ARRAKYS SYSTEM

OPERATING SYSTEMS

PROJECT 1

Paula García & Arnau Castellà

22/05/22

INDEX

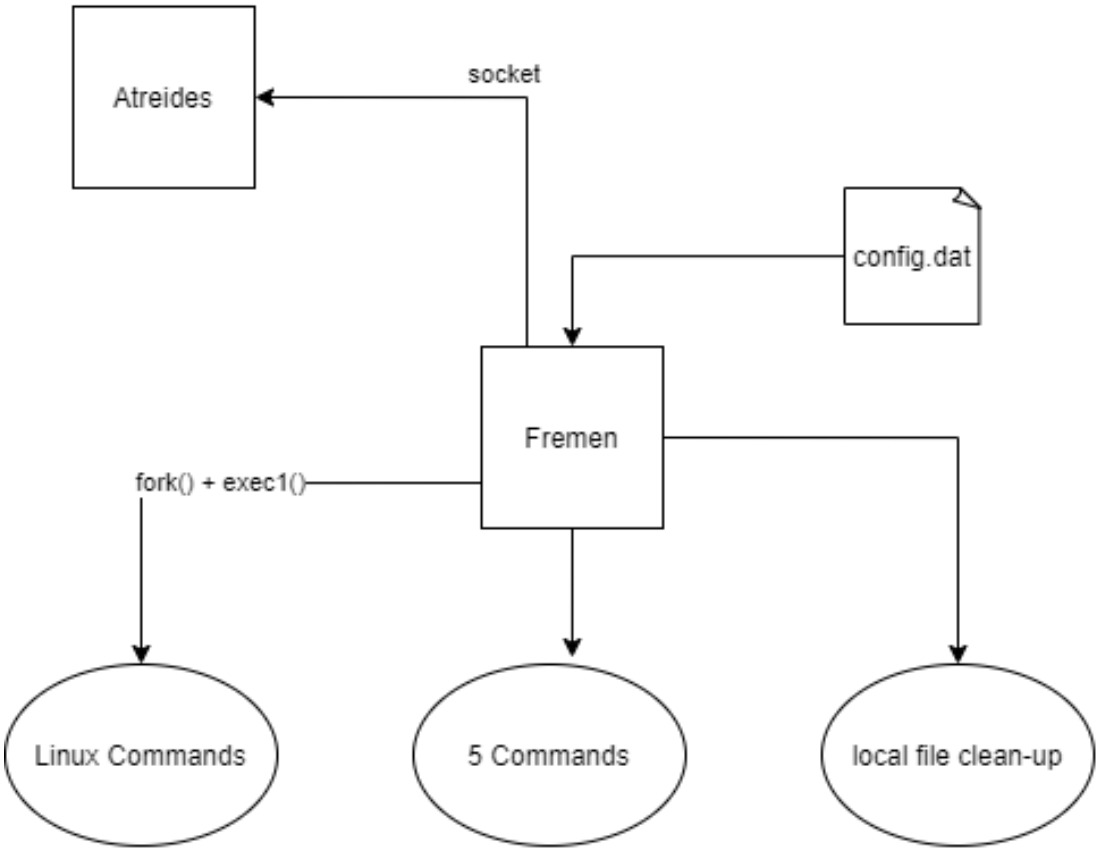
Design.

The design of the Arrakis system consists on a principal server (Atreides), in which different Fremens can connect. At the same time Fremen's can be attacked by Harkonens. Harkonen's objective is to kill Fremen's processes'.

To explain our design, we will first explain the design of each of the three components of the system, an then we will also show in more detail some of the Fremen's functionalities.

FREMEN

DIAGRAM



Fremen works in command line mode, it is able to execute commands that Linux operating systems can execute, and also has 5 more commands programmed.

The extra 5 commands are the following:

LOGIN: This command allows us to connect to the Atreides Server.

SEARCH: In this case we are searching for users with a concret postal code, that have been registered in the Atreides server.

SEND: With this command we can send a photo to Atreides Server, and Atreides will store the photo with the ID of the user sending it.

PHOTO: We can also download photos from Atreides, if there are, with the ID.

LOGOUT: will close all connections, and free memory, and shut down Fremen.

There is one more functionality on Fremen which is launch a local file clean-up every 'x' time.

The configuration time specifies the time that should pass between clean-ups, as well as the IP and PORT we must connect to connect to the server, and also the directory where the Fremen Program is located. Diagrams explaining what processes have been created, the different communications between processes, etc.

Data structures.

We have used one structure to store all the information about the configuration file. As already mentioned in this file we can find the IP and PORT where Atreides Server is, and the directory of the Fremen program.

```
typedef struct {
    char * cleanup_time;
    char * ip;
    char * port;
    char * folder;
} FileInfo;
```

We have used another data structure to store the information about the currently logged in user. We only use this data structure after there has been a successful login.

```
typedef struct {
    int id;
    char * name;
    char * postal_code;
} User;
```

System resources used

As we already mentioned, Fremen connects via socket with the Atreides server. Atreides will then manage the petitions of Fremen. From Fremen's part we only had to establish one connection to Atreides. We will then see how Atreides manages all the connections to different Fremen.

We also mentioned that Fremen must be able to execute Linux commands, to do this we have created a fork, where the child executes the Linux command. To execute the Linux command we used the execl() function.

In order to execute commands from our program, we use the function execvp. In some cases we also use pipes to read the output of the command sent into our main process. For example, when we need to check the md5sum of a file.

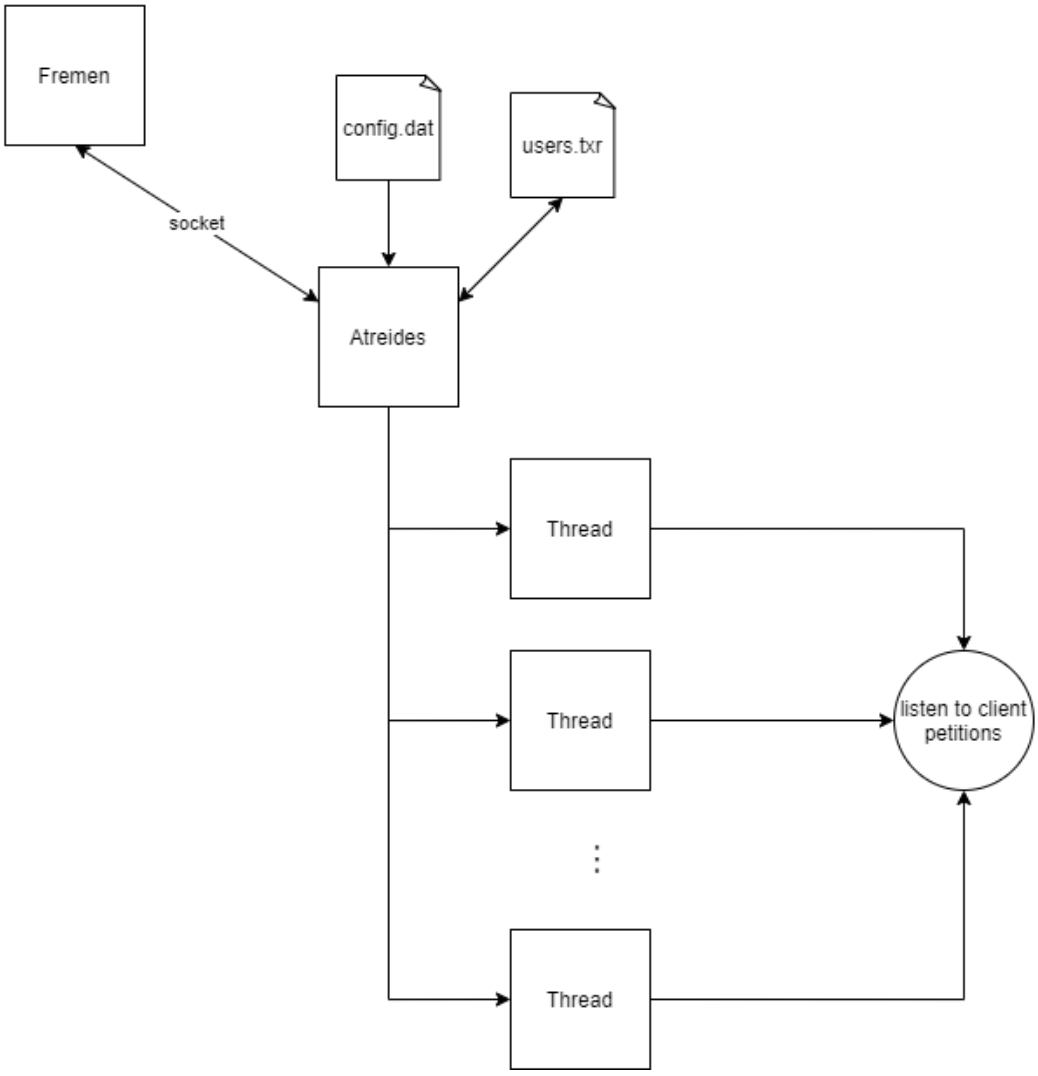
A part from all the commands explained, Fremen should also take into account CTRL+C and other signals. These are singals we used:

- SIGALARM: IT is encharged of avisanros everytime we need to do local file clean-up.
- SIGINT: This will make sure that the process shuts down, and we free all dynamic memory.
- SIGTERM: If Fremen receives an attack from Harkonen, we will receive it through this interruption and we handle this exactly the same as the signal before.
- SIGPIPE: If Fremen tries to send a command to the Atreides server after there has been a disconnection, we will receive this signal to properly manage this scenario.

Optional phases implemented.

ATREIDES

DIAGRAM



As we mentioned different Fremen can connecto to Atreides. When a Fremen is connected there are four petitions atreides can get from it:

- SEARCH: The server will search inside its data structure all the users registered with the postal code specified.
- SEND: Atreides receives a photo, and it will store it in it's directory with the respective ID of the user that sent it.
- PHOTO : Atreides sends a photo, if it exists, of the user of the specified ID.
- LOGOUT: Atreides disconnects the user.

In Atreides we also have a configuration file were it contains it's IP, port and folder where it will store the photos.

Atreides opens a Socket to start listening to petitions for possible Fremen clients. When a petition arrives, it checks that it is correct and if it is, it creates a thread to manage the petitions of that specific Fremen. This way we can keep listening for new client petitions, and still manage the petitions of the Fremens that are already connected.

Data structures.

In the Atreides server we can find three data structures.

The first one we use to store the configuration file information. The first thing Atreides does, is read this file and store the information on the following structure:

```
typedef struct {
    char * ip;
    char * port;
    char * folder;
} FileStruct;
```

The second structure is used to store information about the client. Each client has important information that the server should store. We store the following information about each client:

```
typedef struct {
    int id;
    char * name;
    char * postal_code;
} User;
```

The last one is a linked list implemented by us, used to store the information of the users that are currently connected to the Atreides server. There is no specific functionality related to this list, however it was necessary to control when a user connects to the server, and more importantly, when it disconnects from it.

In each node of the list we store a user, the data structure explained above.

```
typedef struct Node {
    User user;
    struct Node* next;
}Node;
```

System resources used

Once again, Atreides is the principal server where Fremen can connect. Like in Fremen, we use sockets to be able to communicate all the clients with Atreides.

Through this sockets we send and receive frames following the communication format specified. This way the server and the client understand the frames and can handle the petitions.

Another important part about Atreides is the use of threads. The principal objective of this server is to handle connections with the different connections to Fremen clients, however, once the connection is accepted, Atreides should be able to keep handling the petitions of each client. To solve this, we use threads. When a Fremen is connected and accepted, it assigns a thread to it that will manage the concrete petitions of this client.

In other words, the main program will only handle the new petitions, and wait for new petitions. With the different threads we manage all the other functionalities of Atreides.

As we have previously explained, in order to execute commands such as ‘md5sum’, we use the execvp function. To use it, we need to call it from the child process after a fork, and to send the output of the command to our main process we use pipes.

We are missing two more resources used. The first one is semaphores. All the process must be able to access to the list of users Atreides has, to read or modify it. For this reason, to protect this data we use mutual exclusion semaphore. This way, the list can be accessed by one process at a time, and we avoid more than one process wanting to change or read information simultaneously.

The last resource used are signals again. It only uses the signal ‘SIGINT’. We use it to close the server correctly. We free all memory that we used, and we close sockets and finally close the server.

Optional phases implemented.

HARKONEN

DIAGRAM

Harkonen scans all active processes on the system and chooses one random Fremen process and kills it. If there are no Fremen processes, Harkonen waits the indicated time and tries again.

Data structures.

Harkonen is the most simple program in the system. Due to its simplicity and its strictly destructive purpose there was no need to use any data structures.

System resources used

The goal of Harkonen is to simulate attacks with intrusive programs and kill processes running in Fremen, wether it is connected to Artreides or not.

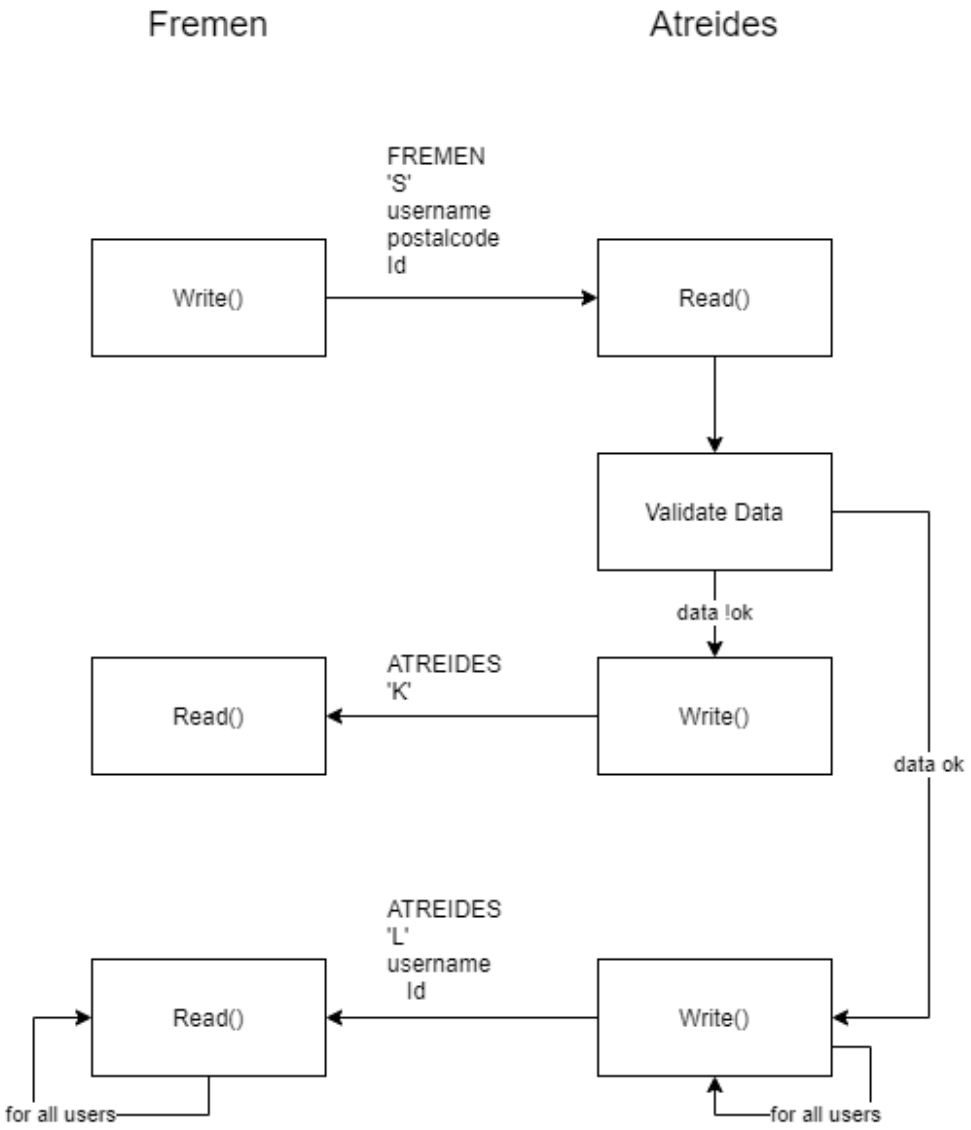
In order to execute commands to kill the Fremen processes found, we use fork to create a child process where we call the execvp function.

We also use forks and execvp to find the actual Fremen processes, and the output of the command executed by execvp is written to an auxiliary file that we use later to select a random process.

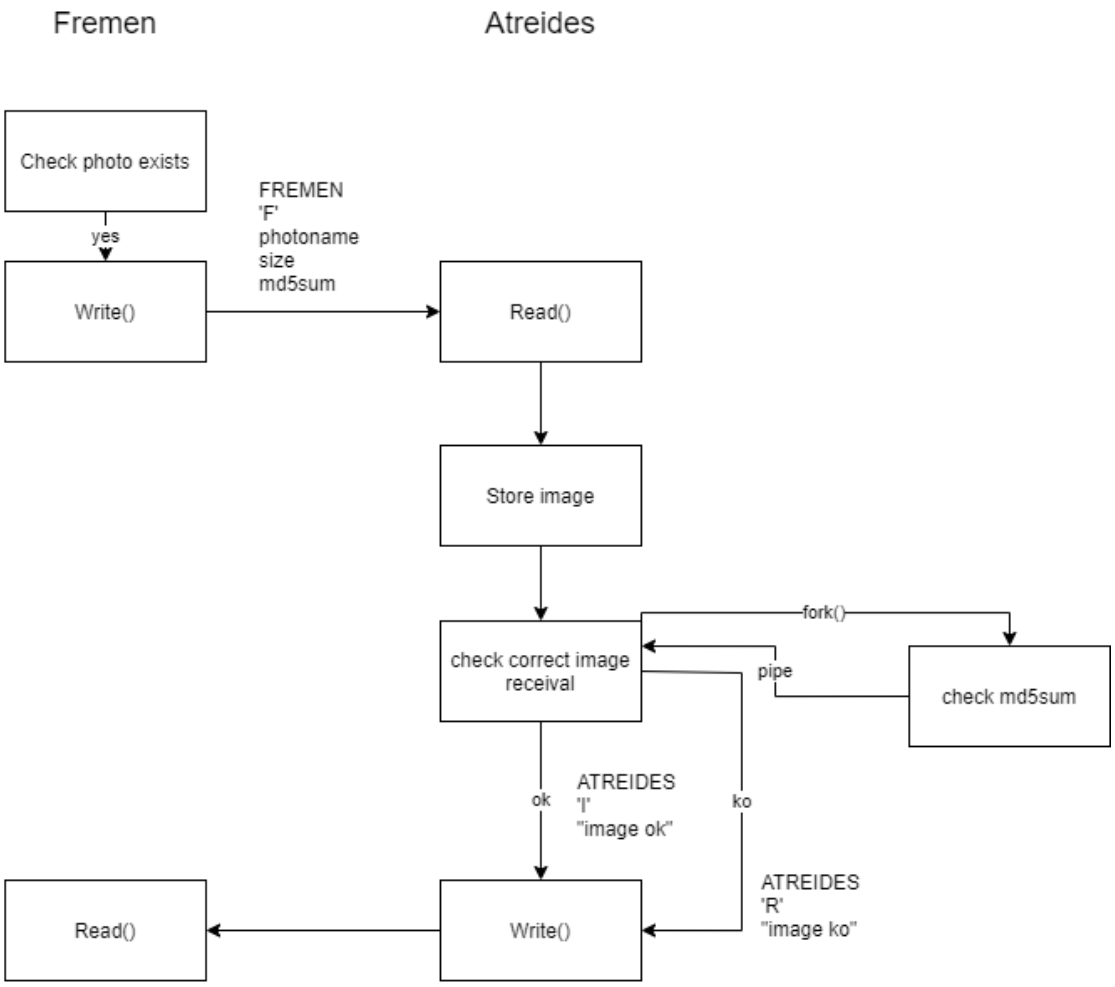
OTHER DIAGRAMS

These diagrams show the communication between Fremen and Artreides while doing a SEND, SEARCH or PHOTO command.

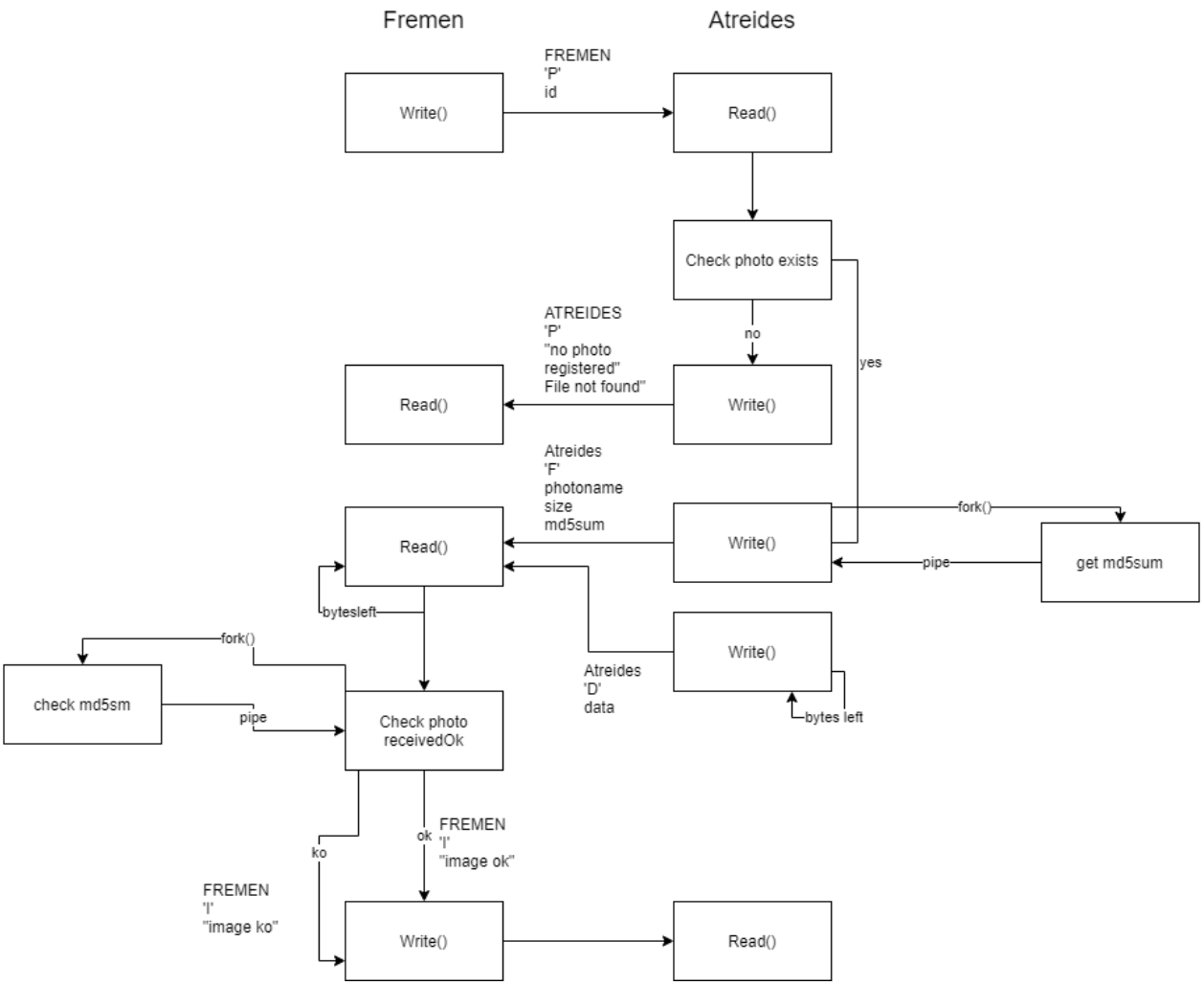
SEND



SEARCH


Fremen
Atreides
Check photo exists
yes
Write()
FREMEN
'F'
photoname
size
md5sum
Read()
Store image
check correct image receival
fork()
pipe
check md5sum
ok
ATREIDES
'I'
"image ok"
ko
ATREIDES
'R'
"image ko"
Read()
Write()

PHOTO


Fremen
Atreides
FREMEN
'P'
id
Write()
Read()
Check photo exists
no
yes
ATREIDES
'P'
"no photo
registered"
File not found"
Read()
Write()
Atreides
'F'
photoname
size
md5sum
Read()
Write()
fork()
pipe
get md5sum
bytesleft
fork()
Write()
Atreides
'D'
data
bytes left
check md5sm
pipe
Check photo
receivedOk
ko
ok
FREMEN
'I'
"image ok"
FREMEN
'I'
"image ko"
Write()
Read()

Observed problems and how they have been solved.

During the development of this project we have encountered many problems. Many of them were relatively easy to solve, but some of them required a lot of research and debugging time.

The valgrind tool was extremely useful to find out if there were any errors in our code such as memory leaks, open file descriptors or errors in memory allocation. This was used during the entire development of the project to ensure we were on the right path and to avoid potential problems.

An error that took some time to solve was the communication of processes when we were sending binary data. We were sending characters checking if they were '\0' which meant that we had reached the end of the string, however binary data can contain this character, so that was a difficult error to spot.

Cleaning the files of the Fremen processes we encountered some problems since we were not managing correctly the directory where the files were stored.

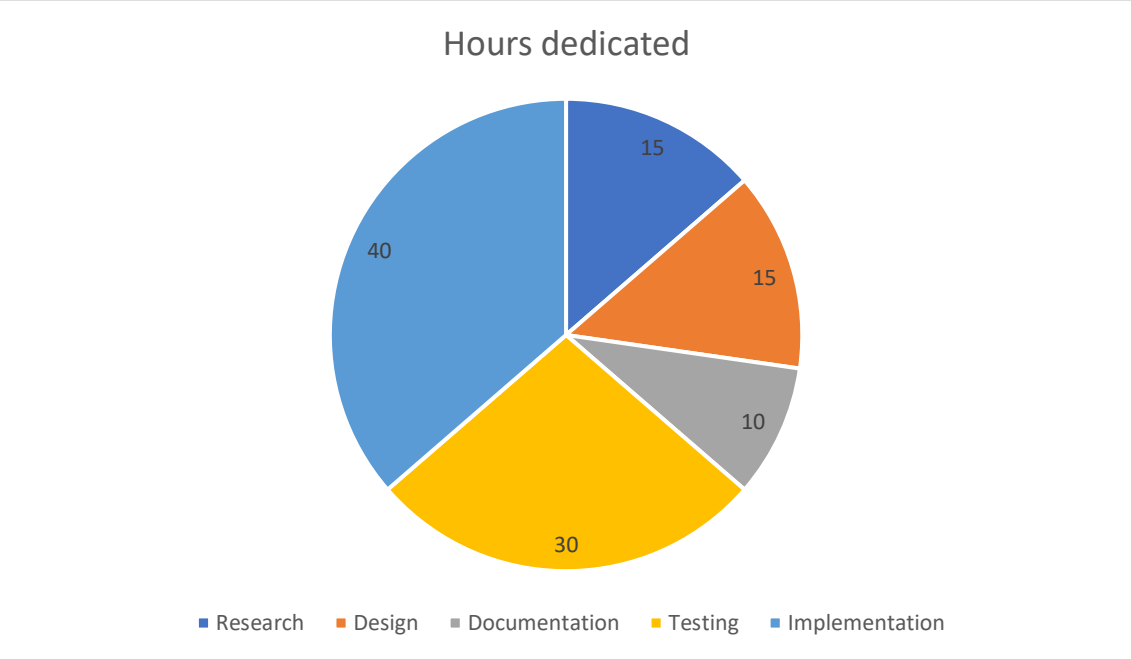
Managing the unexpected disconnection of Fremen and Atreides was also a bit challenging. We were able to do so by checking the return values of the write() and read() functions as well as managing the SIGPIPE signal.

The interruption of the signal to delete images in the Fremen process generated an error in the input that we were reading when this signal was caught. We were storing the input dynamically, and some garbage characters were stored each time the signal was reached. We were able to solve this problem by assigning a static size to the input and doing a single read.

Temporal estimation:

The total estimation of hours dedicated to this practice is about … hours. We will now detail how this hours have been divided and the amount of hours dedicated to each.

- Investigation: hours dedicated to the understanding of the practice, as well as searching on how to do certain functionalities.
- Design: hours dedicated to the design and structure of each module.
- Implementation: hours dedicated to the writing of the code.
- Testing: hours dedicated to debug the practice.
- Documentation: hours dedicated to document all the project and explain our decisions.



Conclusions and proposals for improvement.

To conclude, this practice has been very helpful to understand functions that we have studied in other subjects. We also had to choose wisely our design, to have a correct implementation of the practice.

We had a little trouble with the sessions each week in class, however this practice has helped us to apply the things we saw in class, but to further understand and be more comfortable using them.

WE could not deliver this practice on date and doing it after months of having ended the subject has been a challenge. However, it was just a matter of refreshing concepts and putting in the hours.

We've noticed that we have not used shared memory or queues, but we have not missed them. We actually enjoyed the practice (when it worked), and we really like the liberty to choose our design and implementing it, just with some indications from the statement.

Bibliography used

GeeksforGeeks. (2021). *Handling multiple clients on server with multithreading using Socket Programming in C/C++*. [online] Available at: https://www.geeksforgeeks.org/handling-multiple-clients-on-server-with-multithreading-using-socket-programming-in-c-cpp/ [Accessed 22 May 2022].

man7.org. (n.d.). *opendir(3) - Linux manual page*. [online] Available at: https://man7.org/linux/man-pages/man3/opendir.3.html [Accessed 22 May 2022].

Error! Hyperlink reference not valid.. (n.d.). *C library function - remove() - Tutorialspoint*. [online] Available at: https://www.tutorialspoint.com/c_standard_library/c_function_remove.htm.

Stack Overflow. (n.d.). *stat - How to walk through a directory in c and print all files name and permissions*. [online] Available at: https://stackoverflow.com/questions/20775762/how-to-walk-through-a-directory-in-c-and-print-all-files-name-and-permissions [Accessed 22 May 2022].

Error! Hyperlink reference not valid.. (n.d.). *Creating Directories (The GNU C Library)*. [online] Available at: https://www.gnu.org/software/libc/manual/html_node/Creating-Directories.html#:~:text=The%20mkdir%20function%20creates%20a [Accessed 22 May 2022].

Error! Hyperlink reference not valid.. (n.d.). *C library function - strcpy()*. [online] Available at: https://www.tutorialspoint.com/c_standard_library/c_function_strcpy.htm.

man7.org. (n.d.). *open(2) - Linux manual page*. [online] Available at: https://man7.org/linux/man-pages/man2/open.2.html.

Stack Overflow. (n.d.). *unix - How to capture output of execvp*. [online] Available at: https://stackoverflow.com/questions/762200/how-to-capture-output-of-execvp [Accessed 22 May 2022].

Stack Overflow. (n.d.). *c - How to use execvp()*. [online] Available at: https://stackoverflow.com/questions/27541910/how-to-use-execvp.

Stack Overflow. (n.d.). *How can I get the list of files in a directory using C or C++?* [online] Available at: https://stackoverflow.com/questions/612097/how-can-i-get-the-list-of-files-in-a-directory-using-c-or-c.

man7.org. (n.d.). *read(2) - Linux manual page*. [online] Available at: https://man7.org/linux/man-pages/man2/read.2.html.

man7.org. (n.d.). *scandir(3) - Linux manual page*. [online] Available at: https://man7.org/linux/man-pages/man3/scandir.3.html [Accessed 22 May 2022].

Error! Hyperlink reference not valid.. (n.d.). *C library function - raise()*. [online] Available at: https://www.tutorialspoint.com/c_standard_library/c_function_raise.htm [Accessed 22 May 2022].

GeeksforGeeks. (2017). *pipe() System call*. [online] Available at: https://www.geeksforgeeks.org/pipe-system-call/.

GeeksforGeeks. (2015). *fork() in C*. [online] Available at: https://www.geeksforgeeks.org/fork-system-call/.

GeeksforGeeks. (2017). *Wait System Call in C*. [online] Available at: https://www.geeksforgeeks.org/wait-system-call-c/.

Error! Hyperlink reference not valid.. (n.d.). *Execute a Program: the execvp() System Call*. [online] Available at: http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/exec.html [Accessed 22 May 2022].

Error! Hyperlink reference not valid.. (n.d.). *Computing an MD5Sum in C*. [online] Available at: https://www.unix.com/programming/134079-computing-md5sum-c.html [Accessed 22 May 2022].

man7.org. (n.d.). *write(2) - Linux manual page*. [online] Available at: https://man7.org/linux/man-pages/man2/write.2.html.

man7.org. (n.d.). *read(2) - Linux manual page*. [online] Available at: https://man7.org/linux/man-pages/man2/read.2.html.

man7.org. (n.d.). *listen(2) - Linux manual page*. [online] Available at: https://man7.org/linux/man-pages/man2/listen.2.html [Accessed 22 May 2022].