Specification: Version 1 (subject to modifications!)

Teams will design and implement a file transfer system based on the TFTP specification (RFC 1350) that is available on the course website and that has been discussed in class. The system will consist of TFTP client(s) running on one or several computers, an error simulator, and a multithreaded TFTP server that runs on a different computer. The code will be written in Java, using the Eclipse IDE. *You must design your code to work in the lab environment provided!*

Note that there will be three separate programs: the client(s), the server, and the error simulator(s). Each program will run as a separate Win-32 process, and the programs will communicate via DatagramSocket objects. In “normal” mode, only the client and server programs will run. In “test” mode, all three programs will be used.

You must be able to run multiple main programs (projects) concurrently. See the information at the top of the "Assignments" tab for how to do this in Eclipse.

Your team's code should demonstrate good programming style, and be well documented. For examples of “industrial quality” Java code, have a look at Sun's Java coding conventions, which can be found on the Oracle Web site.

All team members should be familiar with all aspects of the code and diagrams for your group. Working in (at least) pairs for the programming, debugging, and developing the diagrams is recommended. If any team members are not pulling their weight, please notify the TAs and/or instructor as early in the term as possible so that this can be remedied.

Client Specification

The client will be implemented as a Java program that consists of one or more Java threads. The client will provide a *simple* user interface (*a GUI is neither required nor recommended*) that allows the user to input:

- the file transfer operation (read file from server, write file to server)
- the name of the file that is to be transferred

Do **not** prompt for the mode, as, for purposes of this course, it doesn't matter whether it is "netascii" or "octet".

You will also need to prompt for various other information, e.g. normal vs. test, verbose vs. quiet, IP address of server, client/server directory, shut down, etc. More on this under "User Interface" below.

The client will then attempt to establish the appropriate connection with the server and transfer the file. After the current connection has been terminated (either because the file was transferred successfully or because an unrecoverable error occurred), the client should permit the user to initiate another file transfer. When the user indicates that no more files are to be transferred, the client should terminate.

The client should **not** support concurrent file transfers; for example, the client will not be able to concurrently transfer multiple files to and from one or more servers.

Note the need to implement "test" and "normal" modes -- see above, as well as "verbose" and "quiet" modes -- see below (User Interface).

Internet protocols always use the normal (left to right; big endian) notation for block numbers. It is critical that you follow this convention, as well as all other aspects specified in the TFTP protocol, as **your client and server must be able to communicate with those of other groups and commercially available TFTP implementations to be considered correct.**

Server Specification

The server will be implemented as a Java program that consists of multiple Java threads. The server must be capable of supporting multiple concurrent read and write connections with different clients. To accomplish this, the server will have a multithreaded architecture. One thread will wait on port 69 (the "well-known port" for TFTP) for UDP datagrams (that should contain RRQ or WRQ packets). This thread should:

1. create another thread (call it the client connection thread), and pass it the TFTP packet to deal with; and
2. go back to waiting on port 69 for another request.

The newly created client connection thread will be responsible for communicating with the client to (attempt to) transfer a file.

Once started, the TFTP server will run until it receives a “shutdown” command from the server operator. Note that the server operator will type in this request in the server window. *It is neither desirable nor acceptable for a client to request that the server shutdown.* After being told to shut down, the server should **finish all file transfers that are currently in progress, but refuse to create new connections with clients**.

Error Simulation

Your error simulation code must be a completely separate program (or programs) from your client and server code. The error simulator communicates with the client and the server using DatagramSocket objects. The number of threads for this code is your choice. We will use **port 23** for the error simulator. Your error simulator may be single-threaded and is expected to work with one client only. To test multiple clients when using just one machine, send requests directly to port 69. (Once we move to multiple machines, it is recommended that the error simulator runs on the same computer as the client.) However, if you wish to make the error simulator multi-threaded, that is your choice.

User Interface

You must implement both a "quiet" and a "verbose" mode. While a real-world system would not output detailed information, we need the "verbose" mode to ensure the system is functioning correctly. In "verbose" mode the client, error simulator, and server output detailed information about packets received and sent, as well as any timeouts and/or retranmission of packets. The detailed packet information must include:

- whether the packet was sent or received (including any information about timeouts and/or retransmits, as appropriate);
- packet type (i.e. RRQ, WRQ, DATA, ACK, ERROR);
- filename (if applicable);
- mode (if applicable);
- block number as an integer (0 to 65535; if applicable);
- number of bytes of data (0 to 512; if applicable), but **not** the actual data.

Keep the user interface (UI) simple. Reduce the amount of input required each time by having options to change values, rather than having to input them each time (e.g. an option to toggle between "quiet" to "verbose" mode). To avoid confusion, keep things consistent with the specification when appropriate, e.g. use option 1 for RRQ and 2 for WRQ, rather than vice versa.

A GUI is neither recommended nor required! Do not develop a fancy UI comprised of instances of `JButton, JList, JComboBox, JCheckBox`, etc., etc., etc., for two reasons. First, developing a fancy UI is not the focus of this project. You should instead direct your efforts to ensuring that the programs correctly implement TFTP (how will you test ERROR packet creation and handling? How will you test the timeout/retransmit protocol?). Second, most of the methods in the Swing classes are not thread-safe. This means that mutual exclusion is not enforced if user-defined threads and Swing's event-dispatching thread invoke methods in Swing classes concurrently. You should be able to build an adequate UI using regular text windows or by using one or more `JTextArea` objects (possibly placed in `JScrollPane` objects) and various types of dialog boxes that can be created by class JOptionPane.
The `append()` method in `JTextArea` is one of the few Swing methods that is guaranteed to be thread-safe. You should review the `JTextArea` API to see if there are any other thread-safe methods that may be of use. Some notes about JOptionPane will be posted on the course Web site. *If you choose to use any features not mentioned above, you must provide a detailed explanation in your documentation as to the research you did and the reasoning process you went through to convince yourselves that the additional features you used are indeed thread-safe.* **This will likely not be a trivial undertaking!**

Development Process

The project will be developed using an iterative, incremental process. The result of each iteration will be the release of an executable piece of software that constitutes a subset of the final TFTP client and server software. The software will be grown incrementally from iteration to iteration to become the final system. **Note that when submitting iteration "n", it is absolutely not acceptable to include code for iteration "n+1", etc.**

Iteration 0 – Establish Connections for File Transfer without Error Detection and Correction

For this part, assume that no errors occur; i.e., no TFTP ERROR packets will be prepared, transmitted, received, or handled. Also, assume that no packets are duplicated in transit, and that no packets will be delayed or lost in transit, so the TFTP timeout/retransmit protocol is not supported.

In the lectures, we examined the timing diagrams for establishing a connection. Using these timing diagrams as a starting point, develop a skeleton TFTP client, error simulator, and server that run on the same computer. Your programs should permit clients to establish WRQ connections and RRQ connections with the server. For this iteration, do not implement the steady-state file transfer between the client and the server. It's sufficient to be able to demonstrate that clients can establish a RRQ connection and a WRQ connection, and that the server can establish multiple concurrent connections.

The error simulator will just pass on packets (client to server, and server to client), as we are not doing any error simulation at this point in time.

For each RRQ, the server should respond with DATA block 1 and 0 bytes of data (no file I/O). For each WRQ the server should respond with ACK block 0.

As noted earlier, the server must be multithreaded. For this iteration, each newly created client connection thread should terminate after it sends the appropriate acknowledgment to the client that requested the connection.

Also, there must be a nice way to shut down both your server and your client. CRTL-C is NOT a nice way!

Taking your solutions from assignment #1 and evolving that code into the first prototype of the TFTP system should be reasonably straightforward. The main difference for iteration 0 is that you will now need to create multiple threads in the error simulator and server code.

Work Products for Iteration #0:

- None - your first submission is Iteration #1 (below) which includes Iteration #0.

Iteration 1 – Implementation of File Transfer without Error Detection and Correction

The goal of this iteration is to extend the client, error simulator, and server programs to support steady-state file transfer. For this part, assume that no errors occur; i.e., no TFTP ERROR packets will be prepared, transmitted, received, or handled. Also, assume that no packets are duplicated in transit, and that no packets will be delayed or lost in transit, so the TFTP timeout/retransmit protocol is not supported.

In the lectures, we examined the timing diagrams for steady-state file transfer between a client and a server. Add steady-state file transfer capability to the client and server code developed in Iteration 0. As in Iteration 0, your server should create a new client connection thread for each connection with a client. You can have additional threads in the client and server, as long as you can justify them.

Again, the error simulator will just pass on packets (client to server, and server to client), as we are not doing any error simulation at this point in time.

The best way to check that files are being transferred properly is to use DOS command "fc" (file compare).

Don't forget that there must be a nice way to shut down both your server and your client. CRTL-C is NOT a nice way!

Work Products for Iteration #1:

- "README.txt" file explaining the names of your files, set up instructions, etc.
- Breakdown of responsibilities of each team member for this iteration
- UML class diagram
- The Use Cases (not UCMs) for this iteration.
- A state machine diagram for each of the three components
- Detailed set up and test instructions, including test files used
- Code (.java files, all required Eclipse files, etc.)

Iteration 2 – Adding Network Error Handling (Timeout/Retransmission)

Modify the client-server system from Iteration #1 to handle network errors. Packets can be lost, delayed, and duplicated. TFTP's "wait for acknowledgment/timeout/retransmit" protocol helps deal with this. Your program must contain the fix for the Sorcerer's Apprentice bug (i.e. duplicate ACKs must not be acknowledged, and only the side that is currently sending DATA packets is required to retransmit after a timeout, though both sides may retransmit). You may assume that there will be no File Input/Output errors (see Iteration #4). You may assume that there will be no TFTP packet format errors (see Iteration #3), and that there will be no File Input/Output errors (see Iteration #4).

You must submit code to enable us to see that your client and server deal properly with timeouts and retransmits – i.e. update your error simulator. As many of these errors will happen very infrequently in practice, the best way is to build in an **extensive** test menu **in the error simulator** to test both the client and the server that will allow you to test each situation, e.g. 0 : normal operation; 1 : lose a packet; 2 : delay a packet, 3 : duplicate a packet. Then you need to be able to select which packet to lose, delay or duplicate (e.g. RRQ, 2nd DATA, 3rd ACK, etc), and how much of a delay or space between duplicates. To repeat, you need to ensure that we can simulate the loss, delay or duplication of **any** packet, and the amount to delay / space the packets by (if applicable).

Work Products for Iteration #2:

- "README.txt" file explaining the names of your files, set up instructions, etc.
- Breakdown of responsibilities of each team member for this and previous iterations
- Any unchanged diagrams from the previous iterations
- UML class diagram
- Sequence diagrams showing the timeout/retransmit scenarios for this iteration
- The Use Cases (not UCMs) for this iteration.
- A state machine diagram for each of the three components
- Detailed set up and test instructions, including test files used
- Code (.java files, all required Eclipse files, etc.)

Iteration 3 – Adding TFTP Packet Format Errors (ERROR Packets 4, 5)

For this part, assume that errors can occur in the TFTP packets received, so TFTP ERROR packets dealing with this (Error Code 4, 5) must be prepared, transmitted, received, and handled. You may assume that there will be no File Input/Output errors (see Iteration #4).

Add support for ERROR packets as described above to the client and server code from Iteration #2. Note that you must parse every field of each TFTP packet for errors. You must submit code to enable us to see that your client and server deal with these errors – i.e. modify your error simulator. As many of these errors will happen very infrequently in practice, the best way is to build in an **extensive** test menu <u>**in the error simulator** to test both the client and the server by forcing each error to occur</u>, e.g. 0 : normal operation; 1 : invalid TFTP opcode on RRQ or WRQ; 2 : invalid mode, etc. We should be able to simulate any problem with any packet and any field within any packet.

Note that there is not a one to one mapping between your error scenarios and the TFTP ERROR codes! For example, there are <u>many</u> ways to get ERROR code 4!

Work Products for Iteration #3:

- "README.txt" file explaining the names of your files, set up instructions, etc.
- Breakdown of responsibilities of each team member for this and previous iterations
- Any unchanged diagrams from the previous iterations
- UML class diagram
- Sequence diagrams showing the timeout/retransmit scenarios for this iteration
- The Use Cases (not UCMs) for this iteration.
- A state machine diagram for each of the three components
- Detailed set up and test instructions, including test files used
- Code (.java files, all required Eclipse files, etc.)

Iteration 4 – Adding I/O Error Handling (ERROR Packets 1, 2, 3, 6)

For this part, assume that I/O errors can occur, so TFTP ERROR packets dealing with this (Error Code 1, 2, 3, 6) must be prepared, transmitted, received, or handled.

Add support for ERROR packets as described above to the client and server code from Iteration #3. Note that you must catch exceptions thrown by the your Java read/write code and translate the exceptions to the appropriate TFTP error codes. It should be possible to directly test all of these errors (e.g. file not found, etc.), i.e. you should not need to update the error simulator.

Work Products for Iteration #4:

- "README.txt" file explaining the names of your files, set up instructions, etc.
- Breakdown of responsibilities of each team member for this and previous iterations
- Any unchanged diagrams from the previous iterations
- UML class diagram
- Sequence diagrams showing the timeout/retransmit scenarios for this iteration
- The Use Cases (not UCMs) for this iteration.
- A state machine diagram for each of the three components
- Detailed set up and test instructions, including test files used
- Code (.java files, all required Eclipse files, etc.)

Project Demonstration – Iteration #3 or #4 (or #5)

Your team will give a 30 min demo to a TA and instructor of iteration #3, 4 or #5 (your choice). The goal of the demo is to give you feedback to improve your project before the final submission. Ensure that you have soft copies of all the work products to hand during the demo.

Work Products for Project Demonstration – Iteration #3, #4 or #5:

- “README.txt” file explaining the names of your files, set up instructions, etc.
- Breakdown of responsibilities of each team member for each iteration
- All diagrams
- Detailed set up and test instructions, including test files used
- Code (.java files, all required Eclipse files, etc.)

Iteration 5 – Implementation of File Transfer between Different Computers

Modify your client and server programs from Iteration #4 so that clients and the server can reside on different computers. The changes to Iteration #4 should be minimal. You just need to change the client's user interface to allow the user to specify the identity of the host where the server will run, and you'll need to modify the code that looks up the Internet address of the server's host. Think carefully about where the error simulator should run, and include this information in your documentation. This iteration may be included in the Project Demonstration, and will be included in the Final Project Presentation (see below).

Final Project Presentation – Iteration #5

Your team will give an approximately two-hour presentation to a TA of iteration #5. The goal of the presentation is to show us how awesome your team and project are, and to hand in a hard copy and soft copy of your work. Ensure that you have all work products (see below) to hand after the presentation. You will hand in your hard copy at the end of your presentation, as well as submitting your soft copy via cuLearn.

Final Project Presentation Work Products:

You must bring **hard** copy of **all** of the following:

- Team number and team members
- Table of contents
- Breakdown of responsibilities of each team member for each iteration
- All diagrams
- Detailed set up and test instructions

And **soft** copy of all of the above plus:

- Code (.java files, all required Eclipse files, etc.)
- Test files for all iterations

Grading and Milestones:

The due dates are given in cuLearn. See "Term Calendar" under "General Course Information".

Work products for each iteration are to be submitted via cuLearn as per the instructions under the "Project" tab by the date/time specified.

The project is worth 25% of your final grade. Iterations #1 through #4, and the demo are worth two marks each (total of 10 marks). 10 project marks are for the final deliverables. These 20 marks are for the team. However, based on the participation of each member, each individual's mark could be higher, the same, or lower than the team mark. The final five marks are assigned to each individual for their participation in the weekly meetings (see below). As there was a "bonus" IT0 meeting, you may earn up to 6 out of 5 for the meetings. In addition, any student who does not complete a project peer assessment will have a deduction of 1 mark from their individual project mark.