Concurrencia y Paralelismo

Grado en Informática 2020

Práctica 1 – Intercambio de Variables

En esta práctica vamos a implementar un sistema que permita intercambiar los valores de un array que se comparte entre varios threads. Cada thread ejecuta un bucle donde escoge dos posiciones de forma aleatoria e intercambia los valores que hay en ellas. Para evitar que se produzcan problemas por las operaciones concurrentes sobre el array vamos a controlar el acceso al mismo.

El programa admite las siguientes opciones:

- `-i n`, para controlar el número de iteraciones que cada thread va a hacer, es decir, cuantas veces va a intercambiar valores.
- `-t n`, para especificar el número de threads que vamos a crear.
- `-b n`, para controlar el tamaño del array compartido.
- `-d n`, para añadir un retraso entre las distintas operaciones que hace cada thread. Incrementando este valor podemos forzar a que los threads cedan la cpu y hace más probable que podamos ver problemas en nuestro código.

El programa crea el array de tamaño `n` y lo rellena con los números de `0` a `n-1`. Antes de lanzar los threads se imprimen los valores de cada posición del array, y se vuelven a imprimir después de que todos hayan terminado. En una ejecución correcta deberíamos ver que los valores finales son una permutación de los iniciales:

```
$ ./swap -i 3 -t 3
creating 3 threads
Buffer before: 0 1 2 3 4 5 6 7 8 9
Thread 0 swapping positions 3(3) and 4(4)
Thread 1 swapping positions 9(9) and 6(6)
Thread 2 swapping positions 7(7) and 5(5)
Thread 0 swapping positions 2(2) and 3(4)
Thread 2 swapping positions 3(4) and 2(2)
Thread 1 swapping positions 7(5) and 5(7)
Thread 0 swapping positions 6(9) and 0(0)
Thread 2 swapping positions 5(5) and 9(6)
Thread 1 swapping positions 2(4) and 7(7)
Buffer after:  9 1 7 2 3 6 0 4 8 5
```

Si tenemos accesos concurrentes sin controlar de forma correcta veremos que hay valores que se repiten en el array:

```
$ swap -i 3 -t 3
creating 3 threads
Buffer before: 0 1 2 3 4 5 6 7 8 9
Thread 0 swapping positions 9(9) and 4(4)
Thread 1 swapping positions 1(1) and 8(8)
Thread 2 swapping positions 0(0) and 9(9)
Thread 0 swapping positions 1(8) and 8(1)
Thread 1 swapping positions 0(4) and 7(7)
Thread 2 swapping positions 7(7) and 4(9)
Thread 0 swapping positions 3(3) and 5(5)
```

```
Thread 1 swapping positions 8(8) and 9(0)
Thread 2 swapping positions 6(6) and 3(3)
Buffer after:  7 1 2 6 7 3 5 4 0 8
```

Partiendo de este código se pide:

Ejercicio 1 (**Proteger los intercambios de variables con un mutex**) Tal y como está el código los accesos se realizan sin protección. Ejecutando el código varias veces se puede ver que hay valores que empiezan a aparecer repetidos (jugando con el retraso entre operaciones se puede forzar a que ocurra con más facilidad). Añada un mutex para controlar el acceso al array. Este mutex no debería ser una variable global, utilice los parámetros de los threads para compartirlo.

Ejercicio 2 (**Proteger los intercambios de variables con un mutex por posición del array**) En el ejercicio anterior añadimos un mutex para controlar los accesos concurrentes al array. El programa funciona correctamente, pero al haber un solo mutex cada vez que un thread quiere hacer un intercambio tiene que bloquear el acceso a todo el array, aunque podamos tener threads que vayan a intercambiar posiciones distintas. Para ganar en eficiencia vamos a proteger cada posición con un mutex. Para hacer el intercambio de dos variables tendremos, por tanto, que bloquear los dos mutex correspondientes a esas variables.

Tenga en cuenta que en este apartado es necesario bloquear dos mutex para hacer cada intercambio, y que esto puede provocar interbloqueos. Aplique alguna de las técnicas vistas en clase para evitarlo.

Compruebe con el comando `time` que el programa ejecuta los intercambios en un tiempo menor que con un solo mutex.

Implemente este ejercicio en un nuevo fichero swap2.c

Ejercicio 3 (**Implementar iterations global**) El programa acepta un argumento `iterations` para controlar el número de iteraciones que hace el programa. En el código que se proporciona cada thread recibe el número de iteraciones entre sus argumentos, y ejecuta ese número de intercambios de variables. Cambie lo que sea necesario para que el número de iteraciones sea global, es decir, que se realicen exactamente `iterations` intercambios **entre todos** los threads.

Implemente este ejercicio en un nuevo fichero swap3.c partiendo de la implementación del ejercicio anterior.

Entrega

La fecha límite de entrega es el 14 de febrero. Para la entrega deberá crearse un proyecto `cp-p1` en el servidor de gitlab de la facultad: https://git.fic.udc.es. El proyecto debe ser privado, e incluir al profesor de prácticas del grupo para que pueda acceder para la corrección. Puede consultar más información sobre gitlab en https://wiki.fic.udc.es/_media/cecafi:software:gitlab.pdf

Cada ejercicio deberá entregarse en su propia rama, llamadas e1, e2 y e3 respectivamente. Incluya todos los ficheros necesarios para compilar la práctica.