CSCI 53700 – Fall 2019

Assignment Number 2

Due Date: October 31, 2019

This assignment is intended to reinforce the principles of inter-process communication and the end-to-end argument. It will also make you familiar with Java and its networking API. You have to develop a simple distributed file transfer system (similar to the one described in the end-to-end argument paper) using Java's *stream sockets*. This system will consist of a client and a server to be deployed on different machines. The server will provide following functions to the client:

- `register`: The client will register itself with the server using simple credentials (exact details of these credentials are left to you to decide). If the registration process is successful then the server will store these credentials (again, could be in any manner) and send a congratulatory message to the client. In case of an unsuccessful registration, an error message will be sent to the client by the server.

- `create`: The client will provide a file name as the parameter and if that file does not exist on the server then a new file with that name will be created. Once created, this file will be populated with random content – the exact mechanism of creating this content is irrelevant to the assignment – and it will be stored by the server. The server will also send a message, to the client, indicating the completion of the task. If the file creation fails, an error message will sent to the client by the server.

- `list`: The client will ask for a list of all the files on the server. This list will be sent, as plain text, by the server to the client. If there are no files on the server then an error message will be sent to the client by the server.

- `transfer`: The client will request for transferring a file by providing its name as the parameter. If the requested file exists on the server then it will be sent across to the client. The contents of the file will be encrypted, using some simple technique such as the security API provided by Java, before the transmission. If the requested file does not exist on the server then an error message will be sent to the client by the server.

- `summary`: The client will request a summary of a specific file by providing the file name as the parameter. If the file exists on the server then the server will send its summary to the client. This summary will consists of: a) the name of the file, b) the total number of lines in the file, and c) the total number of words in the file. This summary will be sent as plain text to the client. If the requested file does not exist then an error message will be sent to the client by the server.

- `subset`: The client will request a chunk of a specific file by providing the file name as the parameter. If the file exists on the server then the server will randomly select a subset of the file and return this subset using encryption (similar to the file transfer situation above) to the client. If the requested file does not exist then an error message will be sent to the client by the server.

- `delete`: The client will request the deletion of a file by specifying its name. If the file exists on the server then it will be deleted and the server will send back a message, to the client, indicating the success of the deletion operation. If the requested file does not exist then an error message will be sent to the client by the server.

- `close`: The client will request to close the connection with the server. The server will reply by sending a goodbye message and then the connection between the client and the server will be closed.

The server will exhibit Byzantine behavior while performing the `transfer` and `subset` functions. Hence, the client and the server need to provide end-to-end checks to ensure that the contents were successfully transferred (i.e., the contents from the original file on the server and the copy received by the client are identical). You may use any suitable technique for these checks. If the contents are not transferred successfully then an error message should be printed by the client. You may provide a simple text interface to invoke various functions, via the client, on the server.

Your design should consider the principles of the end-to-end argument. You must explicitly indicate, in the report, the design decisions that you have taken, in order to achieve the above mentioned functionality, that have been influenced by the end-to-end argument. Also, the consequences of these decisions should be discussed in the report.

Although, the assignment mentions one client and one server, if you design the server such that it can accept multiple concurrent requests, using the threads of Java, then you will get an extra credit of 10 points. Your program should handle exceptions. The program must run on any two machines (the server on one machine and the client on the other) from the following list:

```
in-csci-rrpc01.cs.iupui.edu 10.234.136.55
in-csci-rrpc02.cs.iupui.edu 10.234.136.56
in-csci-rrpc03.cs.iupui.edu 10.234.136.57
in-csci-rrpc04.cs.iupui.edu 10.234.136.58
in-csci-rrpc05.cs.iupui.edu 10.234.136.59
in-csci-rrpc06.cs.iupui.edu 10.234.136.60
```

In case your system considers multiple concurrent clients, you may use other machines from the above set.

Employ good software engineering principles in your design and implementation and provide adequate documentation of your program. Create a *makefile* and a *readme* file for your program. All these files (source files, readme, makefile, sample outputs, and report) should be submitted via the *submitd* command on tesla.cs.iupui.edu in a zipped folder with the following format (LastNameA2.zip) - e.g., RajeA2.zip. Also, hand-in a hard-copy of the report at the beginning of the class on the due date.