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`)

1. Enunciados

1. Rescate la cuenta corriente utilizada en prácticas anteriores, y obtenga versiones controladas mediante

   - cerrojos de clase `ReentranLock`. 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 para ello la clase `CyclicBarrier`. Guarde el código en `barrera.java`. Comprueba que el comportamiento de la barrera es el esperado.

3. Escriba una solución al problema de los lectores-escritores bajo los mismos presupuestos fijados en la asignación número 7, utilizando ahora un monitor que emplee los recursos del API de alto nivel, que emplee 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 `lecEscAN.java`. Escriba ahora una diseño de hebras que sirva para crear tareas lectoras y escritoras, utilizando implementación de la interfaz `Runnable`. Guarde todo esto en `usaProdConAn.java`. Recuerde utilizar guardas cuando proceda.

4. Escriba una solución al problema del productor+consumidor 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 `prodConAN.java`.

Escriba ahora una diseño de hebras que sirva para tareas productoras y consumidoras, mediante herencia de la clase `Thread`. Guarde todo esto en `usaProdConAn.java`. Recuerde utilizar guardas cuando proceda.

5. Reescriba los monitores desarrollados con el API estándar de control de la concurrencia, en la práctica número 8, pero utilice ahora en su lugar el API de alto nivel.

6. 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`.

7. Pida a ChatGPT que resuelva los ejercicios 1 a 4 de esta asignación, y compruebe qué tal funciona el bot aplicando el API de Java de alto nivel para control de la concurrencia.