1 Problema del Lector-Escritor

Este es otro de los problemas clásicos a la hora de hacer concurrencia:

Supongamos que se tiene un recurso en el que tienen que entrar muchos hilos a leer datos y muchos hilos que quieren entrar al recurso para hacer modificaciones.

Este problema es típico de bases de datos en el que se desea maximizar el rendimiento haciendo que todos los hilos lectores puedan usar el recurso a la vez. Evidentemente sólo un hilo escritor podrá usar el recurso.

Una solución errónea sería:

```
class Lector {
…
      semaforo.acquire();
      // Leer del recurso
      semaforo.release();
…
}
```

```
class Escritor {
…
      semaforo.acquire();
      // Escribir en el recurso
      semaforo.release();
…
}
```

Si se usa esta solución, no se tendrá la máxima concurrencia, pues sólo un hilo lector podrá acceder, cuando es posible que varios hilos lectores usen el recurso a la vez.

Se pueden adoptar dos posturas a la hora de buscar una solución:

- Prioridad de lectura: Cuando un hilo desea leer del recurso, tiene prioridad sobre los hilos que desean escribir, aunque lleven tiempo esperando.
- Prioridad de escritura: Cuando un hilo desea escribir del recurso, tiene prioridad sobre los hilos que desean leer, aunque lleven tiempo esperando.

1.1 Solución con prioridad de lectura

La solución usa las siguientes herramientas:

- Un entero nLectores que representa el número de lectores que están leyendo el recurso en ese instante. Se inicializa a 0.
- Un semáforo semaforoNLectores que va a controlar el acceso a nLectores. Se inicializa a 1.
- Un semáforo semaforoRecurso que va a controlar el acceso al recurso.

La idea es contar cuántos hilos están leyendo el recurso y si es cero se despiertan los hilos Escritor que estén esperando.

Si hay un hilo Escritor usando el recurso, los hilos Lector quedan bloqueados por semaforoRecurso, hasta que el Escritor termine.

Si no hay hilos Escritor o Lector usando el recurso, los hilos pueden entrar al recurso sin problemas.

La solución en Java podría ser:

```
import java.util.concurrent.Semaphore;

class Compartido {
```

```
    public int nLectores;
    public Semaphore semaforoNLectores;
    public Semaphore semaforoRecurso;

    // Recurso: En este caso se usa un array, pero puede ser cualquier
    // otro recurso
    public int buffer[];

    public Compartido() {
        semaforoNLectores = new Semaphore(1);
        nLectores = 0;
        semaforoRecurso = new Semaphore(1);

        // Se inicializa el recurso
        buffer = new int[10];
        for(int i = 0; i < buffer.length; i++)
            buffer[i] = i * 2;
    }
}

class Lector extends Thread {
    private Compartido m;

    public Lector(Compartido compartido) {
        m = compartido;
    }

    public void run() {
        try {
            m.semaforoNLectores.acquire();
            m.nLectores++;

            if(m.nLectores == 1) // Se bloquean los escritores
                m.semaforoRecurso.acquire();

            m.semaforoNLectores.release();

                // Se lee el recurso
                System.out.print("Leyendo");
                for(int i = 0; i < m.buffer.length; i++) {
                    System.out.print(" " + m.buffer[i]);
                }
                System.out.println();

            m.semaforoNLectores.acquire();
            m.nLectores--;

            if(m.nLectores == 0) // El último lector, libera a los escritores
                m.semaforoRecurso.release();

            m.semaforoNLectores.release();
        } catch(InterruptedException e) {
            System.err.println(e);
        }
    }
}

class Escritor extends Thread {
    private Compartido m;

    public Escritor(Compartido compartido) {
        m = compartido;
    }
```

```
    public void run() {
        try {
            m.semaforoRecurso.acquire();

                // Se lee escribe en el recurso
                System.out.println("Escribiendo");
                for(int i = 0; i < m.buffer.length; i++) {
                    m.buffer[i] = m.buffer[i] + 1;
                }

            m.semaforoRecurso.release();

        } catch(InterruptedException e) {
            System.err.println(e);
        }
    }
}

class PrioridadLectura {
    public static void main(String args[]) {
        Lector l[] = new Lector[2];
        Escritor e[] = new Escritor[2];
        Compartido compartido = new Compartido();

        while(true) {
            for(int i = 0; i < l.length; i++) {
                l[i] = new Lector(compartido);
                e[i] = new Escritor(compartido);

                l[i].start();
                e[i].start();
            }

            for(int i = 0; i < l.length; i++) {
                try {
                    l[i].join();
                    e[i].join();
                } catch(InterruptedException err) {
                    System.err.println(err);
                }
            }
        }
    }
}
```

1.2 Solución con prioridad de escritura

La solución usa las siguientes herramientas:

- Un entero nLectores que representa el número de lectores que están leyendo el recurso en ese instante. Se inicializa a 0.
- Un entero nLectoresEsperando que representa el número de lectores que están esperando a entrar en el recurso. Se inicializa a 0.
- Un entero nEscritoresEsperando que representa el número de escritores que están esperando a entrar en el recurso. Se inicializa a 0.
- Un booleano “escribiendo” que representa cuando hay un escritor usando el recurso. Se inicializa a falso.

- Un semáforo “semaforo” que va a controlar el acceso a nLectores, nLectoresEsperando y nEscritoresEsperando. Se inicializa a 1.
- Los semáforos semaforoLector y semaforoEscritor que va a controlar el acceso al recurso por parte de los lectores y escritores respectivamente.

En este caso la solución es un poco más complejo:

```
import java.util.concurrent.Semaphore;

class Compartido {
	public Semaphore semaforo, semaforoLector, semaforoEscritor;
	public int nLectores, nLectoresEsperando, nEscritoresEsperando;
	public boolean escribiendo;

	// Recurso compartido
	public int buffer[];

	public Compartido() {
		semaforo = new Semaphore(1);
		semaforoLector = new Semaphore(0);
		semaforoEscritor = new Semaphore(0);
		nLectores = nLectoresEsperando = nEscritoresEsperando = 0;
		escribiendo = false;

		// Se inicializa el recurso compartido
		buffer = new int[10];
		for(int i = 0; i < buffer.length; i++)
			buffer[i] = i * 2;
	}
}

class Lector extends Thread {
	private Compartido m;

	public Lector(Compartido compartido) {
		m = compartido;
	}

	public void run() {
		try {
			m.semaforo.acquire();
			if(m.escribiendo || m.nEscritoresEsperando > 0) {
				m.nLectoresEsperando++;
				m.semaforo.release();
				m.semaforoLector.acquire();
				//m.semaforo.acquire();
				m.nLectoresEsperando--;
			}
			m.nLectores++;
			if(m.nLectoresEsperando > 0)  // Desbloqueo encadenado se
desbloquean
				m.semaforoLector.release();   // todos los lectores que
estén esperando
			else
				m.semaforo.release();

					// Se lee el recurso
					System.out.print("Lector");
					for(int i = 0; i < m.buffer.length; i++)
						System.out.print(" " + m.buffer[i]);
					System.out.println();
```

```
                m.semaforo.acquire();
                m.nLectores--;
                // Se desbloquean los escritores
                if(m.nLectores == 0 && m.nEscritoresEsperando > 0)
                        m.semaforoEscritor.release();
                else
                        m.semaforo.release();
            } catch(InterruptedException err) {
                System.err.println(err);
            }
        }
}

class Escritor extends Thread {
        private Compartido m;

        public Escritor(Compartido compartido) {
            m = compartido;
        }

        public void run() {
            try {
                m.semaforo.acquire();
                if(m.nLectores > 0 || m.escribiendo) {
                        m.nEscritoresEsperando++;
                        m.semaforo.release();
                        m.semaforoEscritor.acquire();
                        m.nEscritoresEsperando--;
                }
                m.escribiendo = true;
                m.semaforo.release();

                        // Se escribe en el recurso
                        System.out.println("Escritor");
                        for(int i = 0; i < m.buffer.length; i++)
                                m.buffer[i]++;

                m.semaforo.acquire();
                m.escribiendo = false;
                // Se desbloquea primero a los escritores en espera,
                // después se desbloquea a los lectores:
                if(m.nEscritoresEsperando > 0)
                        m.semaforoEscritor.release();
                else if(m.nLectoresEsperando > 0)
                        m.semaforoLector.release();
                else
                        m.semaforo.release();
            } catch(InterruptedException err) {
                System.err.println(err);
            }
        }
}

class PrioridadEscritura {
        public static void main(String args[]) {
            Compartido compartido = new Compartido();
            Lector l[] = new Lector[100];
            Escritor e[] = new Escritor[100];

            while(true) {
                for(int i = 0; i < l.length; i++) {
                        l[i] = new Lector(compartido);
                        e[i] = new Escritor(compartido);
```

```
                        l[i].start();
                        e[i].start();
                  }
                  for(int i = 0; i < l.length; i++) {
                        try {
                              l[i].join();
                              e[i].join();
                        } catch(InterruptedException err) {
                              System.err.println(err);
                        }
                  }
            }
      }
}
```

Las líneas:

```
m.semaforoLector.acquire();
...
if(m.nLectoresEsperando > 0)  // Desbloqueo encadenado: se desbloquean
      m.semaforoLector.release();   // todos los lectores que estén esperando
```

se conocen desbloqueo encadenado. Hay un gran número de hilos esperando a poder usar el recurso y cada hilo que se desbloquea, desbloquea a otro que esté esperando. Esta solución se permite resolver un gran número de problemas de concurrencia.

Ejercicios:

1. Se va a intentar simular un el juego de la silla. Hay 10 sillas y 10 participantes. Se supone que cada hilo es un participante. Los hilos debe consultar las sillas libres (lectores) y elegir una silla. Una vez elegida la silla, deben lanzar un hilo escritor que ocupe la silla. Si el hilo lo consigue, finaliza su ejecución. Si no lo consigue vuelve a repetir el proceso. La salida del programa debe ser similar a:

```
Hilo 5 intentando ocupar silla 6
Hilo 8 ocupa la silla 6
Hilo 3 intentando ocupar silla 7
Hilo 5 intentando ocupar silla 7
Hilo 6 intentando ocupar silla 7
Hilo 3 ocupa la silla 7
Hilo 5 intentando ocupar silla 8
Hilo 6 intentando ocupar silla 9
Hilo 5 ocupa la silla 8
Hilo 6 ocupa la silla 9
Ocupación final de las sillas:
Silla 0 ocupada por hilo 1
Silla 1 ocupada por hilo 9
Silla 2 ocupada por hilo 2
Silla 3 ocupada por hilo 4
Silla 4 ocupada por hilo 10
Silla 5 ocupada por hilo 7
Silla 6 ocupada por hilo 8
Silla 7 ocupada por hilo 3
Silla 8 ocupada por hilo 5
Silla 9 ocupada por hilo 6
```

2. ¿Por qué en el ejercicio anterior la escritura debe tener prioridad sobre la lectura? ¿Podría darse inanición de los escritores?

3. Suponga que dos investigadores tienen varios termómetros que mandan información a un array de datos. Cada investigador se echa una siesta, que dura un tiempo aleatorio y lee los

datos de todos los termómetros. Simule la situación suponiendo que cada termómetro escribe su medida de la temperatura siempre en la misma posición del array. Suponga que la lectura tiene prioridad a la escritura. Se debería tener una salida similar a lo siguiente:

```
Investigador 1 lee las temperaturas  36 39 -7 39 27 24 26 37 7 23
Investigador 1 lee las temperaturas  -5 10 5 -4 13 27 36 12 29 35
Investigador 0 lee las temperaturas  5 24 -1 28 1 11 15 39 28 1
Investigador 1 lee las temperaturas  11 32 8 9 -7 34 20 20 19 36
```