198:214 Systems Programming
pa5 – Multithreaded Bank System
Authors: Yuk Yan (yuky) & Emmanuel Baah (enb25)

# Server

**Signal handlers:**

[SIGCHLD handler]

Every time a connection with the client ends, the child process exits and SIGCHLD is sent to the parent process. Whenever this signal handler is invoked, the parent process wait()s for the child process that has just ended.

[SIGINT handler]

Not absolutely necessary, however a nice message is printed to the server whenever the SIGINT handler is invoked ☺.

**Threads:**

[Main thread]

The main thread is responsible for first initializing the bank and signal handlers. After successful initialization, the main thread then spawns the print accounts thread and session acceptance thread. When the main thread is completed with its responsibilities, then the thread exits.

[Print accounts thread]

This thread is responsible for printing the information of each open bank account. After printing the information, the thread will sleep for 20 seconds. This thread will live as long as the process is running.

[Client service thread]

This is the thread that performs all of the banking functionality. This thread is spawned upon successful connection to a client. This thread parses the command sent in by the client and calls the necessary banking functions (open, start, debit, credit, balance, finish, exit). When this thread exits, it is assumed that the connection is closed and the process will end.

[Forking thread]

Upon each successful connection between the client and server, a new process will be created to handle the client session. This thread is solely responsible for creating the new process. If the process is the parent, it will close the socket descriptor and exit the thread. If the process is the child, then the client service thread is spawned, and this thread exits.

[Session acceptance thread]

This is an ongoing thread that is initialized spawned by main. This thread waits and accepts incoming connections. Each successful connection will create a new process via the forking thread and then will perform banking functionalities via the client service thread.

# Client

**Threads:**

[Main thread]

The main thread is responsible for making sure that the argument passed (host name) is correct. Then an attempt to connect to the server will be made. The main thread will try to connect every 3 seconds until successful, or process is terminated. Upon successful connection, both the command input thread and response output thread is created and the main thread exits. Two separate threads are created in order for asynchronous communication from and to the server. The socket descriptor will be passed as an argument to both the command input and response output threads.

[Command input thread]

This thread is responsible for sending the client commands to the server. The input is throttled so that the commands are sent every 2 seconds. When this thread exits, the connection/session is assumed to be over and the process ends.

[Response output thread]

This thread is responsible for reading the responses sent from the server. When the server shuts down or the communication is lost with the server, then the process ends.

# Makefile

[server]

This creates an executable server starts the server using shared memory.

[servermm]

This creates an executable servermm that starts the server using memory mapping.

[client]

This creates an executable client.

# Other files

[bankaccount]

Contains the definition of a bank account.

[errormessage]

A custom module to print error statements in red with included FILE and LINE number.

## Concurrency

[Printing account information]

When the server attempts to print the information of all bank accounts, a mutex lock is called on each account to ensure that no information is being updated or access during the print. This same lock is locked while updating account information for the same reason.

[Opening new accounts]

Whenever a new account is being opened in the bank, the entire bank is locked to ensure that no two processes are attempting to open an account at the same time.

[Concurrent Sessions]

When a client has started an account, a lock on that account is locked to ensure that no other client may start the same account in session.

## Extra Credit

[Multiprocessing]

Upon each successful connection, a new process is created to handle the client session. In order to access the same data, shared memory is utilized. The initshmBank() function returns the pointer to the segment of shared memory. This function is called when the server is first executed and the pointer is captured. Therefore, upon each fork(), the same pointer is used.

**Please make sure to change the KEY_PATHNAME to a file that is accessible by you in order for ftok() to work.**

[Accessing in session accounts]

When multiple clients attempt to access an account that is in session, instead of silently locking the client with a mutex lock, a trylock is called instead. This way the client may receive feed back from the server that the account is currently in session. The client will attempt to access the account every 3 seconds.

[Memory Mapping]

An alternative to shared memory was memory mapping. The server is identical except that initmmBank() is used to capture the pointer to the bank instead of initshmBank(). The pointer now points to a memory map instead of a segment of shared memory. A file named “bankdata” will be created to serve as the memory map.