ECS713P/ECS713U - Functional Programming

Coursework - 4 : Report

Aim

To simulate a concurrent "banking" system. The main program should spawn ten "customer" threads, and each of these threads model a bank account with a starting balance of £1000. The customers should then (at random intervals) choose one of the other customers (at random) and transfer a random amount of money (between £10 and £50) into their account.

Executing the Program

Run the `stack build` to build the project and then `stack exec bank-exe` to execute it. Read more about that here in this project's Github readme.

Control Flow and Explanation

- The program starts with starting the worker threads to create random transactions - which is done via forking the method to create transactions in a thread. We create 10 threads(each thread for each customer) and perform 100 transactions per thread. Mainly three things happen in a single forking(or in a single thread)
    - The data for transactions are being created randomly - two accounts to be debited and credited from (say A-> B) and the amount to be credited. Appropriate checks are added to that too.
    - The accounts are checked whether it exists, if not we create one with a minimum balance of £1000.
    - Now the transfer process happens, the amount is debited from A and credited to B. here it checks whether A!=B and after debiting A doesn't have a *negative balance*. If everything is fine, the transfer is initiated *atomically.*
- Now the main program waits for all the workers to finish their jobs
- And the amounts are mapped and printed in readable form.
  P.S - The initial sum of all accounts is £10,000. So after these transactions the sums must be the same, ie £10,000. This can be used to verify whether the transactions are correct or not.

Design Choices and Issues faced

- We have used both *TVar* and *MVar* to avoid a few *deadlocking issues* which were caused in the initial stages of development. Source post.
- We tried implementing the transactions more efficiently and real like by introducing the atomicity principle in Databases. It basically follows an all or nothing approach. It was implemented using the STM module. Source link.
- Since we have used TVar Mapping (*Map.Map*) to create the custom datatype *Customer*, we had to use other mapping functions too like *Map.assocs, mapM, Map.empty* etc. Source posts - one and two.
- *Issue (pending)* - The parallel mapping was tried to be implemented using *parallel_* and *parMap* to be used instead of replicating the forkIO. But it was creating issues in the existing computations and even running into locks (and even race conditions). Source post.