Jordan Lyon & Olaolu Emmanuel
Bank Server Readme

client.c
Our bank server is composed of a client.c and server.c files and their executables. The client opens a socket and connects to the server using TCP/IP socket programming. Once connected, the client prompts the user for input. This input is expected to be the command that the user wants the bank server to respond to. In order to interpret the commands we used regular expressions to validate input and pass valid input to the socket to be read by the server.
After the initial connection to the server through the established socket, a thread is created to listen for server responses. The thread is created using pthread libraries. We created a struct to save the port, host, and socket file descriptor for use in the thread. This struct is cast as a void pointer and sent to our thread function where its information is used in creating the receiving thread. This thread listens for a server response and outputs server messages to the client side.

server.c
Our bank server uses the testserver code provided as a starting point. A socket is created and connection with the client is established. The server waits on the client for commands from the client. It then interprets these commands and writes information to the socket to be read by the client thread. Using the signal method, we set a process to run when the sigint signal is received. This is how the server gets killed. In our end method, we kill the bank printing process and remove shared memory segment used for the bank. We dont have to kill any of the processes client side because they have handler methods that terminate themselves when the parent process (the main server) ends. We allocate enough space for the bank in memory and create semaphores for all 20 accounts and the bank itself which restrict concurrent modification of the bank. We spawn a process to print the bank by running exec with another executable to print the banks contents every 20 seconds. We adhere to this interval by using sleep(). Everytime an incoming response comes from the client a new process is spawned to handle it.