README – Deepak Nalla – CS 214
PA5 – Bank Server (Extra credit implemented)

Client Side: Once the server starts, the client can automatically connect if not it will be waiting every 3 seconds until the connection is made. My session acceptor(outputResponse) and client service algorithms(inputCommands) run as separate processes, this is done just to make it easier for the flow of execution. Initial parent spawns child process every time a connection is made with a client. The bank holds up to 20 account structs which each can be have account names upto 100 characters each, a balance, and an insession flag to check if user is in an active session or not

Server: shared memory is attached in the main method along with my signals and timers, and shared memory is cleaned once parent is terminated by SIGINT. I used semaphores to avoid deadlocks or race conditions, as this ensures serializability. Also it is used to stall for every 20 seconds when printing. semaphores in general are good to restrict the number of simultaneous users in the program, and I used mutex to lock each session when a client starts a session with a bank, then we lock it to ensure no other concurrent activity, until the user 'finishes' his/her session. Then, the next user can start their session. I had use parsing methods to parse input, and display output errors (client session & token_parser). I thought of using our tokenizer from PA1, but this wasn't too bad to implement.

We have checking and catching mechanisms for each command, open, start, finish, debit, credit, and balance. You cannot open an account if someone already has the same name.

Idiosyncrancies I implemented: You cannot withdraw more than you credit or have in your account. Just U/I

End note: I really enjoyed this assignment as challenging as it was, I learnt a lot about multi threading, multi processing, how carefully you must manage threads, processes, client server protocols, and semaphores/shared memory