

Arcadia Youlten

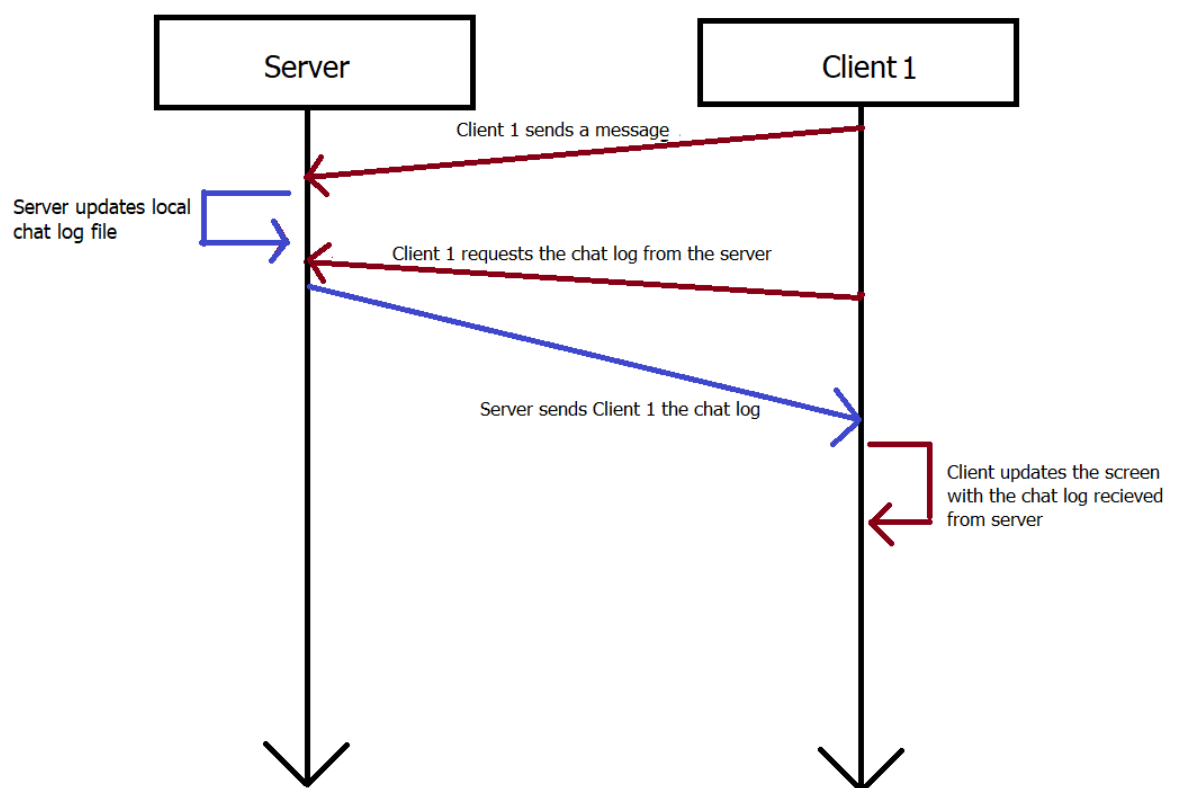
Felipe Perez

Distributed Architecture Projects

30/11/2021

DAP 3 Report – Remote Procedure Calls

The premise of this project was to create a C app which utilized rpcgen to communicate between two chat clients and ncurses to provide the user interface. In this project, we were required to have a client and a server. The server had to be stateless and had to keep a log of all messages from the chat. The client was not allowed to have any local files to store any lines of chat. Therefore, communication had to be done as follows:



(Figure 1: An oversimplified representation of the communication between a single client and the server.)

Although not explicitly stated in the diagram, the clients will periodically request the chat log from the server to update the chat terminals. In our implementation, the clients have a thread dedicated to this task, and wait 1 second to receive information from the server. Additionally, we tested our program with two clients, as this was one of the project requirements. However, the communication between client and server does not change with two clients.

There are two main programs: `test_client.c` and `test_server.c`. The `test_client.c` has two main functionalities: 1) processing messages that the user writes 2) getting the chat log from the server and printing past messages received to the chat windows for the users to see once they try to communicate.

The test_server.c has two main functionalities: 1) writing chat messages to a log file 2) reading information from the log file and sending it to the clients to print. Writing messages to the chatlog is a straightforward process, as the messages the client sends to the server have already been processed by the client. Therefore, in the server, the contents received from the client are simply written to the log.

On the other hand, reading the information from the log file and sending it to the clients is a bit more complicated. First, we seek to the end of the chat log file so we can accurately determine the number of chat lines to send to the client. For example, if the chat log contains more lines than the number of rows on the screen, then we don't want to send the entire file, but rather, the number of chat rows equivalent to the number of rows on the screen (n). We start to read the information from the file in a loop but starting from the end of the file. Once we have read the whole file or as many lines as the file contains, we stop. We save each chat log line to a string array, allocating the necessary memory as we go. Then, we condense the array into a single variable to send all necessary information in a single message. This information is once again allocated in a loop, and the final string with all n lines of chat messages is sent to the client.

Two main tools made this project possible: rpcgen and ncurses. Rpcgen is a tool which provides boiler plate code in C, with the aim of helping developers create programs that can be called remotely. This tool separates out a 'server' and a 'client'. The server contains functions that the 'client' wants to call and must be constantly running. Essentially, the server must, 1) know the functions that the client is going to call, 2) inform the operating system of the function names, and 3) be available any time a client wants to call one of the functions. On the other hand, the client simply must connect to the server, call the functions, and exit.

To implement RPC, we used the following guide: [Remote Procedure Calls](#)¹. One important part of using RPC is informing the operating system of the function name. This is done by using an XDR language to define the function names, and creating a .x file as seen below:

```
struct message {
    string contents<>;
};

program PROGRAM_NAME{
    version ALPHA {
        void myWrite (message) = 1;
        message getChar (void) = 2;
    } = 1;
} = 0x20000001;
```

(Figure 3: the contents of our .x file for the project.)

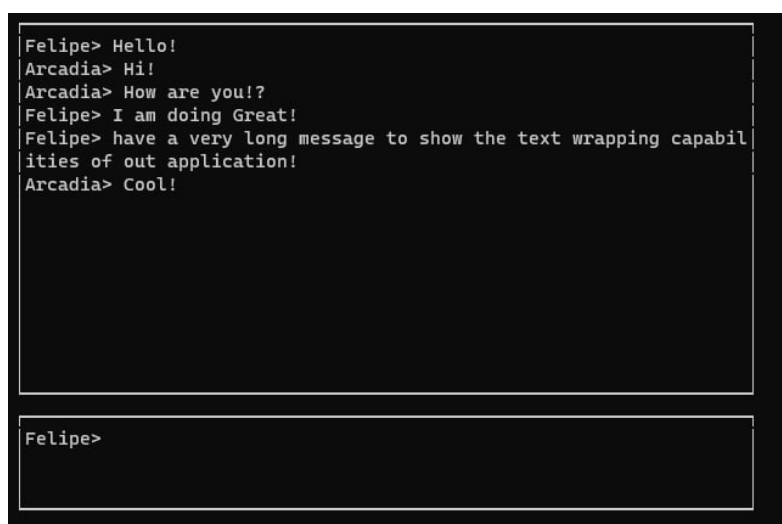
In this case, we've defined the program as PROGRAM_NAME and its unique id as 0x20000001. We've also set the version to ALPHA with version number 1. These are required attributes for the .x file. Then are the function definitions. Each function is set equal to a number, as the client will refer

¹ "RPC (Remote Procedure Call)." Advanced Programming Theory UCO 2011 , https://teoriapa1112-blogspot-com.translate.goog/2012/01/rpc-remote-procedure-call.html?_x_tr_sch=http&_x_tr_sl=auto&_x_tr_tl=en&_x_tr_hl=es&_x_tr_pto=nui.

to the function by number rather than name. Two functions were required to be implemented as remote procedure calls: myWrite, and getChar. myWrite is a write function which adds a new line of text to the log file of the server, and getChar which returns the most recent n lines from the chat log. These files will be called by the client to send and receive chat messages. In this case, myWrite is 1, while getChar is 2. Additionally, we've defined a message struct, as in the myWrite function, we want to pass a string as a parameter. Although it would be easier to simply define a string, the XDR language only accepts predefined structs as the only parameter types.

When compiled, this file generates the seven following files: Makefile.test, test_xdr.c, test.h, test_server.c, test_svc.c, and test_clnt.c. The Makefile is used to compile all files needed for the project and the xdr code is a 'translator' which converts parameters and results into universal machine code. The test_server.c and test_client.c are templates for us to implement our code while still maintaining RPC functionality. Finally, the test_svc.c and test_clnt.c are files used for the RPC in order to function properly, but do not have to be modified to work. The test.h file is for the client to include if it wants to use of the server's functions

The other tool used was ncurses. Ncurses is a library for C/C++ which allows developers to create rudimentary text-based UI in the terminal. In our project, ncurses was used to create the individual windows of the UI as seen in the figure below:



```
Felipe> Hello!
Arcadia> Hi!
Arcadia> How are you!?
Felipe> I am doing Great!
Felipe> have a very long message to show the text wrapping capabilities of out application!
Arcadia> Cool!

Felipe>
```

(Figure 4: The UI generated by ncurses)

Ncurses is used solely in the client-side of the project, and as shown in the figure above, it is used to display the chat, and draw the chat box borders. The client has a thread dedicated to managing the ncurses UI where chat log messages are received and printed to the upper window. To ensure that the messages are the most up to date, and the window is erased and refreshed each second. Additionally, there are two non-threaded functions used to create the UI using ncurses. create_newwin, and destroy_win. Create_newwin is a function which creates a new window given some dimensions and destroy_win is a function which destroys windows when called.

In conclusion, this project was very effective in teaching the basics of the rpcgen and ncurses library. Much of our initial research showed that ncurses is primarily used for games and applications which only require limited user interactions (such as navigating a menu), therefore, it was very interesting to implement ncurses in a way where there were slightly more complicated user interactions. Using ncurses in another exercise probably would have made the project a bit easier and straightforward to do, therefore, perhaps adding another small exercise about ncurses to the course

would be interesting. Additionally, it would be nice if there was a more complete guide on RPC, as even with the provided tutorial, it was still sometimes difficult to understand should have been done in different situations.

Citations

“RPC (Remote Procedure Call).” *Advanced Programming Theory UCO 2011* ,
https://teoriapa1112-blogspot-com.translate.goog/2012/01/rpc-remote-procedure-call.html?_x_tr_sch=http&_x_tr_sl=auto&_x_tr_tl=en&_x_tr_hl=es&_x_tr_pto=nui.