Learning diary: Juuso Korhonen (sn. 652377)

Started the project by delving myself into thread programming. There were lots of good material on the internet about the subject and I picked up a Youtube tutorial-series from channel Engineer man (the first part can be found on: https://www.youtube.com/watch?v=nVESQQg-Oiw). I find that this way suits me the best when introducing myself to a new programming concept. First a youtube tutorial where the concept is most of the time laid out in layman terms. Then later I can just quickly check how something is done through Stack overflow, man pages or other coding archives.

Git commit message: "Got a basic thread call structure to work in the main" (I try to add these here in between, since I have to use git transferring versions of the project between my desktop and laptop, and I thought they would add a nice summary of what I've been up to).

I started to tackle the next obvious big thing in the project: the queue structure. First I pondered about some way to do it the typical way with arrays, but then I decided to look in my old C-programming basic course directory, and again I found something very useful from there: A queue structure which implements a linked list concept. Every queue member itself is a queue structure which has the pointer attribute next that just points to the next member in queue. Then they also have the request ofc as an attribute. A few debugs and few change of names and voila, I had a working queue structure, which had dequeue and enqueue methods to remove and add customer requests into the queues (The structure was originally implemented as a book list in C-programming basic course). I also added the size variable, which I update on every en- and dequeue. This is linked to the global variable server_queue_lenghts in main, from which the entry_function (a function which a separate planned to be master thread is running) can easily check the shortest queue and enqueue the inputted request there.

Git commit: "Added queue structure"

Added structures for bank, server and account. Bank and account hold obviously the balances of each (Bank balance is designed to update with the query), and account has the withdraw and deposit, but server methods is where the main meat of the program processing is done. Process function does the parsing and recognizing which command the string request means and then gets the job done calling static functions like print account balance, which then calls other functions like account_exists to check does the account requested even exist. Did not however get to debug them or call them in main.

Git commit message: "added structures to hold server, account and bank, but not yet used them"

Decided to remove the unnecessary server structure, since I think that a separate server function already makes the code modular enough, and would have had to change the implementation for it to work. Also added create account command to actually start testing the other commands.

Git commit message: “Removed unnecessary server struct and added create account command”

Read the requirements through once again since I think I am quite well ahead of the schedule, and realized that part 2 also required process or program communication. Since I had already done communication through pipes between parent and child processes in assignment 1, I decided to delve into communication between programs. So again, I picked up a tutorial concerning socket programming on youtube as the first step into this new concept. The one I found was from the channel: Eduonix Learning solutions and the first part can be found in address: https://www.youtube.com/watch?v=LtXEMwSG5-8.

The chosen protocol in the tutorial was TCP/IP, and gotta say this was the most interesting part of the assignment yet. I got it to work in an instant (second run try is considered an instant in my programming), but I think it added massive reality value to my program. Now I can type and send the requests and from a different computer (=my laptop) running this TCPClient program, which they instructed in the tutorial how to do. Then I can receive and process the requests in my main program TCPserver, the one I’ve been working on, on my desktop. Quite neat!

Git commit message: “Replaced main with tcpserver, which connects to another program tcpclient communication is now much safer as client doesnt share the memory of the server side.”

Just when I thought that I was well off with the program working and fulfilling minimum requirements, I tested it more thoroughly and discovered that when I create multiple accounts the old ones get corrupted. So frustrating!

Git commit message: “todo: fix messed up ids’”

Managed to find out the line that produces the bug with the Clions great debug tools. It is still quite strange how a simple printf line can corrupt the data. But implemented mutex as nearly the last fix I could think of, and it worked. This is quite strange since no two threads should be accessing the account data the same time!?

Git commit message: “fixed todo with mutexes”

Tried to get server program to send response back through TCP, but ran into problem that it would midigate one of the main reasons I like the implementation now with client program only used for sending commands to server, it is so clean. To also receive messages I would have to compromise with messy command line. Maybe even another program for server side responses, but that would be maybe too complicated. Maybe I’ll just stick to all the operational info outputted on server program.

Git commit message: “tried to get response working”

Checked valgrind. Only few sending maybe uninitialized values, but that problem solved with replacing malloc with calloc at few places.

Git commit message: “checked valgrind”

Added rest of the commands, and they were easy to implements with withdraw and deposit already in place.

Started to implement master query. Firstly I implemented a overtake command that can bypass every command on queue and take the first place. Then I realized that I need someway to wait in main for all the threads to reach certain point. That was taken care of by concept of barrier in thread programming. It’s like that barrier initializing code line is a wall that needs n amount of threads to hit It to break so that everyone can continue and breach the city.

Git commit message: “added rest of the commands and started implementing master thread query.”

Did a bit of cleaning in the code and added comments. This version of the program basically fulfills the requirements.

Git commit message: “Check version”

Almost forgot the required logging. Imported wlog-function from assignment 1 and started wlogging.

Git commit message: “Added log messages”

Sacrificed some thought for the testers of my program. Instead of having to open the server side program ahead of starting the client program, delved myself into systemcalls and how to write to command line programmatically and start the server side program that way in the beginning of the file.

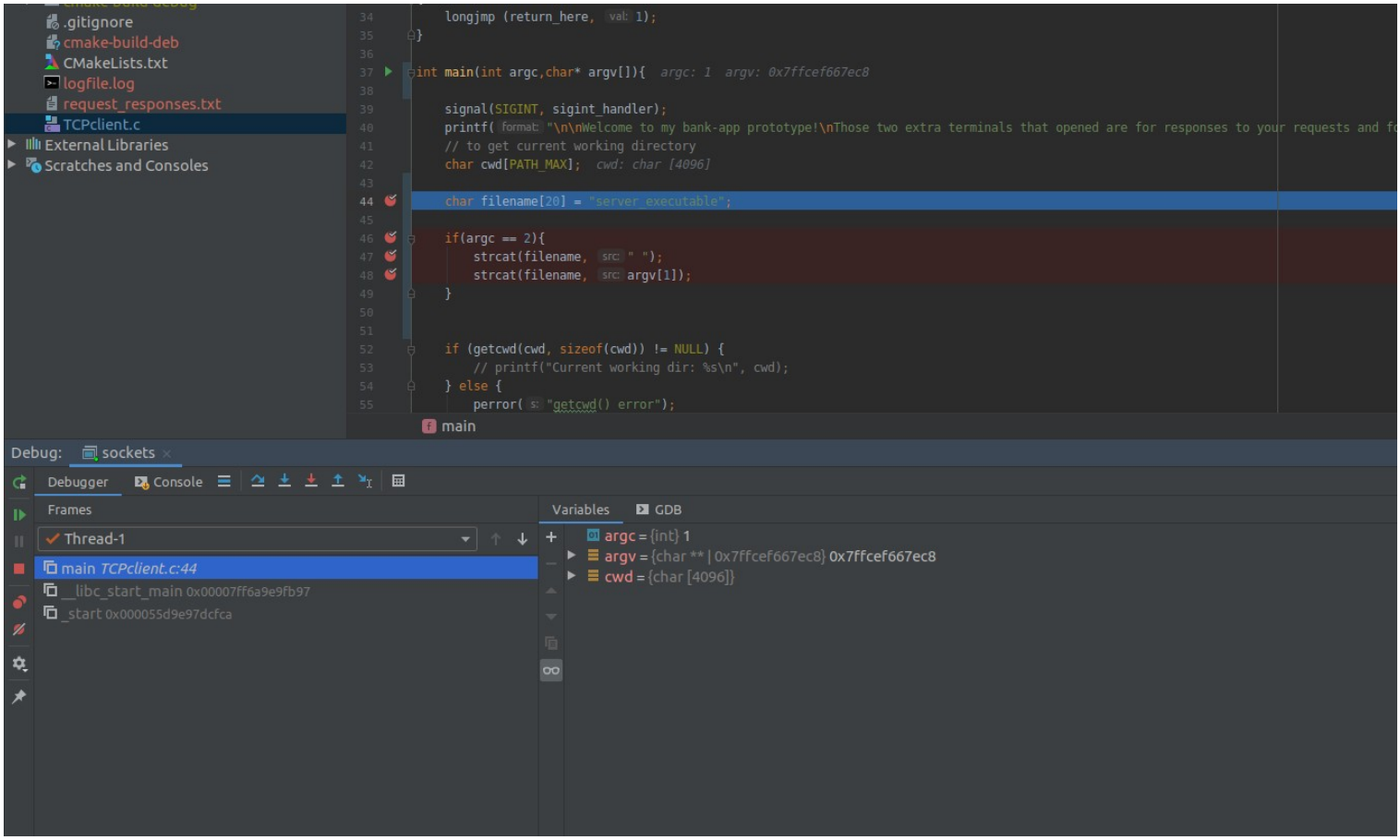
Found out a tidious problem however. Every now and then connection is refused from the server side. Read about it on stack and it seems that the port doesn’t close properly when I end the program.

Found out a solution. Problem seems not to have been an unclosing port, but a normal procedure for the OS not to free the port immedietly for other use. This is done so that the socket buffer has time to empty of all the pending messages, and so that the receiving program doesn’t think that the new connection is meant for it. I was able to pass this procedure however with simple instruction found from stack. This shouldn’t cause any problems since the port number is such that is very rarely in any use in computers.

Git commit message: “lot of work, now can just open client and connection dies properly”

How the code was tested, what works and what doesn’t, and some plans on how the assignment could be made better:

I did the project on IDE called Clion which offered excellent debugger and valgrind support among other useful features. The debugger allowed printed out detailed error message, and allowed me to set breakpoints to the code where I could check all the variables at that point. This is how I found out the cause of most of my bugs. Below is a screenshot of the debugging view.



I also tried to do pretty comprehensive tests before committing to git.

What works:

- I think I achieved all the specified requirements and above

- Especially I am proud of the multiprogram implementation that communicates over TCP/IP which I think adds a lot of realism to the implementation as a whole

- I also tried to avoid unsafe functions on top of separating client and server side, since I think especially in this kind of program the safety is vital.

What doesn’t:

- Sometimes (still) the connection is refused when trying to connect. It has worked for me just to try again, but still it can be quite annoying. I tried to disable the holdout time of the connection by the OS, but still, sometimes it happens.

Plans for further development:

- Somekind of saving of bank information so the data wouldn’t have to be recreated at every startup

- Make the connection between server and client more stable and trustworthy