La Salle 2019-2020

Cypher System

Operating systems

Marc Llort Maulion Almansa and Alex Casanovas 25-12-2019

index

design

Explained

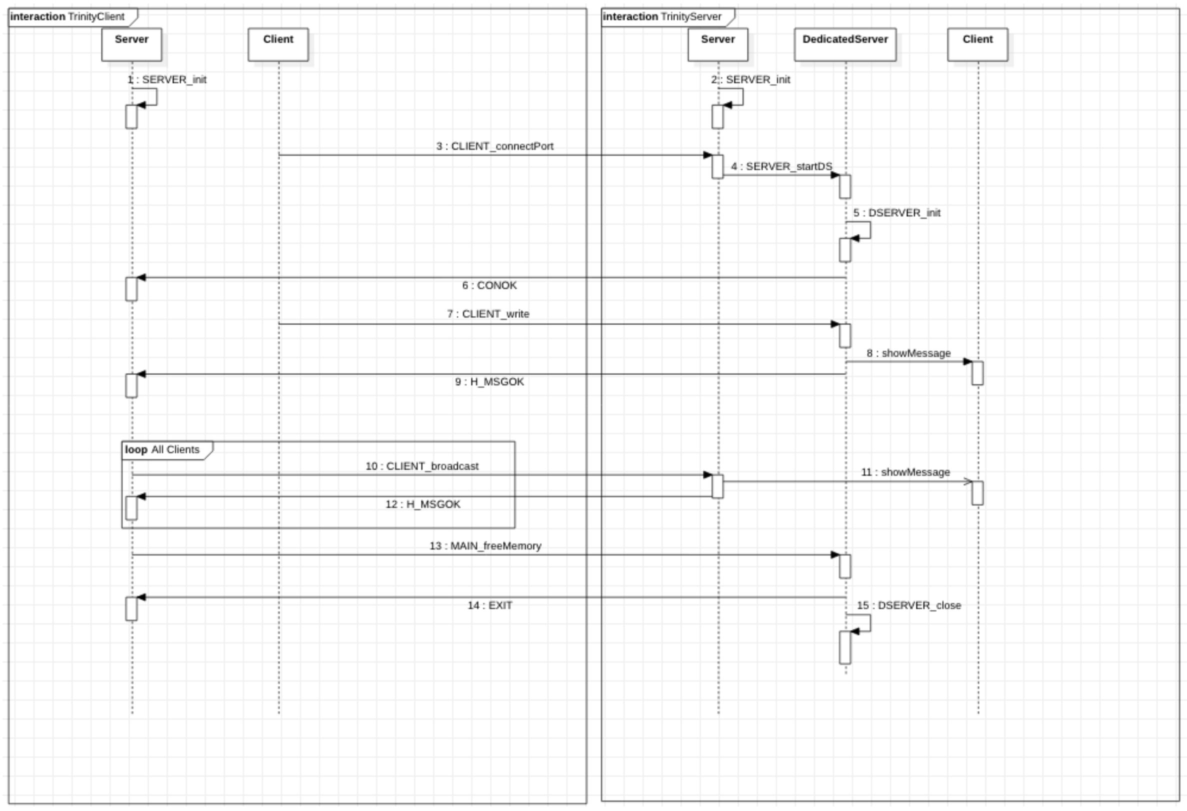
To carry out the implementation of practical design and has tried to think what would be the best structure and then carry it out without too many problems with the design. First, the first phase was not necessary to think much about design as it simply had correctly read all orders were introduced by this stage the most important was to properly manage all dynamic memory the different strings.

The second phase was the most complicated in terms of design but also in terms of the implementation phase as it contains all the logic of communication between different clients. The first idea was that it had two threads, one for the server and one for the client that communicates with other clients using sockets. Thinking it and turns giving him finally decided to make a thread of execution only waited connections and then create another execution thread for each connection to ensure that a customer can send messages simultaneously. Then finally it was made is a thread that is waiting for connections to a specific port when it receives a connection request creates another thread for that particular connection. Finally, when a customer is disconnected,

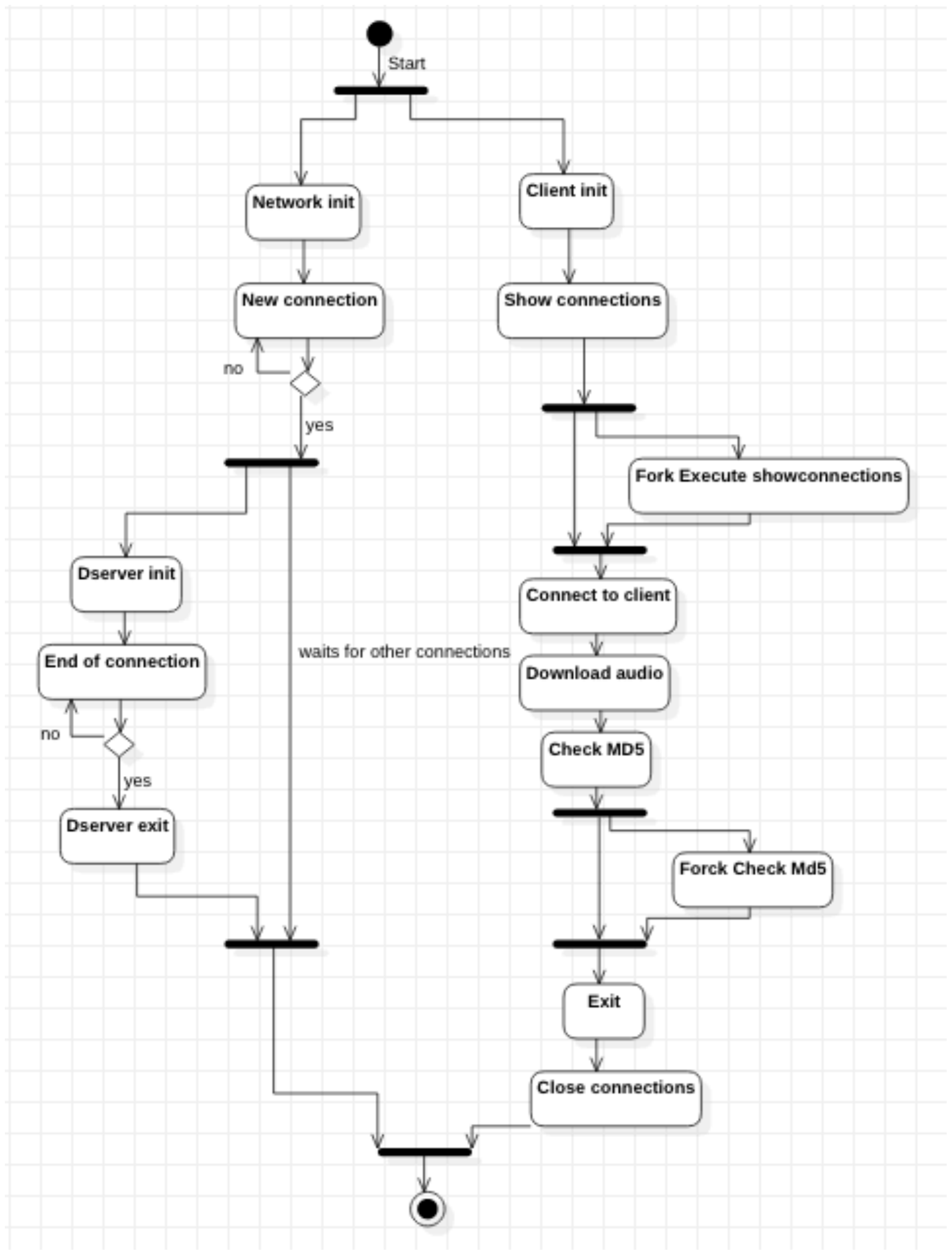
The next phase, Phase 3 is the phase that handles all matters related to the audio for this phase did not require much thought in the design, with the base of the previous phase and all logical messages and connection done, what it required was thinking about how to manage the sending of large files that can not be sent at once using sockets, but the communication protocol to follow, we are quite easy this job. So this phase is only to send the names of the files available in the folder specified in the configuration file and send the file inside the folder specified by dividing it into pieces if it is a large file. A mistake that cost enough to solve this was related to one that was used strcpy,

Finally, phase four, was the easiest of all, because all you had to do was send a broadcast to all connected clients, this functionality was already practically implemented the second phase, as the disconnect a user had to warn all others are disconnected, and could use that function.

diagrams



This sequence diagram shows an example of operation in which there are two users, one that has the role of client and another that has the role of server. First, the client makes a connection to the server. Then sends a message to him and then made a broadcast to all users is connected. Finally, the client disconnects from the server.



In this diagram you can see what the operation of an execution in which the Trinity is a show connections and an md5 check.

Data Structures

As for the structures used to implement the practice, first, that statement is defined by the same practice is the packet structure. This is used for any communication between the two programs trinity, and is composed of a first field is a char and tell us what kind of package is a second field, which is a char * and the header of the packet, a third field with the length of the information itself, which is the type uint_16_t and finally the data are a char *.

As a second structure used also governed by the rules of the memory part is the config structure. This, stores all the information contained in the configuration file is read to start the program. Consists of the username, the folder where the audio, the IP and port of the user, and finally the connections may be available, made by ip and port range to scan.

Apart from these structures have been used different types of structures defined by us. First, a bidirectional list, which has been deployed twice since it took two lists of different objects and try to make a generic problem occurred, so finally decided to implement two of them.

Another structure, created by us, is the server, this is the name of the server, the IP, the name of the audio folder, the port used, the fd of the connection in the thread that will run server function to be executed in this thread, deidicated a list of servers that are active, and a traffic light to change this list Dedicated servers from the server or from a general Dedicated server.

Next, a structure quite similar, it's the Dedicated Server has the same fields that the server adding a State that serves to manage when we close a Dedicated Server.

System resources

In terms of system resources used, we used, firstly sockets for communication between different programs using ports assigned to our group. Second, we used threads to the execution of a communications server that accepts you arrive and create a thread for each creating a dedicated server for each connection with its corresponding thread.

Finally, we used two lights, one for writing the screen to ensure that no two writes simultaneously from two trhreads different, and a second for the correct use of the list Dedicated servers having the server usually because is modified from the general server and Dedicated servers from within.

problems Observed

While performing implementations of each phase we have encountered several problems. Then explain stage by stage what were the problems and how they are solved.

phase 1

Being of implementation fairly simple, did not meet with any great problem as the other phases. In general, problems of memory leaks have not been solved as yet did not know how to run Valgrind and looked manually, so that always escaped without one.

On the other hand, when reading the file we meet any of the configuration files that we created, operated incorrectly. The problem is that some have a space before the "/ n" and those who did not have a space, we lost a letter read from the field.

phase 2

In this phase, which was no more problems arose at the same time implementing it. The first was to recover the first notes and improve a bit bidirectional list to adapt it to our needs. For some reason still unknown, if the list was initialized but empty, if calling functionality to the point of interest at the beginning, that the list was unusable. To correct the mistake, we protect functionality *vesInici* with " *f* "Verifies that it is not empty. Later, when working with *threads* to create dedicated server, but we managed to successfully pass any information received outside *thread.* The problem was simple to fix, we had to move all the logic to receive messages once the connection is established within the dedicated server.

At first, the above problem was some logic from the design, since receiving the first packet sent by the client from out of dedicated server allowed us to identify packets received, and thus filtering packets connection with those *show connections.*
When moving the logic in the dedicated server to resolve the fact that no dedicated servers created by the *show connections,* if we put that in a case of not having field *header,*
close the dedicated server.

While sending and receiving packets, a problem that cost us a lot to solve was that in some cases packets received "dirty" but sent "clean." The problem was that the reading was not reading the "\ 0" or putting it manually, so read more memory than necessary. This error also caused us problems when making *free,* since it is not performed properly, and in some cases generated *SEG Fault* or *memory leaks.*

Moreover, although it is not a problem if we had to implement *show connections*
3 ways, the first time because the script was not yet in eStudy, and second because we make use of a *fopen.* While using the pipes, the only problem we had (which we take to find), it was just that we had not initialized *pipe,* therefore was not working.

phase 3

Despite not having many problems are the most time we had to spend to solve them. The principal was also to implement the functionality of dedicated server *audio show.* In some cases we read correctly the folder, but if you tried the same functionality rerun a second time failed Prints things "random" or failing to make a double free. At first we found it worked a little better to make use of a different function

Scandia to read the directory, as some sites indicated that it was a function that was not thread-safe.

Finally, we realized that the problem was not to read the files, but as we sent to the client. The problem is the same that I said the second phase, where we were reading incorrectly due to "\ 0". Once you have fixed everything worked properly. The second problem was while making the submission file to the client. First we did our experiments with a text file and saw that everything worked correctly, but send audio, although the file size was the same as the original, you could not hear properly. We knew we could not do *strcpy* to send or read, but we were prepared using a function to send and read text above, we did not realize that even though we were reading correctly, one of the functions in the sending of the package, there was a *strcpy* it was that caused all the problems, once arranged everything worked perfectly, we just have to take care of some memory leaks.

phase 4

In the fourth phase, we have not had any problems while making remarkable implementation because it is a simple *broadcast,* which already did one when making functionality *success* customer.

The only small problem we had was that we were not following the correct protocol for communication because we were reading a message directly like functionality *say,*
and the answer was that it brought with type 2 instead of 3 as the protocol says.

estimating Temporary

phase 1

Initially, we spent about 2 hours to design the first phase, already taking into account the structure we wanted to continue throughout the project. We rethink the design during the implementation to make some improvement, because at no time do not programmable in C recalled some of its peculiarities.

In the first phase, the implementation was quite simple, we spend about 8 hours per month 3h testing. Once corrected, finally fix memory leaks and added another error about 5 hours to finish leaving it completed. A total estimated a 18hores.

phase 2

Approximately 2h wanted to design and perform different functions. When implemented, we invest many hours because the functionality *show connections* we implement the three different forms, the first time because the script was not yet in eStudy, and second because we make use of a *fopen.* Other features were relatively simple to implement once worked in shipping *packets* and the *thread.* Implementing we were about 30h.

In testing and corrections of errors found corrected once the first phase about 10am. In total, around 42h.

phase 3

In design, because it was almost the same as in the second phase, we were 30min. However, when we met implement many problems to see the files available in the audio folder and simultaneously sending the audio to run all we invest approximately 60h. In testing and corrections, especially in *memory leaks* 10h a month, so a total estimated 71h.

phase 4

Being of a simple *broadcast,* we spent time on design, implement and test in just 2h proper operation.

In this last phase, where we have invested more time has been fixed error correction phase 3, approximately 5 hours, and perform memory, about 10am. A total of 17h.

Conclusions and suggestions for improvement

Mainly this practice has helped us to take back lost confidence and lost with C you some (not all) fear.

We also have been very practical to carry out a program more structured and designed from the beginning, without putting us to program directly. Thus we have many error estalviar- us we probably had appeared after the practice was already in an advanced state.

To schedule thus cleaner has been very useful to use Valgrind, a tool with which we played very little so far, and you have given us very life when finding memory leaks, and d that way we save errors.

From a more general, has helped us to understand the various concepts that have been explained more theoretical. Especially in the case of the *threads, Forks* and *pipes,* everything that is laboratory sessions we began to play with them, to make the practice done more fully understand its operation.

One of the points that we liked most was to work with client-server program, because normally always differentiate a client and server separately, and although the end is still the same when you have to implement, we think this is a good way to show you a real case of the threads where we have a server running invisibly to the user, while the "top" running the client as usual. As for the improvements you could make to the program, in terms of functionality, we believe it might be interesting to create a system replicates files, like P2P, where instead of selecting who you want to download the file, simply write download the file and pieces of different customers, even though we understand that this is a system too complicated to carry out in practice.

Another improvement would be to mount a login system for each client, and even a second "password" to connect to other guests and download their files.

Bibliography

C program to demonstrate fork () and pipe (). (2017, June 2). Retrieved from https://www.geeksforgeeks.org/c-program-demonstrate-fork-and-pipe/.

C sockets Computer Science. (Nd). Retrieved from http://www.cs.rpi.edu/~moorthy/Courses/os98/Pgms/socket.html.

Fork () in C. (2019, December 9). Retrieved from https://www.geeksforgeeks.org/fork-system-call /.

Manish. (2015, August 24). C Program Files to List in Directory. Retrieved from https://www.sanfoundry.com/c-program-list-files-directory/.

GNU System calls. (Nd). Retrieved from https://www.gnu.org/software/libc/manual/html_node/System-Calls.html.

Using Valgrind to Find Memory Leaks and Invalid Memory Use. (Nd). Retrieved from https://www.cprogramming.com/debugging/valgrind.html.

Programming for UNIX Operating Systems Practice. (Nd). Retrieved from https://estudy.salle.url.edu/pluginfile.php/792145/mod_folder/content/0/LlibreUNIX_14 -15.pdf? Forcedownload = 1