Carleton University
Department of Systems and Computer Engineering
SYSC 3303 - Real-Time Concurrent Systems - Winter 2019

Assignment 1

Background

The goal of this assignment is to build a very basic three part system consisting of a client, an intermediate host, and a server. The client sends requests to the intermediate host, which sends them on to the server. The server sends responses to the intermediate host, which sends them on to the client. From the client's point of view, the intermediate host appears to be the server. From the server's point of view, the intermediate host appears to be the client. In this assignment, the intermediate host will not change the packets, it will just send them on. The intermediate host could be updated to change packets and thus become an error simulator for the system.

Note that this assignment requires knowledge of the material presented on UDP/IP and Java's DatagramPacket and DatagramSocket classes, as well as conversion between Strings and arrays of bytes. It includes some very basic knowledge of UML Class and Collaboration Diagrams and basic UCMs, but does not require any knowledge of Java threads or TFTP.

Specification

The client algorithm is:

- the client creates a DatagramSocket to use to both send and receive
- repeat the following 11 times:
  - the client creates a DatagramPacket
    - the packet is either a "read request" or a "write request" (alternate between read and write requests, five each) with #11 an invalid request
      - read request format:
        - first two bytes are 0 and 1 (these are binary, not text)
        - then there's a filename converted from a string to bytes (e.g. *test.txt*)
        - then a 0 byte
        - then a mode (*netascii* or *octet*, any mix of cases, e.g. *ocTEt*) converted from a string to bytes
        - finally another 0 byte (and nothing else after that!)
      - write request format:
        - just like a read request, except it starts with 0 2 instead of 0 1
  - the client prints out the information it has put in the packet (print the request both as a String and as bytes)
  - the client sends the packet to a well-known port: 23 on the intermediate host
  - the client waits on its DatagramSocket
  - when it receives a DatagramPacket from the intermediate host, it prints out the information received, including the byte array

The intermediate host algorithm is:

- the host creates a DatagramSocket to use to receive (port 23)
- the host creates a DatagramSocket to use to send and receive
- repeat the following "forever":
  - the host waits to receive a request
  - the host prints out the information it has received (print the request both as a String and as bytes)
  - the host forms a packet to send containing exactly what it received

- the host prints out this information
- the host forms a packet to send containing exactly what it received the host prints out this information
- the host sends this packet on its send/receive socket to port 69 it waits to receive a response
- it prints out the information received
- it forms a packet to send back to the host sending the request
- it creates a DatagramSocket to use to send this request
- it prints out the information being sent
- it sends the request

The server algorithm is:

- the server creates a DatagramSocket to use to receive (port 69)
- repeat the following "forever":
  - the server waits to receive a request
    - the packet should be either a "read request" or a "write request" (see details above)
    - the server should parse the packet to confirm that the format is valid: for purposes of this assignment that means that the packet contains:
      - 0 1 or 0 2
      - some text
      - 0
      - some text
      - 0
      - nothing else after that!
  - the server prints out the information it has received (print the request both as a String and as bytes)
  - if the packet is invalid (as per the above), the server throws an exception and quits
  - if the packet is a valid read request it sends back 0 3 0 1 (exactly four bytes)
  - if the packet is a valid write request it sends back 0 4 0 0 (exactly four bytes)
  - the server prints out the response packet information
  - it then creates a DatagramSocket to use just for this response
  - and sends the packet via the new socket to the port it received the request from
  - and closes the socket it just created

Hints

- The Echo Client-Server example discussed in class and posted on the web site will be useful.
- The APIs for the DatagramSocket, DatagramPacket, and String classes, as well as information on Java arrays which are available through the http://www.oracle.com/technetwork/java/api-141528.html help facility, may also prove useful.
- For this assignment and for the project you need to be able to run multiple main programs (projects) concurrently. Ensure that Eclipse is configured correctly for running multiple programs. See the course reference material for more information.
- Don't get mixed up between text and bytes. For example, the requests start with a 0 byte, not the character 0 converted to a byte.
- Ensure that you follow the specification above. Your datagram sockets (the total number, ports used, send and/or receive) and packet formats must be exactly as described. In the project you will be following a similar specification. Your implementation is considered to be wrong if you do not follow the specification!
- The TAs will mark your assignments in the lab environment. It is your responsibility to ensure that your code works in that environment, and that any software required for viewing any text/diagrams is also present in the lab. You must use the Eclipse Java IDE.

Work Products

1. A "README.txt" file explaining the names of your files, set up instructions, etc.
2. One UML Sequence diagram, showing the client, intermediate host, and server.
3. One or more UML Class diagrams showing your system.
4. The source code for all three parts of the system, as well as any files required to run these files in Eclipse. (You may submit test classes if you wrote them, but be sure to explain the files you have submitted – see #1 above.) Your code should demonstrate good programming style, and be well documented, etc.

For parts 2 and 3, hand-drawn scanned diagrams are acceptable, as long as they are neatly drawn and your handwriting is legible, and the software required to view them is present in the lab. As an alternative, you can use Violet UML.

Submitting Assignments

Assignments are to be submitted electronically using cuLearn. Emailed submissions will not be accepted. See the course outline for the procedure to follow if illness causes you to miss the deadline.

Due: Saturday, January 19th at midnight SHARP!