Prácticas Concurrencia y Distribución (2015/16)

Arno Formella, Francisco Rodríguez Martínez, David Olivieri

16 de marzo de 2016

Introducción

Según la guía docente las prácticas se evaluan con los siguientes aspectos:

(P3) Informes: Elaboración de informes (según una guia) que recogen los principales desarrollos y resultados obtenidos por el/la estudante. Partes de los informes se elaboraran en pequeños grupos.

(P4) Pruebas prácticas: Demonstración de los desarrollos e implementaciones de las tareas de programación y experimentos de estudio.

(P5) Resolución de problemas: Elaboración de algoritmos y análises con cierto nivel de formalismo para comprobar la corrección y estudios de rendimiento.

(P6) Presentaciones: Breves presentaciones orales con medios audiovisuales de desarrollo y resultados obtenidos por el/la estudiante.

Eso se traduce en la realidad de este curso a:

(P3) Informes

Informes de realización de las prácticas a entregar en pequeños grupos de 2 o 3 personas con posibles variaciones/sugerencias como puedan surgir durante las clases presenciales por la tutoría del profesor.

Un informe deberá constar de y cumplir con:

- Una identificación clara de sus autores (nombres completos y DNIs).
- Una sección para cada apartado y subapartado que se deba incluir en el informe, siguiendo la misma numeración que la de los apartados de los anuncios de las prácticas, si procede.
- Un informe no debe superar las tres (3) páginas.
- Los informes se entregan a través de la plataforma TEMA en el plazo que ahí se indica.
- Se debe entregar también un archivo comprimido con todos los ficheros del código fuente que fueron elaborados para realizar las prácticas con su documentación embebida como se genera con una herramienta adecuada (como por ejemplo `doxygen`).

(P4) Pruebas prácticas

El profesor mirará durante las prácticas los desarrollos realizados y evalua las competencias adqueridas en el diálogo con los estudiantes.

(P5) Resolución de problemas

Se distribuye problemas que se pueden resolver en el plazo fijado con la entrega del informe correspondiente.

(P6) Presentaciones

Se ofrece la posiblidad de realizar pequeñas presentaciones sobre el contenido y los resultados de las actividades previamente acordados con el profesor en clases prácticas (10 min) y se evaluan las competencias correspondientes.

Evaluación y competencias

Los informes se evaluarán y computarán dentro del apartado de realización de informes y memorias de prácticas de la materia. Para considerar que el informe alcanza los requisitos mínimos para ser considerado adecuado, será necesario que todos los apartados estén resueltos de manera correcta, y que el código elaborado para cada apartado funcione correctamente y realice la tarea pedida. Para la evaluación del informe se tendrán en cuenta la corrección del código elaborado en cada apartado, su claridad, idoneidad, eficiencia y buen diseño, control de errores y excepciones, finalización correcta, comentarios, modularidad; y las respuestas a las preguntas planteadas y mediciones realizadas (completitud y corrección de las respuestas, número e idoneidad de los ejemplos aclaratorios, número y adecuación de las pruebas realizadas, realización de las cuestiones opcionales, número de máquinas utilizadas para hacer pruebas si es el caso, adecuación y completitud de gráficas y datos presentados, conclusiones obtenidas etc.).

1. Actividad: Introducción a la concurrencia en Java

Objetivos: Adquirir conocimientos básicos sobra la forma como está implementada la concurrencia en Java, las estructuras básicas para crear hilos, y su utilización.

Metodología: Esta práctica se realizará en horas de prácticas presenciales y en horas no presenciales. El tiempo de dedicación estimada es de 2 semanas. Al tratarse de la primera práctica de esta asignatura, además de las actividades descritas se llevarán a cabo tareas de familiarización con las herramientas y métodos de trabajo, y la explicación general del enfoque de las prácticas de la asignatura.

Material adicional: Son de especial interés los siguientes enlaces

- {`http://docs.oracle.com/javase/7/docs/`}
- {`http://docs.oracle.com/javase/8/docs/`}
- {`http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html`}
- {`http://www.stack.nl/~dimitri/doxygen/index.html`}
- {`http://en.wikipedia.org/wiki/Markdown`} y todo lo que ya se presentó en clases de teoría.

Requisito general a todos los programas es: siempre termina el programa con un mensaje como *Program of excercise X has terminated*, es decir, todos los componentes del programa concurrente terminan correctamente su ejecución.

1.1. Introducción a los hilos en Java

1. Examina las dos formas que provee Java para crear un hilo: la clase `Thread` y la interfaz `Runnable`.

2. Utiliza ambas formas para crear un programa que cree y ejecute un hilo que imprima en pantalla un mensaje como *Hello world, I'm a java thread*.

   ¿Hay alguna diferencia de funcionamiento entre ambas formas? ¿A nivel de diseño, cuál te parece preferible, y por qué?

3. Modifica tu programa para que el hilo tarde aproximadamente un segundo en mostrar el mensaje. ¿Qué método usarás para ello?

4. Cuando arrancas un hilo desde el programa principal, ¿cuántos hilos hay activos? Saca una lista de los hilos activos por pantalla.

1.2. Creación de múltiples hilos

1. Escribe un programa en Java que mediante parámetros de línea de commando reciba cuantos hilos se debe crear. El programa creará y ejecutará el número de hilos indicado. Cada hilo debe imprimir en pantalla un mensaje como *Hello, I'm thread number X*, y después de un segundo,

un mensaje como *Bye, this was thread number X*, siendo *X* el valor de un contador que se va incrementando con el número de hilos creados.

2. Queremos ahora medir el tiempo que tardan en ejecutarse todos los hilos que se crean. Copia y modifica el programa anterior para que muestre este tiempo. Asegúrate de quitar cualquier código que implemente esperas en el hilo, que pudieras haber añadido anteriormente, para medir realmente sólo el tiempo de ejecución de los hilos.

3. ¿Cómo debemos hacer para asegurarnos de que la medida de tiempo sea fiable? Es decir, que realmente se mide el tiempo desde que se arranca el primer hilo hasta que todos los hilos hayan finalizado. ¿Qué errores de medida de tiempo pueden ocurrir, si no nos aseguramos de que todos los hilos han acabado?

4. Copia y modifica el programa para que cada hilo mida él mismo el tiempo que tarda en ejecutarse y guarde el valor en un atributo accesible públicamente. Una vez finalizados todos los hilos, haz que el programa principal sume todos estos valores y muestre en pantalla el resultado. ¿Este valor será mayor o menor que el tiempo global que ya estabas midiendo? ¿Por qué? ¿Te sirve esta suma para algo? ¿Qué harías tú con estos valores para tener algo interesante? Compara tus mediciones con la salida del comando `time` en Linux que te ofrece tiempo real, tiempo del *user* y tiempo del *sistema*.

5. Trata de modificar el programa para poder distinguir entre *tiempo de creación de hilos*, *tiempo de ejecución de los hilos* y tiempo de sincronzación final de los hilos*. Si tan pronto como se crea cada hilo ya se intenta ejecutar, ¿se puede distinguir entre estos tiempos? ¿Qué consecuencias tiene esto a la hora de diseñar y evaluar soluciones concurrentes a un problema?

6. Copia y modifica el programa para que los hilos o bien muestren algo por pantalla o bien que hagan por ejemplo unas operaciones matemáticas suficientemente complejas. Distingue el modo de ejecución mediante un parámetro en línea de comando. La ejecución del programa por prueba debe durar unos segundos para obtener mediciones interpretables.

   Pruébalo con un número creciente de hilos para conseguir finalmente una función de tiempo de ejecución en relación al número de hilos trabajadores (y modo de ejecución). ¿Cambia mucho el tiempo de ejecución respecto a cuándo se mostraba algo en pantalla? ¿Por qué crees que ocurre esto? ¡Interpreta detenidamente los gráficos que obtienes!

   ¡Es interesante que antes de realizar las mediciones intentes predecir los resultados por lo menos cualitativamente y averiguar si tu predicción coincida con los resultados medidos!

   ¡Es interesante realizar las mediciones en diferentes entornos, respecto a hardware, sistema operativo, y máquina virtual de Java!

1.3. Control básico de múltiples hilos

1. Estudia en detalle la utilidad del método `interrupt()` de la clase `Thread`. ¿Qué limitaciones tiene? ¿Cómo debe ser utilizado? Aunque existen otros métodos en la clase `Thread` como `stop()` o `suspend()` ¿Por qué crees que el método `interrupt()` es el método recomendado para detener un hilo?

2. ¿Qué diferencia hay entre los métodos `interrupted()` y `isInterrupted()`? Elabora un programa en Java que demuestre claramente cómo funcionan los dos y en qué se diferencian.

3. Utiliza uno de estos dos métodos (`interrupted()` o `isInterrupted()`) para elaborar un programa que lance un hilo que debe:

   - mostrar un mensaje de inicio;
   - luego debe dejar pasar unos 10 segundos, imprimiendo un mensaje aproximadamente cada segundo,
   - y luego mostrar otro mensaje de fin, y terminar.

   El usuario debe poder enviar una señal de interrupción al hilo. Si esta señal llega antes de que hayan transcurrido 5 segundos, el hilo debe mostrar un mensaje de que ha recibido la señal, y volver a ponerse a esperar, hasta terminar los 10 segundos (aproximadamente). Si llega una vez pasados 5 segundos, debe mostrar un mensaje de que ha sido interrumpido y acabar de inmediato.

2. Actividad: Concurrencia simple

Objectivos: Adquirir conocimientos básicos como distribuir el trabajo por realizar entre diferentes hilos para mejorar el rendimiento del sistema.

Metodología: Esta práctica se realizará en horas de prácticas presenciales y en horas no presenciales. El tiempo de dedicación estimada es de 2 semanas.

2.1. Repaso de la práctica anterior

1. Sabemos como lanzar cierto número de hilos basado en parámetros de la línea de comando.
2. Sabemos sincronizar los hilos al final del programa/hilo principal.
3. Sabemos medir el tiempo tanto de la parte concurrente del programa como de la parte secuencial del programa.

2.2. Uso de hilos para operaciones matriciales

1. Escribe una clase para matrices cuadradas NxN (el valor de N se pasa en el constructor), rellenándolas de valores de diferente índole (por ejemplo, aleatorios, un valor igual en todas las posiciones, una matriz de identidad, etc.) durante la construcción.
2. Implementa un método de la clase que reciba las dos matrices A y B como parámetros, y devuelve una nueva matriz C que será la suma de A+B. Cada elemento de la matriz C será la suma de los elementos de la posición correspondiente de A y B.
3. No te olvides verificar que el código realiza la operación matricial correctamente.
4. Para hacer la suma, este método deberá usar el número de hilos disponibles que trabajan concurrentemente.
5. Los hilos trabajadores puedes crear como miembros de la clase o gestionarlos de otra manera que te ocurra. El número concreto se pasa como parámetro de línea de comando al programa.
6. El programa deberá contar con la posibilidad de medir el tiempo empleado para realizar la suma.
7. Analisa el comportamiento del programa respecto al tiempo de ejecución dependiendo del tamaño de las matrices (hasta bastante grandes) y el número de hilos trabajadores (también grande). Visualiza los tiempos de ejecución gráficamente.

3. Actividad: Sincronización de hilos

Objectivos: Comprender cómo se pueden comunicar varios hilos de ejecución compartiendo variables comunes de manera segura usando sincronización. Saber definir tipos de datos simples que sean seguros en entornos concurrentes usando sincronización. Conocer las principales operaciones sobre hilos relacionadas con la sincronización.

Metodología: Esta práctica se realizará en horas de prácticas presenciales y en horas no presenciales. El tiempo de dedicación estimada es de 2 semanas.

3.1. Repaso de las prácticas anteriores

1. Sabemos como lanzar cierto número de hilos basado en parámetros de la línea de comando.
2. Sabemos sincronizar los hilos al final del programa/hilo principal.
3. Sabemos medir el tiempo tanto de la parte concurrente del programa como de la parte secuencial del programa.
4. Sabemos distribuir una carga de trabajo a varios hilos. Los hilos realizan su trabajo de forma cuasi-paralelo e independiente y se sincronizan cuando todas las tareas hayan terminado.

3.2. Un simple contador concurrente

1. Elabora una clase `Contador` que tiene un método `Incrementar(n)` que ejecuta un bucle de *n* iteraciones que incrementa una variable interna cada cierto tiempo, y al acabar devuelve el valor actual de dicha variable.

   Elabora un programa que lanza varios hilos que comparten tal objeto contador. Cada hilo, cada cierto tiempo, debe llamar al método `Incrementar(n)` con *n* adecuado e visualizar el valor devuelto por ese método.

   ¿Podemos garantizar que cuando un hilo ejecuta el método `Incrementar()`, no interferirá con otro hilo? ¿Cómo podemos solucionarlo (hay varias posibilidades)?

3.3. Introducción a la sincronización de hilos

1. Implementa una clase `A` que contenga un solo método `EnterAndWait()`. Este método debe hacer lo siguente, en este orden:

   - imprimir un mensaje indicando cual es el hilo que está comenzando a ejecutarlo;
   - luego se detendrá durante unos segundos;
   - y vuelve a imprimir otro mensaje indicando el hilo que está acabando de ejecutar el método.

2. Realiza una clase `B` que implemente `Runnable`, que reciba como parámetro un objeto de la clase `A`, y en el método `run()` simplemente llame al método `EnterAndWait()` de ese objeto.

3. Una clase con un método `main()`, en el que se crea un objeto de la clase `A`, y varios objetos de la clase `B` a los que se les pasa a todos como parámetro el mismo objeto de la clase `A`. Es decir, todos los objetos de la clase `B` compartirán el mismo objeto de la clase `A`. Después se crearán y ejecutarán el mismo número de hilos que de objetos de tipo `B` tengamos, pasándoles como parámetros los objetos de la clase `B`, de forma que cada método `run()` de cada objeto de clase `B` se ejecute en un hilo diferente.

4. ¿Cuál es el resultado? ¿Cuántos hilos pueden estar simultáneamente ejecutando el método `EnterAndWait()`?

5. Modifica el código para que solo un hilo pueda estar ejecutando el método `EnterAndWait()` en cualquier instante. Pruébalo. ¿Qué supone respecto del grado de concurrencia del código este cambio?

6. ¿Tendría el mismo efecto el hacer que sólo un hilo pueda ejecutar el método `run()` de la clase `B`? ¿Por qué o por qué no?

4. Actividad: Sincronización total de hilos

Objectivos: Comprender cómo se pueden intercalar varios hilos de ejecución compartiendo variables comunes de manera segura usando sincronización. Saber definir tipos de datos simples que sean seguros en entornos concurrentes usando sincronización. Conocer las principales operaciones y dificultades sobre hilos relacionadas con la sincronización.

Metodología: Esta práctica se realizará en horas de prácticas presenciales y en horas no presenciales. El tiempo de dedicación estimada es de 2 semanas.

4.1. Repaso de las prácticas anteriores

1. Sabemos como lanzar cierto número de hilos basado en parámetros de la línea de comando.
2. Sabemos sincronizar los hilos al final del programa/hilo principal.
3. Sabemos medir el tiempo tanto de la parte concurrente del programa como de la parte secuencial del programa.
4. Sabemos distribuir una carga de trabajo a varios hilos. Los hilos realizan su trabajo de forma cuasi-paralelo e independiente y se sincronizan cuando todas las tareas hayan terminado.
5. Sabemos usar variables simples compartidos.
6. Sabemos usar las posibilidades de Java para sincronizar hilos de forma simple.

4.2. Un juego: ping-pong-múltiple

Queremos hacer un programa que simule un juego de ping-pong (pero con mucho más jugadoras que pasan la pelota entre ellas en un orden preestablecido). Para ello hemos de definir una clase llamada `Ping` que simulará una jugadora con su método `run()` que será ejecutado en su hilo. Es necesario que cada hilo saque un mensaje indicando qué jugadora juega.

Para realizar esta actividad:

1. Experimenta con el material presentado en clases de teoría.
2. Usa—parecido al juego real—un objeto que haga de *pelota*, de forma que cada jugadora sepa cuando tiene que jugar, que será cuando la pelota esté *en su posesión*.

   Averigua para ello cómo se puede usar un objeto en Java para sincronizar varios hilos.
3. Usa el hilo principal con el papel de *árbitro*, es decir, es el hilo que inicia y termina el partido.
4. Implementa diferentes criterios de parada, por ejemplo: número de jugadas, tiempo de jugadas, interrupción... El programa debe terminar todos los hilos participantes de forma explícita. Como siempre, el hilo principal termina el programa con el último mensaje.
5. Utiliza en tu solución los métodos `wait()`, `notify()` y `notifyAll()` de Java. ¿Observas diferencias entre `notify()` y `notifyAll()`?

6. Queremos que el árbitro decida quien será el hilo que comience a jugar.

7. Argumenta de forma semi-formal que tu programa es correcto y garantiza la semántica del juego. ¿Cómo puedes comprobar automáticamente que la salida del programa es la deseada?

8. Mide el tiempo de ejecución—con un criterio de terminación razonable—de tu partido variando el número de jugadoras de unas pocas a unas miles. ¿Qué resultado esperas para un número constante de *pasar-pelota* independientemente del número de jugadores? ¿Qué dicen tus mediciones? ¿Cuál es el método más eficiente?

9. Si se interrumpe un hilo que participa en el juego ¿qué pasa con los demás? ¿Puedes sugerir un remedio?

10. ¿Funciona tu programa concurrente con una sola jugadora también? Si no lo hace ¿qué deberías cambiar?