AOS Project 1

Name: Intekhab Naser ID: ZC11577

Design Document

Program Design: The program model consists of a Client-Server architecture.

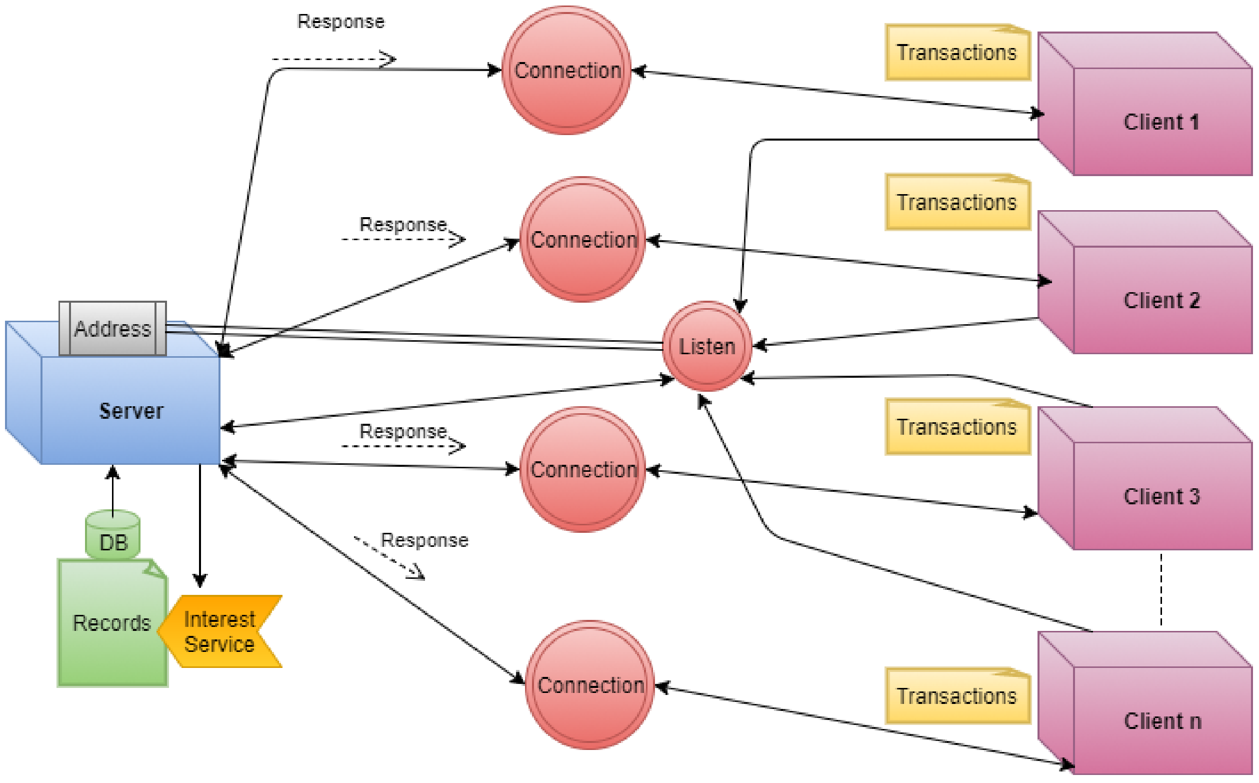


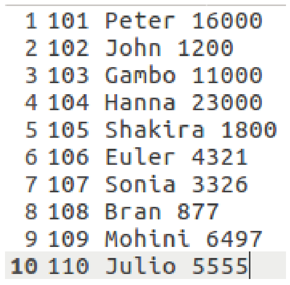
Fig. 1. Block Diagram of Centralized Multi-User Concurrent Bank Account Manager

Server: The server acts as a centralized Bank which maintains a coherent record of all bank account holders in its database. Server can accept multiple client requests (amount deposit or withdrawal) simultaneously and it can process these requests concurrently. Server validates all client requests for safe amount withdrawal limits and valid account numbers, before processing them, to maintain the correctness and safety of records. It also takes care of concurrent access to individual records using locking by mutexes, to maintain consistency of records. Further, the server periodically deposits interest to each eligible account, based on fixed rate of interest.

Clients: Clients act as bank customers in the form of ATM machines, Online Transactions, etc. There may be multiple clients requesting the server to deposit/withdraw funds to/from an account number. These requests are sent to the server periodically based on some timestamps associated with each request. The success or failure status (with reason for failure) of each request is returned from server and printed by client. To indicate the completion of all transaction requests, client sends a message ‘end’ to server so that the connection may be terminated.

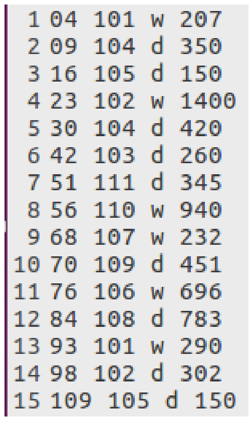
The structure of typical Records.txt and Transactions.txt files may be shown as:

Records.txt



```
101 Peter 16000
102 John 1200
103 Gambo 11000
104 Hanna 23000
105 Shakira 1800
106 Euler 4321
107 Sonia 3326
108 Bran 877
109 Mohini 6497
110 Julio 5555
```

Transactions.txt



```
04 101 w 207
09 104 d 350
16 105 d 150
23 102 w 1400
30 104 d 420
42 103 d 260
51 111 d 345
56 110 w 940
68 107 w 232
70 109 d 451
76 106 w 696
84 108 d 783
93 101 w 290
98 102 d 302
109 105 d 150
```

The data structure of typical Bank Record and Client Transaction Request may be shown as:

Bank Account Records

```
struct bank_record
{
        long acc_number;
        string name;
        double balance;
        pthread_mutex_t lock;
}
```

Client Transaction Requests

```
struct client_requests
{
        float start_time;
        long acc_number;
        char type;
        double amount;
}
```

How it works:

All the commands needed to successfully run this program are handled using the make command from Makefile file. The following commands may be used:

(i) make clean: This will delete all the object files in the directory.
(ii) make compile: This will compile the files server.cpp and client.cpp, and create two linux executables, server and client.
(iii) make run_server: This will start the server process and begin execution.
(iv) make run_client: This will start the client process and begin execution.

If the IP address or port number needs to be changed for client and server, the changes may be made in the Makefile file.

Design Tradeoffs:

There are a few design tradeoffs made in this program which may be listed as follows:

(i) New accounts can NOT be created in this program. If needed, new entries can be added in the Records.txt manually.

(ii) The program does not handle any invalid transaction requests apart from account number and amount withdrawal limits. Eg. An undefined transaction type ‘k’ instead of a deposit ‘d’ or withdrawal ‘w’ is not handled by the program.

Multiple threads and synchronization:

The program is designed to handle multiple clients and concurrent access of shared resources in the following ways:

(i) The server runs in endless loop and continuously keeps listening for client connection requests and spawns a thread for each new connection accept.

(ii) Mutex locks are defined separately for each account in the bank records. If any client thread needs access to a particular account, it needs the corresponding lock.

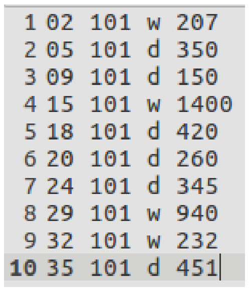
Program Evaluation Test Cases:

Case (i): Concurrency Control and Correctness Evaluation

To evaluate the concurrency control and correctness, we design a testcase with two clients. Both clients run concurrently, use same transactions.txt file and perform all transactions on a single account e.g. Account number 101.

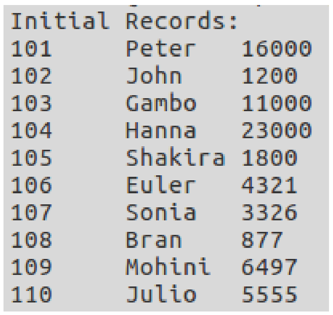
Following Transactions.txt file is used for evaluation:

Transactions.txt



```
02 101 w 207
05 101 d 350
09 101 d 150
15 101 w 1400
18 101 d 420
20 101 d 260
24 101 d 345
29 101 w 940
32 101 w 232
35 101 d 451
```

Records.txt

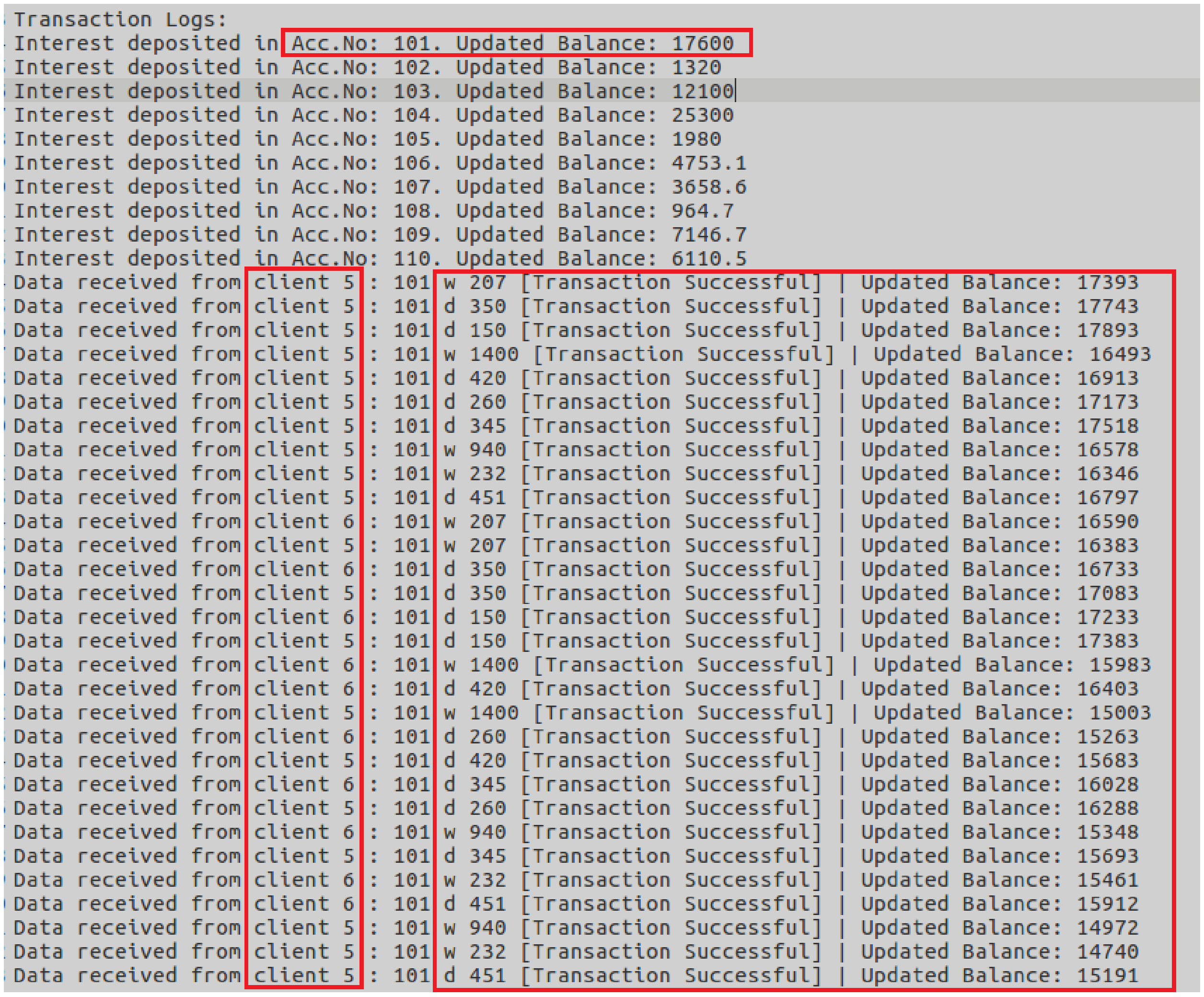


```
Initial Records:
101     Peter   16000
102     John    1200
103     Gambo   11000
104     Hanna   23000
105     Shakira 1800
106     Euler   4321
107     Sonia   3326
108     Bran    877
109     Mohini  6497
110     Julio   5555
```

The Initials Records before clients perform any transactions, can be noted as follows:

Note that, the initial balance amount of account number 101 is ‘16000’

Now, we run both clients simultaneously and evaluate the result (Concurrency and Correctness)



```
Transaction Logs:
Interest deposited in Acc.No: 101. Updated Balance: 17600
Interest deposited in Acc.No: 102. Updated Balance: 1320
Interest deposited in Acc.No: 103. Updated Balance: 12100
Interest deposited in Acc.No: 104. Updated Balance: 25300
Interest deposited in Acc.No: 105. Updated Balance: 1980
Interest deposited in Acc.No: 106. Updated Balance: 4753.1
Interest deposited in Acc.No: 107. Updated Balance: 3658.6
Interest deposited in Acc.No: 108. Updated Balance: 964.7
Interest deposited in Acc.No: 109. Updated Balance: 7146.7
Interest deposited in Acc.No: 110. Updated Balance: 6110.5
Data received from client 5 : 101 w 207 [Transaction Successful] | Updated Balance: 17393
Data received from client 5 : 101 d 350 [Transaction Successful] | Updated Balance: 17743
Data received from client 5 : 101 d 150 [Transaction Successful] | Updated Balance: 17893
Data received from client 5 : 101 w 1400 [Transaction Successful] | Updated Balance: 16493
Data received from client 5 : 101 d 420 [Transaction Successful] | Updated Balance: 16913
Data received from client 5 : 101 d 260 [Transaction Successful] | Updated Balance: 17173
Data received from client 5 : 101 d 345 [Transaction Successful] | Updated Balance: 17518
Data received from client 5 : 101 w 940 [Transaction Successful] | Updated Balance: 16578
Data received from client 5 : 101 w 232 [Transaction Successful] | Updated Balance: 16346
Data received from client 5 : 101 d 451 [Transaction Successful] | Updated Balance: 16797
Data received from client 6 : 101 w 207 [Transaction Successful] | Updated Balance: 16590
Data received from client 5 : 101 w 207 [Transaction Successful] | Updated Balance: 16383
Data received from client 6 : 101 d 350 [Transaction Successful] | Updated Balance: 16733
Data received from client 5 : 101 d 350 [Transaction Successful] | Updated Balance: 17083
Data received from client 6 : 101 d 150 [Transaction Successful] | Updated Balance: 17233
Data received from client 5 : 101 d 150 [Transaction Successful] | Updated Balance: 17383
Data received from client 6 : 101 w 1400 [Transaction Successful] | Updated Balance: 15983
Data received from client 6 : 101 d 420 [Transaction Successful] | Updated Balance: 16403
Data received from client 5 : 101 w 1400 [Transaction Successful] | Updated Balance: 15003
Data received from client 6 : 101 d 260 [Transaction Successful] | Updated Balance: 15263
Data received from client 5 : 101 d 420 [Transaction Successful] | Updated Balance: 15683
Data received from client 6 : 101 d 345 [Transaction Successful] | Updated Balance: 16028
Data received from client 5 : 101 d 260 [Transaction Successful] | Updated Balance: 16288
Data received from client 6 : 101 w 940 [Transaction Successful] | Updated Balance: 15348
Data received from client 5 : 101 d 345 [Transaction Successful] | Updated Balance: 15693
Data received from client 6 : 101 w 232 [Transaction Successful] | Updated Balance: 15461
Data received from client 6 : 101 d 451 [Transaction Successful] | Updated Balance: 15912
Data received from client 5 : 101 w 940 [Transaction Successful] | Updated Balance: 14972
Data received from client 5 : 101 w 232 [Transaction Successful] | Updated Balance: 14740
Data received from client 5 : 101 d 451 [Transaction Successful] | Updated Balance: 15191
```

In the above logs, we can see that 10% interest was added to account number 101. So updated balance becomes 17,600.

Then a concurrent execution of transactions on by client ‘5’ and client ‘6’ starts.

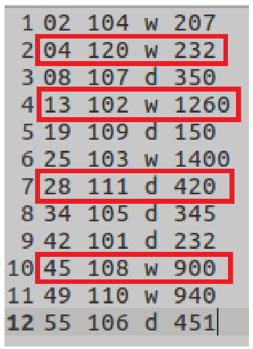
1) In the ‘left’ <red marked rectangle>, we see that client 5 and client 6 have their order of execution intermingled (they are not sequential).
   - ➢ This confirms that both clients are executing **concurrently**.
2) In the ‘right’ <red marked rectangle>, we can see and calculate the updated balance from the transaction type (d = deposit, w = withdraw) and transaction amount.
   - ➢ This confirms that **correctness** of balance is handled even in concurrent account access.

Case (ii): Consistency Evaluation

To evaluate consistency, we design a simple testcase with a single client. In this case, we deliberately put some inconsistent transactions in the Transactions.txt file.

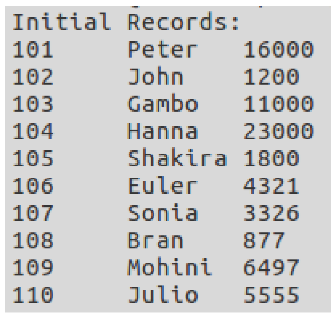
Following Transactions.txt file is used for evaluation:

Transactions.txt



```
02 104 w 207
04 120 w 232
08 107 d 350
13 102 w 1260
19 109 d 150
25 103 w 1400
28 111 d 420
34 105 d 345
42 101 d 232
45 108 w 900
49 110 w 940
55 106 d 451
```

Records.txt

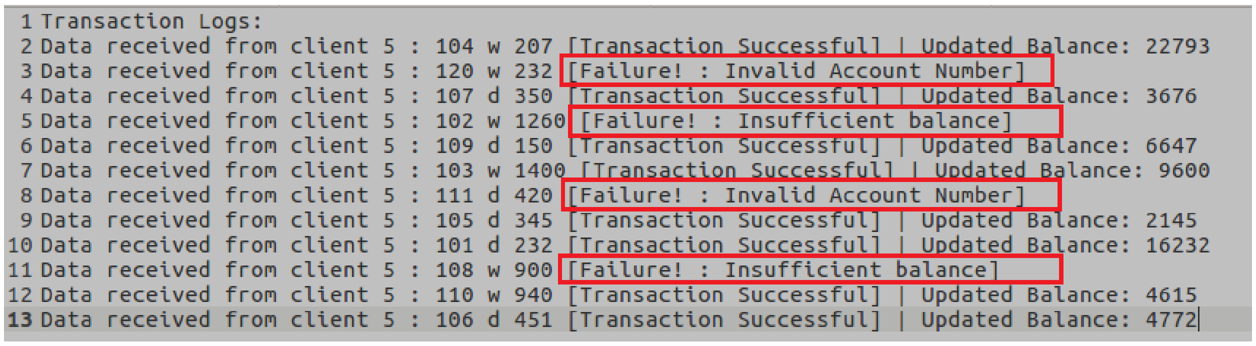


```
Initial Records:
101      Peter    16000
102      John     1200
103      Gambo    11000
104      Hanna    23000
105      Shakira  1800
106      Euler    4321
107      Sonia    3326
108      Bran     877
109      Mohini   6497
110      Julio    5555
```

In this file, there are four inconsistent transactions.

- The 'first' and 'third' <red marked rectangle>, shows an account number '120' and '111' respectively, which are not in the Bank Records.
- In 'second' and 'fourth' <red marked rectangle>, we are withdrawing amounts from account numbers '102' and '108' respectively, which are greater than their balances.

These transactions are then performed on the server to check if these cases are handled in the program. Following is the result of these transactions:



```
Transaction Logs:
Data received from client 5 : 104 w 207 [Transaction Successful] | Updated Balance: 22793
Data received from client 5 : 120 w 232 [Failure! : Invalid Account Number]
Data received from client 5 : 107 d 350 [Transaction Successful] | Updated Balance: 3676
Data received from client 5 : 102 w 1260 [Failure! : Insufficient balance]
Data received from client 5 : 109 d 150 [Transaction Successful] | Updated Balance: 6647
Data received from client 5 : 103 w 1400 [Transaction Successful] | Updated Balance: 9600
Data received from client 5 : 111 d 420 [Failure! : Invalid Account Number]
Data received from client 5 : 105 d 345 [Transaction Successful] | Updated Balance: 2145
Data received from client 5 : 101 d 232 [Transaction Successful] | Updated Balance: 16232
Data received from client 5 : 108 w 900 [Failure! : Insufficient balance]
Data received from client 5 : 110 w 940 [Transaction Successful] | Updated Balance: 4615
Data received from client 5 : 106 d 451 [Transaction Successful] | Updated Balance: 4772
```

We see that, all the inconsistencies introduced in the transactions in this case are handled in the program. In each failure instance, there is no update performed on the corresponding account number balance and consistency is maintained.

Possible improvements and extensions to the program

Further improvements in this program could include adding new functionalities like a utility for adding new accounts in the bank record.

References:

[1] Distributed Systems: Principles and Paradigms by Andrew S. Tanenbaum

[2] Stackoverflow.com

[3] Wikipedia.org