Diana Noveanu
gr. 935

First PDP Lab Assignment Documentation

(1) 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 chosen, and with randomly chosen 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 assess 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.

(2) Problem Statement (no. 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 executed on multiple threads. Each operation transfers a given amount of money from one account to another 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 in the logs of the source or destination of the transfer.

(3) Setup

MacBook Pro (Retina, 13-inch, Early 2015)
Processor 2.7 GHz Intel Core i5
Memory 8 GB 1867 MHz DDR3
Graphics Intel Iris Graphics 6100 1536 MB

Language of implementation: Java

(4) Description

Compared with the sequential run, 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 (see functions in Helper file)
- Each thread performs transactions in a given range [start, stop), computed based on the number of transactions and the number of threads

(5) Implementation

Classes used:

- Account (name, id, balance)
- Log (id, serial number, list of records)
- Multithread (extends Java Thread; hashmap of accounts, record, mutex, hashmap of logs)
- Record (id, transaction)
- Transaction (id, ids of both user accounts, timestamp, amount, type)

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.

(6) Tests Conducted

a. First test

- Data size: 10000 accounts and 250 transactions
- Number of threads: 5
- Time consumed:

```
Parallel time: 0.0032
Sequential time: 0.006200000000000001
Consistency: true
```

b. Second test

- Data size: 20000 accounts and 2500 transactions
- Number of threads: 5
- Time consumed:

```
Parallel time: 0.0058
Sequential time: 0.008799999999999999
Consistency: true
```

c. Third test

- Data size: 20000 accounts and 2500 transactions
- Number of threads: 200
- Time consumed:

```
Parallel time: 0.005
Sequential time: 0.007000000000000001
Consistency: true
```

d. Fourth test

- Data size: 20000 accounts and 2500 transactions
- Number of threads: 1
- Time consumed:

```
Parallel time: 0.0056
Sequential time: 0.0064
Consistency: true
```

e. Fifth test

- Data size: 20000 accounts and 500 transactions
- Number of threads: 10
- Time consumed:

```
Parallel time: 0.0024000000000000007
Sequential time: 0.0064
Consistency: true
```

Many other tests can be conducted, yet the results (the correct execution of the program) would almost certainly prove to be the same as the ones presented in this documentation.

Conclusively, parallel execution (when done right) will always yield better (optimal) results when compared to sequential execution.