

DistributedOS

Environments:

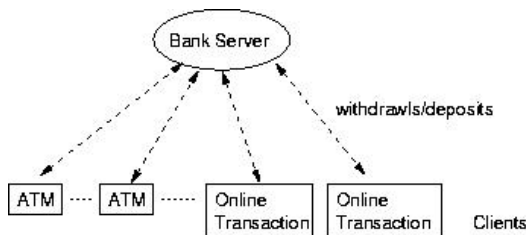
All tests are on Ubuntu 16.04 LTS, g++ (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609.

Project 1: A Centralized Multi-User Concurrent Bank Account Manager (Multithreading, Synchronization, Mutex)

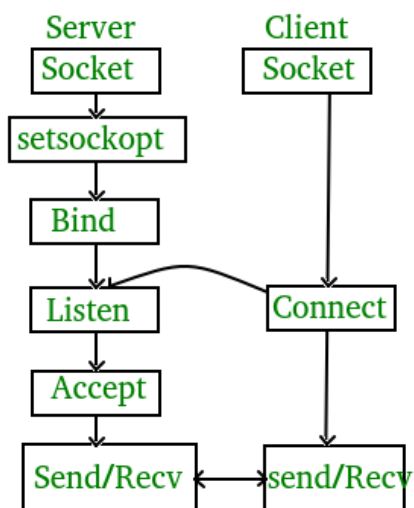
The system has two important components:

1. **Bank Server:** The server program that services online requests for account manipulations and maintains all customer records correctly.
2. **Clients:** Customers are clients of the bank server and use its services to update bank accounts. The operations that can be performed on an account are: withdrawal of an amount from an account and deposit of an amount into an account. Additionally, the bank server can have its own service that periodically deposits an interest amount to each account based on some fixed rate.

System architecture:



State diagram for server and client model:



More assignment details PDF [here](#), HTML [here](#).

Project 1 Implementation:

Check out this directory.

- Phase 1 (Version 1): Socket programming (without multithreads).
- Phase 2 (Version 2): Add multithreads on Phase 1.
- Phase 3 (Version 3): Add mutex (lock) on each account at server side Phase 2. (Add account initialization functions first.)
- Phase 4 (Version 4): Add other necessary operation functions. Complete.
 - Write a gen_transactions_file.cpp file to generate high volume transactions.
 - read transactions
 - withdraw / deposit
- Phase 5 (Version 5): Tailor for scalability.
 - Remove client terminate input request
 - Add bash script to launch multiple ./client parallelly
 - Count time for each transaction
 - Conduct experiments for scalability

Explanation

1. Synchronization

The assignment says "It should provide locking/protection for access to an account records during shared access (i.e., a user might be depositing money into his account and at the same time an online billing agent might be withdrawing money from the same account). Such cases need to be correctly handled by protecting variables in the critical section."

I set up a mutex lock for each account. So when d/w it will ask for the lock first and then finish the transaction operation, followed by unlock corresponding account. Check out server.cpp line 132 transaction_oper() function.

2. High transaction volume (synchronization proof)

If the transaction amount is small, like 1000, the server deal with them quickly and cannot generate the running time overlap between multiple clients. The server executes transactions so fast, making the execution become sequential, i.e. it finishes all 1000 transactions for client 1 (because so fast) and then finishes all 1000 transactionf for client 2.

So I set up 10000 transactions (rate = 10, every one transaction per 10 milliseconds) and all on the same account 101, insuring there will be a running time overlap between two clients in my demonstration. This generated a about 2 seconds overlap, i.e. the last 2 s running time of transactions of client 1 and the first 2 s running time of transactions of client 2. During these 2s the 2 client threads competed for the lock of the account 101.

To run it, open first terminal, run

```
cd proj1_centralized_multiuser_bank/src/v4_add_transaction_operations
```

```
make
```

```
./gentransc
```

```
./server
```

Open second terminal, run

```
./client
```

Open third terminal, run

```
./client
```

This run the server with 2 clients, check out v4_add_transaction_operations/server_log.txt of my print-out of server side.

Below is the screenshot that shows Synchronization.

The screenshot shows a VS Code editor with the file `server_log.txt` open. The log contains the following entries:

```
53491 Transaction No.2970 ongoing: 28.75 101 d 1
53492 Transaction No.2970 Done!
53493 Server: sent to client tid(139970494994176) with msg 'From server: tid(139970494994176) Transaction 28.750000 101 d 1 has been finished'
53494
53495 Last transaction for client 1
53496 ***** Account Info *****
53497 There are totally 8 accounts.
53498 Account No.: 101 Name: Peter Remain balance: 9999
53499 Account No.: 102 Name: John Remain balance: 1200
53500 Account No.: 103 Name: Gambo Remain balance: 11000
53501 Account No.: 104 Name: Neil Remain balance: 12000
53502 Account No.: 105 Name: Jasson Remain balance: 13000
53503 Account No.: 106 Name: Emma Remain balance: 14000
53504 Account No.: 107 Name: Jack Remain balance: 15000
53505 Account No.: 108 Name: Phebe Remain balance: 16000
53506 *****
53507
53508 Server: rcv from client tid(139970494994176): '28.760000 101 w 1'
53509 Transaction No.2971 ongoing: 10.95 101 d 1
53510 Transaction No.2971 Done!
53511 Server: sent to client tid(139970486601472) with msg 'From server: tid(139970486601472) Transaction 10.950000 101 d 1 has been finished'
53512
53513 ***** Account Info *****
53514 There are totally 8 accounts.
53515 Account No.: 101 Name: Peter Remain balance: 10000
53516 Account No.: 102 Name: John Remain balance: 1200
53517 Account No.: 103 Name: Gambo Remain balance: 11000
53518 Account No.: 104 Name: Neil Remain balance: 12000
53519 Account No.: 105 Name: Jasson Remain balance: 13000
53520 Account No.: 106 Name: Emma Remain balance: 14000
53521 Account No.: 107 Name: Jack Remain balance: 15000
53522 Account No.: 108 Name: Phebe Remain balance: 16000
53523 *****
53524
53525 Server: rcv from client tid(139970486601472): '10.960000 101 w 1'
53526 Transaction No.2972 ongoing: 28.76 101 w 1
53527 Transaction No.2972 Done!
53528 Transaction No.2973 ongoing: 10.96 101 w 1
53529 Transaction No.2973 Done!
53530 Server: sent to client tid(139970494994176) with msg 'From server: tid(139970494994176) Transaction 28.760000 101 w 1 has been finished'
53531
53532 ***** Account Info *****
53533 There are totally 8 accounts.
53534 Account No.: 101 Name: Peter Remain balance: 9998
53535 Account No.: 102 Name: John Remain balance: 1200
53536 Account No.: 103 Name: Gambo Remain balance: 11000
53537 Account No.: 104 Name: Neil Remain balance: 12000
53538 *****
53539
```

Annotations on the screenshot:

- Deal with transaction for client 2 and finished. (Received at earlier, before log line 53491)
- Lock account 101 at this moment (check code server.cpp line 139)
- Received another transaction assignment (10.96 101 w 1) from client 2
- Transaction got its no 2972, server in working on it now
- Deal with another transaction (10.96 101 w 1) for client 2
- This transaction has been finished for client 1
- Unlock account 101 at this moment (check code server.cpp line 154)

3. Scalability

Transaction avg complete time (ms) over clients amount

Use v5_scalability/gen_transactions_file.cpp and v5_scalability/launch_multi_clients.sh.

To run it, open first terminal, run

```
cd proj1_centralized_multiuser_bank/src/v5_scalability
```

```
make
```

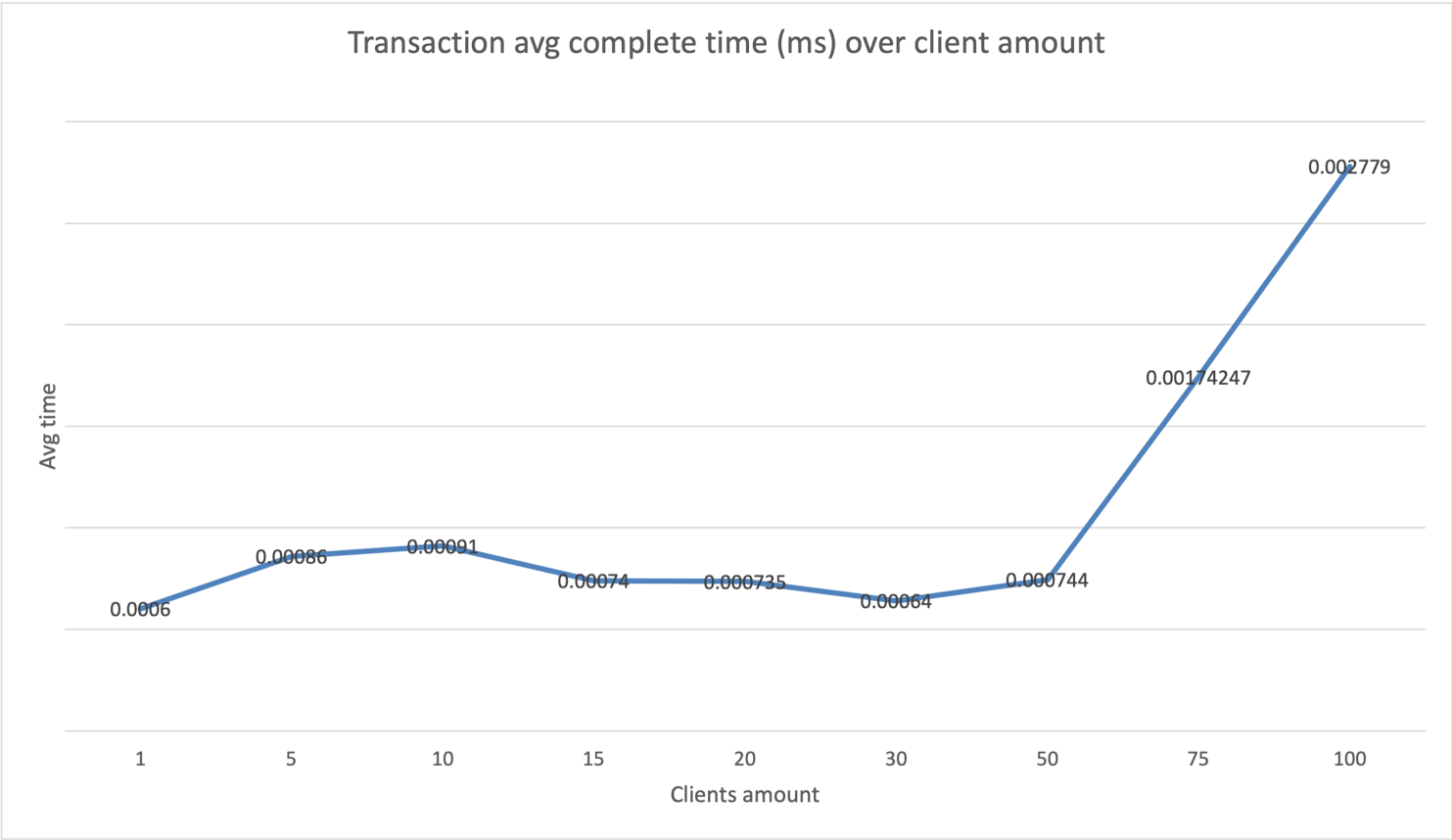
./gentransc

./server

Open second termial, run

./launch_multi_clients.sh

Clients amount	1	5	10	15	20	30	50	75	100
Transaction avg complete time (ms)	0.0006	0.00086	0.00091	0.00074	0.000735	0.00064	0.000744	0.00174247	0.002779



Transaction avg complete time (ms) over transaction rate (one transaction every rate milliseconds)

Use v5_scalability/gen_transactions_file_w_rate.cpp and v5_scalability/launch_25_clients_w_rate.sh.

To run it, open first terminal, run

cd proj1_centralized_multiuser_bank/src/v5_scalability

make

./server

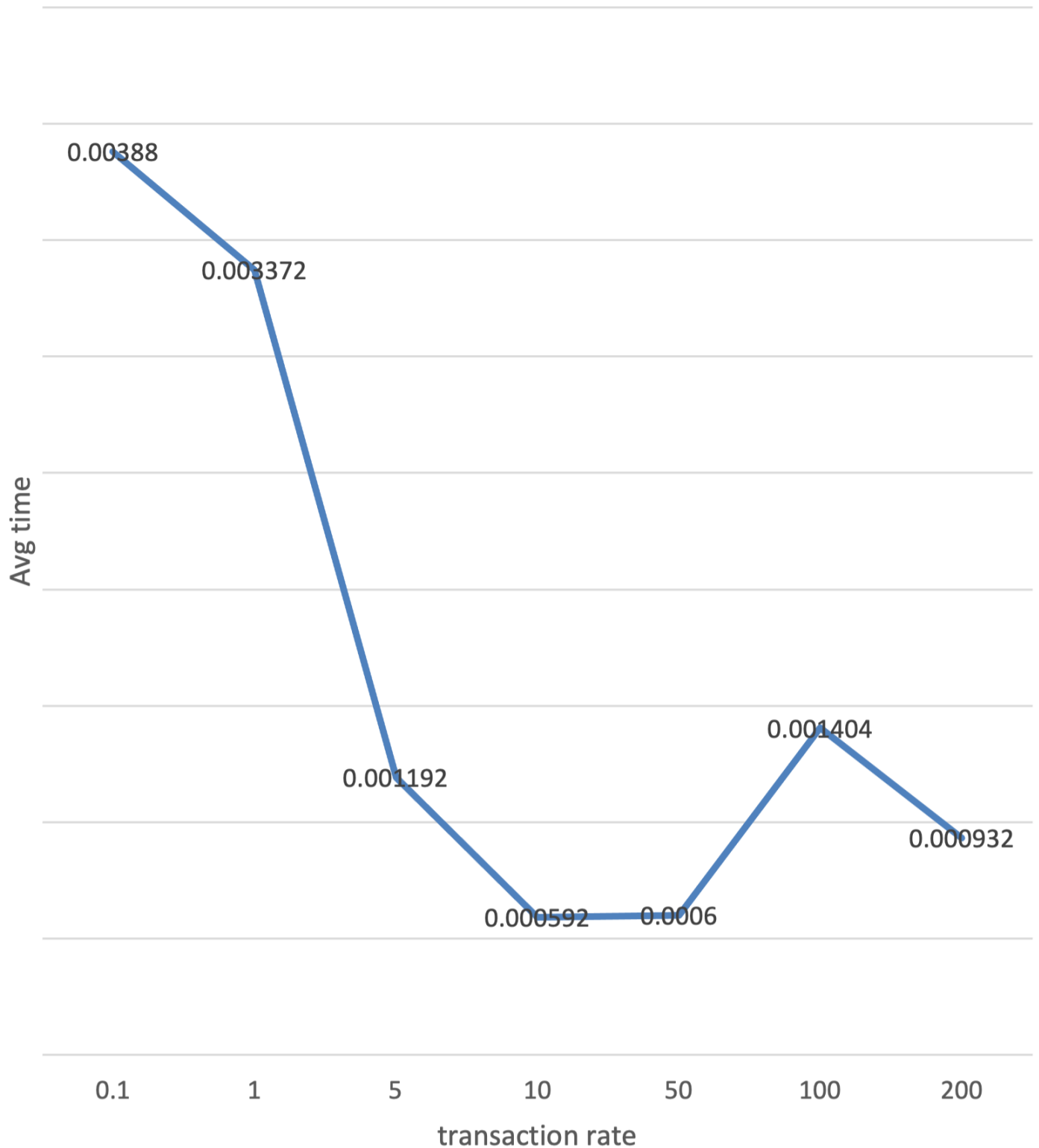
Modify the rate as you want in the file "launch_25_clients_w_rate.sh".

Then open second termial, run

```
./launch_25_clients_w_rate.sh
```

rate	0.1	1	5	10	50	100	200
Transaction avg complete time (ms)	0.00388	0.003372	0.001192	0.000592	0.0006	0.001404	0.000932

Transaction avg complete time (ms) over
transaction rate (one transaction every rate
milliseconds)



References:

Socket:

1. Socket programming example 1
2. Socket programming example 2

Multithread:

3. pthread_join need to be outside of pthread_create loop

Mutex:

4. pthread_mutex

C++ programming tricks:

5. use typedef for struct
6. c++ need cast the result of malloc
7. c++ static cast
8. c++ calc duration time using milliseconds
9. c++ append string at the end of a file
10. c++ <unistd.h> sleep() round down, sleep(0.9) = sleep(0)

Linux:

11. Launch multiple ./client programs in parallel from a bash script