Candet Andrei-Gabriel
Group 932/1
Parallel and Distributed Programming

Lab. 1 – "Non-cooperative" multi-threading

1. 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 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.

2. Machine Specifications

Processor: Intel® Core™ i5-5200U CPU @ 2.20GHz (4 CPUs), ~2.2GHz
Cores: 2
Threads: 4
Memory: DDR3 8 GBytes

3. My Solution

Account Class:	Log Class:
- account_id : int - balance : float - owner : string - logs : vector<Logs>	- id : long - description : string - target : int - source : int - amount : float - target_balance : float - transaction : Transaction<enum> {send, receive }

The main method manages the transactions and the threads. The program can be started as following:

output -w <number> -t <number> -a<number>

- w = workers (threads)
- t = transactions (operations/worker or threads)
- a = accounts

Candet Andrei-Gabriel
Group 932/1
Parallel and Distributed Programming

At the beginning, each account is created with 500 monetary units. Then, the integrity thread is started and detached. From time to time, the threads sum up the money on each account and checks if the sum is equal to A*500 (account numbers*500 monetary units).

Then, the T threads are created and started. These threads work synchronously by a global mutex. This mutex is also acquired by the checking thread, just to check the sums. Before the threads start and at the end, the main method remembers the time, so we can measure the time performance and elapsed time.

4.

Threads	Transactions/Threads	Accounts	Time	Observations
1	1000	4	0.0034388s	Increasing the number of threads increases the execution time.
2	1000	4	0.0021494s	
3	1000	4	0.0033056s	
4	1000	4	0.0041474s	
6	1000	4	0.0131346s	
8	1000	4	0.0100419s	

The most efficient run is the one with 1 worker thread. At some point, when the threads number is bigger than the number of available cores, it becomes more expensive to create them than to use them.