

Programación Concurrente y de Tiempo Real

Grado en Ingeniería Informática

Asignación de Prácticas Número 9

En esta asignación aplicará control de exclusión mutua y sincronización, utilizando para ello el API de alto nivel de Java, a diferentes situaciones que se le plantean. Documente todo su código con etiquetas (será sometido a análisis con `javadoc`). Si lo desea, puede también agrupar su código en un paquete de clases, aunque no es obligatorio.

1. Enunciados

1. Rescate la cuenta corriente de la asignación número 2, y obtenga versiones controladas mediante

- cerrojos de clase `ReentrantLock`. Guárde el código en `cCRL.java`
- semáforos de clase `Semaphore`. Guárde el código en `ccSem.java`

2. Escriba un programa que cree tres hebras concurrentes que se citen en una barrera utilizando la clase `CyclicBarrier`. Guarde el código en `barrera.java`

3. Escriba una solución al problema de los filósofos utilizando un monitor que emplee los recursos del API de alto nivel, utilizando cerrojos de clase `ReentrantLock` y variables de condición modeladas con la interfaz `Condition`. Para ello, dispone en la carpeta de la práctica de un documento que describe la «Técnica de Diseño de Monitores en Java con el API de Alto Nivel». Guarde el monitor en `filoApiAN.java`. Escriba ahora una tarea `Runnable` que sirva para modelar a los cinco filósofos, y actívelos mediante un ejecutor de capacidad fija. Guarde todo esto en `usaFiloApiAn.java`. Recuerde utilizar guardas cuando sea necesario.

4. Escriba una solución al problema del productor consumidor utilizando un monitor que emplee los recursos del API de alto nivel, empleando cerrojos de clase `ReentrantLock` y variables de condición modeladas con la interfaz `Condition`. Para ello, dispone en la carpeta de la práctica de un documento que describe la «Técnica de Diseño de Monitores en Java con el API de Alto Nivel». Guarde el monitor en `prodConAN.java`. Escriba ahora una diseño de hebras que sirva para tareas productoras y consumidoras, y actívelos mediante un ejecutor de capacidad fija. Guarde todo esto en `usaProdConAn.java`. Recuerde utilizar guardas cuando sea necesario.

5. Deseamos conocer la rapidez en tiempo de las siguientes técnicas de control de exclusión mutua en java: cerrojos `synchronized`, semáforos de clase `Semaphore`, cerrojos de clase `ReentrantLock`, y objetos `atomic`. Escriba segmentos de código con una estructura similar a la siguiente:

```
public long f(long iter){
    ini=activar-cronometro;
    for(long i=0; i<iter; i++){
        pre-protocolo;
        n++; //seccion critica
        post-protocolo;
    }
    fin=parar-cronometro;
    return(fin-ini);
}
```

Los protocolos de acceso y salida pre y post de la sección crítica los construirá a partir de la plantilla anterior, implementándolos con las técnicas ya citadas, y hará pruebas con un número creciente de iteraciones con cada técnica de control de exclusión mutua. Guarde el código en `tiempos.java`. Tome ahora tiempos y construya una gráfica que incluya las curvas de tiempo para cada técnica de control como una función del número de iteraciones. Esa curva le dirá, previsiblemente, que técnica es más rápida. Guárdela en `curva.pdf`.

2. Procedimiento de Entrega

PRODUCTOS A ENTREGAR

- Ejercicio 1: `cCRL.java` y `yccSem.java`
- Ejercicio 2: `barrera.java`
- Ejercicio 3: `filoApiAN.java` y `usaFiloApiAn.java`
- Ejercicio 4: `prodConAN.java` y `usaProdConAn.java`
- Ejercicio 5: `tiempos.java` y `curva.pdf`

MÉTODO DE ENTREGA: Tarea de Moodle.