PROGRAMACIÓN CONCURRENTE

EXAMEN ORDINARIO

Grado en Ingeniería de Computadores. Universidad Rey Juan Carlos

20 diciembre 2018

Nombre y apellidos: _____

DNI: _____

Entrega esta hoja y todas las demás que hayas necesitado. Pon el nombre en todas ellas y numéralas.

1. (1 punto) Explica lo que es la exclusión mutua generalizada y da un ejemplo de aplicación.

Solución: diapositivas 161 a 164 del tema 2

3. (1,25 punto) Compara el paralelismo simulado con el real, describiendo ambos modelos y explicando sus características y ganancias o pérdidas de velocidad respecto a la ejecución secuencial.

Solución: diapositivas 155-160 del tema 1

4. (1,25 puntos) Describir el problema de crear un hilo por cada petición en un servidor web, y explica la solución a dicho problema.

Solución: diapositivas 15, 16 y 18 del tema 5.8

5. (2,5 puntos) Programa la clase "SimpleSemaphore", que tendrá la misma funcionalidad (y los mismos tres métodos públicos) que la clase SimpleSemaphore de la biblioteca SimpleConcurrent. Prográmala utilizando únicamente la filosofía de la espera activa, y los métodos "enterMutex()" y "exitMutex()" de la biblioteca SimpleConcurrent. Te recomiendo que lo hagas en borrador y, una vez esté todo claro, lo pases a limpio (y me entregas sólo lo limpio, no el borrador).

Solución:

```
public class SimpleSemaphore {
     private volatile int permisos;

     public SimpleSemaphore (int permisosIniciales) {
          permisos=permisosIniciales;
     }

     public void adquire () {
          boolean continuar = true;
          while (continuar) {
               enterMutex();
               if (permisos > 0) {
                    permisos--;
```

```
                    continuar=false;
                }
                exitMutex();
            }
        }

        public void release() {
            enterMutex();
            permisos++;
            exitMutex();
        }
}
```

6. (4 puntos) Tenemos el problema de los filósofos comilones estudiado en clase, pero con una salvedad: una vez que un filósofo termina de comer, no puede volver a intentar empezar a comer hasta que todos los demás filósofos hayan terminado a su vez de comer. Por lo tanto, una vez que todos los filósofos ya hayan comido, ya pueden todos empezar a intentar comer de nuevo. Y así sucesivamente. Implementa esta solución en Java, utilizando los mecanismos de concurrencia que prefieras. Se valorará que la solución dé nombres descriptivos a las variables y a los métodos, y divida el código correctamente en métodos. Se valorará el grado de concurrencia alcanzado. El número de filósofos lo tendremos en una constante. El número de veces que comen los filósofos es infinito. No es necesario poner los "import". Te recomiendo que lo hagas en borrador y, una vez esté todo claro, lo pases a limpio (y me entregas sólo lo limpio, no el borrador).

Solución (una de las posibles): los filósofos se implementan utilizando cualquiera de las soluciones descritas en el ejercicio 17 del tema 2 (filósofos comilones), pero implementando una barrera cíclica con CyclicBarrier (diapositivas 63 a 65 del tema 5.5).