Ideas para pensar y practicar la concurrencia

Concurrencia
GRADO EN INGENIERÍA INFORMÁTICA/ GRADO EN MATEMÁTICAS E INFORMÁTICA/
2BLE. GRADO EN INFORMÁTICA Y ADE
Universidad Politécnica de Madrid

11 de febrero de 2019

Creación de tareas

Ejercicio 1 (SleepSort)**.** En el folclore informático se conoce como *SleepSort* a un curioso (pseudo)algoritmo de ordenación que hace uso de la concurrencia. La idea es la siguiente: Inicializamos un vector con n números positivos que queremos ordenar. Posteriormente lanzamos n tareas que hacen lo siguiente: leen una posición del vector (una distinta cada tarea) y esperan un tiempo proporcional al valor del número en esa posición. Tras esa espera lo imprimen. Teóricamente, deberíamos obtener los números impresos en orden.

Adapta el código de tu primera entrega para que implemente esta idea y juega con el factor de proporcionalidad para descubrir por qué no usamos este algoritmo en la práctica.

Ejercicio 2 (Integración concurrente)**.** Modifica tu solución a la primera entrega para efectuar una integración aproximada de una función real en un intervalo. La idea es subdividir el intervalo en n subintervalos y sumar el resultado de integrar la función en cada uno de los subintervalos.

Presta atención a cómo efectuar esa suma de manera que no aparezcan situaciones de carrera.

Ejercicio 3 (Supercomputación para pobres)**.** Es muy probable que estés ejecutando Java en un ordenador con varios núcleos y que la máquina virtual de Java sea capaz de sacar partido de ello. Modifica el código del ejercicio anterior para calcular el tiempo dedicado a la integración de cada subintervalo y el que se tarda en realizar la integración completa. Encuentra el valor óptimo de n.

Ejercicio 4 (Barreras rápidas)**.** Hemos visto cómo un hilo puede esperar por la terminación de n tareas que ha creado y lanzado mediante la invocación de `join()` sobre cada una de ellas. Esto es lo que se denomina una *barrera* de n tareas. ¿Se te ocurre una manera de acelerar esto? ¿En qué situaciones sería más rápido? ¿En qué situaciones sería útil?

Un día en las carreras

Ejercicio 5 (Deadlock Empire)**.** Accede a `https://deadlockempire.github.io` y realiza los tutoriales 1 y 2, que tienen que ver principalmente con el entrelazado de operaciones no atómicas y las situaciones de carrera.

Ejercicio 6 (Situaciones de carrera y *bytecode* de Java)**.** Hemos visto que la ejecución concurrente de operaciones aritméticas no atómicas sobre una variable compartida puede dar lugar a situaciones de carrera. Localiza en el código resultante de compilar tu solución a la entrega 2 las instrucciones responsables de la situación de carrera entre incrementadores y decrementadores.

Ejercicio 7 (BubbleSort modo "sin reglas")**.** La ordenación por burbuja (*BubbleSort*) es un método de ordenación de una secuencia que consiste en intercambiar pares adyacentes de elementos desordenados hasta que la secuencia esté ordenada. Cuando lo implementamos de manera secuencial, el orden en que escogemos los pares a comparar y permutar define algoritmos que resultan ser variaciones del tema fundamental — *SelectionSort*, *ElevatorSort*, etc.

En principio, nada impide realizar la ordenación de varios pares de la secuencia de manera simultánea. En este ejercicio te pedimos que programes una pseudoversión concurrente de la ordenación por burbuja donde el hilo principal asigna valores a un vector de n enteros y luego lanza $n-1$ hilos permutadores que se encargan de estar permanentemente comparando un par de posiciones consecutivas y permutándolas si es necesario. El hilo principal, tras lanzar los n obreros, se dedica a mostrar por pantalla los contenidos de la secuencia.

Como lo que queremos es provocar situaciones de carrera, no debes programar ningún mecanismo de sincronización. Tampoco es necesario que programes la terminación de las tareas.

Ejercicio 8 (Entrelazados)**.** Muestra un ejemplo lo más sencillo posible de cómo entrelazar la ejecución de dos permutadores para que se corrompa el vector.

Ejercicio 9 (Corrupción en AED)**.** Hemos visto en clase cómo corromper un contador compartido con operaciones de incremento o decremento. Hemos visto en los dos ejercicios anteriores cómo corromper una secuencia de enteros entrelazando permutaciones. Escribe un pequeño programa que corrompa una estructura de datos que hayas usado en AED (pilas, colas, árboles, etc.) mediante accesos concurrentes sin sincronización.