Lab Session 3:
A concurrent bank server
21653-Sistemes Operatius

1 Introduction

The objective of this lab session is to continue practicing multi-threading and socket communication concepts. We will implement a concurrent bank server. You will need to deliver three files bankMonitors.c, bankServer.c and bankClient.c solving the tasks of the lab session with a small document describing your solution.

2 A Threaded Bank program

We have provided you with a file bank.c which presents a solution without synchronization of the basic bank operations. The bankMonitors.c will be based on the code bank.c, will manage a bank database and will serve bank operations from clients: deposit, withdraw and transfer. As you can see in bank.c, each account is a struct like data in the code:

```
typedef struct sAccount {
        int id;
        double balance;
        bool bInUse; // the account is being used?
        pthread_mutex_t mutex; // lock to use/modify
        pthread_cond_t  freeAcc; // we wait when inUse
} Account;
```

The method bankInit initializes the bank with a fixed amount to each account: 5 accounts of 100 euros each. In addition we have provided three basic operations:

- withdraw (which takes an amount of money from an account and returns true if it was possible and false otherwise),
- deposit (which adds a quantity of money to an account)
- transfer which given two accounts (from and to) withdraws an amount of money from the first account and deposits in the second one.

3 Client / Server application

We have provided you with the code of a client.c and server.c. Take a look to the code. Remember the steps of the server: call to method socket, bind, listen, accept (blocking call until a client connects). Remember the steps of the client:

- call to method socket.

  ```
  int socket(int domain, int type, int protocol);
  ```

  Domain is the addressing domain of socket, which for our purposes is an internet socket or AF_INET, type would be SOCK_STREAM.

- Call to method connect: given a socket socket and a socket address address, try and connect the socket to that foreign address.

  ```
  int connect(int socket, const struct sockaddr *addr, socklen_t addr_len);
  ```

Compile it and test it, each process launched in different consoles:

```
./server 9000
./client 127.0.0.1 9000
```

4 Exercice 1 : Synchronized Bank

Based on the provided file bank.c you will implement file bankMonitors.c which will create a number of threads and each thread will do 10000 transfer random operations between random accounts. Test that everything works correctly and that the total bank budget is preserved.

5 Exercice 2 : Bank server and client

Based on files bankMonitors.c, server.c and client.c you will need to implement a threaded bank server which serves client transfer operations. Whenever a server receives a client connection it creates a thread and serves its transfer operation that is transferred via the socket using the write system call.

We will establish a code for a client to send a transfer operation to the server. You have to options: you can either transform the operation into a string char pointer code :

```
sprintf(buff,"T%02d%02d%02.2f",inAcc,toAcc,amount);
```

or to directly write an operation code and ints for the account numbers and a float for the amount:

```
char c = 'T';
write(sfd, &c, sizeof(char));
write(sfd, &inAcc, sizeof(int));
write(sfd, &toAcc, sizeof(int));
write(sfd, &amount, sizeof(float));
```