Kinh Hoang khh39
Tim Goetjen tag135

Bank System – Readme

Introduction:
We have created a bank system program, consisting of a client and server pair that interact with each other to mimic a bank. The client connects to the server and is prompted by the server to perform commands that interact with the bank structure, such as open, start, credit, debit, balance, finish, and exit. Our client program is multithreading, using two threads to simultaneously listen and send out commands. Our server is multiprocessing, creating different processes for each client connection and an additional process to print out the status of the bank.

We have completed the extra credit specifications as outline by BKR (to be explained in the design portion):
-Multiprocessing/Multithreading server/client system (100 points)
-Shared memory (100 points)
-Using mmap (40 points)
-Server sends wait message when account in session (lock retries every 2 seconds)
-Client service process messages
-Child process cleanup messages
-Shared memory set up and removal messages

Server Design:
Upon start, the server prepares the socket using getaddrinfo(), bind(), and listen(). It then sets up a signal handler to clean up after child processes and prevent zombies.

The server then initializes the bank structure in shared memory utilizing a memory mapped file (mmap).

The bank structure itself consists of an integer size, a semaphore initialized to 1 in order to behave like a mutex for critical sections, and an array of 20 account structures. Each account structures consists of a name (up to 100 characters), a balance, an additional semaphore in order to lock down an account in session (for finer granularity), an unsigned int used to determine the validity of the account, and an unsigned int used to determine if the account is in session. (**As there is no client command to close and remove an account from the bank, the bank will be reinitialized every time the server is run. To prevent saturation, the memory mapped file will not retain information between server runs.**)

The server then fork()s to create the separate process which locks the bank down every 20 seconds, using semaphores, to ensure a constant state, and prints out the bank information. If an account is in session instead of current balance it prints "IN SESSION". Unlocks bank after print completed.

After this, the server then waits for a client connection, and upon accept(), it forks to create a process dedicated to that specific client.

Now, the server prompts the client for bank session commands: open, start, exit.

Open: locks the bank struct, edits it to create a new account, unlocks bank after account created
Start: locks the bank struct to search for specified account. Unlocks the bank and starts an account session for the specified account if it has been found
Exit: Shuts down the process dedicated to the current client

If a customer creates an account session through start, the server locks the account and then prompts the client for account session commands: credit, debit, balance, finish. (If the account has been locked by another client, it will retry the lock every 3 seconds until success)

Credit: adds specified amount to current account balance
Debit: subtracts specified amount from current account balance. If debit amount exceeds account balance, client is notified of insufficient funds and action is not carried out
Balance: notifies customer of current account balance
Finish: returns client to bank session and unlocks the account the client held a session for

If the client disconnects, the process dedicated to that client is shut down safely.

Client Design:
The client requires the bank server hostname as a command line argument.

Upon start, the client attempts to connect with the bank server using getaddrinfo() and connect(). On failure, the client will continue to attempt a connect() every 3 seconds until success.

The client then spawns a new thread to send commands to the server, and uses the original thread to consistently read()/recv() messages from the server. If the connection with the server is broken, the client closes the extra thread, all sockets and descriptors, and terminates the client program.