|  |  |  |
| --- | --- | --- |
| SYSC 3303: Final Project | June 9  2015 | |
| By: Mohammed Hamza, Scott Savage, Matthew Pepers, Hidara Abdallah, Adhiraj Chakraborty, Anuj Dalal | | Group #2 |

Table of Contents

[Breakdown of Responsibilities 2](#_Toc421546972)

[Iteration #1: 2](#_Toc421546973)

[Iteration #2: 2](#_Toc421546974)

[Iteration #3: 3](#_Toc421546975)

[Iteration #4: 4](#_Toc421546976)

[Iteration #5: 4](#_Toc421546977)

[Set-Up Instructions 5](#_Toc421546978)

[INSTALL/RUN 5](#_Toc421546979)

[USE 5](#_Toc421546980)

[CLIENT 5](#_Toc421546981)

[SERVER 6](#_Toc421546982)

[ERROR SIMULATOR 6](#_Toc421546983)

[TESTING 6](#_Toc421546984)

[ERROR CODE 03 7](#_Toc421546985)

[ERROR CODE 03 (ALTERNATE) 7](#_Toc421546986)

[PATHS THROUGH ERROR SIMULATION MENU 7](#_Toc421546987)

[DISCLAIMER 10](#_Toc421546988)

[Diagrams 10](#_Toc421546989)

[UCM 10](#_Toc421546990)

[Timing Diagrams 10](#_Toc421546991)

[UML Class Diagram 10](#_Toc421546992)

# Breakdown of Responsibilities

## Iteration #1:

**Hidara Abdallah:** UM Class diagram and aided in ErrSim code.

**Matthew Pepers:** Worked on majority of server code.

**Mohammed Hamza:** Helped with server and client code.

**Adhiraj Chakaborty:** Worked on client code and aided with all other programming aspects.

**Scott Savage:** Worked on Errsim Code.

**Anuj Dalal:** Worked on UCM and aided with all other coding.

## Iteration #2:

For this iteration, our team decided to split the tasks up into blocks of code instead of having only one or two people focus on ALL the client or server code. The task division is as follows:

**Matthew Peper:** Dealt with Parsing of errors and creating error packets on the server side, assisted with coding of threads on the server and establishing a connection between client and server, migrating previous code over and easing the process of optimization through segmenting the code into smaller chunks.

**Mohammed Hamza:** Was assigned methods to deal with parsing incoming data on the client side, also assisted with parsing incoming error packets as the logic was similar, also assisted in quality review and debugging.

**Hidara Abdallah**: Dealt with all methods that had to do with creating error packets and transmitting them as well as parsing received error packets, also reviewed other member’s methods and helped to debug problems. Created this document for team contributions and UCM diagram.

**Anuj Dalal:** Dealt with parsing ACK packets on client and server side, debugged other member’s code throughout entire process, aided with quality review as well as creation of class diagram for this iteration.

**Scott Savage:** Dealt with creation of ACK packets and DATA packets on client side. Aided with overall debugging and code review and optimization.

**Adhiraj Chakraborty:** Dealt with creating and receiving datagram packets from client to server, also aided in code debugging and optimization from previous iteration. Aided with receive and send data methods between client and server.

## Iteration #3:

**Matthew Peper:** Creating code on server for handling of error codes 4 +5.

**Mohammed Hamza:** Timing diagrams, and assisted with Errsim GenericError method.

**Hidara Abdallah:** Did ErrorSim user interface, and assisted with initial skeleton of Errsim.

**Anuj Dalal:** Did class diagrams and helped with server code and debugging.

**Scott Savage:** Did CreateError method 5 and helped debug code in Errsim.

**Adhiraj Chakraborty:** Helped with code in Errsim namely UI ErrSimQuit and createError4.

## Iteration #4:

**Mathew Pepers:** Worked on code for Server Client and Error simulator. Was responsible for most of the implementation of theory and planning in terms of what methods needed to be coded. Mathew acted as the main programmer for iteration 4.

**Mohammed Hamza:** Mapped out Pseudo code for changes made in error simulator after being advised by TA to re-model current error simulator. Created skeleton methods (methods to be worked on) in project and helped to assign to group members.

**Adhiraj Chakraborty:** Contributed to code in error simulator behind simulating error code 4, and helped with testing multiple different scenarios in terms of errors, also made class diagram.

**Scott Savage:** Contributed to code behind error code 5, providing useful insight from his previous work with the create error 5 method. Also did timing diagrams for iteration 4.

**Anuj Dalal:** Aided in creating bulk of error simulator, including error code 4 simulating tampering with the mode in a RRQ/WRQ. Also worked on code to delay packets.

**Hidara Abdallah:** Worked on several error simulating traits to do with error code 4 such as tampering with file name, opcode and block number.

## **Iteration #5**:

**Mathew Pepers:** Acted as one of the testers for this iteration. Implemented new methods for iteration 5 to deal with several clients. Aided with debugging the code from previous iterations, and made improvements suggested during demo of iteration 4.

**Mohammed Hamza:** Worked on diagrams and acted as tester for this final implementation as well as worked on final documentation.

**Adhiraj Chakraborty:** Acted as tester, as well as wrote suggested improvement methods for error simulator. Made class diagram for iteration 5.

**Scott Savage:** Contributed to timing diagrams (with experience from past iterations) also acted as tester for this final iteration.

**Anuj Dalal:** As the main tester for the demo of iteration 4, he was the go-to tester for this iteration’s beginning. He aided with deriving new techniques for solving bugs from previous code. Helped with adding improvement code.

**Hidara Abdallah:** Acted as tester for newer methods created, as well as to make sure older ones were functioning in conjunction with newer ones.

# Set-Up Instructions

## INSTALL/RUN

* create a new Java Project in Eclipse
* add Client.java, Server.java, and ErrorSim.java to the project
* compile and run Server.java, then ErrorSim.java, then Client.java from within Eclipse
* if the .java files are in your project \src\ folder, put your \files\ folder at the same level as \src\
* the Client's files are in '\files\client\'
* the Server' files are in '\files\server'
* you should put the files that you want to send and receive in their respective folders

## USE

### CLIENT

* follow onscreen console prompts from the Client
* first enter the InetAddress of the Server. This may also be entered as a machine name.
* next choose whether you want to send the read/write request directly to the (S)erver, or through the (E)rror Simulator
* then choose whether you want to send a (R)ead Request, a (W)rite Request, (E)nter a new destination,or (Q)uit
* next type in the name of the file that you want to read/write
* test text files for the Client and Server are included in their respective folders to facilitate testing, but any file may also be put into those folders in order to transfer between client/server
* once a file transfer has ended, the onscreen console prompts will ask you if you want to send another request or quit

### SERVER

* when the Server starts, it listens on port 69 for request packets
* you can (Q)uit any time by typing 'Q' and pressing 'Enter'. This will quit the Server, but any file transfers that are still ongoing will continue until they complete or timeout

### ERROR SIMULATOR

* run ErrorSim on the same machine as the Server is running
* when ErrorSim starts, a console prompt within Eclipse will ask the user if they want to start in (N)ormal mode, or (E)rror Simulator mode
* Normal Mode facilitates the file transfers by sending packets back and forth between the Client and Server and not interfering with them
* Error Simulator Mode allows the user to simulate errors happening during the file transfer, and the user will be prompted to choose a variety of settings with which to simulate those errors
* the Error Simulator must be closed and restarted between each transfer

# TESTING

* the Client and Server both have socket timeouts of
* 2 seconds, in order to be both short enough for testing, and long enough to notice when reading console output
* most testing can be done through following the onscreen console prompts in the Error Simulator, but for those errors that can't be simulated through the Error Simulator:
* putting a space in the filename generates error from RRQ but not from WRQ from client to server
* when changing block # on data packet from server (ie packet 3 to block # 1) look for reception of ack # 1 at server

### ERROR CODE 03

* to test Error Code 03 (Disk full), the Eclipse project should be installed and run from a USB drive
* while the Client, Server, and ErrorSim are running on your USB drive, check how many bytes are free and usable on the USB drive
* open a Windows Command Prompt (cmd), and type the following:

fsutil file createnew D:\hugefile.tmp SIZE

* replace 'SIZE' with the number of bytes that you found were free on your USB drive, and replace 'D' with the drive letter of your USB drive
* this will create a new file on your USB drive to take up the remaining space
* next try sending a RRQ or WRQ, and you will get Error Code 03, because the disk is full, and files are not able to be written

### ERROR CODE 03 (ALTERNATE)

* if you are not able to use the fsutil command above because of lack of privilages, you can add a bunch of files to your USB drive manually, until it is full

### PATHS THROUGH ERROR SIMULATION MENU

* when choosing Error Simulation Mode, instead of Normal Mode, the following paths through the menu can be taken to simulate a variety of errors:

#### - 1. Lose a packet.

- 1. RRQ

- coming from Client? or Server?

- 2. WRQ

- coming from Client? or Server?

- 3. DATA

- which DATA (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

- 4. ACK

- which ACK (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

- 5. ERROR

- which ERROR (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

#### - 2. Delay a packet.

- 1. RRQ

- how long of a delay?

- coming from Client? or Server?

- 2. WRQ

- how long of a delay?

- coming from Client? or Server?

- 3. DATA

- how long of a delay?

- which DATA (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

- 4. ACK

- how long of a delay?

- which ACK (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

- 5. ERROR

- how long of a delay?

- which ERROR (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

#### - 3. Duplicate a packet.

- 1. RRQ

- how long of a delay between duplicated packets?

- coming from Client? or Server?

- 2. WRQ

- how long of a delay between duplicated packets?

- coming from Client? or Server?

- 3. DATA

- how long of a delay between duplicated packets?

- which DATA (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

- 4. ACK

- how long of a delay between duplicated packets?

- which ACK (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

- 5. ERROR

- how long of a delay between duplicated packets?

- which ERROR (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

#### - 4. Send a packet.

- 1. RRQ

- enter a filename

- which packet (1st, 2nd, 3rd, etc.)

- sending to Client? or Server?

- 2. WRQ

- enter a filename

- which packet (1st, 2nd, 3rd, etc.)

- sending to Client? or Server?

- 3. DATA

- enter a block number (0-65535)

- which packet (1st, 2nd, 3rd, etc.)

- sending to Client? or Server?

- 4. ACK

- enter a block number (0-65535)

- which packet (1st, 2nd, 3rd, etc.)

- sending to Client? or Server?

- 5. ERROR

- enter an error code (0-8, 8 is invalid)

- which packet (1st, 2nd, 3rd, etc.)

- sending to Client? or Server?

#### - 5. Edit a packet.

- 1. RRQ

- 1. make opcode invalid

- coming from Client? or Server?

- 2. make filename invalid

- coming from Client? or Server?

- 3. make mode invalid

- coming from Client? or Server?

- 4. change TID

- coming from Client? or Server?

- 2. WRQ

- 1. make opcode invalid

- coming from Client? or Server?

- 2. make filename invalid

- coming from Client? or Server?

- 3. make mode invalid

- coming from Client? or Server?

- 4. change TID

- coming from Client? or Server?

- 3. DATA

- 1. make opcode invalid

- which DATA (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

- 2. change block number (0-65535)

- which DATA (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

- 3. delete data field

- which DATA (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

- 4. change TID

- which DATA (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

- 4. ACK

1. make opcode invalid

- which ACK (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

2. change block number (0-65535)

- which ACK (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

3. change TID

- which ACK (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

- 5. ERROR

1. make opcode invalid

- which ERROR (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

2. change error code (0-8, 8 is invalid)

- which ERROR (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

3. change TID

- which ERROR (1st, 2nd, 3rd, etc.)

- coming from Client? or Server?

### DISCLAIMER

* + Many of the previous paths will produce the same errors, and many paths won't affect file transfer at all (eg: if you choose a packet number, but the file transfer isn't long enough to make it to that packet). It is up to the user of the Error Simulator to understand how file transfer program works, and how to correctly test for various TFTP errors.

# Diagrams

## UCM

-See Figures at end of document

## Timing Diagrams

-See Figures at end of document

## UML Class Diagram

See Figure at end of document