Basic Functioning of server and client

Server

Creating a TCP socket → Binding the socket to “” host and a port → Listen for any incoming clients → Accept the incoming client connections → Keep the connection persistent → Follow the client’s commands until connection is closed by client → If connection is closed again wait for the incoming connections

Binding to “” host so that socket is reachable by any address the machine happens to have.

Client

Creating a TCP socket → Connecting to the server → Send commands to server which are taken as input through console → Close connection on “QUIT” command

Mode of Layering

Layering is done as writing the code in such a flow that first network layer is instantiated(TCP/IP Socket), then while sending data just call function MyEncrypt when data is to be sent and for receiving call MyDecrypt to get decrypted data all other function between sending and receiving are implemented using the OS apis.

Protocol

Network Layer

Protocol used is TCP

Crypto Layer

A general protocol is followed for all data sent whether it is client side or the server side. A header is added for indicating encryption type to every set of bytes sent. This is the first header is at the front of bytes always apart from any second or third header a set of bytes may have.
“ca-” header indicates that data is caesar ciphered
“tr-” header indicates that data is transposed that is content is with each word are reversed
“pl-” header indicates that data is simply plain text there is no encryption

Top Most Layer

I also have followed a protocol at each command level

For “CWD” command

Client end
The input from the console that is “CWD” is directly sent.
Client request data is of format “`<encryption header>CWD`”

Server end

On request of client when the server sends the current working directory data it adds an additional header of “cwdr ” so that when the client receives the response from server it knows that response corresponds to CWD command it sent and it appropriately prints the received cwd data on console.
Server response is of format “<encryption header>cwdr <current working directory in server>”

For “LS” command

Client End
The input from the console that is “LS” is directly sent.
Client request data is of format “<encryption header>LS”

Server End
Same thing as the “CWD” command. But this time the header of the server response is “lsr ”. And the main data that is the list of files/ folder is sent as the name of file/folder separated by space.
Server response is of format “<encryption header>lsr <file/folder1> <file/folder2> …”

For “CD <dir>” command

Client End
The input from the console that is “CD <dir>” is directly sent
Client request is of format “<encryption header>CD <dir>”

Server End
Again similar protocol followed as for “CWD” command. But this time the header of the server response is “cdr ”. And depending on whether the request is successfully executed the main response data is status ok/nok
Server response is of format “<encryption header>cdr STATUS: <OK/NOK>”

For “DWD <file>” command

Client End
The input from the console that is “DWD <file>” is directly sent.
Client request is of format “<encryption header>DWD <file>”

Server End
Here I had to send a file so again the first thing is “dwdr ” header so that the server can recognise it is in response to a file download request and get ready to receive the file. The main data is sent and after a break of 1 second a footer “dwdrend” is sent. This footer is sent so that the client can stop expecting any more file data. Then again after one second a status is sent for whether the file is correctly sent or not with a “fdwd ” header.

Server response is of format
"<encryption header>dwdr"
"<encryption header><filedata>"
"<encryption header>dwdrend"
"<encryption header>fdwd STATUS: <OK/NOK>"

For "UPD <file>" command

Client End
The input from the console that is "UPD <file>" is directly sent. Then the main data that is file data is sent. This is followed by a wait of 1 second and then a footer is sent so that the server knows the file has been sent completely by the client.

Client request is of format
"<encryption header>UPD file"
"<encryption header><filedata>"
"<encryption header>updrend"

Server End
Depending on whether the server receives the file properly or not. The server responds with main data that is status ok/nok which is preceded by a header "fupd"

Server Response is of format
"<encryption header>fupd STATUS: <OK/NOK>"

Challenges
A common challenge in the protocol followed for all commands is that if the UDP socket is used it is possible(although maybe rare) where data may not be sent in order. The protocol defined by me could fail. Although it identifies that response is for what type of command but does not do one on one matching with client request and server response

Proof Of Execution of commands

Both client and server are running on the local server
Server Directory State

```
┌──(kali㉿kali)-[~/Downloads/newproj/server]
└─$ ls
server.py  server_test_file.txt
```

Client Directory State

```
┌──(kali㉿kali)-[~/Downloads/newproj/client]
└─$ ls
client.py  client_test_file.txt
```

Server Started

```
┌──(kali㉿kali)-[~/Downloads/newproj/server]
└─$ python server.py
Socket Created at server end
Listening At Port : 49154
```

Client Started and connected

```
┌──(kali㉿kali)-[~/Downloads/newproj/client]
└─$ python client.py
Socket Created at client end
Connnected to Server IP= 127.0.0.1
```

Server Connected with client

```
┌──(kali㉿kali)-[~/Downloads/newproj/server]
└─$ python server.py
Socket Created at server end
Listening At Port : 49154
Connection established with client IP= 127.0.0.1 | Port: 49154
```

“CWD” command

```
┌──(kali㉿kali)-[~/Downloads/newproj/client]
└─$ python client.py
Socket Created at client end
Connnected to Server IP= 127.0.0.1
CWD
/home/kali/Downloads/newproj/server
```

“LS” command

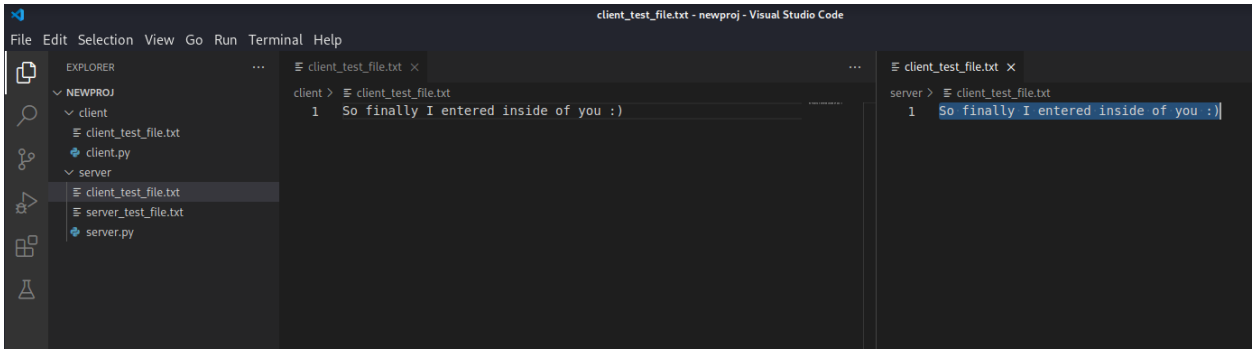
```
┌──(kali㉿kali)-[~/Downloads/newproj/client]
└─$ python client.py
Socket Created at client end
Connnected to Server IP= 127.0.0.1
CWD
/home/kali/Downloads/newproj/server
LS
server_test_file.txt server.py
```

“CD <dir>” command

```
┌──(kali㉿kali)-[~/Downloads/newproj/client]
└─$ python client.py
Socket Created at client end
Connnected to Server IP= 127.0.0.1
CWD
/home/kali/Downloads/newproj/server
LS
server_test_file.txt server.py
CD ..
STATUS: OK
CWD
/home/kali/Downloads/newproj
```

“UPD <file>” command

```
┌──(kali㉿kali)-[~/Downloads/newproj/client]
└─$ python client.py
Socket Created at client end
Connnected to Server IP= 127.0.0.1
CWD
/home/kali/Downloads/newproj/server
LS
server_test_file.txt server.py
CD ..
STATUS: OK
CD server
STATUS: OK
UPD client_test_file.txt
STATUS:OK
```



“DWD <file>” command

```
LS
client_test_file.txt server_test_file.txt server.py
DWD server_test_file.txt
STATUS: OK
```