Concurrencia y Paralelismo

Grado en Informática 2022

Práctica 1 – Cuentas de Banco

En esta práctica vamos a implementar un sistema que simula un banco donde se realizan operaciones contra varias cuentas de forma simultánea. Cada thread simula una persona realizando un ingreso en una cuenta. Tanto la cuenta como la cantidad se generan aleatoriamente.

Los accesos al valor depositado en las cuentas forman la sección crítica. Para forzar que los threads coincidan y sea más fácil ver problemas de concurrencia cada thread realizará varios ingresos, y esperará un tiempo entre la consulta del saldo y el incremento.

El programa admite las siguientes opciones:

- `-i n`, para controlar el número de iteraciones que cada thread va a hacer, es decir, cuantos ingresos hace.
- `-t n`, para especificar el número de threads que vamos a crear.
- `-a n`, para controlar el número de cuentas en el banco.
- `-d n`, para cambiar el tiempo de espera entre operaciones.

El programa crea las cuentas y las inicializa con saldo 0 . Durante la ejecución imprime las operaciones de ingreso que va realizando cada thread. Al finalizar se muestra el total ingresado por cada thread, y el balance total de cada cuenta. Si los accesos a la sección crítica están protegidos correctamente, la suma de ingresos debería ser igual a la suma de los balances de las cuentas.

```
$ ./bank -t 3 -i 2 -a 3
creating 3 threads
Thread 1 depositing 7 on account 2
Thread 0 depositing 0 on account 2
Thread 2 depositing 19 on account 2
Thread 0 depositing 7 on account 1
Thread 2 depositing 12 on account 0
Thread 1 depositing 4 on account 1

Net deposits by thread
0: 7
1: 11
2: 31
Total: 49

Account balance
0: 12
1: 11
2: 26
Total: 49
```

Si tenemos accesos concurrentes sin controlar de forma correcta veremos que los totales no coinciden, y si hacemos la suma de los ingresos realizados en cada cuenta no coincidirán con el saldo final:

```
$ ./bank -t 3 -i 2 -a 3
creating 3 threads
```

```
Thread 0 depositing 11 on account 2
Thread 2 depositing 15 on account 2
Thread 1 depositing 13 on account 0
Thread 0 depositing 3 on account 1
Thread 2 depositing 19 on account 0
Thread 1 depositing 16 on account 1

Net deposits by thread
0: 14
1: 29
2: 34
Total: 77

Account balance
0: 32
1: 16
2: 15
Total: 63
```

Partiendo de este código se pide:

Ejercicio 1 (**Proteger el acceso a cada cuenta del banco con un mutex**)

Añada un mutex por cuenta para controlar el acceso. Estos mutex no deben ser variables globables, sino que deberían estar en la estructura que protegen.

Ejercicio 2 (**Añadir transferencias**)

Añada threads a la simulación que hagan una transferencia entre dos cuentas de una cantidad entre 0 y el saldo total de la cuenta origen. Debería haber tantos como threads ingresando, y deberían hacer el mismo número de iteraciones que éstos. Imprima un mensaje cuando la transferencia se realiza, con las cuentas implicadas y la cantidad.

Haga que estos threads se ejecuten cuando hayan terminado los depósitos.

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

Ejercicio 3 (**Totales**)

Para verificar que las transferencias se protegen de forma adecuada, añada un hilo que recorra las cuentas calculando e imprimiendo el saldo total repetidamente. Este thread debería iniciarse al mismo tiempo que las transferencias, y debería estar ejecutándose hasta que las transferencias terminen. Si el número de mensajes que imprime es muy alta, añada una espera al final de cada recorido de todas las cuentas igual al tiempo de espera entre operaciones.

El acceso al saldo de cada cuenta debería hacerse bloqueando el mutex correspondiente. Asegúrese de que este thread termina sin dejar ningún mutex bloqueado.

Si las transferencias se hacen de forma correcta, el saldo impreso por este thread debería ser siempre el mismo.

Ejercicio 4 (**Iteraciones**) Ahora mismo cada thread realiza un número de operaciones especificado con la opción -i. Cambie el comportamiento para que ese número de iteraciones sea el realizado entre todos los threads de cada tipo.

Es decir, si se especifica que hay que hacer 100 iteraciones, deberían hacerse 100 depósitos y 100 transferencias entre todos los threads.

Entrega

La fecha límite de entrega es el 20 de febrero. El código inicial está disponible a través de github classroom https://classroom.github.com/a/4l21-BXC, donde cada estudiante podrá crear un repositorio para acceder al código y subir las sucesivas implementaciones de los ejercicios. Cubra su nombre y login en el fichero authors del repositorio.

La corrección se hará sobre el contenido del repositorio al final del día 20 de febrero.