Christian Suasi & Alec Fray

programming assignment #5
Banksystem

We used sockets, shared memory, multithreading, and multiprocessesing, and semaphores

1.sockets: so that we can communicate between the client and server cleanly

2.shared memory: so that our accounts list is shared between multiple processes and that other processes can access it

3. multithreading: so that our print accounts every 20 seconds function can work within the same main() program without using an entire process

4. multiprocessing: so that multiple clients can access the server and use the command syntax functions the server provides. It then uses the shared memory to access its account from the account list and semaphores so that it cannot open accounts while another client is in session.

5. semaphores: so that when an account is in session, another process cannot access/use the command syntax functions while a client account is in session

systems programming 198:214 with Brian Russell
——————————Compiling——————————
for the client use:
gcc -g client.c -o client

for the server use:
gcc -g server.c -o server -lpthread

-the client is compiled straightforward with -o
-the server is compiled with -lpthread in order to make it multithreaded where we implement the printAccounts which prints the accounts every 20 seconds, it should run concurrently with the main program which is tasked with all the command syntax functions
---------- Command Line ----------
for the server:
./server 201201

for the client:
./client localhost 201201

The server has to go first in order to create a server with a unique socket id that anyone can connect to if they also use the same socket

id.

---------- Algorithmic Analysis ----------
open() runs in O(n) time because the function first has to traverse the list to make sure that an account with a the input name is not already in the list

start() runs in O(n) time because the function has to traverse the list of accounts list and find the input string name that matches the account list. If the start() cannot find the name, then it doesn't start any account so it still ran through the entire list which happens in worst case the 20 accounts or (n) time.

credit() runs in O(1) time because it only works when an account is in session. Thus it already knows which account to credit and will then simply add the desired amount.

debit() runs in O(1) time because it only works when an account is in session. Thus it already knows which account to dedit and will then simply subtract the desired amount.

balance() runs in O(1) time because it only works when an account is in session. Thus it already knows which account to return the balance for and will then simply return the balance.

finish() runs in O(1) time because if the input string from the client is "finish" then it simply exits the current session, however the client is still running and can use other commands such as open() or start(). Nonetheless it's a simple function that does 1 thing without any traverses.

exit() runs in O(1) time because if the input string from the client is "exit" then it simply exits the current client process. The client is done and cannot be accessed because it was closed, you are done with the bank. However the server is still running thus other and multiple other clients can access the server.