Report : End of semester assignment - Log server project

Computer systems and programming
2023/2024

Thomas Korpal Matricola : 2118445

This report will quickly present the end of semester project in Computer systems and programming. We needed to create a log server and a client to interact with it.

We will then talk about these two parts and see how are they coded.

Finally we will test the program.

(A part of this work has been inspired by my work on another project which you can find in this link : https://github.com/ThomasKorpal/Claire-ThomasBank)

Part 1 - Server side

The server.c file hosts the server side of the application. There are a few functions in this file:

- Die : a function which prints an error message and execute the exit syscall with value EXIT_FAILURE
- isANumber : function that takes a string as an input and returns 1 if the string corresponds to a number and 0 if not
- generateUniqueFilename : function that create a unique filename for the log file
- createLogFile : function that create the log file with name generated by generateUniqueFilename in directory stored in the global variable log_dir

These are the minor functions of the program. There is still three main functions that we will go over now :

1. setConfigValuesServer :

This function is tasked with setting the values of the configuration variables. That is why it takes the standard arguments as an input.

To start, the function takes the configuration file for the server and gets the values written in it : server port and log directory.
To do that, a combination of reads from the file and some calls to strtok to split the strings will get us the value in both the server_port and log_directory variables.
The program closes the configuration file afterwards.

Then, in case some options have been added in the command line, the program will modify these values. For every case possible (number of options and ordering of these options), the program will isolate the value from the string, verify its validity (is it a valid number for the server_port, is the format correct for the log directory) and modify the associated variable.

The function will also test if the given log directory already exists or not and create it if needed.

2. HandleClient :

This function is tasked with the management of the client's messages. To start, the function will open the log file ("protected" by semaphores). It will write a message stating that a new client is now connected to the server.

It will then enter an infinite loop to wait for incoming messages from the client. When a message arrives, the function will write them on the log file ("protected by semaphores) and then reset the buffer in order to be prepared in case a new message arrives.

3. main :

The main function starts by calling setConfigValuesServer since these values are critical for the rest of the program.
We then initialize the semaphores.

Then we'll start to configure the sockets part. We first create the server socket as a stream socket. We then construct the sockaddr_in struct of the server to give details on incoming addresses and port details.

We then bind the server socket and then make the socket listen for an incoming connection.

After that, the log file is created using the createLogFile function. Its name is printed on the terminal.

The server will then enter an infinite loop to wait for clients to connect. When a client manifests itself and connects to the server, a fork is being done to handle each possible client. The parent process will not handle any client.
In the child processes, we then call the HandleClient function.

We also want to monitor the standard input stdin for any command. Two commands are implemented :

- 'file' : prints out the current log file that is used by the system
- 'exit' : server shutdown command

To do that, we make use of the select call to monitor the standard input and the server socket. stdin also has been put in non-blocking mode to work.

Part 2 - Client side

Let’s now turn to the client side of the application. A lot of things are similar between the two parts of the program like the setup of the config values.

We’ll only go over the differences :

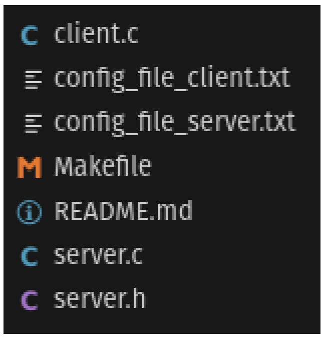
- the isIPValid function. This function takes a string as an input and returns 1 if this string is a valid IP address and 0 if not

- the setConfigValuesClient function. While very similar to the one found in the server part of the program, this instance checks for the server port and the IP address of the server. This is why the isIPValid function is important.

- the main function. The network part of the function is where most differences lie. We first create the client socket and the sockaddr_in of the server. We then try to connect to the server using the specified inputs or the default values. When this is done, the program starts to create the buffer that will be sent to the server. It first waits for the user input on the message that it wants to send. It then creates the message header using the name that the user gave just before, its IP address and the current time (hour, minute and second). It then appends the message to this header and then sends the whole to the server. All the buffers used are then reset for a new message to be sent.

Part 3 - Testing the program

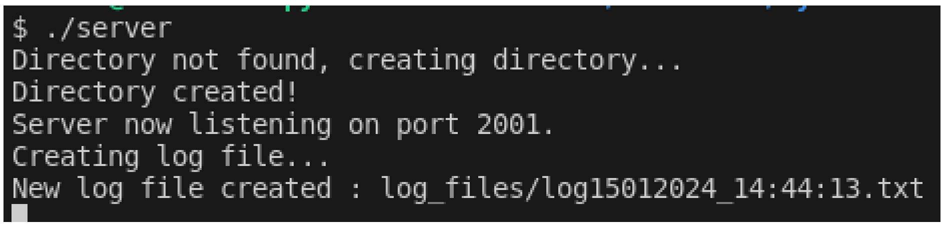
For this part, I will take the Correctness part of the project specifications. Here is the current directory when testing :



```
client.c
config_file_client.txt
config_file_server.txt
Makefile
README.md
server.c
server.h
```

Before doing anything, I execute the command “make” to create both the server and the client executable. We can now test everything :

1. Server startup (start the server only)



```
$ ./server
Directory not found, creating directory...
Directory created!
Server now listening on port 2001.
Creating log file...
New log file created : log_files/log15012024_14:44:13.txt
```

As we can see from the state of the current directory, there wasn’t any directory dedicated for storing the log files. As I only launched the ./server command with no options, the program had to get the default values in the configuration file :

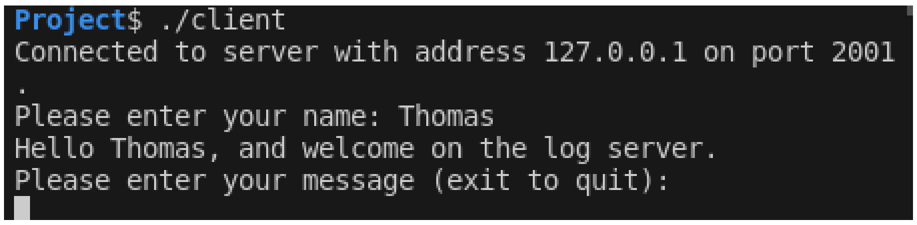
--server_port=2001

--log_directory=log_files/

The log_files directory, as it didn’t exist previously, had to be created by the program. Then, still using the configuration file, the program created the server socket and gave it the 2001 port. After all that has been done, the program can now create the log file with the current date and the current hour, minutes and seconds of its creation.

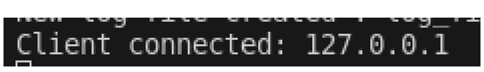
Finally, the server is now waiting for clients to connect.

2. Client connect



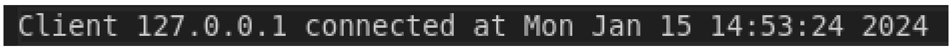
```
Project$ ./client
Connected to server with address 127.0.0.1 on port 2001
.
Please enter your name: Thomas
Hello Thomas, and welcome on the log server.
Please enter your message (exit to quit):
```

We then launch on another terminal the ./client command. It first get the default values from the configuration file as I didn't specified any options in the command. The client then connect to the server using these informations. From the server side :



```
Client connected: 127.0.0.1
```

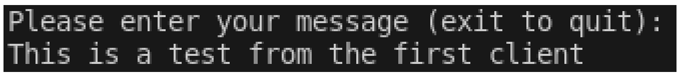
A line stating that a client is now connected has appeared. In the log file :



```
Client 127.0.0.1 connected at Mon Jan 15 14:53:24 2024
```

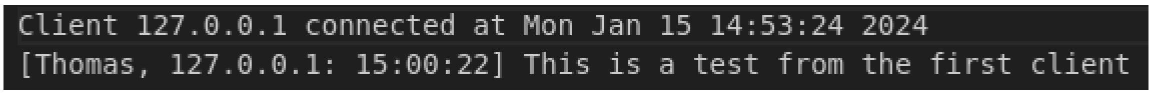
The client program also ask for a name which will be used in the header of log messages. After that, the client waits for the user's input.

3. Client message



```
Please enter your message (exit to quit):
This is a test from the first client
```

The message has been typed on the standard input and was send to the server which will log it. We can now see this in the log file :

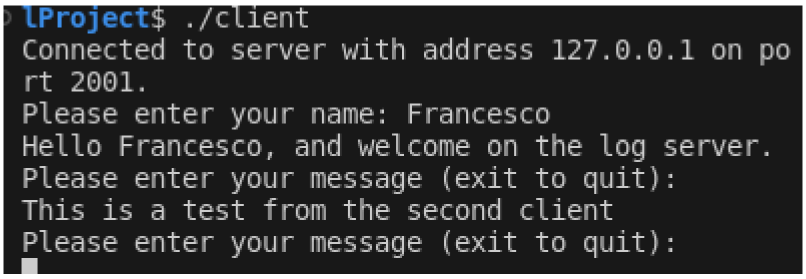


```
Client 127.0.0.1 connected at Mon Jan 15 14:53:24 2024
[Thomas, 127.0.0.1: 15:00:22] This is a test from the first client
```

As we can see, the header is composed of the name and the ip address of the client as well as a timestamp. Since the files are named with the current date, the header only includes the time of the day.

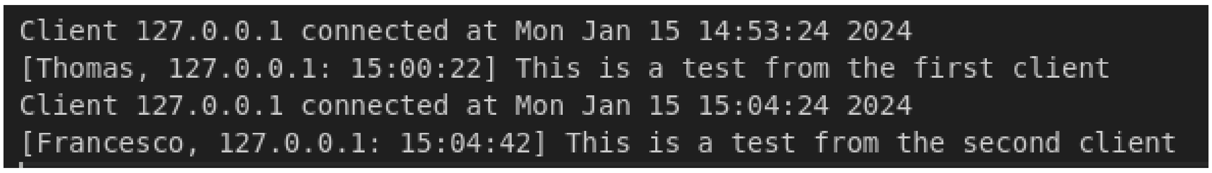
4. Second client message

Here the process is the same as with the first client. On a third terminal, we launch the "./client" command and enter a name (in this case, the name will be Francesco). We then enter a message :



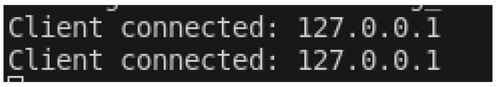
```
lProject$ ./client
Connected to server with address 127.0.0.1 on po
rt 2001.
Please enter your name: Francesco
Hello Francesco, and welcome on the log server.
Please enter your message (exit to quit):
This is a test from the second client
Please enter your message (exit to quit):
```

The state of the log file is now :



```
Client 127.0.0.1 connected at Mon Jan 15 14:53:24 2024
[Thomas, 127.0.0.1: 15:00:22] This is a test from the first client
Client 127.0.0.1 connected at Mon Jan 15 15:04:24 2024
[Francesco, 127.0.0.1: 15:04:42] This is a test from the second client
```

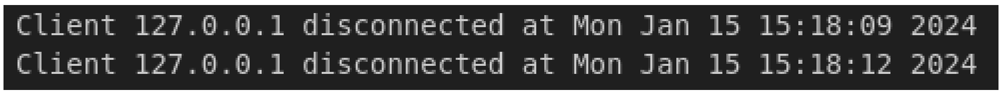
The connection also has been registered on the server terminal :



```
Client connected: 127.0.0.1
Client connected: 127.0.0.1
```

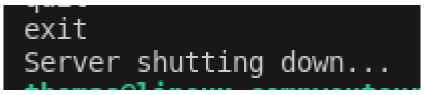
5. Client shutdown

As we saw from the input prompt in the client terminals, if you type exit, the client program will shutdown. The server is programmed to register these specific queries coming from the clients and to log their disconnection in the log file :



```
Client 127.0.0.1 disconnected at Mon Jan 15 15:18:09 2024
Client 127.0.0.1 disconnected at Mon Jan 15 15:18:12 2024
```

6. Server shutdown



```
exit
Server shutting down...
```

The exit command on the server terminal makes the server shutdown.