**SYSC3303 Project:**

Full Project Presentation

**05/06/17**

**Group #1 (1000000):**

* Ben Croskery (100973306)
* Ben Earle (100970237)
* Dillon Verhaeghe (100965889)
* Patrick Perron (100965104)
* Shane Corrigan (100965710)

**Table of Contents**

* **Responsibilities for each Iteration - 3**
* **Assumptions made for conditions not specified by TFTP - 4**
* **Java Files - 4**
* **Diagrams - 5**
* **Test Files - 6**
* **Instructions for running - 7**
* **Instructions for Test Cases - 8**
* **Github Source Link - 12**

**Responsibilities for each Iteration**

**Iteration 1:**

* **Ben Croskery**: Provided base code, util classes, FileRead/Writer Classes, Test Classes, Client Read/Write
* **Ben Earle:** Server main, control, read and write threads, command line interfaces, formatting
* **Dillon Verhaeghe:** Client side support, verbose mode, test mode, conforming to TFTP protocal
* **Patrick Perron:** Writing threads for server, debugging read/write transfers, Main ErrorSimulator code
* **Shane Corrigan:** UMC Diagrams, Server main user interface, documentation

**Iteration 2:**

* **Ben Croskery:** unit testing, terminate incompleted files in bad transfers, testing of packet checking
* **Ben Earle:** UML Sequence diagrams, UDP helper class and replacing all old udp code, client code
* **Dillon Verhaeghe:** Created packet checking methods for various packet types, fixed bugs
* **Patrick Perron:** Error Sim interface, sabotaging packets with Error Sim, Debugging
* **Shane Corrigan:** Updated client to use UDPHelper, UML class, client debugging

**Iteration 3:**

* **Ben Croskery:** Testing and utility scripts.
* **Ben Earle:** Typed for group programing for err code 1, 2, and 3
* **Dillon Verhaeghe:** Research and design for err code 1, 2, and 3
* **Patrick Perron:** Research and design for err code 1, 2, and 3
* **Shane Corrigan:** Research and design for err code 1, 2, and 3

**Iteration 4:**

* **Ben Croskery:** Fixing bugs, diagrams, verification testing
* **Ben Earle:** Fixing bugs, diagrams, verification testing
* **Dillon Verhaeghe:** Fixing bugs, diagrams, verification testing
* **Patrick Perron:** Error Sim Interface, Duplication, Loss, Delay, etc.
* **Shane Corrigan:** ixing bugs, diagrams, verification testing

**Iteration 5:**

* **Ben Croskery:**
* **Ben Earle:**
* **Dillon Verhaeghe:**
* **Patrick Perron:**
* **Shane Corrigan:**

**Assumptions made for conditions not specified by TFTP**

* We chose to delete files if there was an error in the transfer. We did not want half complete files to exist on the client or server.
* We chose to NOT allow the client and server to overwrite files. This means that our TFTP client and Server will be sending error code 6 if a file already exists on the server in a WRQ.
* The client and server each try resend packets up to 3 times if no response is received. Quits after the 3rd retransmission

**Java Files**

**Main Classes:**

* **Server.java:** Code for Server User Interface, launches ControlThread
* **Client.java:** Code for Client that will send Read/Write requests
* **ErrorSimulator.java:** Code for Error simulator, that forward messages back and forth between client and server. Sabotages packets to test errors if desired
* **DelayedSendThread.java:** Thread for the ErrorSim to send a packet to the server or client after a delay specified in ms
* **ReadThread.java:** Server Thread for handling read request from server
* **WriteThread.java:** Server Thread for handling write request from server
* **ControlThread.java:** Server Thread for listening for requests from clients and creating threads to handle them
* **ClientResponseThread.java:** Abstract class with shared code for read/write threads

**Util Classes:**

* **util/ErrorScenario.java:** Describes an error case to test for ErrorSimulator
* **util/Var.java:** Contains shared Constants
* **util/Log.java:** Contains methods for controlling logging to console
* **util/FileReader.java:** Class to handle writing bytes to a file
* **util/FileWriter.java:** Class to handle readinh bytes from a file
* **util/TFTPErrorHelper.java:** Static class that checks packets for specific errors
* **util/UDPHelper.java:** Class to faciliate UDP send and receive operations

**Diagrams**

**Iteration 1:**

* **Diagrams/Iteration1/IT1 RRQ Connection.png**
* **Diagrams/Iteration1/IT1 WRQ Connection.png**
* **Diagrams/Iteration1/IT1 RRQ Data Transfer.png**
* **Diagrams/Iteration1/IT1 WRQ Data Transfer.png**
* **Diagrams/Iteration1/UML\_default.PNG**
* **Diagrams/Iteration1/UML\_util.PNG**

**Iteration 2:**

* **Diagrams/Iteration2/IT2\_Bad\_RRQ.png**
* **Diagrams/Iteration2/IT2\_Bad\_WRQ.png**
* **Diagrams/Iteration2/IT2\_Client\_Receives\_Bad\_ACK.png**
* **Diagrams/Iteration2/IT2\_Client\_Receives\_Bad\_Data.png**
* **Diagrams/Iteration2/IT2\_Client\_Receives\_Packet\_From\_Unkown\_Sender.png**
* **Diagrams/Iteration2/IT2\_Server\_Receives\_Bad\_ACK.png**
* **Diagrams/Iteration2/IT2\_Server\_Receives\_Bad\_DATA.png**
* **Diagrams/Iteration2/IT2\_Server\_Receives\_Packet\_From\_Unkown\_Sender.png**

**Iteration 3:**

* **Diagrams/Iteration3/IT3\_RRQ\_Code1**
* **Diagrams/Iteration3/IT3\_RRQ\_Code2**
* **Diagrams/Iteration3/IT3\_RRQ\_Code3**
* **Diagrams/Iteration3/IT3\_WRQ\_Code3**
* **Diagrams/Iteration3/IT3\_WRQ\_Code6**

**Iteration 4:**

* **Diagrams/Iteration4/IT4\_**
* **Diagrams/Iteration4/IT4\_**
* **Diagrams/Iteration4/IT4\_**
* **Diagrams/Iteration4/IT4\_**
* **Diagrams/Iteration4/IT4\_**

**Iteration 5:**

* **Diagrams/Iteration5/IT5\_**
* **Diagrams/Iteration5/IT5\_**
* **Diagrams/Iteration5/IT5\_**
* **Diagrams/Iteration5/IT5\_**
* **Diagrams/Iteration5/IT5\_**

**Test Files**

* **c\_0.txt, s\_0.txt:** Empty file
* **c\_512.txt, s\_512.txt:** File with 512 bytes of ASCII characters
* **c\_1221.txt, s\_1221.txt:** File with 1221 bytes of ASCII characters
* **c\_bee.png, s\_bee.png:** Picture of bee in png format
* **c\_bee.txt, s\_bee.txt:** Script to "Bee Movie" stored as ASCII characters
* **c\_jpg.jpg, s\_jpg.jpg:** Basic jpg file stored as binary
* **50mb.zip:** Large file to test wraparound of block numbers

\*\*NOTE: s\_\* and c\_\* convention for test files is used to indicate if file originally existed in client root folder or server root folder. Client files in /src/testFile/, Server files in /src/testFile/server/.

**Instructions for running**

**Setup:**

1. Compile and run Server.java

* For VERBOSE mode: enter 'v' or 'verbose' to toggle verbose after startup, OR pass 'v' as an initial argument

1. (OPTIONAL) Compile and run ErrorSimulator.java

* Select desired error(1,2,3,4,5) or no error mode(0). Follow instructions to set up scenario

1. Compile and run Client.java

* For VERBOSE mode: enter 'v' or 'verbose' to toggle verbose after startup, OR pass 'v' asan initial argument
* For ERROR SIM mode: enter 't' or 'test' to toggle test mode after startup, OR pass 't' asan initial argument

**Write:**

1. Enter 'w' on client console
2. Enter a file name found in the client root from Test Files mentioned above.

* File should be written as c\_\* to indicate it is in client root (i.e. c\_512.txt, c\_bee.txt)

1. In /src/testFile/server/, an identical copy of file should be found

**Read:**

1. Enter 'r' on client console

* Enter a file name found in the server root from Test Files mentioned above.

1. File should be written as s\_\* to indicate it is in server root (i.e. s\_0.txt, s\_jpg.jpg)
2. In /src/testFile/, an identical copy of file should be found

**Quitting:**

1. Enter 's' on client console
2. Type 's' on Server console
3. (OPTIONAL) Ctrl+C on Error Simulator console

**Instructions for Test Cases**

**Lost Packet:**

1. Start server and client in verbose and test modes
2. Start Error Simulator. Setup desired lost packet error scenario using error simulator interface
3. Chose Error Category 1
4. Chose desired packet type to lose (READ, WRITE, DATA, ACK, ERROR)
5. If DATA or ACK, chose packet number to trigger the fault
6. Start the required Read or Write operation to trigger the lost packet.
   * To simulate a lost RRQ packet, start read request
   * To simulate a lost WRQ packet, start write request
   * To simulate a lost DATA or ACK packet, start read or write request
   * To simulate a lost ERR packet, start read request with a file that does not exist in the server dir
7. In /src/testFile/, an identical copy of file should be found
8. In the logs of each of the 3 programs, the system behaviour for a missing packet is shown

**Delayed Packet:**

1. Start server and client in verbose and test modes
2. Start Error Simulator. Setup desired lost packet error scenario using error simulator interface
   1. Chose Error Category 2
   2. Chose Desired packet type to delay (READ, WRITE, DATA, ACK, ERROR)
   3. Chose a time in ms to delay packet by
   * Socket Timeout time is 1000 ms.
   * If delay is less than 1000 ms, transfer is not affected except for small delay
   * If delay is more than 1000 ms, client/server will resend packets in place to continue the transfer
   * If delay is too large, the delayed packet will be received after the end of the transfer.
   * Try to keep delays less than 1050 ms, i.e.(1010 - 1050 ms), to ensure the packet is retransmitted during the transfer.
   1. If DATA or ACK, chose packet number to trigger the fault

* If timeout is more than 1000 ms, use a lower ack packet number (i.e. 2,3,4), or the delayed packet will be received after the end of the transfer.

1. Start the required Read or Write operation to trigger the lost packet.

* To simulate a lost RRQ packet, start read request
* To simulate a lost WRQ packet, start write request
* To simulate a lost DATA or ACK packet, start read or write request
* To simulate a lost ERR packet, start read request with a file that does not exist in the server dir

1. In /src/testFile/, an identical copy of file should be found
2. In the logs of each of the 3 programs, the system behaviour for a delayed packet is shown

**Duplicate Packet:**

1. Start server and client in verbose and test modes
2. Start Error Simulator. Setup desired lost packet error scenario using error simulator interface
   1. Chose Error Category 3
   2. Chose Desired packet type to duplicate (READ, WRITE, DATA, ACK, ERROR)
   3. Chose a time in ms between duplicate packets
   * If delay is too large, the duplicayed packet will be received after the end of the transfer.
   * Try to keep delays less than 50 ms, i.e.(10 - 50 ms), to ensure the packet is retransmitted during the transfer.
   1. If DATA or ACK, chose packet number to trigger the fault
3. Use a lower ack packet number (i.e. 2,3,4), or the duplicated packet will be received after the end of the transfer.
4. Start the required Read or Write operation to trigger the lost packet.

* To simulate a lost RRQ packet, start read request
* To simulate a lost WRQ packet, start write request
* To simulate a lost DATA or ACK packet, start read or write request
* To simulate a lost ERR packet, start read request with a file that does not exist in the server dir

1. In /src/testFile/, an identical copy of file should be found
2. In the logs of each of the 3 programs, the system behavior for a duplicate packet is shown

**Github Source Link**

https://github.com/BenEarle/SYSC3303-Project

\*Note: GitHub project is private - if you would like access, please let us know.