Catargiu Georgiana Ecaterina

Group 932/1

Parallel and Distributed programming

Laboratory1: “Non-cooperative” multithreading

**Setup**

Asus VivoBook S14 - Processor: Intel® Core™ i5-8250U CPU @ 1.60GHz 1.80Ghz

**Problem chosen: 2. Bank Accounts:**

At a bank, we have to keep track of the balance of some accounts. Also, each account has an associated log (the list of records of operations performed on that account). Each operation record shall have a unique serial number, that is incremented for each operation performed in the bank.

We have concurrently run transfer operations, to be executer on multiple threads. Each operation transfers a given amount of money from one account to some other account, and also appends the information about the transfer to the logs of both accounts.

From time to time, as well as at the end of the program, a consistency check shall be executed. It shall verify that the amount of money in each account corresponds with the operations records associated to that account, and also that all operations on each account appear also in the logs of the source or destination of the transfer.

**Requirements:**

1. The problems will require to execute a number of independent operations, that operate on shared data.
2. There shall be several threads launched at the beginning, and each thread shall execute a lot of operations. The operations to be executed are to be randomly choosen, and with randomly choosen parameters.
3. The main thread shall wait for all other threads to end and, then, it shall check that the invariants are obeyed.
4. The operations must be synchronized in order to operate correctly. Write, in a documentation, the rules (which mutex what invariants it protects).
5. You shall play with the number of threads and with the granularity of the locking, in order to asses the performance issues. Document what tests have you done, on what hardware platform, for what size of the data, and what was the time consumed.

**Description**

In the parallel run we can start more transactions at the same time, every thread performing an optimal number of transactions, whereas in the sequential run transactions are performed one by one.

*Problem Solution* - Based on the number of threads, the accounts and the transactions were randomly generated so that each thread performs transactions in a given range [start, stop), computed based on the number of transactions and the number of threads given by the user.

**Implementation**

Classes used:

- *BankAccount* (serialNumber, initialAccountBalance, currentAccountBalance, logstTransactions, id)

- *Transactions* (transactionNumber, transactionId, firstAccount, secondAccount, amount, transactionType( false->sends money, true->receives money)

- *Repository*(accounts, allMutexes)

- *Controller*(threads, repository, transactions)

***Mutex invariant***: *ReentrantLock* on the accounts, which performs a transaction at a given time for securing parallel and correct access to the data, protecting the modification of the balance.

***Consistency check*:** After the parallel execution, the sequential execution takes place, and the final values of the accounts is compared for correctitude validation.

***Granularity:*** The access on two accounts was blocked at a time, there being only one mutex for all operations.

**Tests**

1. **First test:**

* Size of data: 10000 accounts and 250 transactions
* Threads: 5
* Time: 2,68s

1. **Second test:**

* Size of data: 20000 accounts and 250 transactions
* Threads: 5
* Time:3,01s

1. **Third test:**

* Size of data: 20000 accounts and 2500 transactions
* Threads: 200
* Time:

1. **Fourth test:**

* Size of data: 20000 accounts and 2500 transactions
* Threads: 1
* Time: 3,15s

1. **Fifth test:**

* Size of data: 20000 and 500 transactions
* Threads: 10
* Time: 5,56s