Programming Assignment 1:

Design Doc

Brandon Dotson
CS 550-01
9/30/2017

Design Overview

While working on Programming Assignment 1, my project went through two separate iterations. The original version of the project was going to function with sockets. Using these sockets, the plan was to combine the server and peer code into one file and have loops, dependent on the kind of process in use. For example, if the process were used as a server, an endless loop would run as the server took care of its duties. Meanwhile, the peer version of this process would act as a read-eval-print loop (REPL). At the beginning of the loop, the user would be prompted to enter a command. Once a command and any other parameters were entered, the program would process and print out the results. As multiple versions of this process ran their loops, they would have to interact with the others. This is where difficulties arose, as any protocols for communicating had to be thought of and coded from scratch. The socket code was also implemented in C while my program would mostly rely on C++. This meant there were many parts of the code that needed to be converted from C code to C++ code to work correctly. Also, the peer-to-peer communication, arguably the most important aspect of the file sharing program, did not work. Ultimately, this caused too much trouble and cost me most of the time required for our project.

Because we were given additional time to complete the project, I decided to abandon the originally iteration completely and start over. This time, I decided to use a library that implemented remote procedure calls called rpclib. It was well documented and seemed like a better choice when compared to the socket approach. The basic framework of the program would be the same with minor changes. For example, the nature of RPC meant the server would no longer require its own loop. In addition to this, I would no longer need to find ways of making the server and peers communicate, as this was already handled by the library. Finally, issues involving peer-to-peer communication were solved.

This new version of my program was much simpler and easier to understand. When the process starts, it prompts the user with a choice between using it as a server and a peer. If the server option is chosen, the process binds the peer user commands to the port and listens for any peers. It runs indefinitely if not stopped with the stop signal (via Linux) or a peer command. Whenever it receives a remote procedure call from a peer, it executes whichever procedure is called and returns a result.

The peer is a little more complex. When the peer option is chosen, a while loop begins where the user is repeatedly prompted to enter a command. When a command is received, the peer takes the string and converts it into a numerical equivalent. For example, “r” or “registry” are equivalent to the integer “1”. This integer is used with a switch statement to determine which procedure to call. In the case of the command “obtain”, the matching case must break out of the switch statement in order to connect to the required peer.

Upon creation, each of the peers spawn a thread which listens for a connection. If a connection is established, this thread is used to transfer the contents of text files, line by line, using for-loops and a string vector. This vector is passed back to the caller, who then copies the vector into a file.

Tradeoffs

In order to restart the entire project, some planned features had to be removed. In order to keep the system simple, I decided to make peers with different directories rather than have peers from entirely different machines. Error handling was also left out also. This means it is possible to “download” a file that is not actually present. Another problem is the ability to add the same files to the index many times with each taking up a new slot. These kinds of cases are not handled well. A few other issues like disfunctional commands (close and quit) are present. The good aspect of starting over was that I was able to implement a much cleaner approach for the file sharing system. The rpclib library made adding features quick and easy.

Future improvements or Extensions

In order to improve on this project, I would focus on three major aspects. The first would be to fully implement networking capabilities for the peers. This would mean my peers could interact using different machines and networks. The next would be to work on error handling. I would look for all the situations that do not have proper control for unexpected problems and fix them. Lastly, I would try to improve the overall feel of the program, making adjustments to how text prompts look, adding the capability of parts of the commands to register as the command (i.e. “reg” → register command), and even fixing the two broken commands.