Assignment: Socket programming

Introduction

After reviewing the video lectures and the examples about socket programming that are available at the Virtual Campus of this course, the students should be able to develop their own application protocol for a simple client-server application

Objective: Implementing a simple FTP server.

The goal of this assignment is to practice the implementation of a simple FTP server. The basic functions of the server will be implemented, so that, it allows to upload and download files from a single folder.

1 FTP (File Transfer Protocol)

The File Transfer protocol is thought for transferring files between a client and a server in order to store or retrieve them from the server. This protocol bases on the TCP transport protocol and uses the well-known ports 20 and 21. Two TCP connections are involved: The control connection (port 21) and the data connection (port 20). The control connection is used to transfer commands and the replies to these commands. The transfer of a file needs to use several commands. Note that here the word "command" does not refer to the commands that the user types into the command line of the FTP client program, it refers to the textual commands, defined in the FTP protocol, that the client will send to the server.

Two different file transfer modes can be distinguished:

- **Active mode**: The client connects from a unprivileged port N to the server command port (port 21). Then, the client starts listening to port M and sends the FTP command PORT M to the FTP server. The server will then connect back from port 20 to port M of the client. Figure 1 illustrates the active mode.

  The main problem with this procedure is that the server connects back to the client, which often is a problem with firewalls that drop non known incoming connections.

- **Passive mode**: To solve the issue that arises when using active mode, the clients support passive mode. In passive mode, the client initiates both connections (control and data).

  When opening an FTP connection, the client opens two random unprivileged ports locally. The first port contacts the server on port 21. Then the client issues a `PASV` command and the server responds with a port number. After that the client connects to that port in order to transfer the data. This is illustrated in figure 2

2 Read documentation about system calls related with sockets.

It is recommended to study the Linux System Calls related to sockets. The related Linux `man` pages have enough documentation about these functions. The resources and videos available at the Virtual Campus are also useful. You should study at least the following system calls: `socket, bind, connect, listen, accept, send, recv, shutdown, close`. You also have to read the man pages of the `fdopen` function, which will be useful in your development.

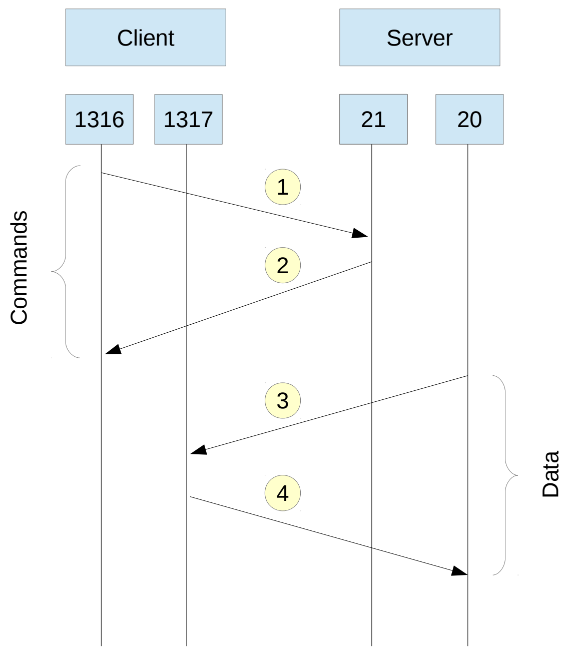

Figure 1: Active mode.

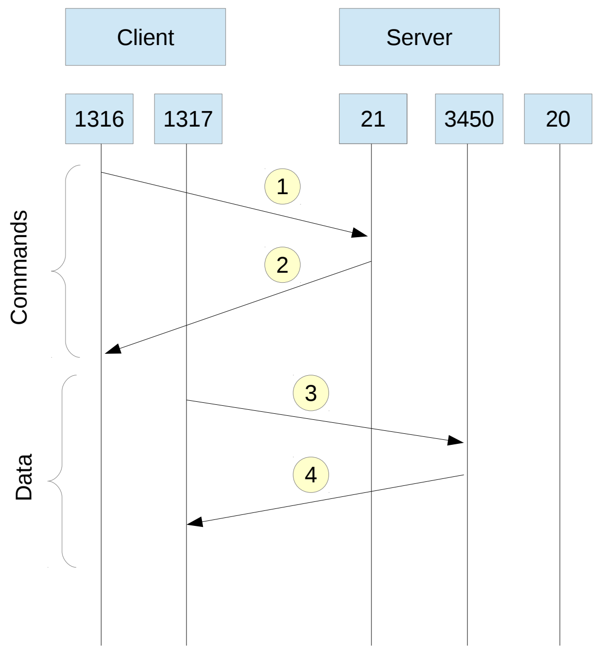

Figure 2: Passive mode.

3 Code template.

To facilitate the implementation, this assignment provides a code template that can be used. This template is composed of the following files:

- `common.h`: Common functions
- `ftp_server.cpp`: Main program.
- `FTPServer.h`: C++ class definition of the FTP server.
- `FTPServer.cpp`:C++ class implementation of the FTP server.
- `ClientConnection.h`: C++ class definition of the `ClientConnection` class that manages the information between the server and each client.
- `ClientConnection.cpp`: Implementation of the `ClientConnection` class.
- Makefile.

Check the comments in the files in order to know where you have to modify them. Mostly you have to change `FTPServer.cpp` and `ClientConnection.cpp`. Other minor changes may be necessary in other files depending on your implementation.

4 FTP protocol commands to implement.

Each server should implement at least the following FTP commands: `PORT`, `PASV`, `STOR`, `RETR`, LIST. Other necessary commands are currently implemented in the code template. Each command sent over the control connection must return an error code and a descriptive message. The different codes and messages can be found in RFC 959. The server should allow both transfer modes, passive and active. The control connection will be done over the port 2121, instead of the well known port for FTP (21), in order to run the program as normal user.

5 Common issues

During the development process you may experience some common issues that you shold take into account:

- The only part to develop is the server (the code template is for the server). The Unix `ftp` command will be used for debuging purposes.
- Client and server are executed on the same machine. If you execute both programs in the same folder and a `get` or `put` command is performed, the uploaded or downloaded file is simultaneously read and written and the process will fail and probably the file will be truncated to zero.
- Each command must return an error code and a text message that is specified in RFC 959. Please, follow the standard. Otherwise, strange behavours may happen.
- Do not confuse the FTP protocol commands with the commands of the Linux console FTP client. These are different things. FTP protocol commands are sent over the control socket while user interface commands are typed by user into the CLI (Command Line Interface) of the FTP client.
- The order in which you implement the FTP commands is important. Some commands like RERT,STOR and LIST depend on the result of previously executed commands, like PORT or PASV. The should be implemented first. Otherwise you will not be able to debug your program.

6 Debugging and testing.

In order to test the implemented server, the `ftp` command should be used with the debug flag (`-d`). This shows the FTP protocol messages sent by the client, that normally are hidden.

The following sections show the expected behavior of each of the use cases that you have to implement. The lines beginning with `--->` are due to the debug flag and show the protocol commands sent by the client.

6.1 Getting a file in active mode

```
user@localhost:~/pruebas_ftp$ ftp -d
ftp> open localhost 2121
Connected to localhost.
220 Service ready
ftp: setsockopt: Bad file descriptor
Name (localhost:jonas):
---> USER jonas
331 User name ok, need password
Password:
---> PASS XXXX
230 User logged in
---> SYST
215 UNIX Type: L8.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> get README
local: README remote: README
---> TYPE I
200 OK.
ftp: setsockopt (ignored): Permission denied
---> PORT 127,0,0,1,229,250
200 OK.
---> RETR README
150 File status okay; about to open data connection
226 Closing data connection.
2480 bytes received in 0.00 secs (33176.4 kB/s)
ftp>
```

6.2 Getting a file in passive mode

```
user@localhost:~/pruebas_ftp$ ftp -d
ftp> open localhost 2121
Connected to localhost.
220 Service ready
ftp: setsockopt: Bad file descriptor
Name (localhost:jonas): jonas
---> USER jonas
331 User name ok, need password
Password:
---> PASS XXXX
230 User logged in
---> SYST
215 UNIX Type: L8.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> passive
```

```
Passive mode on.
ftp> get README
local: README remote: README
---> TYPE I
200 OK.
ftp: setsockopt (ignored): Permission denied
---> PASV
227 Entering Passive Mode (127,0,0,1,129,232).
---> RETR README
150 File status okay
226 Closing data connection.
2480 bytes received in 0.00 secs (41756.5 kB/s)
```

6.3 Putting a file in active mode

```
user@localhost:~/pruebas_ftp$ ftp d
ftp> open localhost 2121
Connected to localhost.
220 Service ready
ftp: setsockopt: Bad file descriptor
Name (localhost:jonas): jonas
---> USER jonas
331 User name ok, need password
Password:
---> PASS XXXX
230 User logged in
---> SYST
215 UNIX Type: L8.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put README
local: README remote: README
> TYPE I
200 OK.
ftp: setsockopt (ignored): Permission denied
---> PORT 127,0,0,1,223,168
200 OK.
---> STOR README
150 File creation ok, about to open data connection
226 Closing data connection.
2480 bytes sent in 0.00 secs (86495.5 kB/s)
ftp>
```

6.4 Putting a file in passive mode

```
user@localhost:~/pruebas_ftp$ ftp -d
ftp> open localhost 2121
Connected to localhost.
220 Service ready
ftp: setsockopt: Bad file descriptor
Name (localhost:jonas):
---> USER jonas
331 User name ok, need password
Password:
```

```
---> PASS XXXX
230 User logged in
---> SYST
215 UNIX Type: L8.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> passive
Passive mode on.
ftp> put README
local: README remote: README
---> TYPE I
200 OK.
ftp: setsockopt (ignored): Permission denied
---> PASV
227 Entering Passive Mode (127,0,0,1,190,81).
---> STOR README
150 File creation ok
226 Closing data connection.
2480 bytes sent in 0.00 secs (89699.1 kB/s)
```

6.5 `ls` in active mode

```
ftp> ls
ftp: setsockopt (ignored): Permission denied
---> PORT 127,0,0,1,162,188
200 OK.
---> LIST
ftp_server
README
Makefile
250 List completed successfully.
```

6.6 `ls` in passive mode

```
ftp> ls
---> TYPE A
200 OK.
ftp: setsockopt (ignored): Permission denied
---> PASV
227 Entering Passive Mode (127,0,0,1,231,72).
---> LIST
125 List started OK.
ftp_server
README
Makefile
250 List completed successfully.
```

7 Report

After completing the assignment, each group of students has to write a report (max. 5 pages) covering the following topics:

- A brief description of the developed application.
- A description of the developed protocol.

- A guide for the compilation of the source code and the necessary steps to execute the server program.
- Test cases
- Appendix: Source code.

The report must be written in ENGLISH LANGUAGE.