Riley Deal
CSCI 53700 Distributed Computing
Assignment 2

Overview

My implementation of assignment 2 consists of a server which sits listening on a socket for clients to connect. Once a client connects it passes that connection to a thread. This allows multiple clients to connect at any given time. This thread will sit and listen for the client to issue requests such as transfer, register, etc. and complete them to the best of its ability. If there are issues it will notify the client.

On the client side, when a client is made it attempts to connect to the server IP/Port to see if a connection can be established. Once it has been created the user can issue commands to the server, where the client will wait for a response between issued commands. Any errors which occur on the backend which are useful to the client will be sent over (file not found, etc.) for the client to deal with.

End-to-end Analysis

Due to Java sockets using TCP/IP by default, a good deal of the end-to-end argument is present in my design. Since TCP provides no guarantees internally for retransmission in the case of message drops, corrupted messages, etc. the endpoints will need to handle this. The standard way of cleanly handling this is to implement ACK/NACK protocols, but for the sake of this assignment I left any retry logic to the endpoints. This means that all errors are handled on the endpoints either by the server hiding the error if it is not directly impactful to the client, or by sending the error to the client for them to figure out. One example of this is to tell the user the

file was corrupted during transmission, and not saving it. If they wish to retry the transmission they can do so.

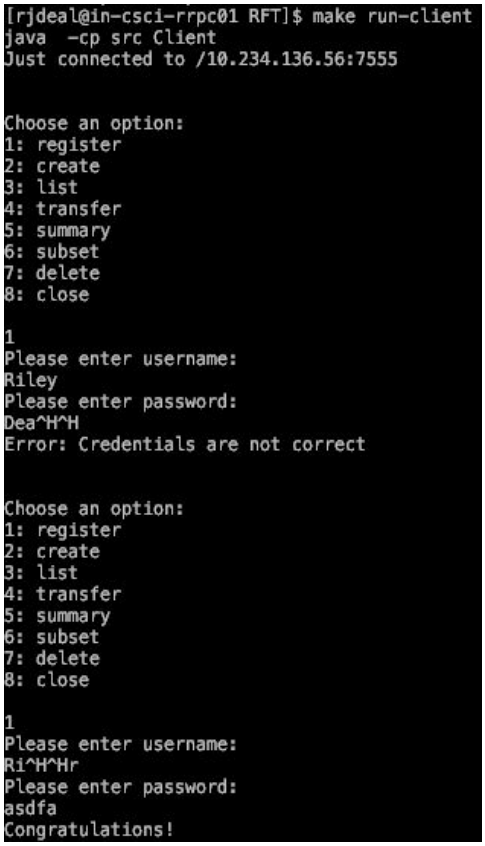
Another design decision that was made in relation to the end-to-end argument is that the endpoints handle all encryption/decryption of data. This means that basically anything could be happening to the data between the endpoints, all we know is when it comes out the other side its in an acceptable format. This also keeps our encryption secure as there are less potential places for someone looking to break the encryption to attack, as they would need to access the endpoints/

Methods

Register

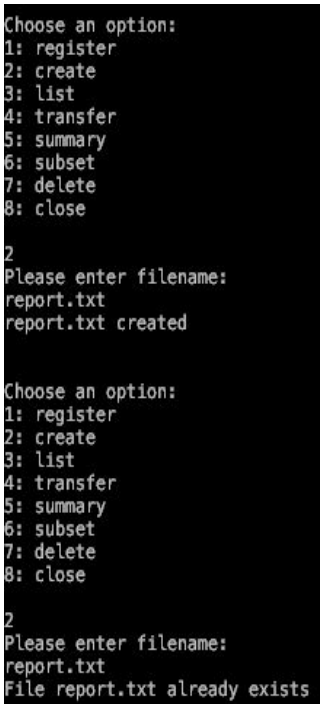
The register method accepts credentials from the user, encrypts them, and passes them to the server.

If the credentials are correct, the user is given a congratulations message. If the credentials are not correct, which is currently only the name “Riley” then the user is told their credentials are not correct.



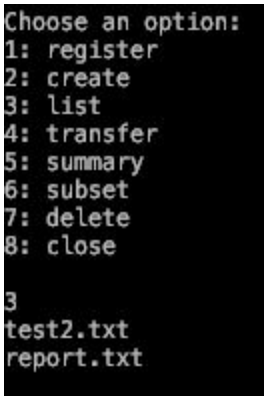
Create

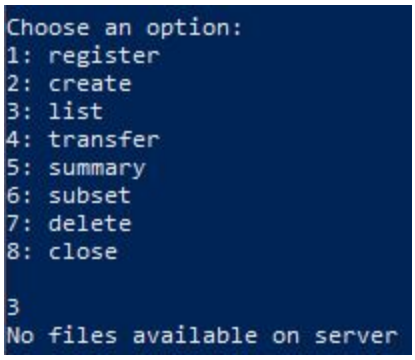
The create method takes a file name from the user. If the file does not exist, it is created populated with 100 random characters from lowercase, uppercase, and space. The user is then notified their file is created. If the file already exists, the user is notified of this.



List

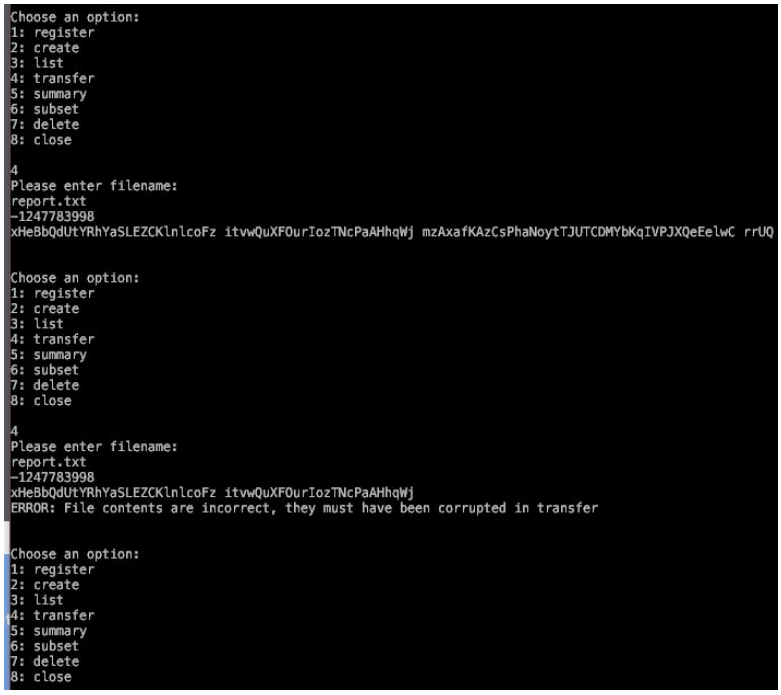
The list method asks the server for a list of all the files on it. If files are available they are sent as plain text. If there are no files, the client is sent an error message.

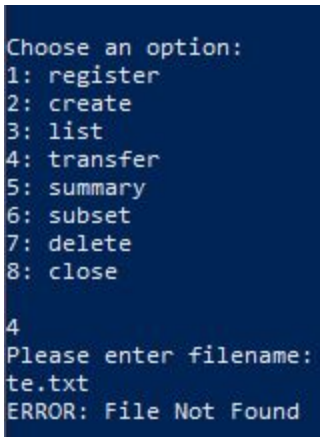




Transfer

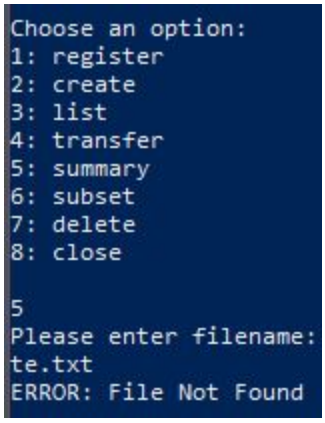
The transfer method takes a file name from the user and sends it as a request for transfer from the server. If the file exists on the server, it will be encrypted and sent back to the user. It is possible for byzantine behavior to happen here (1/10 chance by default) which will only send a subset of the file. In order to check for this, the server also sends a hash of the string to be used as a checksum with what the client receives. If the checksums do not match, then corruption happened during transmission. If the file does not exist on the server in the first place, the client will be notified with an error message.





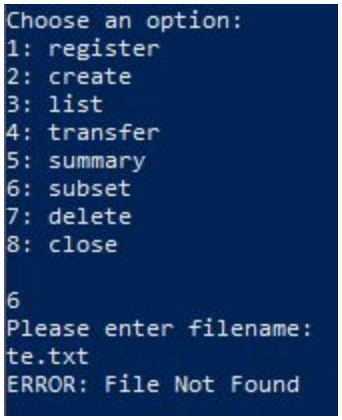
Summary

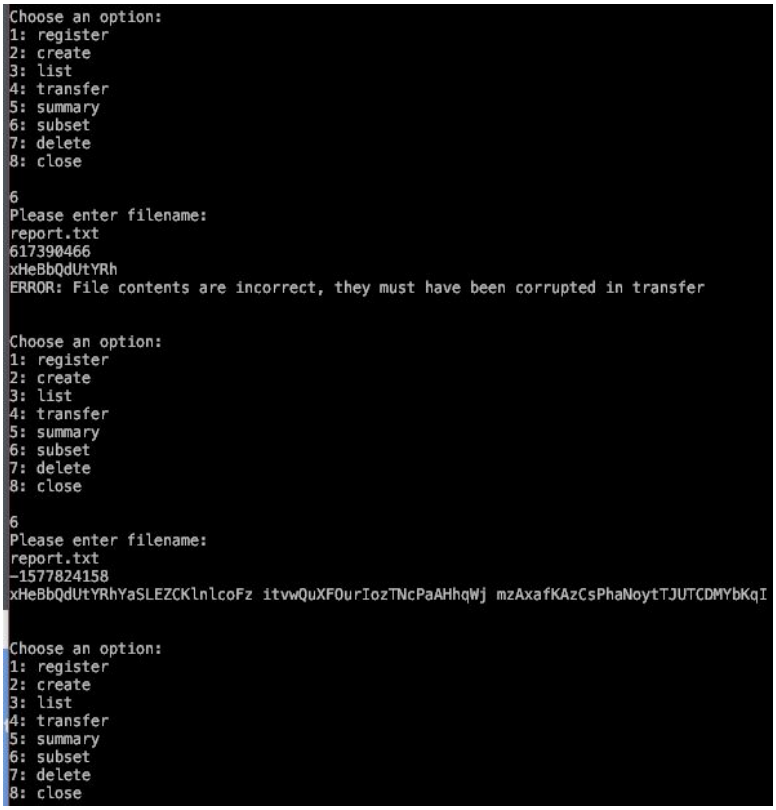
The summary method takes a file name from the user and sends it to the server. If the file exists, the server will send back the name, number of lines, and number of words in the file. Number of lines is done via a count of the \n character. Number of words is done via a count of whitespace between other characters. If the file does not exist then an error message is sent to the user.



Subset

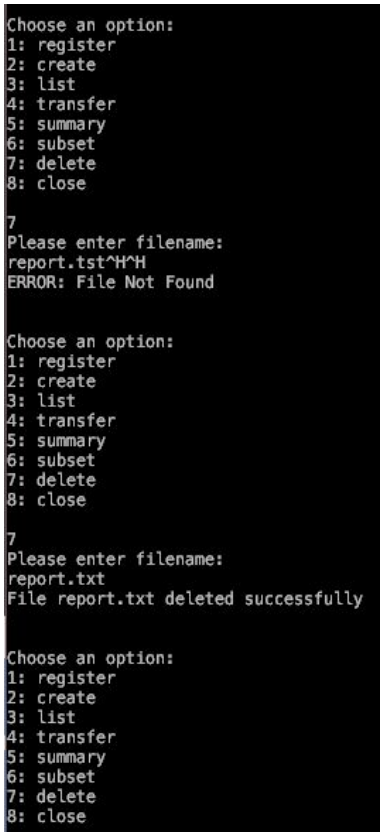
The subset method takes a file name from the user and sends it to the server. A random chunk from the start to the end of the file will be selected, encrypted, and sent back to the user. It is possible for byzantine behavior to happen here (1/10 chance by default) which will only send a subset of the subset. In order to check for this, the server also sends a hash of the string to be used as a checksum with what the client receives. If the checksums do not match, then corruption happened during transmission. If the file does not exist on the server in the first place, the client will be notified with an error message.





Delete

The delete method takes a file name from the user and sends it to the server. If the file exists it is deleted and the user is notified. If there is an issue deleting the file or the file does not exist, an error message is sent to the user.



Close

The close method sends a request to close to the server. The server sends a goodbye and then closes the connection. The client then ends its connection as well.