Computación Concurrente - Tarea 2

Damián Rivera González
Alexis Hernandez Castro

September 15, 2019

1. 1.Considera el algoritmo de Peterson:

```
1. public void lock() {
2.     int i = ThreadID.get();
3.     int j = 1 - i;
4.     flag[i] = true;
5.     victim = i;
6.     while (flag[j] && victim == i) {};
7. }
```

   a) Supón que intercambias las líneas 4 y 5. ¿El algoritmo sigue cumpliendo con las propiedades de exclusión y no deadlock? Demuestra o da una ejecución en donde no se cumpla.

   b) Supón ahora que la instrucción de la línea 6 se cambia por:

   $$while(flag[j] \;\&\&\; victim == j)$$

   Da un ejemplo en donde se cumpla la exclusión mutua y otro en donde no.

2. Considera el siguiente algoritmo para dos procesos:

Algoritmo para Alice:

1. Levanta su bandera
2. Espera hasta que la bandera de Bob este abajo
3. Libera la mascota
4. Baja la bandera cuando la mascota regresa

Algoritmo para Bob:

1. Levanta su bandera
2. Mientras la bandera de Alice este arriba:
   1) Baja su bandera.
   2) Espera hasta que la bandera de Alice este abajo
   3) Levanta su bandera
3. En cuanto la bandera de Alice está abajo y la suya arriba, libera su mascota
4. Baja su bandera cuando la mascota regresa.

a) Demuestra que resuelve el problema de la exclusión mutua.

b) Demuestra por qué no cumple con la propiedad libre de hambruna.

c) El algoritmo cumple con la propiedad de no deadlock?

d) Cuál es la desventaja del algoritmo? Argumenta por qué es difícil de generalizar.

3. Supón que tienes la instrucción swap(A,B) que intercambia de manera atómica los valores de las variables A,B.

   a) Da una propuesta de codificación para un objeto Look que resuelva el problema de la exclusión mutua para dos procesos.

   b) Demuestra que tu algoritmo cumple con la propiedad de exclusión y no deadlock.

4. Considera la siguiente modificación de la clase contador, que es ejecutada por dos procesos concurrentemente cuyos IDś son (0 y 1):

```
public class Counter implements Runnable {
    private int counter = 0;
    private static final int rounds = 10;
    private boolean wantToEnter[] =  { false, false };

    public void run() {
        long ID = Thread.currentThread().getId();
        for(int i = 0; i < Counter.rounds; i++) {
            wantToEnter[ID] = true;
            while(wantToEnter[1 - ID]) {
                wantToEnter[ID] = false;
                wantToEnter[ID] = true;
            }
            this.counter++;
            wantToEnter[ID] = false;
        }
    }
}
```

   a) Muestra una ejecución en donde los hilos llegan a un estado de deadlock.

   b) Demuestra que el algoritmo no tiene una condición de carrera, es decir, toda ejecucin que termina siempre imprime el mismo resultado.

5. ¿Porqué requerimos definir una sección de entrada (doorway), porqué no podemos definir un algoritmo de exclusión mutua que cumpla con la propiedad FCFS basado en el orden en el que los hilos ejecutan la primer instrucción del método lock? Argumenta tu respuesta caso por caso según la naturaleza de la primera instrucción ejecutada por el método lock(): una lectura o una escritura, a registros separados o a un mismo registro.

6. Muestra que el algoritmo del filtro permite que algunos hilos superen a otros un número arbitrario de veces.

7. Supón que en el algoritmo del filtro se intercambia la operación $\geqslant$ por $>$ en la condición:

$$while((Ek\ !=\ \text{me})\ (level[k] > i\ \&\&\ victim[i] == me))$$

8. Una forma de generalizar el algoritmo de Peterson para n hilos (supón que n es potencia de 2) es utilizar varios candados de 2 hilos en un árbol binario. A cada hilo es asignado un candado en una hoja que comparte con otro hilo.Cada candado trata a cada hilo como hilo 0 o hilo 1. En el método de adquisición del árbol de candados (como el método lock() en Peterson), el hilo adquiere cada candado de Peterson de la hoja de ese hilo a la raíz (camino de la hoja de ese hilo a la raíz). El método que libera al candado (como el unlock() en Peterson) para el árbol de candados, desbloquea cada candado de Peterson que el hilo haya adquirido desde la raíz hasta su hoja. En cualquier momento un hilo puede retrasarse por un tiempo finito. Para cada propiedad, da un esbozo de prueba que la propiedad se cumple o describe una ejecución posiblemente infinita donde se viola:

   - Exclusión mutua
   - Libre de deadlok
   - Libre de hambruna

   ¿Hay una cota superior en el número de veces que el árbol de candados puede ser adquirido o liberado entre el tiempo en el que un hilo empieza el adquirir el candado y cuando lo logra?

9. Supón que n hilos ejecutan el mtodo visit() de la siguiente clase:

```
class Bouncer {
    public static final int DOWN = 0;
    public static final int RIGHT = 1;
    public static final int STOP = 2;
    private boolean goRight = false;
    private ThreadLocal<Integer> myIndex;
    private int last = -1;

    int visit() {
        int i = myIndex.get();
        last = i;
        if (goRight) return RIGHT;

        goRight = true;
        if (last == i) return STOP;
        else return DOWN;
    }
}
```

   a) A lo más un hilo obtiene el valor STOP
   b) A lo más n - 1 hilos obtienen el valor DOWN
   c) A lo más n - 1 obtienen el valor RIGHT