Reverse Engineering w/ Design Critique

Assignment 07

Stephanos, Koi

2020

# 1.

* The filler characters are added in the request methods in the client.cpp files contained in the tcp and udp fixed length message exchange folders in level 7 of the PISCES directory. They are used to pad the string being transmitted to enforce the fixed length.
* 6) I don’t think there is much wrong with this design, but you could use the str library resize method which would make for cleaner code.

# 2.

* In all five of the programs, the default buffer size or message length is set to 40 in the main.cpp for each program respectively.
* 2) This is potentially very problematic in terms of maintainability. You have two sides of a client-server relationship both setting their buffer size in their main files. If someone wanted to change this buffer size, they would need to know to go to both sides of the relationship. Worse, if someone were not aware of this, a change in one buffer size could have dramatic consequence on the transmission of data. I think a good solution is to have both sides inform each other as to their buffer sizes upon securing the connection, resolving the issue at run-time.

# 3.

* This is good example of content coupling. This is because both ends of the program fail to ensure the other functions a certain way, and instead just assumes it has parallel logic and constants.
* Like the answer for number two, I think you could establish buffer size on connection. To do this you could set up the server socket with a larger buffer size, say 100 in this case. Then, start the client communication by sending the number of bytes in the buffer, so the server knows where to stop parsing. You run the risk of allocating too much space and occasionally having to deny connections that request an excessive buffer size, but overcome the coupling concern.

# 4.

* Within the port\_number.cpp file there is a small class called allocatable\_port\_number\_class. This class contains code to validate that a requested port number is in the valid range, in which the lowest non-reserved port number available was 2000, and throw an error if a bad request came through
* 5) I have no real issues with the way this is handled. The design could potentially be improved by moving some of the code to its own file and by moving the lowest non-reserved port number from a method constant to a class constructor parameter with a default value. These revisions would help make the codebase cleaner and more flexible.

# 5.

* std.h
  + This should be removed. Including the std library this was is considered bad practice, and potentially pulls in a ton of code that isn’t necessary.
* bool.h
  + This can be removed from the library, C++ now has Booleans built in.
* dllinit.h/.cpp
  + This can also be removed. You say it right there in the header:
    - Newer Windows C++ environments, including (I believe) Visual C++ v5.0

and Borland C++ v4.52, load the Winsock DLL by default, without the need

for any special code in a user's application.

* platform.h
  + This can perhaps be refined or potentially just removed. Several of the backup C++ platform are not frequently in use, so at the very least those could be removed. I don’t entirely understand the justification for isolating IP implementations, but this can also be handled more easily with modern platforms and the newest version of C++.

# 6.

* Depending on how you view the socket/server relationship, this could be done in different ways. I think have the sockets inherited from each other is good. For example, tcp\_server\_socket inheriting from tcp\_socket which inherits from socket works well as each step in the inheritance is still essentially doing the same thing with an extra level of detail and specificity. I would, however, convert the tcp\_fixed\_message\_length\_server\_class a has-a relationship with tcp\_server\_socket\_class. They are clearly related, but a server is not a socket in the same sense that inheritance relationships have been addressed thus far. On that note, I would keep many of the relationships the same, only changing the socket to server relationships to has-a.

# 7.

* socket\_select, multiple\_socket\_select:
  + I think both names are a little confusing and are named after actions as opposed to a noun. In other files names, for example tcp\_fixed\_message\_length\_selectable\_server, the order is changed resolving this issue, so I would do the same here. I think the names selectable\_socket and multiple\_selectable\_socket are much better.
* getopt:
  + This is an interesting class, in that much of the functionality is in the constructor and the class essentially just wraps the helper methods, constants and custom exceptions. Since this is essentially just a method anyway, it should be made one, and placed inside a CLOptionHandler class, or something to that effect. That would better represent the logic, and then getopt could stand alone as a method.

# 8.

* The marshalling and unmarshalling code is contained within the request and confirm methods respectively for each of the different client server implementations.
* XDR is a general cross-language protocol for encoding data transmitted on networks. A good overview can be found [here](http://osr507doc.sco.com/en/netguide/dnfsC.xdrstand.html). The marshalling that occurs in this code is not set up to handle many of the examples found in the XDR overview. Thankfully, there is a library for encoding and decoding XDR format available in C++. As a result, most of the logic could stay the same, provided an extra step to call the encode/decode library methods was made before/after transmission of the data.

# 9.

* multiple\_socket\_select
* Looking at the documentation for Winsock Sockets reveals the reason PISCES would never sense a write event. A write event is triggered when there is space detected in a target buffer, that way the sending socket knows it can send more data. Since we are using the Winsock select method, however, all other events are suppressed. We are essentially opting out of receiving those messages and handling the buffer ourselves.

# 10.

* If you added callbacks to PISCES, you would need to assign those callback functions at runtime. As of right now, we detect the flags, fetch a pointer to the corresponding socket, and then call an internal method of that socket to handle the corresponding event. That part of the logic would be quite similar, we would just change the method call from the class method to the callback method. The major change is we would need to assign those methods upon instantiation earlier in main before they are used.
* Callbacks would allow the codebase to be more flexible and quickly swap between implementations. Separating the code from the individual modules might also be easier to maintain, as most of the handler code is very similar. This does add a bit of complexity, however, and you would need to decide where to store the call back methods. They likely merit their own class, but depending on nuances for the various implementations, might also be contained within the main methods for each separately.

# 11.

* No, TCP sockets and UDP sockets can not communicate. Took a little digging but found a good quote from the UDP source docs on [StackOverflow](https://stackoverflow.com/questions/32410444/is-it-possible-to-connect-a-tcp-server-to-a-udp-client-in-winsock/32410687): "UDP address space, the space of UDP port numbers (in ISO terminology, the TSAPs), is completely disjoint from that of TCP ports." This, unfortunately, is a non-starter.

# 12.

* One big point of emphasis throughout our course was decision management. The big takeaway I took was that in most cases, massive switch statements or series of if-else blocks should be replaced with a more cohesive dispatch table. Current C++ allows for the creation of dispatch tables, so implementing such a change would not be terribly difficult. Perhaps the best specific case where this change could be made is within socket\_apis.cpp. At line number 383, a switch statement begins that doesn’t end until line number 630. This is a massive chore to maintain or even read, and the logic would be much better represented in a dispatch table (or perhaps even a chain of responsibility).