problem

- Readers: processes that access data for reading only
- Writers: threads that access the data for writing
- DB: shared data

Readers/Writers: the problem

- Readers: processes that access data for reading only
- Writers: threads that access the data for writing
- DB: shared data

Goal: different *types* of accesses to shared resources

Readers/Writers: the problem

- Readers: processes that access data for reading only
- Writers: threads that access the data for writing
- DB: shared data

Goal: different *types* of accesses to shared resources

Spec:

One or more readers concurrently,
with no writer

At most one writer
with no conccurrent reader

Readers/Writers: the problem

- Readers: processes that access data for reading only
- Writers: threads that access the data for writing
- DB: shared data

Goal: different *types* of accesses to shared resources

Spec:

$$\texttt{num\_readers} \geq 0$$
$$\texttt{num\_writers} \in \{0, 1\}$$

One or more readers concurrently,
with no writer

At most one writer
with no conccurrent reader

Readers/Writers: the problem

- Readers: processes that access data for reading only
- Writers: threads that access the data for writing
- DB: shared data

Goal: different *types* of accesses to shared resources

Spec:

$$\texttt{num\_readers} \geq 0$$
$$\texttt{num\_writers} \in \{0, 1\}$$

$$(\texttt{num\_readers} > 0) \rightarrow \texttt{num\_writters=0}$$

At most one writer
with no conccurrent reader

Readers/Writers: the problem

- Readers: processes that access data for reading only
- Writers: threads that access the data for writing
- DB: shared data

Goal: different *types* of accesses to shared resources

Spec:

$$\texttt{num\_readers} \geq 0$$
$$\texttt{num\_writers} \in \{0, 1\}$$

$$(\texttt{num\_readers} > 0) \rightarrow \texttt{num\_writters=0}$$

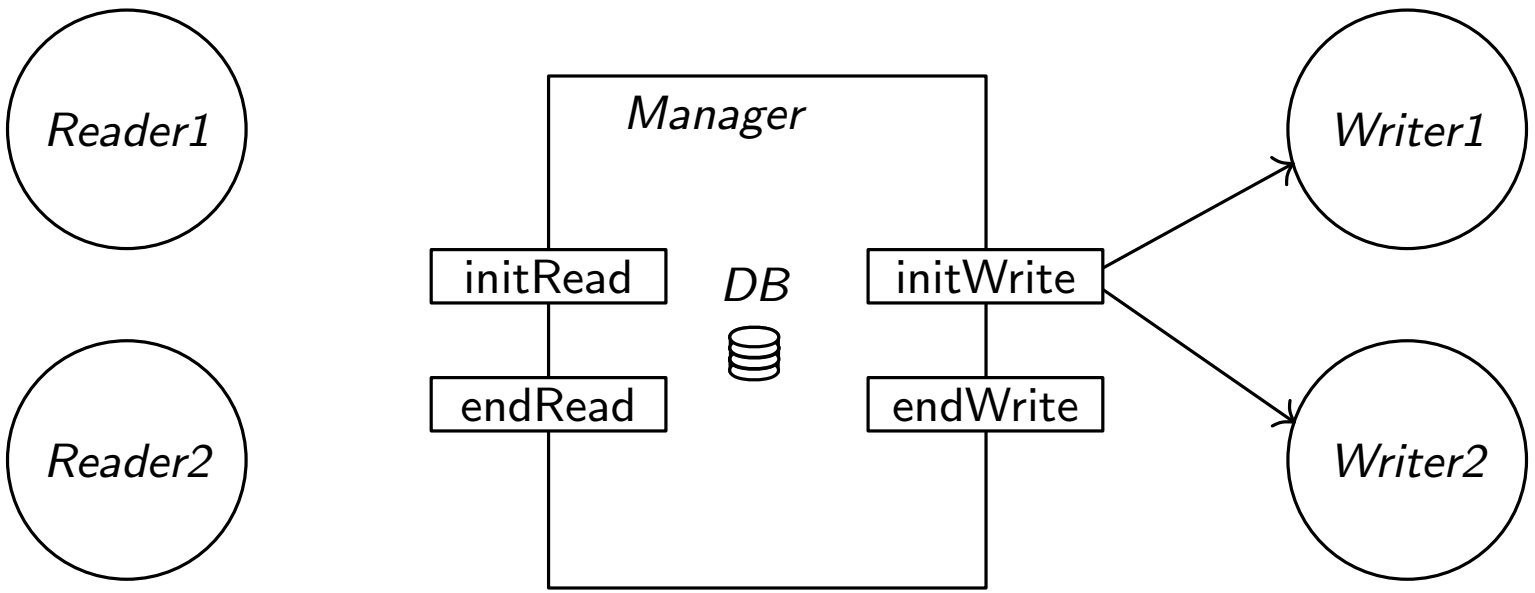
$$(\texttt{num\_writers} = 1) \rightarrow \texttt{num\_readers=0}$$

Readers/Writers

- Readers: processes that access data for reading only
- Writers: threads that access the data for writing
- DB: shared data

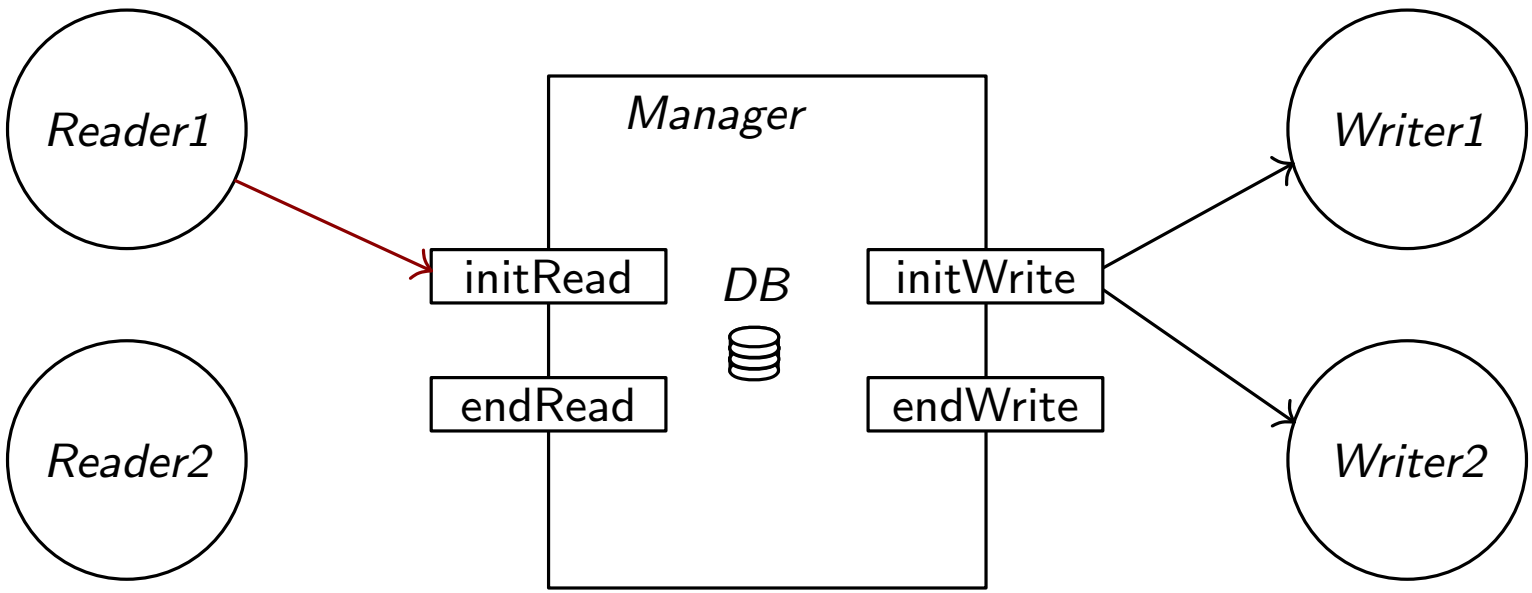
Readers/Writers

- Readers: processes that access data for reading only
- Writers: threads that access the data for writing
- DB: shared data



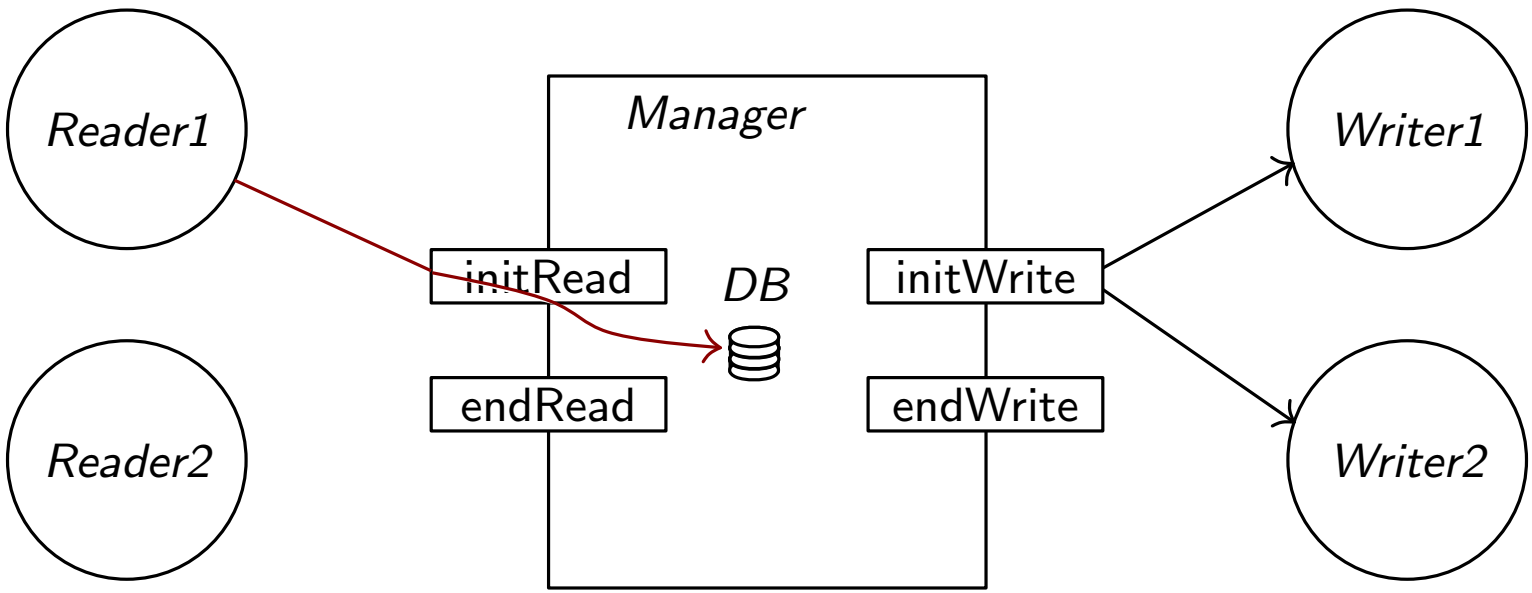
Readers/Writers

- Readers: processes that access data for reading only
- Writers: threads that access the data for writing
- DB: shared data



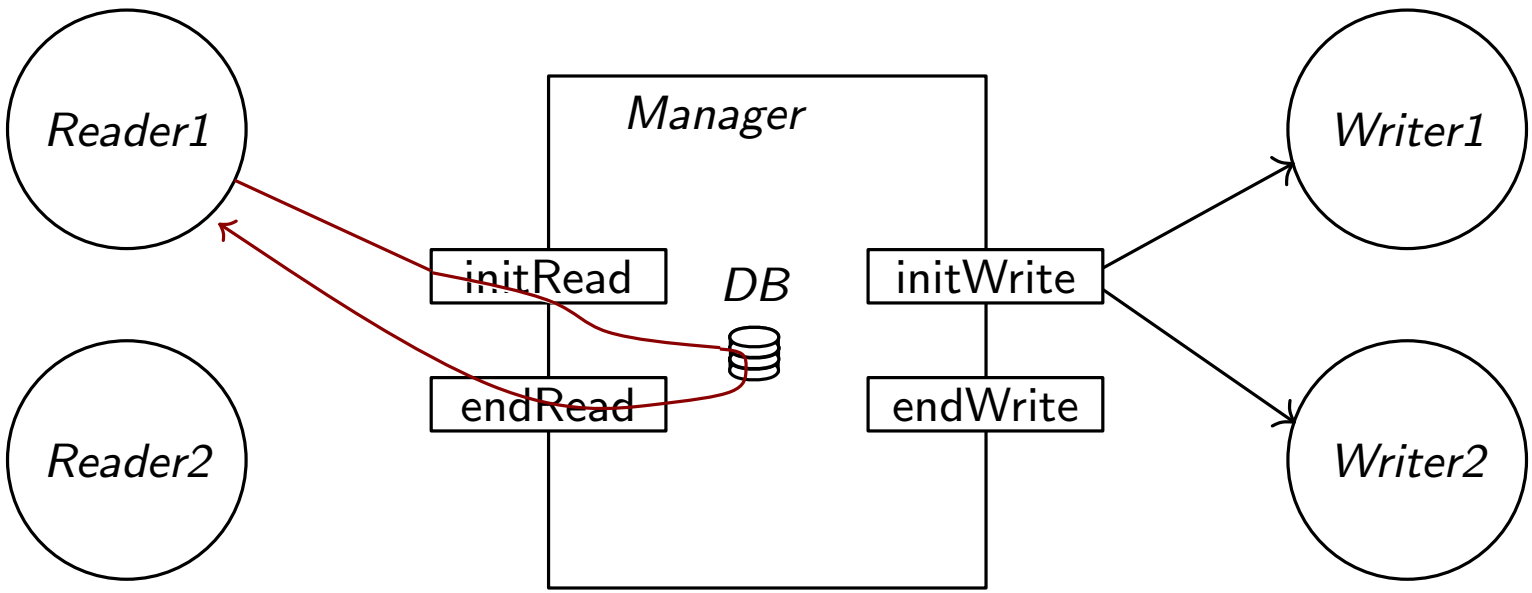
Readers/Writers

- Readers: processes that access data for reading only
- Writers: threads that access the data for writing
- DB: shared data



Readers/Writers

- Readers: processes that access data for reading only
- Writers: threads that access the data for writing
- DB: shared data



Readers/Writers. Skeleton

```
public static class Escritor extends Thread {
  private Gestor_LE gestor;
  public Escritor(Gestor_LE g){
     this.gestor = g;
  }
  public void run(){
     while(true) {
        gestor.inicioEscribir();
        ConcIO.printfnl("empiezo a escribir");
        //...
        ConcIO.printfnl("termino de escribir");
        gestor.finEscribir();
        //...
     }
  }
}
```

Readers/Writers. Skeleton

```
public static class Lector extends Thread {
  private Gestor_LE gestor;
  public Lector(Gestor_LE g){
     this.gestor = g;
  }
  public void run(){
     while(true) {
        gestor.inicioLeer();
        ConcIO.printfnl("empiezo a leer");
        //...
        ConcIO.printfnl("termino de leer");
        gestor.finLeer();
        //...
     }
  }
}
```

Readers/Writers. Skeleton

```
    public static class Gestor_LE {
        public void inicioLeer() {
   //...
        }

        public void finLeer() {
   //...
        }

        public void inicioEscribir() {
   //...
        }

        public void finEscribir() {
   //...
        }
    }
}
```

Readers/Writers. Solution 1

Readers/Writers. Solution 1

```
public static class Gestor_LE {
   Semaphore mutex;
   Semaphore write;
   public Gestor_LE(){
      mutex   = new Semaphore(1);
      write   = new Semaphore(1);
   }
   public void inicioLeer() {
      mutex.await();
      readers++;
      if (readers == 1) {
         write.await();
         }
      mutex.signal();
   }
   public void finLeer() {
      mutex.await();
      readers--;
      if (readers == 0) {
         write.signal();
      }
      mutex.signal();
   }
   public void inicioEscribir() {
      write.await();
   }
   public void finEscribir() {
      write.signal();
   }
}
```

Readers/Writers. Solution 1

```
public static class Gestor_LE {
   Semaphore mutex;
   Semaphore write;
   public Gestor_LE(){
      mutex   = new Semaphore(1);
      write   = new Semaphore(1);
   }
   public void inicioLeer() {
      mutex.await();
      readers++;
      if (readers == 1) {
         write.await();
         }
      mutex.signal();
   }
   public void finLeer() {
      mutex.await();
      readers--;
      if (readers == 0) {
         write.signal();
      }
      mutex.signal();
   }
```

```
   public void inicioEscribir() {
      write.await();
   }
   public void finEscribir() {
      write.signal();
   }
}
```

Problems?

Readers/Writers. Solution 1

```
public static class Gestor_LE {
   Semaphore mutex;
   Semaphore write;
   public Gestor_LE(){
      mutex   = new Semaphore(1);
      write   = new Semaphore(1);
   }
   public void inicioLeer() {
      mutex.await();
      readers++;
      if (readers == 1) {
         write.await();
         }
      mutex.signal();
   }
   public void finLeer() {
      mutex.await();
      readers--;
      if (readers == 0) {
         write.signal();
      }
      mutex.signal();
   }
```

```
   public void inicioEscribir() {
      write.await();
   }
   public void finEscribir() {
      write.signal();
   }
```

Problems?

Writers starvation!

Readers/Writers. Solution 2

Readers/Writers. Solution 2

```
public static class Gestor_LE {
   Semaphore rmutex,wmutex,r_entry,readTry, resource; /
   public Gestor_LE(){
      //... all semaphores to 1
   }
   public void inicioLeer() {
         r_entry.await(); //
         readTry.await(); // reader trying to enter
         rmutex.await(); // reader entry/exit mutex
         readers++;
         if (readers==1) {
            resource.await(); // readers vs writer
         }
         rmutex.signal();
         readTry.signal();
   }
   public void finLeer() {
      rmutex.await();
      readers--;
      if (readers == 0) {
         resource.signal();
      }
      rmutex.signal();
```

Readers/Writers. Solution 2

```
public static class Gestor_LE {
   Semaphore rmutex,wmutex,r_entry,readTry, resource; /
   public Gestor_LE(){
      //... all semaphores to 1
   }
        public void inicioEscribir() {
         wmutex.await();
         writers++;          //report yourself as a writer entering
         if (writers == 1) { //checks if you're first writer
            readTry.await();
         }
         wmutex.signal();
         resource.await();
        }

        public void finEscribir() {
         resource.signal();
         wmutex.await();
         writers--;
         if (writers == 0) {
            readTry.signal();
         }
         wmutex.signal();
```

Readers/Writers. Solution 2

```
public static class Gestor_LE {
   Semaphore rmutex,wmutex,r_entry,readTry, resource; /
   public Gestor_LE(){
      //... all semaphores to 1
   }
       public void inicioEscribir() {
        wmutex.await();
        writers++;          //report yourself as a writer entering
        if (writers == 1) { //checks if you're first writer
           readTry.await();
        }
        wmutex.signal();
        resource.await();
       }

       public void finEscribir() {
        resource.signal();
        wmutex.await();
        writers--;
        if (writers == 0) {
           readTry.signal();
        }
        wmutex.signal();
```

Problems?

Readers/Writers. Solution 2

```
public static class Gestor_LE {
   Semaphore rmutex,wmutex,r_entry,readTry, resource; /
   public Gestor_LE(){
      //... all semaphores to 1
   }
        public void inicioEscribir() {
         wmutex.await();
         writers++;          //report yourself as a writer entering
         if (writers == 1) { //checks if you're first writer
            readTry.await();
         }
         wmutex.signal();
         resource.await();
        }

        public void finEscribir() {
         resource.signal();
         wmutex.await();
         writers--;
         if (writers == 0) {
            readTry.signal();
         }
         wmutex.signal();
```

Problems?

Readers starvation!

Readers/Writers. Solution 3

Readers/Writers. Solution 3

```
public static class Gestor_LE {
   Semaphore mutex,read,write;
   public Gestor_LE(){
      //... all semaphores to 1
   }
        public void inicioLeer() {
         read.await();
         mutex.await();
            readers++;
         if (readers == 1) {
            write.await();
         }
         mutex.signal();
         read.signal();
        }

        public void finLeer() {
         mutex.await();
            readers--;
            if (readers == 0) {
                write.signal();
            }
         mutex.signal();
```

Readers/Writers. Solution 3

```
public static class Gestor_LE {
   Semaphore mutex,read,write;
   public Gestor_LE(){
      //... all semaphores to 1
   }
        public void inicioEscribir() {
         read.await();
         write.await();
        }

        public void finEscribir() {
         write.signal();
         read.signal();
        }
    }
```

Lesson learned

Programming with semaphores is tricky!!