IN2140 DESIGN DOCUMENT

connection.c

The connection.c file is pretty boilerplate in my delivery. I did make some decisions which did impact it pretty heavily though. Rather than watching the cbra videos I instead chose to read about socket programming from a book. The book is called “Beej's Guide to Network Programming” this is where I got the inspiration to use getaddrinfo, which I don’t think is used in the cbra videos. I got the impression from the author of the book that getaddrinfo was a more modern approach, hence the reason I used it. I chose to do either IPv4 or IPv6 (future-proofing woho!), along with SOCK_STREAM to only have TCP connections and AI_PASSIVE since we’re using the socket in bind later(that’s how I understood the man pages). I then got my results in the results struct and we’re ready to connect!

There is however a pretty frustrating issue that occurred while working on the connection.c file, no matter how hard I try I couldn’t seem to get tcp_wait_timeout to actually timeout. I tried putting the tv.tv_sec and usec into a while loop, no luck. I tried it without the while loop and still no luck. I looked up countless ways to do it, but none of them seemed to work. Now the way to get around this, was to ctrl+c out of the anyreceiver session. It’s a pretty big bug, but with that workaround you still get to execute the cleanup code.

recordFromFormat.c

recordFromFormat.c turned out to be a way bigger job than I anticipated, but I tried to separate the code into meaningful functions. For the XML part, I chose to make a function called get_value. The functions purpose is to look for the key name in the XML file, then skip the key length, plus two characters (to avoid the =”) ahead. This results in getting the first char of the value, or a quote sign, upon getting a quote sign the loop breaks. This lets me count the length of the value for the username.

I then broke up the course_code and grade into functions too, which makes a call to get_value, and processes the value from there. The same also applies to get_number_uint32 and get_number_uint8. This way I have a general function to get a value, that can also be used in the more specialized functions to then treat the value.

Now that the XML parser was done, all that was left to do was the binary parser. It took me a while to understand how the bitwise operations worked, but after some googling I think I got it working. I read in the first byte from the buffer, then proceed to make conditions where I AND the byte with the FLAG_SRC etc. I then increment the read_bytes with one to keep track of where in the buffer we are. If the condition passes, I make a call to the relevant setX function, almost just like in the XML file. There was a bit of uncertainty for me if I should also use ntohl for the bytes on ID and group, you clearly stated that the username should do it, but didn’t say anything about those. I chose to do it, reason being, why would the length of the username use it, but not the ID or GROUP number.

Proxy.c

This file has seen a lot of changes, but it (hopefully) works now. handleNewClient takes in the clients array, an integer pointer with amount of clients and the fd_set. I then make a new client and assign ID and is_xml with the first tcp_read() values. It's added to the fd_set and clients array. I then return the sock_fd to keep track of the biggest fd in the set.
Now we move on to a function which may look questionable without the proper context. The purpose of recordSplitter was to be able to take in multiple records, split them up and add them to a passed Array. I got it working for XML records, but the time ran out and I didn't make it fit for the binary records. That's why you have a read_binary variable which breaks the loop after one binary record has been read. If it's a xml record it will break after it has no <record> available in the buffer. Finally it returns the amount of records, so that I can loop over them later.
removeClient was very simply made, I just removed the socket from the set and closed it. I could also free() the client here, but chose to do it at the last part of the main loop.
forwardMessage is also pretty simple, take in the msg, sock_fd and is_xml, then you convert it to a record depending on the type, then write it in the tcp_write_loop, then free it at last.
get_client_from_id was just a function to get a client from the ID given in a record.
In handle client I chose to keep a records array, with 100(!) records, just to be safe. I then read what the sockets had written, passed it to the recordSplitter. From there you loop through the records you got, sending them to a receiver if they're available, then delete the record after. Free the buffer at last.
My main loop pretty much got the active_socket from select in tcp_wait, if its the server sock its a new connection, otherwise its an existing connection. Update biggest_fd with server sock number plus amount of clients(maybe not so safe?).
Then I free every client at last.

The "book" I referred to using: https://beej.us/guide/bgnet/html/split/