Concurrency and Paralellism

Degree in Computer Science 2022

Lab Assignment 1 – Bank Accounts

We are going to simulates a bank with several accounts where operations are performed concurrently. Each thread represents a person making deposits of a random amount on a random account.

Accounts are shared, so any read/write operation on the balance will create a critical section. In order to make concurrency problems easier to observe, we will increase the probability of having several threads in the critical section at the same time by:

1. Making each thread loop so that if performs several deposits.
2. Sleeping after every operation in the critical section, so that each thread spends more time there.

The program accepts the following command line arguments:

- `-i n` sets the number of deposits each thread will perform.
- `-t n` sets the number of threads.
- `-a n` sets the number of accounts in the bank.
- `-d n` sets the delay (in μs) between operations in the critical section.

All the account balances start at 0. When a thread makes a deposit, it prints a message with the account and amount. After the threads have finished, the program will print the total deposits made by each thread, and the final account balances. If we protect the critical section successfully the sum of the deposits should be equal to the sum of the account balances.

```
$ ./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
```

If we have concurrent execution in the critical section our totals will be different, and for some accounts, the sum of all the deposits will be different from the balance.

```
$ ./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
```

Add the following features:

Exercise 1 (**Protect each account with a different mutex**)

Add a mutex to each account to manage concurrent accesses. Do not add the mutexes as global variables. Instead, add them to the account structure.

Exercise 2 (**Add transfers**)

Add a new type of thread that performs transfers between two random accounts. The amount should be between 0 and the balance of the first account. Use the same number of operations and transfering threads as we had for deposits in the previous exercise. When a thread performs a transfer it should print its number, the source and destinations accounts, and the amount.

Start the transfer threads once all the deposits have been completed.

Note that we have to lock two mutexes to transfer money between two accounts, and that makes deadlocks possible if you are not careful.

Exercise 3 (**Balance**) Add a thread that sums the balances of the accounts and prints the total repeteadly while the transfers are in progress. If you notice that the number of messages is high add a delay at the end of each printing loop.

Lock the mutex of each account as you add it to the total. Make sure that the thread is not holding any mutexes when it finishes.

If the transfers are working correctly the total printed should always be the same.

Exercise 4 (**Iterations**) Currently, the number of iterations is the number of operations (deposits or transfers) performed by each thread. Change it so that it is the total number of deposits and transfers performed between all the threads.

For example, if we ask for 100 iterations, there should be 100 deposits and 100 transfers in total insted of 100 * number of threads.

Submission

Assignments are due on February 20. Register for the assignment in github classroom at https://classroom.github.com/a/4l21-BXC. When you register github will create a repository for you with the starting code for the assignment. Push your solutions to that repository. Please fill in your name and login in the authors file.

We will review your submissions during the following week (february 21).