A Rust Implementation of FTP Server and Client

Pachev Joseph
PID: 5699044
CNT 4713

I. Introduction

In this project, the task was to implement the File Transfer Protocol(FTP) client and server in accordance to RFC 959[1]. This was accomplished using the systems programming language Rust[2]. To do this, extra research was necessary into the intricacies of that certain programming language. The result is two extremely fast and reliable applications that work well with each other and with any other application that meets the RFC standard.

The client is able to perform the most common commands that are necessary for a seamless FTP experience. Similarly, the server is able to handle multiple clients and process the most common requests that an RFC compliant client will send. Further, They are both able to work together or as standalone applications.

II. Problem Statement

The File Transfer Protocol is used to transfer files between computers[1]. One computer is the server, which is able to service requests from clients. The clients are normally users who wish to retrieve files from the server. In order to get a better understanding of communication on the internet, an FTP client and server was developed. The problem was to implement these two applications in accordance to RFC 959 standards with the guidelines from a project specification; all done in a given time.

III. Methodology

The simplest place to start on this project was the FTP client. The client sends requests to the server and the server responds by performing these requests. The client is written in the Rust programming language using tcp sockets, a form of communication between two computers on a network[3]. On the client side, the process involves creating a socket, connecting to a remote connection, sending data, receiving data, and lastly closing the connection.

In addition to creating the socket, the client needed to process the commands that the user will input. Once a command is received, the client verifies that if can perform the command, then sends the appropriate request to the server. In the case that the user wants to send or receive multiple files, the client uses threads to handle each file. Threads are a small tasks within a process that execute code[3].

The FTP server is also written in Rust and uses sockets for communication. However, the process is longer than that of the client. The server must create a socket, bind to a specific address and port, listen for incoming connections, then accept those connections, send data, receive data, and lastly close the socket. Beyond this overview, several details needed to be handled within the server. First, the server also listens on a separate port called the service port. This port is used by an administrator level user to start, pause, and stop the server remotely.

Next, the server needs to handle multiple clients simultaneously. This was also done using threads. By using threads, multiple clients are able to connect to the server and have their request handled as opposed to waiting one by one. The service port is also running in the background through its own thread. The server keeps a database of users with different access roles. This was accomplished using a file called ‘users.cfg’, which contained the list of users. Each time the application is started, the users are initialized. If they don’t already exist, a separate directory is created under the root directory for that user. Lastly, the server does it’s most important task; which is handle the many requests that are sent by the client.

Both the FTP client and server use configuration files that give them default behaviors. In addition to the default behaviors, command line arguments can be passed to the applications upon startup. Furthermore, in accordance with RFC 959, both applications use a command port to send and receive commands, and a data port to send and receive files. There are two modes of doing this,

passive and active mode. Passive mode tells the server to assign a data port to which the client can create a socket and connect to. Active mode tells the client to perform this assignment of data port. The final product supports both modes in the client and in the server.

IV. Results

The following figures demonstrate the results of several tests that were ran between the client and the server simultaneously. Figure1 shows the side by side results of successful login and file upload. Figure2 shows the results of an unsuccessful login by failing to login more than three times. Figure3 shows the results of successful retrieval, storage, and deletion of files. In contrast, figure 4 shows the results of trying to create files that already exist, removing and retrieving nonexistent files.

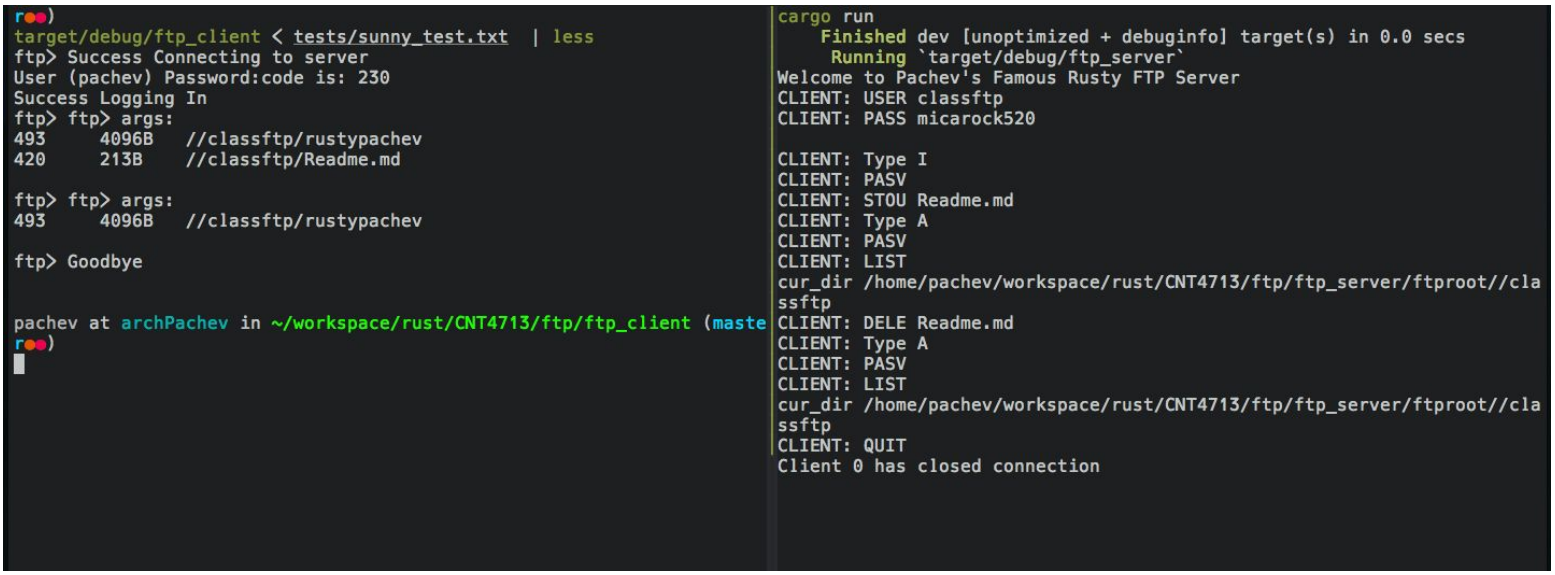

Figure1: Result of First Sunny Test

```
pachev at archPachev in ~/workspace/rust/CNT4713/ftp/ftp_client (master●●)
target/debug/ftp_client < tests/rainy_test.txt  | less
ftp> Success Connecting to server
User (pachev) Password:code is: 430
Login Failed
ftp> User (pachev) Password:code is: 430
Login Failed
ftp> User (pachev) Password:code is: 430
Login Failed
ftp> Goodbye

pachev at archPachev in ~/workspace/rust/CNT4713/ftp/ftp_client (master●●)
```

```
pachev at archPachev in ~/workspace/rust/CNT4713/ftp/ftp_server (master●)
cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.0 secs
     Running `target/debug/ftp_server`
Welcome to Pachev's Famous Rusty FTP Server
CLIENT: USER classftp
CLIENT: PASS micarock521

CLIENT: USER classftp
CLIENT: PASS micarock522

CLIENT: USER classftp
CLIENT: PASS micarock523

Client 0 has closed connection
```

Figure2: Result of three unsuccessful logins

```
----> PASV

SERVER: 227 Entering Passive Mode (127,0,0,1,107,108).

----> LIST

args:
SERVER: 150 Openning ASCII mode data for file list

493     4096B   //classftp/rustypachev
493     4096B   //classftp/test_dir

SERVER: 226 Transfer Complete

ftp> ----> MKD test_dir

SERVER: 257 test_dir creation success

ftp> ----> CWD test_dir

SERVER: 250 CWD Command Success

ftp> ----> Type I

SERVER: 200 Type set to I

----> PASV

SERVER: 227 Entering Passive Mode (127,0,0,1,107,108).

----> STOU readme.md

SERVER: 150 Opening binary mode to receive readme.md

SERVER: 226 Transfer Complete

ftp> ----> Type I

SERVER: 200 Type set to I

----> PASV

SERVER: 227 Entering Passive Mode (127,0,0,1,107,108).

----> STOU nono.txt

SERVER: 150 Opening binary mode to receive nono.txt

SERVER: 226 Transfer Complete
```

```
CLIENT: Type A
CLIENT: PASV
CLIENT: LIST
cur_dir /home/pachev/workspace/rust/CNT4713/ftp/ftp_server/ftproot//classftp
CLIENT: MKD test_dir
CLIENT: CWD test_dir
user path: /home/pachev/workspace/rust/CNT4713/ftp/ftp_server/ftproot//classftp
user path: /home/pachev/workspace/rust/CNT4713/ftp/ftp_server/ftproot//classftp
new cur path: /home/pachev/workspace/rust/CNT4713/ftp/ftp_server/ftproot//classftp/test_d
ir
CLIENT: Type I
CLIENT: PASV
CLIENT: STOU readme.md
CLIENT: Type I
CLIENT: PASV
CLIENT: STOU nono.txt
CLIENT: Type I
CLIENT: PASV
CLIENT: RETR nono.txt
/home/pachev/workspace/rust/CNT4713/ftp/ftp_server/ftproot//classftp/test_dir/nono.txt re
quested file
CLIENT: Type I
CLIENT: PASV
CLIENT: STOR Readme.md
CLIENT: PASV
CLIENT: STOR trial.txt
CLIENT: Type A
CLIENT: PASV
CLIENT: LIST
cur_dir /home/pachev/workspace/rust/CNT4713/ftp/ftp_server/ftproot//classftp/test_dir
CLIENT: DELE readme.md
CLIENT: DELE nono.txt
CLIENT: DELE trial.txt
CLIENT: Type A
CLIENT: PASV
CLIENT: LIST
cur_dir /home/pachev/workspace/rust/CNT4713/ftp/ftp_server/ftproot//classftp/test_dir
CLIENT: CDUP
CLIENT: Type A
CLIENT: PASV
CLIENT: LIST
cur_dir /home/pachev/workspace/rust/CNT4713/ftp/ftp_server/ftproot//classftp
CLIENT: RMD test_dir
CLIENT: Type A
CLIENT: PASV
CLIENT: LIST
cur_dir /home/pachev/workspace/rust/CNT4713/ftp/ftp_server/ftproot//classftp
CLIENT: QUIT
Client 0 has closed connection
```

Figure3: Results of Successful File creating and Deletions

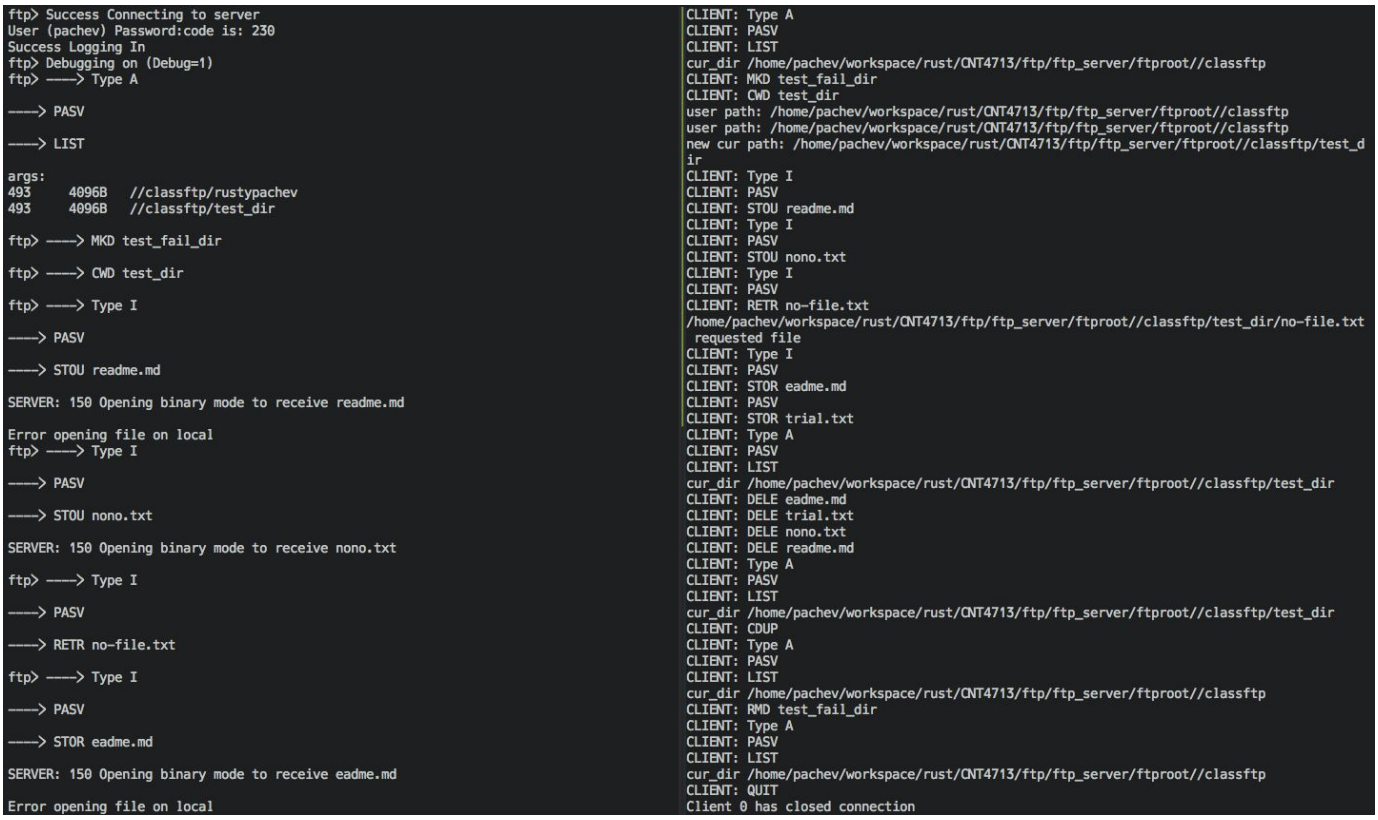

Figure4: Results of unsuccessful attempts of getting and putting nonexistent files

Although not as comprehensive as unit tests, these systems tests demonstrate the use of the FTP client with the server. Both are able to communicate with one another for the most common FTP commands.

V. Analysis

The process of designing, writing, and testing the two applications took two weeks. In those two weeks, the problems that were encountered dealt with the inner workings of the new programming language. Rust focuses on type safety. This is a great advantage for a developer as you do not need to worry about memory leaks or stray behavior due to side effects. However, to achieve this, Rust uses an ownership system[2]. The ownership system is Rust's way of ensuring speed without the use of a garbage collector. In short, each object can only be used once; afterwards, it is removed from memory. To extend the use of

objects, one must borrow and transfer ownership. To someone who is unfamiliar with the language, this can become a great barrier to cross.

Other interesting problems were logging, and using linux redirection for test files. The logging was solved by using a global object that logged everything and only displayed them if certain flags were set. For testing, instead of parsing a text file and running it through the program, redirection was used to send contents of a text file as input to the FTP client. This allowed the most flexibility for testing as long as the file was formatted properly.

The project gave a very good insight on communication throughout the web. It was also a great opportunity to learn the Rust programming language. The experience of writing the client and server reinforces everything that was taught in class. These includes sockets, threads, tcp and ip. Overall, the project was a great learning experience in solving problems in an unfamiliar area.

VI. References

[1] J. R. J. Postel, “RFC 959,” *IEFT*, 01-Oct-1985. [Online]. Available: https://www.ietf.org/rfc/rfc959.txt. [Accessed: 04-Mar-2017]

[2] “The Rust Programming Language,” *Rust Book*, 09-Feb-2017. [Online]. Available: https://doc.rust-lang.org/book/. [Accessed: 11-Feb-2017]

[3] K. W. R. James F. Kurose, *Computer Networking: A Top Down Approach*, 6th Edition. Pearson Ed, 2013.