

Implementação de um servidor FTP

Pedro Augusto Araujo da Silva de Almeida Nava Alves

1 de dezembro de 2019

Contents

1	Introdução	3
2	Organização	3
2.1	USER	3
2.2	PASS	3
2.3	PORT	3
2.4	LIST	4
2.5	CWD	4
2.6	RETR	4
2.7	STOR	4
2.8	QUIT	5
2.9	Not Implemented	5
3	Diagrama de Mensagens Trocados	6
4	Exemplo	10
5	Conclusão	10

1 Introdução

Este trabalho possui como objetivo aplicar os conhecimentos vistos em sala ao implementar um servidor FTP. O servidor FTP é capaz de se comunicar com cliente FTP tercerizados com passagem de mensagens tradicionais segundo a documentação. Foram implementados os seguintes comandos: `cd`, `ls`, `get` e `put`.

O servidor foi Implementado em Python através da biblioteca nativa de `sockets`. O servidor aceita conexões `localhost`, e é implementado em `threads`, criando até 5 `threads`, podendo então aceitar até 5 conexões com clientes diferentes ao mesmo tempo.

2 Organização

O servidor é implementado tal que ele fique ouvindo a porta 21 esperando alguma conexão de cliente FTP, assim que há requisição de conexão, ele aceita a conexão e cria um objeto servidor thread que irá lidar com a conexão, processando as mensagens enviadas pelo cliente, podendo criar até 5 `threads` de servidor. O objeto servidor thread armazena o `socket` e o endereço da conexão assim que é criado, ele envia o código 200, informando que está pronto para uso e entra em loop esperando as mensagens do cliente servidor. As mensagens podem ser: `USER`, `PASS`, `PORT`, `LIST`, `CWD`, `RETR`, `STOR` e `QUIT`.

2.1 USER

A mensagem `USER` informa o nome do usuário usando o servidor. Como o servidor não possui uma tabela de usuários salvos, ele aceita qualquer usuário, enviando o código "331 OK", informando que o usuário foi aceito e requer senha.

2.2 PASS

A mensagem `PASS` informa a senha do usuário usando o servidor. Como o servidor não possui uma tabela de usuários salvos, ele aceita qualquer usuário independentemente de senha, enviando o código "230 OK", informando que foi feito o login.

2.3 PORT

A mensagem `PORT` envia para servidor o endereço e porta utilizados para o `socket` de dados, ele é geralmente enviado quando o cliente ou servidor estão prestes a utilizar o `socket` de dados. Assim que o servidor recebe este comando, ele processa a mensagem enviada calculando o endereço e porta da conexão de dados, salvados-os, e enviando ao cliente que a porta foi recebida com o código "200 Port Received".

2.4 LIST

A mensagem LIST envia para servidor que o usuário está requerindo as informações dos arquivos presentes no diretório atual. Assim que o servidor recebe este comando, ele informa o cliente que está prestes a abrir o socket de dados com o código "150 Data Socket Opening". A partir do endereço e porta de dados previamente salvos, ele tenta abrir a conexão, em caso de falha ele envia o código "425 Could Not Open Data" informando a falha, em caso de sucesso, o servidor obtém os arquivos no diretório atual, e os envia pela conexão de dados, por fim o servidor fecha o socket de dados, informando o cliente que o diretório foi enviado com o código "226 Directory Sent".

2.5 CWD

A mensagem CWD serve para mudar o diretório atual de acordo com o diretório passado pelo cliente. Assim que o servidor recebe este comando, ele calcula se o diretório passado é global ou local, chamando então uma chamada do sistema para alterar o diretório local, por fim, informa o cliente do sucesso enviando o código "250 OK". Caso o diretório não seja localizado, o servidor informa a falha enviando o código "550 Folder not Found".

2.6 RETR

A mensagem RETR serve para obter um arquivo no servidor. Assim que o servidor recebe este comando, ele abre o arquivo passado, em caso de falha, ele informa o cliente com o código "550 File not Found", em caso de sucesso informa o cliente que está prestes a abrir o socket de dados com o código "150 Data Socket Opening". A partir do endereço e porta de dados previamente salvos, ele tenta abrir a conexão, em caso de falha ele envia o código "425 Could Not Open Data" informando a falha, em caso de sucesso, o servidor entra em loop lendo 1Kb do arquivo e enviando imediatamente pelo socket de dados, em cada iteração, até ter lido o arquivo completamente. Por fim, o servidor fecha o socket de dados, informando o cliente que o arquivo foi enviado com o código "226 Transfer Complete".

2.7 STOR

A mensagem STOR serve para armazenar um arquivo no servidor. Assim que o servidor recebe este comando, ele cria o arquivo passado, em caso de falha, ele informa o cliente com o código "450 Cant open new file", em caso de sucesso informa o cliente que está prestes a abrir o socket de dados com o código "150 Data Socket Opening". A partir do endereço e porta de dados previamente salvos, ele tenta abrir a conexão, em caso de falha ele envia o código "425 Could Not Open Data" informando a falha, em caso de sucesso, o servidor entra em loop lendo 256B do socket de dados, e escrevendo imediatamente no arquivo criado em cada iteração, até ter recebido todos os dados pelo socket de dados.

Por fim, o servidor fecha o socket de dados, informando o cliente que o arquivo foi recebido com o código "226 Transfer Complete".

2.8 QUIT

A mensagem QUIT serve informar que o cliente deseja desconectar do servidor. Assim que o servidor recebe esta mensagem, ele corta a conexão enviando o código "221 Goodbye".

2.9 Not Implemented

Caso o cliente envie alguma outra mensagem diferente destas vistas anteriormente, como o servidor não tem conhecimento de como trata-las, ele apenas informa o cliente que tal funcionalidade não foi implementada com o código "500 Not Implemented".

3 Diagrama de Mensagens Trocados

A seguir estão os diagramas das mensagens trocadas nos 4 principais comandos: cd, ls, get e put.

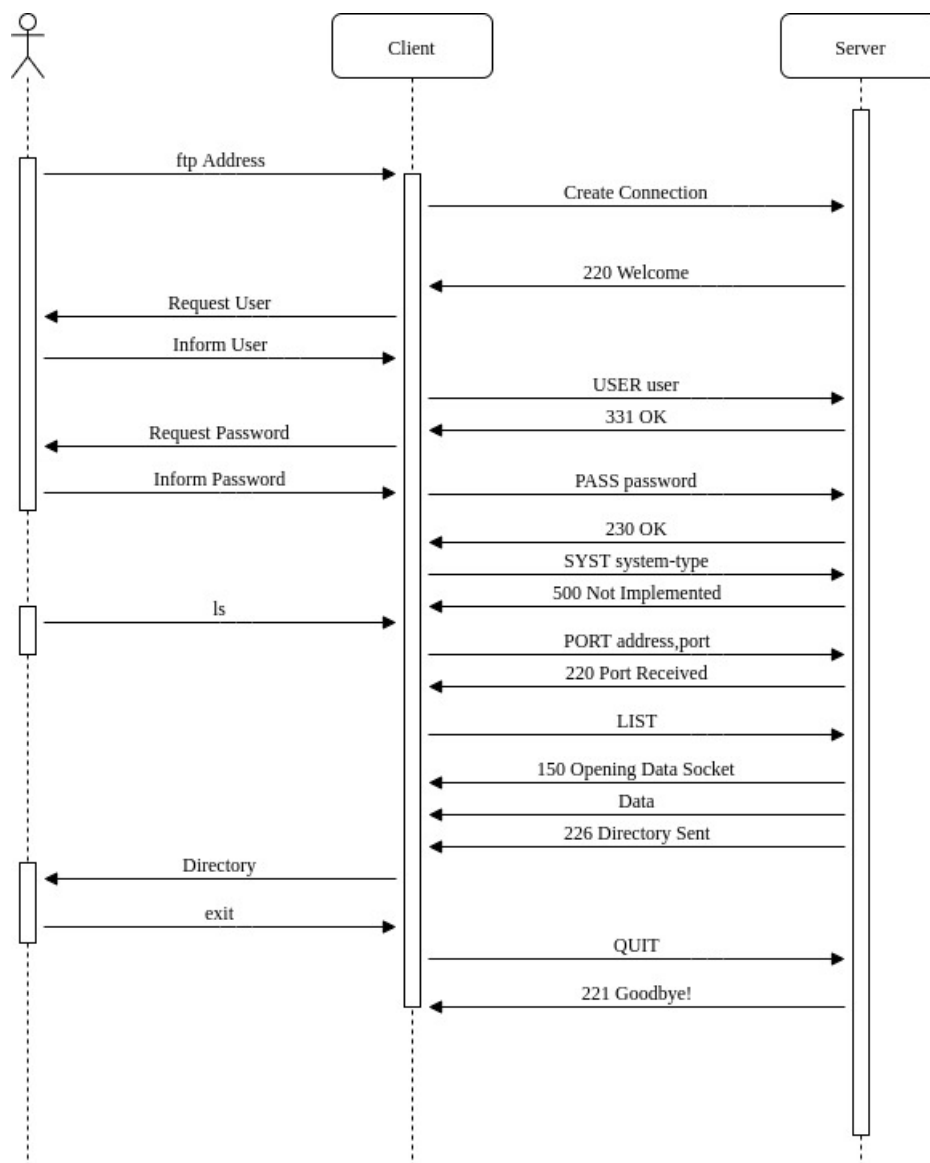


Figure 1: Diagrama ls

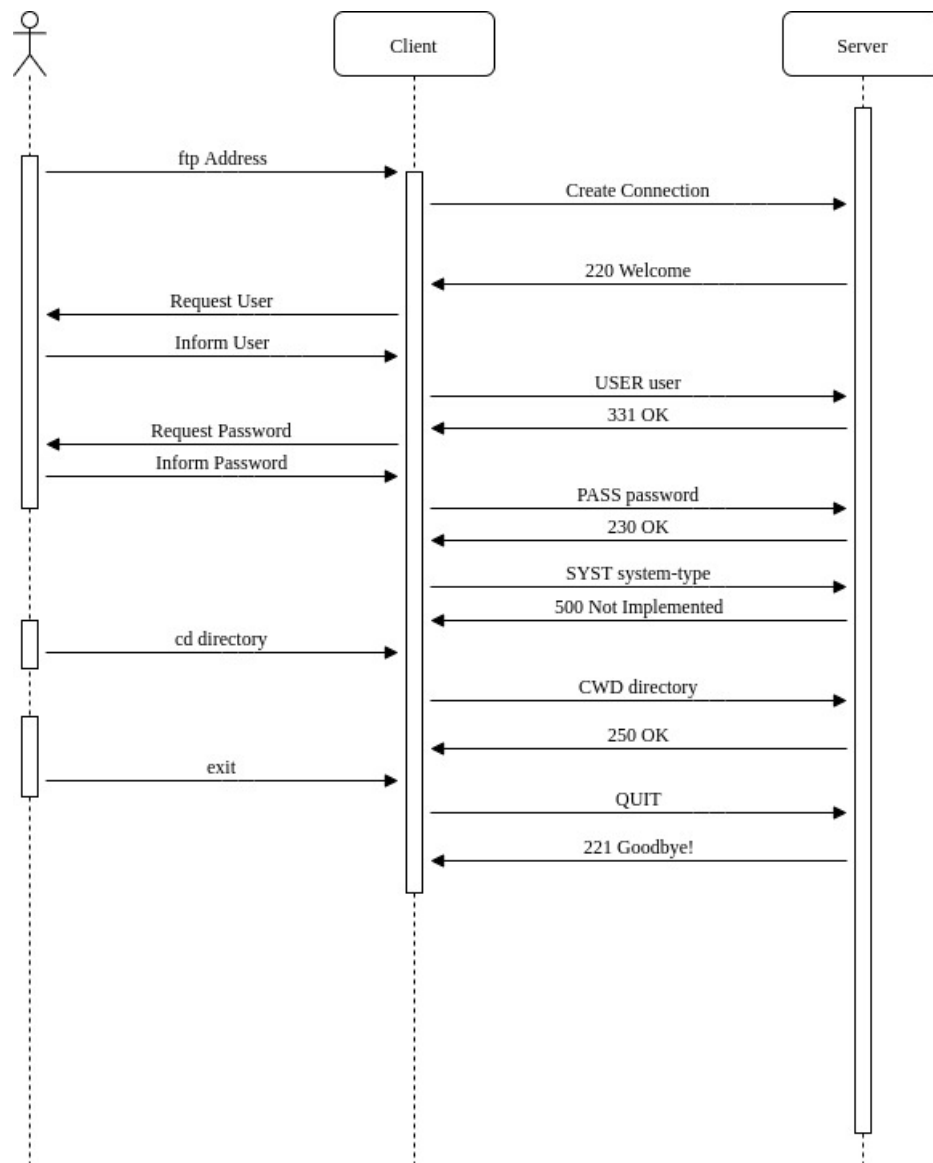


Figure 2: Diagrama cd

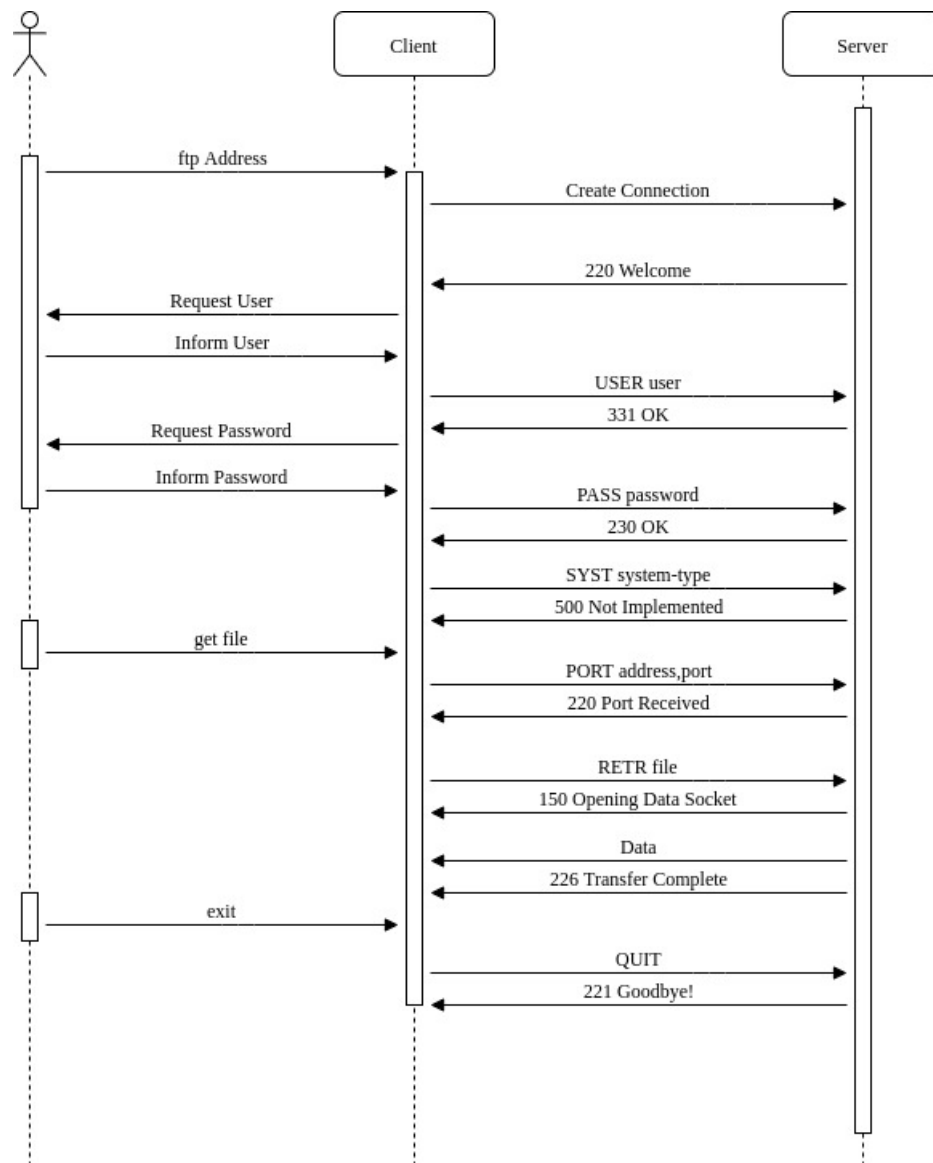


Figure 3: Diagrama get

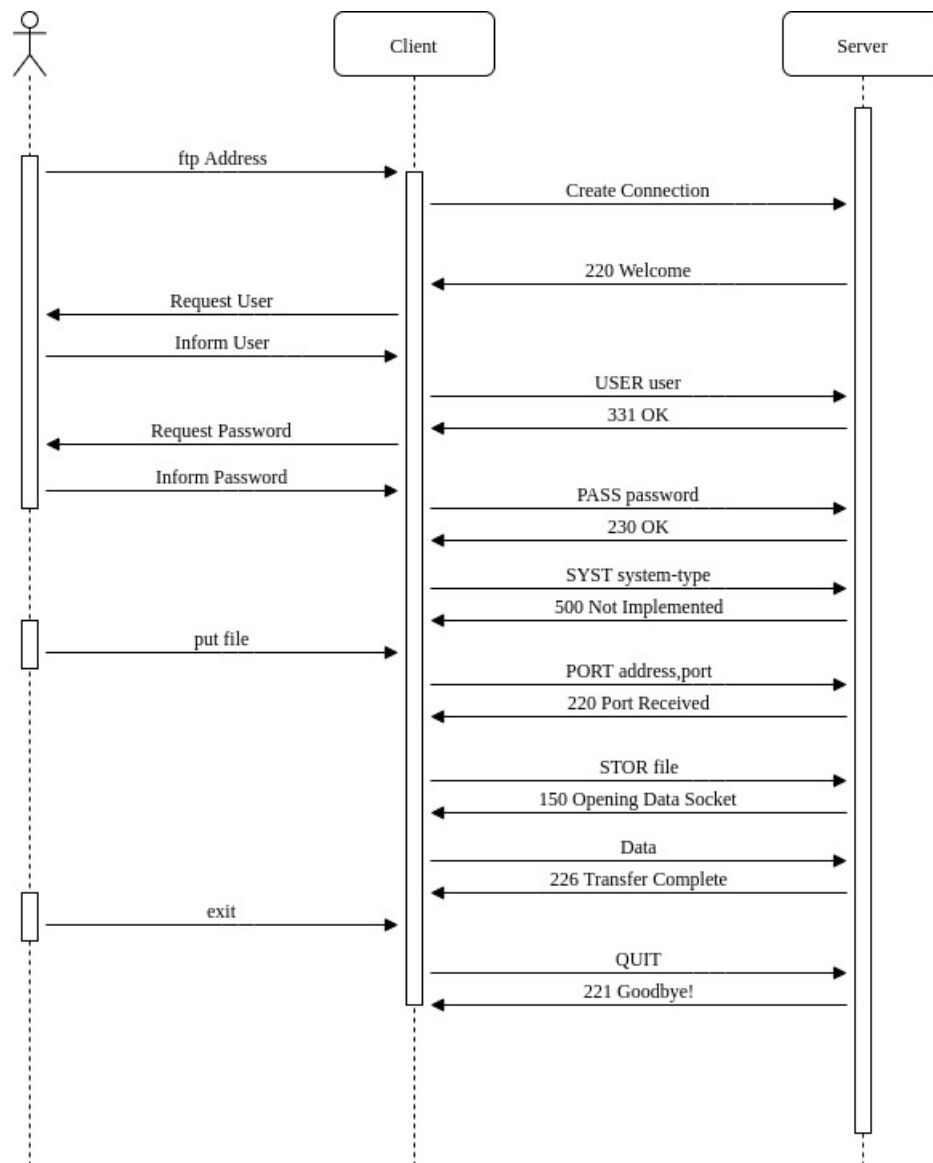
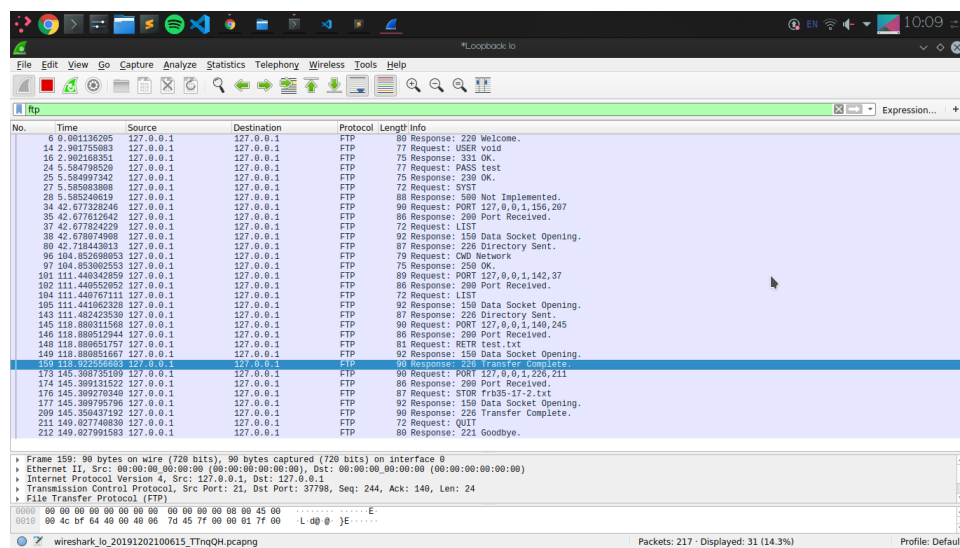


Figure 4: Diagrama put

4 Exemplo

Como demonstração do servidor funcionando, foi executado o servidor e em seguida o cliente servidor do Linux, e foi feita a conexão localhost e então utilizados os comandos cd, ls, get e put. o cliente e servidor foram executados em diretórios diferentes para houver a transferência efetiva de arquivo com get e put. Enquanto testado, foi capturado as mensagens passados pelo programa Wireshark, a seguir está a imagem da captura as mensagens mostrados no Wire-shark. Os comandos usados no teste foram respectivamente ls, cd, ls, get e put.



No.	Time	Source	Destination	Protocol	Length	Info
0	0.001130295	127.0.0.1	127.0.0.1	FTP	80	Response: 228 Welcome.
14	2.981755083	127.0.0.1	127.0.0.1	FTP	77	Request: USER void
16	2.982168351	127.0.0.1	127.0.0.1	FTP	75	Response: 331 OK.
24	5.584798520	127.0.0.1	127.0.0.1	FTP	77	Request: PASS test
25	5.584997342	127.0.0.1	127.0.0.1	FTP	75	Response: 230 OK.
27	5.585083888	127.0.0.1	127.0.0.1	FTP	72	Request: SYST
28	5.585248619	127.0.0.1	127.0.0.1	FTP	88	Response: 500 Not Implemented.
34	42.677328246	127.0.0.1	127.0.0.1	FTP	90	Request: PORT 127,0,0,1,156,207
35	42.677612642	127.0.0.1	127.0.0.1	FTP	86	Response: 200 Port Received.
37	42.677824229	127.0.0.1	127.0.0.1	FTP	72	Request: LIST
38	42.678074988	127.0.0.1	127.0.0.1	FTP	92	Response: 150 Data Socket Opening.
80	42.718443913	127.0.0.1	127.0.0.1	FTP	87	Response: 226 Directory Sent.
96	104.852688853	127.0.0.1	127.0.0.1	FTP	79	Request: CWD Network
97	104.853062553	127.0.0.1	127.0.0.1	FTP	75	Response: 250 OK.
101	111.448342859	127.0.0.1	127.0.0.1	FTP	89	Request: PORT 127,0,0,1,140,245
102	111.448525252	127.0.0.1	127.0.0.1	FTP	86	Response: 200 Port Received.
104	111.448767111	127.0.0.1	127.0.0.1	FTP	72	Request: LIST
105	111.441062328	127.0.0.1	127.0.0.1	FTP	92	Response: 150 Data Socket Opening.
143	111.482423930	127.0.0.1	127.0.0.1	FTP	87	Response: 226 Directory Sent.
145	118.880311560	127.0.0.1	127.0.0.1	FTP	90	Request: PORT 127,0,0,1,140,245
146	118.880512944	127.0.0.1	127.0.0.1	FTP	86	Response: 200 Port Received.
148	118.880651757	127.0.0.1	127.0.0.1	FTP	81	Request: RETR test.txt
149	118.880851667	127.0.0.1	127.0.0.1	FTP	92	Response: 150 Data Socket Opening.
150	118.882455003	127.0.0.1	127.0.0.1	FTP	90	Response: 226 Transfer Complete.
163	145.309155109	127.0.0.1	127.0.0.1	FTP	90	Request: PORT 127,0,0,1,120,211
174	145.309131522	127.0.0.1	127.0.0.1	FTP	86	Response: 200 Port Received.
176	145.309278340	127.0.0.1	127.0.0.1	FTP	87	Request: STOR PrtGS-17-2.txt
177	145.309785796	127.0.0.1	127.0.0.1	FTP	92	Response: 150 Data Socket Opening.
209	145.356437192	127.0.0.1	127.0.0.1	FTP	90	Response: 226 Transfer Complete.
211	149.027740830	127.0.0.1	127.0.0.1	FTP	72	Request: QUIT
212	149.027991583	127.0.0.1	127.0.0.1	FTP	80	Response: 221 Goodbye.

Frame 150: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0
Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 21, Dst Port: 37798, Seq: 244, Ack: 140, Len: 24
File Transfer Protocol (FTP)

0000 00 00 00 00 00 00 00 00 00 00 00 00 00 45 00E
0010 00 4c bf 64 40 00 40 06 7d 45 7f 00 00 01 7f 00L.d@.jE.....

Packets: 217 · Displayed: 31 (14.3%) Profile: Default

Figure 5: Trace do Wireshark

5 Conclusão

A partir do trace do Wireshark mostrado anteriormente, é possível concluir que o servidor FTP foi implementado com sucesso, respondendo de acordo com a documentação FTP, para outros clientes FTP, para os comandos implementados, cd, ls, get e put. As mensagens passadas no teste, mostrados no trace estão de acordo com o diagrama de mensagens, e na mesma ordem, além de estarem de acordo com a documentação do FTP, o que confirma que o servidor foi implementado de forma correta e pronto para o uso com clientes FTP.