# Design Doc

Medhansh Singh

Code is available on GitHub: https://github.com/medhanshgit/ComputerNetworks_CS433

1. FILE SERVICE: The layer is responsible for executing the RFS commands and uses OS API. The os library helps us in doing so.
   The challenge faced was learning the commands to execute the required task.
   The functions were implemented as follows:
   a. CWD: returns the current working directory using the **os.getcwd()** function. CWD command is sent to the server, and the server sends the path to the client.

```
##CURRENT DIR-------------------------------------
    if req == "CWD":
        dir = s.recv(1024).decode()
        print(dir)
```

CWD (client)

```
##CURRENT DIR-------------------------------------
    if req == "CWD":
        dir = os.getcwd()
        client_socket.send(dir.encode())
```

CWD (server)

   b. LS: lists all the files in the current directory using the **os.listdir()** function. It returns a list.

```
##LIST DIR----------------------------------
    if req == "LS":
        dir = s.recv(1024).decode()
        print(dir)
```

LS (client)

```
##LIST DIR-----------------------------------------
    if req == "LS":
        path = os.getcwd()
        dir = os.listdir(path)
        client_socket.send(format(dir).encode())
```

LS (server)

c. CD: Changes the current directory using **os.chdir()** function. The current path is sent to the server by the client, which then uses os.chdir to change the directory. The path is passed as an argument with CD with space as a delimiter.

```
##CHANGE DIR-------------------------------------
    if req == "CD":
        path = s.recv(1024).decode()
        print("Initial directory:",path)
        s.send(file.encode())
        path = s.recv(1024).decode()
        print("Current directory:",path)
```

CD (client)

```
##CHANGE DIR-----------------------------------------
    if req == "CD":
        path = os.getcwd()
        client_socket.send(format(path).encode())
        dir = client_socket.recv(1024).decode()
        os.chdir(format(dir))
        path = os.getcwd()
        client_socket.send(format(path).encode())
```

CD (server)

d. UPD: The filename to be uploaded to the server is sent as an argument with UPD with space as a delimiter. When this command is executed, it first asks for the type of encryption that needs to be implemented. Then the file name, as well as the type of encryption, is sent to the server. Then the file is encrypted, and the data in the file is sent to the server using a loop which runs until all the data in the file is sent to the server.
Similarly, on the server side, it first receives the name of the file to be uploaded to the server along with the type of encryption used to encrypt the file. Then a copy of the encrypted file is made on the server using a loop which runs until all the data sent by the client is written to the file on the server, and then the file on the server is decrypted.

```
##UPLOAD-----------------------------------------------------------------------------
    if req == "UPD":
        filename = file
        e = input("which encryption out of (plaintext,substitution,transpose):")
        s.send(filename.encode())
        cip.encrypt(e,filename)
        s.send(e.encode())
        # start sending the file
        with open(filename, "rb") as f:
            while True:
                # read the bytes from the file
                bytes_read = f.read(BUFFER_SIZE)
                if not bytes_read:
                    # file transmitting is done
                    print("Sent")
                    s.shutdown(socket.SHUT_WR)
                    break
                s.send(bytes_read)
                print("Sending")
        cip.decrypt(e,filename)
        break
```

UPD (client)

```
##UPLOAD-----------------------------------------------------------------
    if req == "UPD":
        filename = client_socket.recv(1024).decode()
        e = client_socket.recv(1024).decode()
        # remove absolute path if there is
        filename = os.path.basename(filename)
        # start receiving the file from the socket
        # and writing to the file
        with open(filename, "wb") as f:
            while True:
                # read bytes from the socket (receive)
                bytes_read = client_socket.recv(BUFFER_SIZE)
                if not bytes_read:
                    # nothing is received
                    # file transmitting is done
                    print("Received")
                    break
                print("Receiving")
                # write to the file the bytes we just received
                f.write(bytes_read)
        cip.decrypt(e,filename)
        break
```

UPD (server)

e. DWD: The filename to be downloaded to the client is sent as the argument along with the DWD command with space as a delimiter. When this command is executed, it first asks for the type of encryption that needs to be implemented. Then the file name, as well as the type of encryption, is sent to the server. Then the server receives the filename and the type of encryption to be implemented and encrypts the file on the server to be sent to the client. Then the file is sent in the encrypted form the client using a loop which runs until all the data in the file is sent to the client.
Then on the client side, it makes a copy of the file from the server and writes it with encrypted data sent by the server. Then the file is decrypted.

```
##DOWNLOAD-------------------------------------------------------------------------------
    if req == "DWD":
        filename = file
        e = input("which encryption out of (plaintext,substitution,transpose):")
        s.send(filename.encode())
        s.send(e.encode())
        # start writing to the file
        with open(filename, "wb") as f:
            while True:
                # read bytes from the socket (receive)
                bytes_read = s.recv(BUFFER_SIZE)
                if not bytes_read:
                    # nothing is received
                    # file transmitting is done
                    print("Received")
                    break
                # write to the file the bytes we just received
                print("Receiving")
                f.write(bytes_read)
        cip.decrypt(filename, e)
        break
```

DWD (client)

```
##DOWNLOAD-------------------------------------------------------------
    if req == "DWD":
        filename = client_socket.recv(1024).decode()
        e = client_socket.recv(1024).decode()
        # remove absolute path if there is
        filename = os.path.basename(filename)
         # start sending the file
        cip.encrypt(e,filename)
        with open(filename, "rb") as f:
            while True:
                # read the bytes from the file
                bytes_read = f.read(BUFFER_SIZE)
                if not bytes_read:
                    # file transmitting is done
                    print("Sent")
                    client_socket.shutdown(socket.SHUT_WR)
                    break
                client_socket.send(bytes_read)
                print("Sending")
        cip.decrypt(e,filename)
        break
```

DWD (server)

2. Crypto Layer: This layer is responsible for encrypting the data that we sent over the socket. The way it is implemented is that the functions for encryption and decryption are executed in a different file called **cip.py** and the functions inside them encrypt and decrypt the data that is passed to them.
   **encrypt** and **decrypt** take as an argument the type and the name of the file they will encrypt/decrypt.
   a. Plaintext: It does nothing and simply returns the file as it is.
   b. Substitution: To implement this method of encryption, I used Caesar cipher module from python library. Each line of the file to be encrypted is passed through the Caesar cipher function and it encrypts it and then the lines are stored in a list which is then used to overwrite the file with the encrypted text.
   c. Transpose: In this method, every line is reversed in the given file. To do so each line is traversed in the file to be encrypted and reversed and then overwritten with the help of a list which acts as an auxiliary space for overwriting.

```python
from caesarcipher import CaesarCipher
def encrypt(type,filename):
    if type == "plaintext":
        return
    if type == "substitution":
        l = []
        with open(filename) as f:
            while True:
                line = f.readline()
                if not line:
                    break
                # print(line.strip())
                cipher = CaesarCipher(line,offset=2)
                l.append(cipher.encoded)
            # print(l)
        with open(filename,'w') as f:
            f.writelines(l)
        return
    if type == "transpose":
        l = []
        with open(filename) as f:
            while True:
                line = f.readline()
                if not line:
                    break
                # print(line.strip())
                line = line[::-1]
                l.append(line)
            # print(l)
        with open(filename,'w') as f:
            f.writelines(l)
        return
```

Encryption code

```python
def decrypt(type,filename):
    if type == "plaintext":
        return
    if type == "substitution":
        l = []
        with open(filename) as f:
            while True:
                line = f.readline()
                if not line:
                    break
                # print(line.strip())
                cipher = CaesarCipher(line,offset=2)
                l.append(cipher.decoded)
            # print(l)
        with open(filename,'w') as f:
            f.writelines(l)
        return
    if type == "transpose":
        l = []
        with open(filename) as f:
            while True:
                line = f.readline()
                if not line:
                    break
                # print(line.strip())
                line = line[::-1]
                l.append(line)
            # print(l)
        with open(filename,'w') as f:
            f.writelines(l)
        return
```

Decryption code

3. Network layer: This layer is implemented using sockets in python. We use IPv4 addresses and TCP protocol for sending data using sockets.

Challenge faced:

Writing function for encryption and decryption of text files is easy. The problem arises when images are sent over using this method of encryption as this implementation doesn't cover encryption of files of image format, as these formats do not contain data in the form of strings.

Another challenge I faced was while uploading and downloading, the respective function for their task was not terminating until a keyboard interrupt would not cause the socket to close.

The solution for the same was discussed by Prof. Sameer Kulkarni in which he insisted that we must check the length of the data that is being sent over the connection and break when its length is smaller than the defined buffer size, we should stop sending/writing data after writing the last received data. However, I pursued a non-persistent connection in my implementation while uploading and downloading.

Error and exceptions will cause the socket to close, and it would require a restart for completing tasks.

Improvements:

1. Error and exception handling can be added.
2. Persistent connection can be implemented for upload and download as well.
3. Encryption/decryption can be implemented for other file formats as well.
4. Other Encryption techniques may be added.

Screenshots of execution:

```
PS C:\Users\DC\Desktop\programming\projects\Python\Computer Networks> python client.py
[+] Connecting to 127.0.0.1:12345
[+] Connected.
enter command:CWD
C:\Users\DC\Desktop\programming\projects\Python\Computer Networks\server
enter command:
```

CWD command

```
enter command:LS
['cip.py', 'dataserver.csv', 'server.py', '__pycache__']
enter command:
```

LS command

```
PS C:\Users\DC\Desktop\programming\projects\Python\Computer Networks> python client.py
[+] Connecting to 127.0.0.1:12345
[+] Connected.
enter command:CD ../
Initial directory: C:\Users\DC\Desktop\programming\projects\Python\Computer Networks\server
Current directory: C:\Users\DC\Desktop\programming\projects\Python\Computer Networks
enter command:
```

CD command

```
PS C:\Users\DC\Desktop\programming\projects\Python\Computer Networks> python client.py
[+] Connecting to 127.0.0.1:12345
[+] Connected.
enter command:DWD dataserver.csv
which encryption out of (plaintext,substitution,transpose):plaintext
Receiving
Received
PS C:\Users\DC\Desktop\programming\projects\Python\Computer Networks>
```

DWD dataserver.csv (plaintext)

```
PS C:\Users\DC\Desktop\programming\projects\Python\Computer Networks> python client.py
[+] Connecting to 127.0.0.1:12345
[+] Connected.
enter command:DWD dataserver.csv
which encryption out of (plaintext,substitution,transpose):substitution
Receiving
Received
```

DWD dataserver.csv (substitution)

```
PS C:\Users\DC\Desktop\programming\projects\Python\Computer Networks> python client.py
[+] Connecting to 127.0.0.1:12345
[+] Connected.
enter command:DWD dataserver.csv
which encryption out of (plaintext,substitution,transpose):transpose
Receiving
Received
```

DWD dataserver.csv (transpose)

```
PS C:\Users\DC\Desktop\programming\projects\Python\Computer Networks> python client.py
[+] Connecting to 127.0.0.1:12345
[+] Connected.
enter command:UPD dataclient.csv
which encryption out of (plaintext,substitution,transpose):plaintext
Sending
Sent
```

UPD dataclient.csv (plaintext)

```
PS C:\Users\DC\Desktop\programming\projects\Python\Computer Networks> python client.py
[+] Connecting to 127.0.0.1:12345
[+] Connected.
enter command:UPD dataclient.csv
which encryption out of (plaintext,substitution,transpose):substitution
Sending
Sent
```

UPD dataclient.csv (substitution)

```
PS C:\Users\DC\Desktop\programming\projects\Python\Computer Networks> python client.py
[+] Connecting to 127.0.0.1:12345
[+] Connected.
enter command:UPD dataclient.csv
which encryption out of (plaintext,substitution,transpose):transpose
Sending
Sent
```

UPD dataclient.csv (transpose)

References:


1. https://www.geeksforgeeks.org/socket-programming-python/
2. OS API: https://linuxize.com/post/python-get-change-current-working-directory/#:~:text=To%20find%20the%20current%20working,chdir(path)%20.
3. https://www.thepythoncode.com/article/send-receive-files-using-sockets-python
4. https://realpython.com/python-sockets/#echo-client-and-server
5. https://pypi.org/project/caesarcipher/
6. Textbook: Learning Python Network Programming