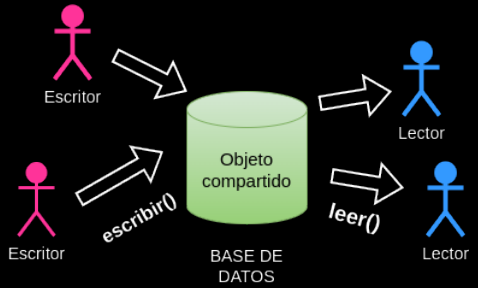
Problema de lectores y escritores

Rodríguez Barrios José Eduardo

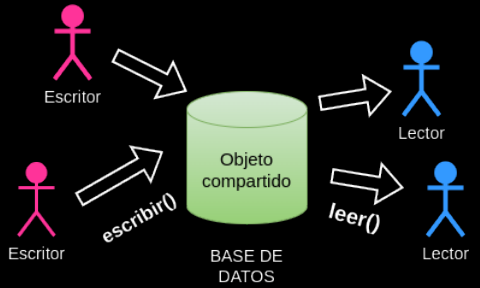
Facultad de Ingenieía, UNAM

26 de noviembre de 2021

El problema tiene los siguientes elementos:

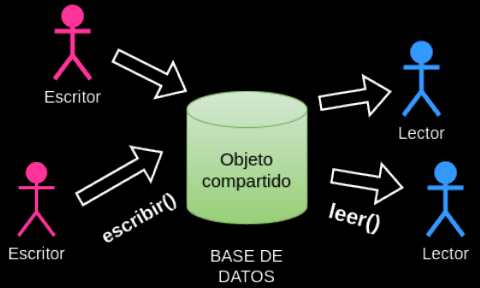


El problema tiene los siguientes elementos:



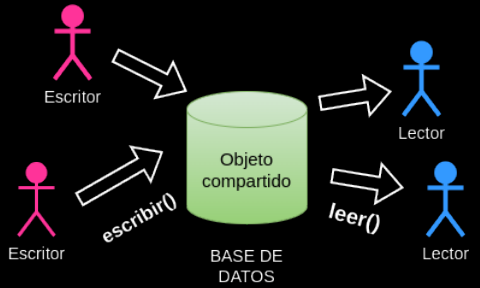
- Un objeto de datos que se va a compartir entre varios procesos concurrentes.

El problema tiene los siguientes elementos:



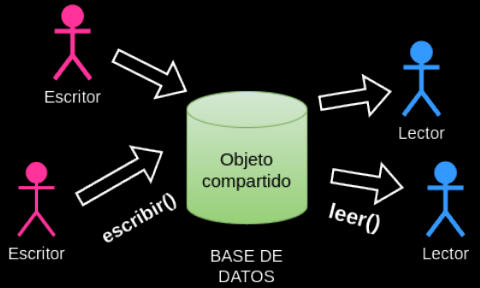
- Un objeto de datos que se va a compartir entre varios procesos concurrentes.
- Existen dos tipos de procesos:

El problema tiene los siguientes elementos:



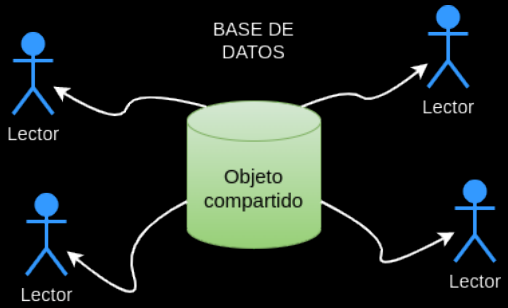
- Un objeto de datos que se va a compartir entre varios procesos concurrentes.
- Existen dos tipos de procesos:
  - Escritores: Los interesados en sólo querer modificar el objeto compartido.

El problema tiene los siguientes elementos:

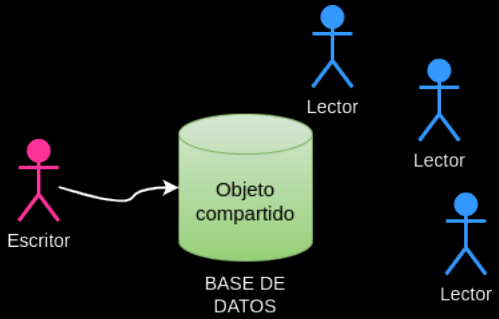


- Un objeto de datos que se va a compartir entre varios procesos concurrentes.
- Existen dos tipos de procesos:
  - Escritores: Los interesados en sólo querer modificar el objeto compartido.
  - Lectores: Los que sólo están interesados en leer el objeto compartido.

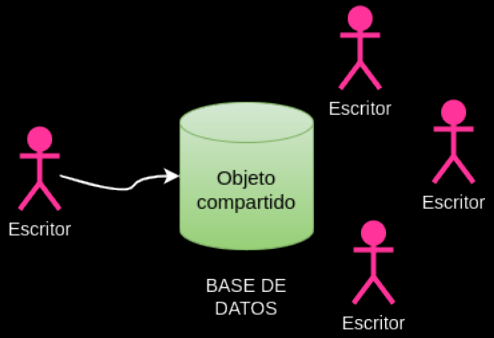
Cualquier número de lectores puede leer el área de datos simultáneamente.



Si un escritor está accediendo al objeto compartido ningún lector puede leerlo.



Sólo puede escribir en el área compartida un escritor en cada instante.

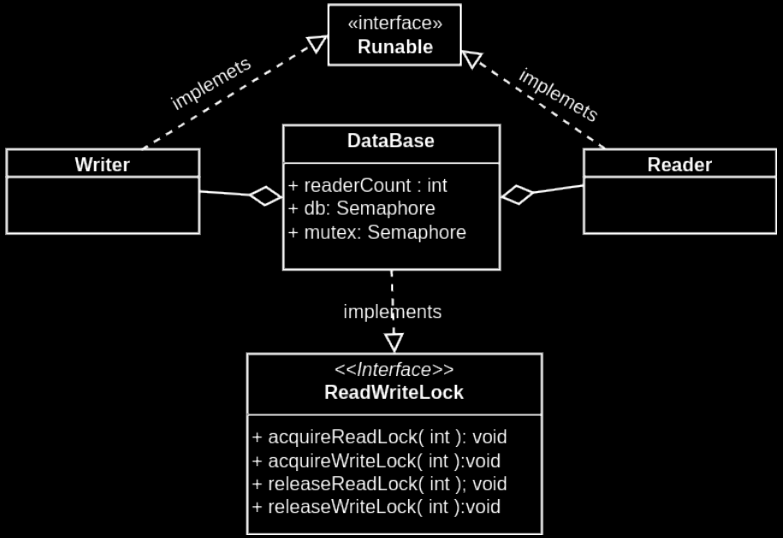


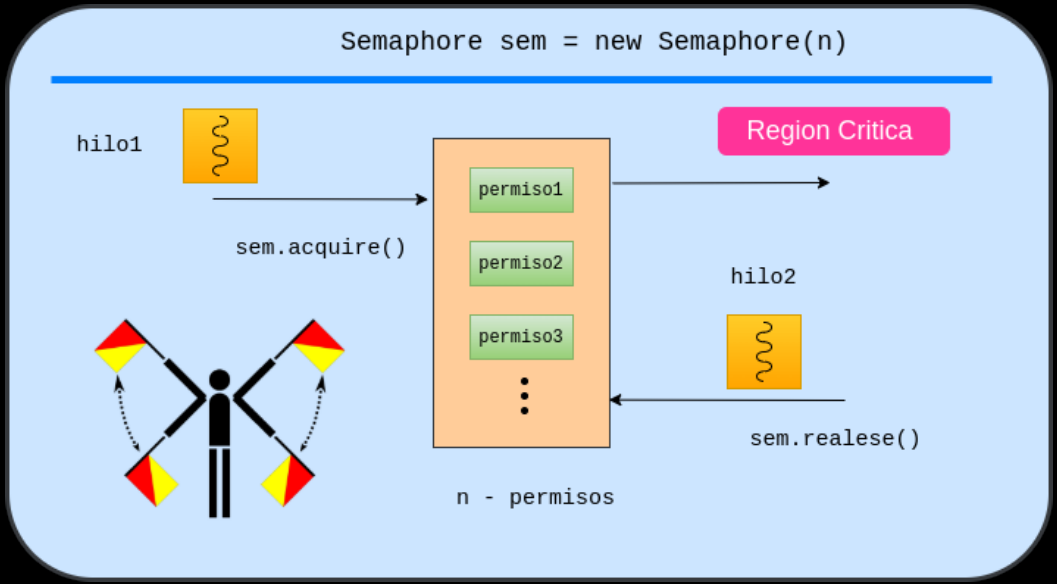
- **Primer problema de lectores y escritores**:
  Exige que no se mantenga esperando a ningún lector a menos que un escritor, ya haya obtenido permiso para usar el objeto compartido. (prioridad a los lectores)

- **Primer problema de lectores y escritores**:
  Exige que no se mantenga esperando a ningún lector a menos que un escritor, ya haya obtenido permiso para usar el objeto compartido. (prioridad a los lectores)
- **Segundo problema de lectores y escritores**
  Se requiere que, úna vez que un escritor está listo, realice su escritura lo antes posible. Dicho de otro modo, si un escritor está esperando acceder al objeto, ningún lector nuevo puede comenzar leer. (prioridad a los escritores)

Solución del primer problema de lectores y escritores en Java

El programa tiene la siguiente estructura:




Semaphore sem = new Semaphore(n)
hilo1
Region Critica
permiso1
permiso2
permiso3
sem.acquire()
hilo2
sem.realese()
n - permisos

```
public class DataBase implements ReadWriteLock
{
  // Número de lectores activos.
  private int readerCount;

  /**
  * El semáforo mutex se utiliza para garantizar la
  * exclusión mutua cuando readerCount se actualiza.
  */
  private Semaphore mutex;
  /*
  * El semáforo db funciona como una exclusión mutua
  * para los escritores, solo debe estar un escritor
  * escribiendo.
  */
  private Semaphore db;
  ...
}
```

Solución del primer problema de lectores y escritores en Java

Lector

```
public class Reader implements Runnable{

 private ReadWriteLock db;
 int readerNum;

 public Reader(int readerNum, ReadWriteLock db) {
   this.db = db;
   this.readerNum = readerNum;
 }

 public void run() {
   while (true) {
     // siesta por un rato
     SleepUtilities.nap();

     System.out.println("lector " + readerNum
             + " quiere leer.");
     db.acquireReadLock(readerNum);
     // ahora lee de la base de datos
     SleepUtilities.nap();

     db.releaseReadLock(readerNum);
   }
 }
}
```

Solución del primer problema de lectores y escritores en Java

Lector

```
public class Reader implements Runnable{

  private ReadWriteLock db;
  int readerNum;

  public Reader(int readerNum, ReadWriteLock db) {
    this.db = db;
    this.readerNum = readerNum;
  }

  public void run() {
    while (true) {
      // siesta por un rato
      SleepUtilities.nap();

      System.out.println("lector " + readerNum
              + " quiere leer.");
      db.acquireReadLock(readerNum);
      // ahora lee de la base de datos
      SleepUtilities.nap();

      db.releaseReadLock(readerNum);
    }
  }
}
```

Escritor

```
public class Writer implements Runnable{

  private ReadWriteLock db;
  int writerNum;

  public Writer(int writercount, ReadWriteLock db) {
    this.db = db;
    this.writerNum = writercount;
  }
  public void run() {
    while (true) {
      // siesta por un rato
      SleepUtilities.nap();

      System.out.println("escritor " + writerNum
              + " quiere escribir.");
      db.acquireWriteLock(writerNum);

      // ahora escribe en la base de datos
      SleepUtilities.nap();

      db.releaseWriteLock(writerNum);
    }
  }
}
```

Flujo de instrucciones de un lector o escritor

En ambos casos, un lector o escritor hará lo siguiente:

Flujo de instrucciones de un lector o escritor

En ambos casos, un lector o escritor hará lo siguiente:

1. Se tomará una siesta entre 0 y 5 segundos.

Flujo de instrucciones de un lector o escritor

En ambos casos, un lector o escritor hará lo siguiente:

1. Se tomará una siesta entre 0 y 5 segundos.
2. Luego, pedirá acceso a la base de datos ya sea para escribir o leer.

Flujo de instrucciones de un lector o escritor

En ambos casos, un lector o escritor hará lo siguiente:

1. Se tomará una siesta entre 0 y 5 segundos.
2. Luego, pedirá acceso a la base de datos ya sea para escribir o leer.
3. Si tiene acceso, realizará su tarea entre 0 y 5 segundos. Si no esperará hasta tener acceso.

Flujo de instrucciones de un lector o escritor

En ambos casos, un lector o escritor hará lo siguiente:

1. Se tomará una siesta entre 0 y 5 segundos.
2. Luego, pedirá acceso a la base de datos ya sea para escribir o leer.
3. Si tiene acceso, realizará su tarea entre 0 y 5 segundos. Si no esperará hasta tener acceso.
4. Una vez realizada su tarea libera la base de datos, ya sea para un escritor o lector según el caso.

Flujo de instrucciones de un lector o escritor

En ambos casos, un lector o escritor hará lo siguiente:

1. Se tomará una siesta entre 0 y 5 segundos.
2. Luego, pedirá acceso a la base de datos ya sea para escribir o leer.
3. Si tiene acceso, realizará su tarea entre 0 y 5 segundos. Si no esperará hasta tener acceso.
4. Una vez realizada su tarea libera la base de datos, ya sea para un escritor o lector según el caso.

Esto lo hará todo el tiempo que dure en ejecución el programa.

Lector - mecanismo de acceso

```
public void acquireReadLock(int readerNum) {
  try {
    mutex.acquire();
    /**
    * El primer lector indica que la base
    * esta siendo leida
    */
    ++readerCount;
    if (readerCount == 1)
      db.acquire();
    System.out.println("lector " + readerNum
      + " está leyendo. Lectores Activos = " + readerCount);
  } catch (Exception e) {

  }
  finally{
    mutex.release();
  }
}
```

Lector - mecanismo de liberación

```
public void releaseReadLock(int readerNum) {
    try {
      mutex.acquire();
      /**
      * El último lector indica que
      * la base de datos ya no se esta leyendo.
      */
      --readerCount;
      if (readerCount == 0)
        db.release();
      System.out.println("Lector " + readerNum
        + " ha terminado de leer. Lectores  Activos = "
        + readerCount);
    } catch (Exception e) {

    }
    finally{
      mutex.release();
    }
  }
```

Escritor - mecanismo de acceso

```
public void acquireWriteLock(int writerNum) {
    try {
      db.acquire();
      System.out.println("escritor " + writerNum
        + " está escribiendo.");
    } catch (Exception e) {}
  }
```

Escritor - mecanismo de liberación

```
public void releaseWriteLock(int writerNum) {
  try {
    System.out.println("escritor " + writerNum
      + " ha terminado de escribir.");
    db.release();
  } catch (Exception e) {}
}
```

1. Abraham Silberschatz, Peter B. Galvin, Greg Gagne, *Operating System Concepts with Java*, Wiley, 2009.
2. Tanenbaum, Andrew S., *Sistemas operativos modernos*, Pearson, 2011.