SYSC 3303 – TFTP Project - 1000000

# ITERATION 3

|  |  |
| --- | --- |
| Colin Kealty:  100855810  Kais Hassanali:  100861319 | Mohammed Ahmed-Muhsin:  100855437  Samson Truong:  100848346 |
| Last Updated: 5/31/2014 12:22:10 PM | |

## COMPILATION AND UTILIZATION

1. File>Import
2. select General > Existing Projects into Workspace
   1. "Select root directory" - browse and navigate to the folder of iteration
   2. If not already checked, check the box in the "Projects:" box next to the project which you imported
3. Ensure that there is a Server folder in the working directory. In this directory, you can put any files that you want to transfer FROM the server to the client
4. Compile the 5 files to ensure that all intended features work properly. The three steps outlined below must be followed in sequence to ensure full utility:
5. Run Server.java
6. Run ErrorSim.java
   1. Select the ErrorSim operation mode
7. Run ClientUI.java
   1. Select if you are reading or writing
      1. If you are Reading, make sure the file exists in the Server directory
   2. Type in the file name
   3. Type in the proper directory
8. After a successful transfer, you need to select the operation mode for the ErrorSim again by switching consoles to the ErrorSim and then switching back to the client console to select the file to transfer

Notes:

* *All output for a Read will be in the directory you specified in the name: output[nameOfFile].txt*
* *All output for a Write will be in the Server directory in the name: output[nameOfFile.txt]*
* The Client and Server will timeout 5 times before truly timing out
* Our client and server allows for files to be overwritten with no error message being thrown. This was up to us for the design of our system and we have implemented an over writing rule if the file exists already on both, the client and the server

## KNOWN ISSUES

* Duplicating a RRQ and WRQ partially works
* You can only invoke one operation in the client at a time. Once it is done, you can continue to do any other operation
* There is no Server shut down implemented

## TESTED AND WORKING FOR

Windows 7 at Carleton University Lab AA 508 running JAVA 1.7.0\_25

Windows 8.1 i7 960 @ 3.20GHz 8 GB RAM running JAVA 1.7.0\_25

Windows 8.1 i5 3570k @ 3.80GHz 16 GB RAM running JAVA 1.7.0\_25

## IDE AND JAVA DEVELOPED ON

Eclipse IDE Release 4.3.2 running JAVA 1.7.0\_25

## TESTING RESULTS

Upon using any of the provided files (512.txt which is a file that is exactly 512 bytes OR Over 512.txt which is a file that is over the 512 bytes), you can use <http://www.diffchecker.com/> to check the output file and the source file.

## CHANGELOG

### CLIENT:

Includes ClientUI.java and Client.java

#### 5.0:

* Improved packet recognition to shut down at the appropriate time
* New behavior which can handle error packets with error packets, 1 2 3 and 6

#### 4.0:

* Implemented a client UI to choose the mode and the which file to transfer
* Implemented the handling of errors

#### 3.0:

* Added file I/O support to actually receive and send a file

#### 2.0:

* Improved documentation and in-line comments
* Improved system messages on what is being done by the program
* Implemented a single read, write, and invalid request to be sent
* Improved information printing regarding the bytes and the string
* Create separate methods for the read, write, and invalid request

#### 1.0:

* All DatagramSockets and Packets implemented
* Basic message sent to intermediate to ensure all connections working
* Basic information about the packet displayed at each instance

### ERRORSIM

Includes ErrorSim.java and ConnectionManagerESim.java:

#### 5.0:

* User is able to select exactly which packet to lose, delay or duplicate and how much by
* Fixed issues in the duplication of a DATA or ACK packet for both a READ and a WRITE request
* Improved user input interface to indicate which error they would like to simulate

#### 4.0:

* Mode selection has been added
  + **Lost**: A predetermined percentage of packets will be lost (ie. simply not transferred) to the client or server
  + **Delayed**: A predetermined percentage of packets will be delayed by a random time (XXXXs - XXXXXs) to the client of server
  + **Duplicated**: A predetermined percentage of packets will be duplicated (with a random delay between them) to the server or client.
* User can properly shut down the ERRORSIM
* Bug fix where the ERRORSIM will only send to the server's Well-Known port (69). ERRORSIM will now send the secondary packets to the correct port.

#### 3.0:

* Added multi-threading capabilities to program
* Will remain to be sending and receiving files as needed with no errors simulated

#### 2.0:

* Improved documentation and in-line comments
* Improved system messages on what is being done by the system
* Improved information printing regarding the bytes and the string
* Reworked the sendSocket to remain open throughout the session

#### 1.0:

* Implemented all DatagramSockets required (receive, sendReceive, and send)
* Implemented all DatagramPackets required (client and server)
* Exception handling with try/catch blocks
* Basic information about the packet displayed at each instance

### SERVER

Includes ConnectionManager.java and Server.java:

#### 4.0:

* Implemented error handling

#### 3.0:

* Separated into a listening module which spawns out the threads to deal with request
* Multi-threaded requests to handle the reading and writing of the file

#### 2.0:

* Improved documentation and in-line commenting
* Improved the printing of information about the packet
* Added a parser which will recognize a valid request (read or write) and an invalid one
* Keep the server alive forever until killed

#### 1.0:

* Implemented a procedure that can receive a general request and send a response
* Print all necessary information about the packet
* Create a sending DatagramSocket and close it after a successful send

## DEBUGGING

By default, the ErrorSim runs in verbose mode. For silent mode operation, please change line 192 in ErrorSim.java to be: “**Thread** connectionmanager = **new** ConnectionManagerESim(false, *userChoice*, delayAmount, packetType, packetNumber, data, receiveClientPacket.getPort(), receiveClientPacket.getLength(), verifyReadWrite(receiveClientPacket));

This will ensure that the first parameter being passed is false, meaning, not verbose mode.

In future implementation, this will be an option to choose at the ErrorSim startup.

## SUPPORT

For technical support or to report a bug, please contact:

* KaisHassanali@cmail.carleton.ca
* SamsonTruong@cmail.carleton.ca
* MohammedAhmedMuhsin@cmail.carleton.ca
* ColinKealty@cmail.carleton.ca