Sujana Oruganti

30 October 2018

CSI 53700

Assignment 2 Report

(Interprocess Communication & End-to-End Argument)

1. **Introduction**

A simplified version of the File Transfer Program (FTP) was developed in the program based on the end-to-end argument. The program was implemented as per the requirements using the stream sockets available in java.net package. The implementation has the Client and the Server objects to achieve the file transfer protocol. The report explains the implementation details and the corresponding end-to-end arguments in the following sections.

1. **Implementation Details**

This section explains the implementation details of the project. Now, as mentioned above, the File Transfer Protocol runs primarily on the two objects – Client and Server. Apart from these, the implementation has the ClientDB, ChecksumMethod and the AES helper classes. Before explaining the functioning of the Client and Server objects, the helper classes will be discussed for a better understanding of the whole implementation.

Firstly, the **ClientDB**, used by the Server object, maintains the login credentials of the users or the clients. The ClientDB class utilizes the functionalities of a HashMap which is initialized with username and password pairs by the values provided in the csv file. This initialized HashMap can be used by the server to authenticate the returning clients with accounts. Additionally, the ClientDB is notified by the Server whenever a new user registers to the server with their login credentials. The ClientDB then, writes/adds the given credentials of the new user to the csv file for future authentication purposes.

The other two helper classes are the ChecksumMethod and the AES. Commonly used by the Client and the Server object, the **ChecksumMethod** calculates the checksum of a given string. It generates a checksum value for the Server when sending contents to the clients. And for the clients, the ChecksumMethod evaluates the received checksum to see if they received the contents correctly without any errors.

Lastly, **AES** is another helper class which is also used by both the Client and the Server object. The AES class has the methods encrypt and decrypt used by the Server and the Client respectively. The encrypt method encrypts a given string using the AES algorithm in the CBC mode with the PKCS5Padding. On the other hand, the decrypt method returns the decrypted string given an encryption with the Key. Without the key, neither the encryption nor the decryption can happen.

The **Server** object is a multi-threaded, executable class that connects with clients using a simple file transfer protocol. The Server object is made multi-threaded to accept multiple concurrent requests from various clients.

Initially, the Server will be waiting for a client on a given port and once the client connects, it gets the login credentials – username and password – from the clients. Using these credentials, the Server either adds the new client to the ClientDB or authenticates the returning client. For the returning clients, the Server authenticates their credentials using the HashMap from the ClientDB class. In the case of invalid credentials, the Server notifies the client that invalid credentials have been provided and closes the connection.

If the given credentials are valid, the Server accepts a filename to be transferred. If the requested file does not exist, then the server will indicate an error message and close the socket connection. If the requested file exists, then it will be sent across to the client along with the checksum generated from the ChecksumMethod class. The contents of the file will be encrypted using the encrypt method from the AES class using a key (key is known both to the client and the server), before the transmission. The Server prints on it’s own console and to the client that the “File has been sent…,” once the file transfer is successfully done without any mishaps.

As the assignment requires the Server to show a **Byzantine behavior**, it was implemented using a random number generator between 1 to 100. If the random number generator produces a number greater than 80, then the Byzantine behavior will be implemented. The Byzantine event in this ftp program was to append a string to the encrypted file contents before sending it to the client, so that the checksum calculated on the client side will be erroneous.

Besides the Server, the **Client** is another important class in the ftp implementation. The Client object is responsible for getting the login credentials from the user and if the client is a returning user or a new user. The client then sends this information to the Server which performs corresponding actions based on the Client’s input. Based on the Server’s response on the login credentials, the Client initiates the file transfer protocol. If the Client sends invalid credentials, the Server informs the client about this and closes the connection. On the other hand, if valid credentials were provided, then the client will ask the user for a filename to be transferred from the Server and sends the corresponding request to the Server.

The Server will transfer the file, if it exists. The Server will send an encrypted version of the file contents along with the checksum of the encryption. Now the client evaluates the checksum to see if the file was transferred without any errors. If there are no errors, then the received file contents will be decrypted by calling the decrypt method of the AES class which uses the shared key for decryption. If the checksum does not return the expected value, the client then automatically retries for the file transfer. If the file transfer fails even after 5 retries, then the socket connection will be closed.

In the appendix are the screenshots of the results of certain conditions from above implementations.

1. **Discussion of the End-to-End Arguments**

In the paper “End-to-End Arguments in System Design, ” the authors provide a new analysis of function placement and its associated design principles. “The problem consists of creating a design principle that would generate system designs that would increase performance by placing functionality at the most appropriate level. Briefly, the authors argue that end-to-end functionality, that is, putting the functionality at the application level, is most often the best approach in system design” [1]. After understanding the implementation details of the ftp program using the java stream sockets, below are the design choices made based on the End‑to‑End arguments and their pros and cons.

Firstly, considering the file transfer, the errors can come up in any level. The error checking at the ends of the application(Client and the Server) is done by computing a checksum at each end and comparing. Now, a checksum is a simple type of redundancy check that is used to detect errors in data. During the process of a file transfer, errors can occur anywhere – while writing/reading to the disk, over the network during transmission, etc. Even though the errors are very small, like a single incorrect bit, the damage can greatly affect the quality of the data. For instance, during the occurrence of byzantine events in the implementation, a string is added to the encrypted file contents. Fortunately, these changes/errors are detected by the checksum algorithm which would make the transmitted data useless otherwise.

In the implementation, the checksum is calculated by the server object and sent to the client object. When the data is retrieved by client object, a new checksum is calculated and compared against the received checksum. If these two do not match, then it is implied that the data received is erroneous and if the checksum matches, it implies the absence of errors, but only that the algorithm was not able to detect any. For instance, the checksum implementation cannot detect reordering of the bytes, inserting or deleting zero-valued bytes and multiple errors that cancel each other out. However, these errors can also be detected using advanced techniques such as the Cyclic Redundancy Checks that require greater system resources.

Another argument considered during the implementation was the secure file transfer. As the data will be in the clear and thus vulnerable as it passes through the network to the client from server, the file transfer system performs the end-to-end encryption. The file transfer system is made responsible to ensure the security of the file transmission via encryption and decryption. The client and the server share the keys to encrypt and decrypt the file contents. This avoids the malicious people from accessing the data and the authentication of the message will be protected during and after the transmission. However, a flaw in the system is that a user with a malicious intention can register as a user and access the files in the Server end. Another mishap that could happen is that the Server might read the data incorrectly before encrypting it to send to the client. The implementation does not perform any integrity check to see if the Server reads the data accurately from the local file system.

Delivery Guarantees of Messages is another argument considered during the implementation of the ftp system. During most cases, the messages or the input/output are assumed to be delivered. Besides, as the paper mentions, the target host acting upon the received message is more important than knowing whether the message was delivered to the target host. Now, when the file contents are delivered and the checksum mismatches, the client retries 5 times before displaying the error message and closing the connection. However, there are no explicit implementations to guarantee the delivery of the messages and the input/output values between the two ends.

FIFO message delivery is expected to be implemented by default as the input reader on one end is blocked until there is an output from the other end. But in the cases of out of order arrival, no implementations were made, and the system cannot detect out of order messages in a distributed implementation file transfer protocol.

Other arguments such as the Transaction management and performance of the end-to-end implementation were not considered in the program. The performance aspect couldn’t be considered as all the implementation was done on the ends of the system.

Furthermore, having defined endpoints – client and server – made the implementation of the ftp program easier rather than having to identify these end points and then performing integrity checks. Additionally, the cost of the ends performing the error checking is not very high due to the simplicity of the program. Again, it wouldn’t have been the case if the endpoints were not very clear. Another reason having the integrity checks at the endpoints is a better approach for this program is that it is easier to know about the reliability of its local system (either the client or the server), but more difficult for it know about the reliability of the intermediaries that its data must pass through as it crosses a network.

In conclusion, this implementation aids in achieving a deeper understanding of the Interprocess communication and end-to-end argument.

1. **References**

|  |  |
| --- | --- |
| [1] | "Paper Review: End to End Arguments in System Design (SRC84)," [Online]. Available: http://zoo.cs.yale.edu/classes/cs633/Reviews/src84.mm446.html. |

1. **Appendix**

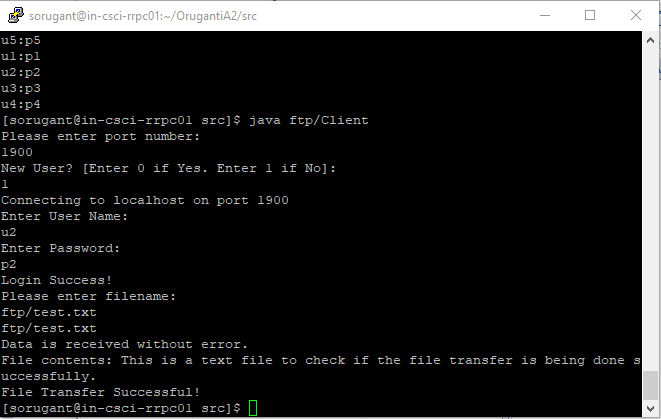


Figure Client Side Implementation Output

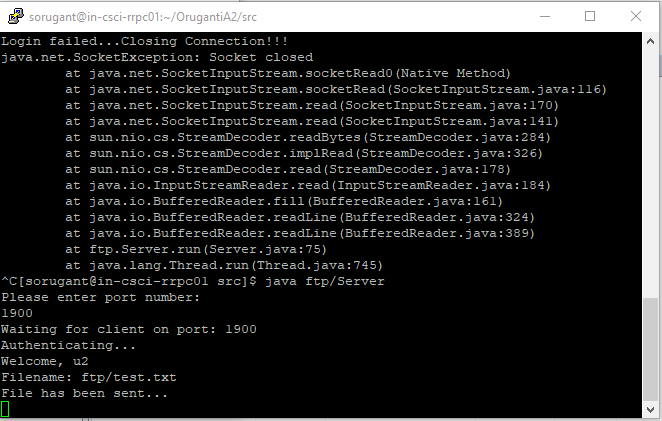


Figure Server-Side Implementation Output

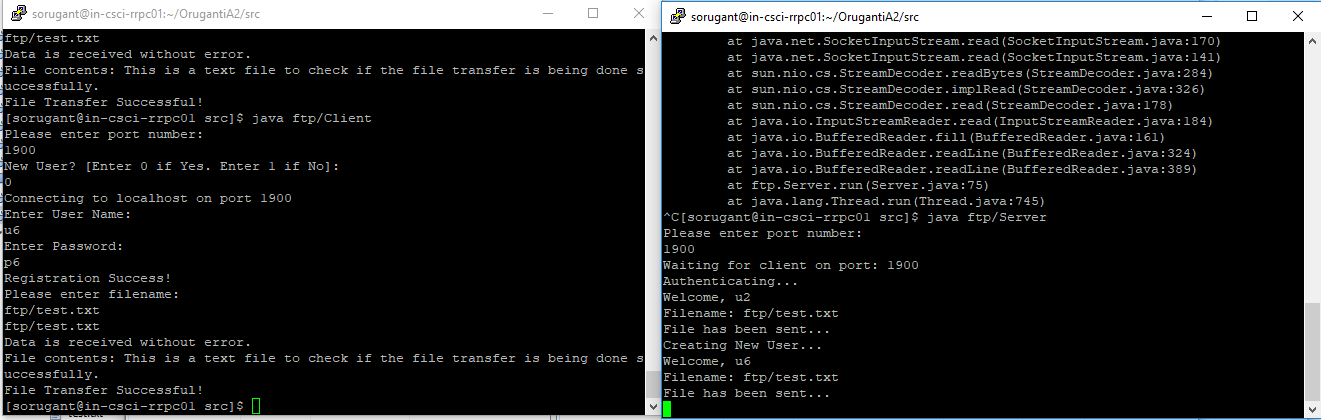


Figure Client Registration Output

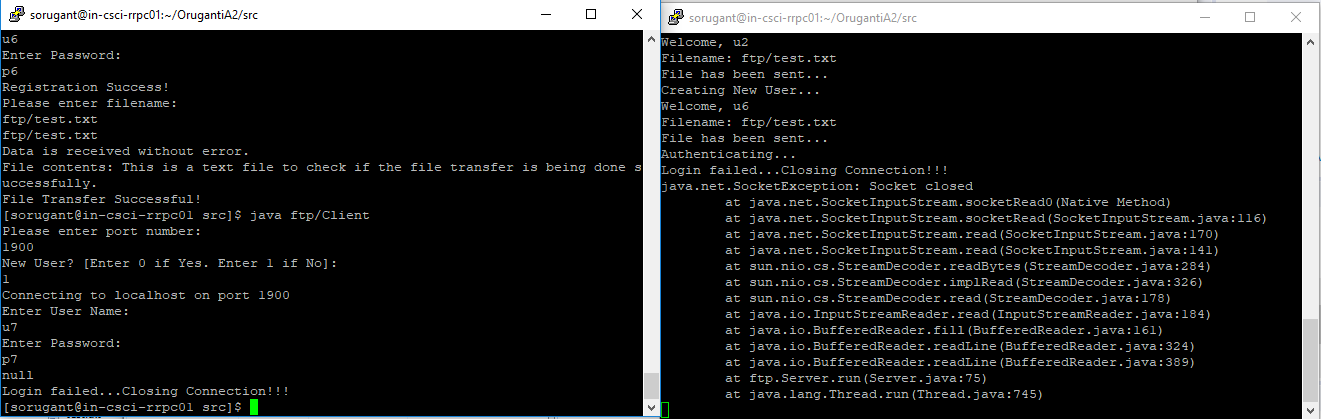


Figure Invalid Login Credentials Error

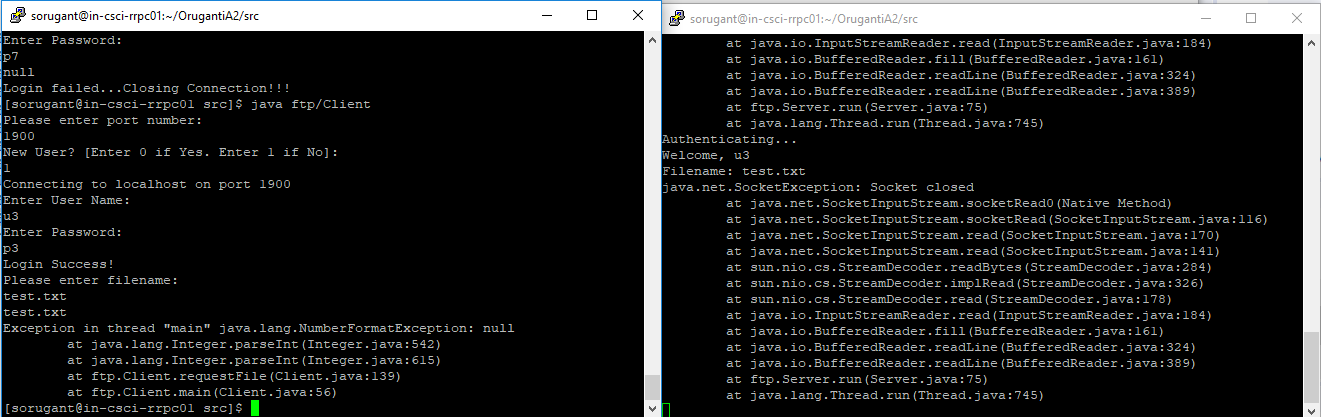


Figure File Not Found Error