Our bank system implementation deals with multiprocessing in shared memory in the server program, and multithreading in the client program.

Client:

Our client program attempts to set up a connection with the server program, stating that a connection has been established if successful. Afterwards, two threads are created. One invokes command_input(), which reads in from the server, and prints messages from it, and the other invokes response_output(), which prompts the user for input every two seconds, reads from stdin, and writes out to the server. If the user can no longer read from or write to the server, the connection is closed, and the program exits. If the user invokes SIGINT, signal_handler() is invoked. It writes the "exit" command to the server, disconnecting the client, and exiting the program.

Server:

Our server program attempts to accept a connection from a client program through a socket descriptor. Upon receiving the socket descriptor, the parent process, which is the bank, forks, creating a child process that handles the communication between the client and the bank. We apply mutex locks to the bank during searches for accounts, which take place in openAccount(), startAccount(), and printBankInfo(). Mutex locks on accounts were invoked when accounts were created and in session. Upon trying to lock the bank or an account, the client is disconnected from the server. Locks were implemented through pthread_mutex_trylock(). We have also included a signal handler that is invoked whenever SIGCHLD is invoked. When the child process finally dies, the parent process cleans up the shared memory and closes the socket, ending the program.

Test plan:

We made 25 test cases for our program. They cover a wide variety of potential problems. Our program was able to handle all of them.

Extra credit:

We successfully implemented multiprocessing in shared memory and pthread_mutex_trylock().