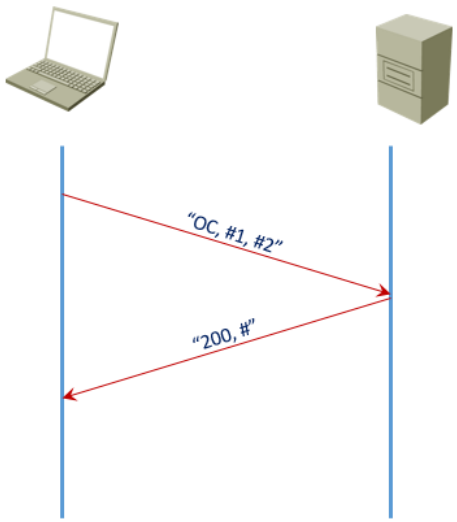
CMPSCI 453: Computer Networking
Spring 2017
Programming Assignment 1

In this assignment you will write a TCP and UDP client server program in which the server acts as a (simple) calculator to act on two numbers sent to it by the client.



Part I: Simple Client Server in a reliable environment

Write this part using both UDP and TCP (i.e. a client/server in UDP and a client/server in TCP).

The Server:

The server performs the operation (OC) requested on the two numbers it receives from the sender and returns the result. More specifically the server

1. Open a socket as a server and
2. listen to the socket
3. Receive a request which consist of an operation code (OC) and two numbers
4. Check the OC and the numbers to make sure they are valid (OC can be one of +,-,*, and /, the numbers should be integers).
5. If the request is not valid, send a return status code of 300 and the result of -1 (to be consistent) and goes to Step 2

   Invalid requests are:

   - Invalid OC (i.e. not +, -, *, or /)
   - Invalid operands
     - Operands not integers
     - Division by zero (0).

6. If the request is valid, perform the operation and return a status code of “200” and the result.
7. Goe to Step 2

The client:
The client sends an Operation Code (OC), and two numbers.
OC can be: Addition (+), Subtraction (-), Multiplication (*), and Division (/)

To make the problem simple, your client sends two integer numbers.

The two numbers and the OC are read from the user via keyboard. Your program
1. Read and display the OC and numbers first; stop if requested.
2. In the case of TCP, open a TCP socket to the server
3. Send them to the server, and wait for the results back
4. Receive the status code and the result from the server.
5. If the status code is OK (200), display the result. And if the status code is anything else (say 300), warn the user of the failure.
6. Go to step 1

To be thorough, you will handle exceptions and if anything goes wrong, to notify the user and halt the program. You can also choose any code to represent the Operation Code. This becomes part of your “Application Layer Protocol”

Part II: Simple Client Server in an unreliable environment, using UDP

In this part you want to simulate errors and how to handle them. When you use TCP, the transport protocol takes care of network unreliability, being dropped packets or erroneous transmission. But when you use UDP, your application should take care of reliability issues. **In this exercise you use UDP to write the recovery code**.

Because you run both client and server on your computer, it is difficult to simulate transmission error. Hence you simulate network/transmission error by randomly dropping (ignoring) packets at the server
The scenario is basically the same as part I, but you simulate dropping packets at the server.

The UDP Server:
The server performs the operation (OC) requested on the two numbers it receives from the sender and returns the result. More specifically the server must
1. Open a socket as a server and
2. listen to the socket
3. Receive a request
4. Randomly (with probability .5) drop the request and go to step 2.
5. Parse the request (which consist of an operation code (OC) and two numbers)
6. Check to OC and the numbers to make sure they are valid (OC can be one of +, -, *, and /, the numbers should be integers).

7. If the request is not valid, it sends a return status code of 300 and the result of -1 (to be consistent) and goes to Step 2
   Invalid requests are:
   - Invalid OC (i.e. not +, -, *, or /)
   - Invalid operands
     - Operands not integers
     - Division by zero (0).
8. If the request is valid, performs the operation and returns a status code of "200" and the result.
9. Goes to Step 2

The UDP client:
The client sends an Operation Code (OC), and two numbers.
OC can be: Addition (+), Subtraction (-), Multiplication (*), and Division (/)
To make the problem simple, your client sends two integer numbers.

There is a possibility that the client does not receive a reply and it repeats sending the request. The client uses a technique known as exponential backoff, where its attempts become less and less frequent. This technique is used in other well-known communication protocols, such as CSMA/CD, the underlying protocol of Ethernet.

The client sends it initial request and waits for certain amount of time (in our case d=0.1 second). If it does not receive a reply, it retransmits the request, but this time waits twice the previous amount, 2*d. It repeats this process, each time waiting for a time equal to twice the length of previous cycle. This process is repeated until the wait time exceed 2 second. At which time, the client sends a warning that the server is "DEAD" and aborts.

The two numbers and the OC are read from the user via keyboard. Your program
1. Read and displays the OC and numbers first,
   a. Set d=0.1
2. Send the request to the server
3. Wait (start a timeout) for d seconds
4. If timeout expires (before a reply comes back),
   a. Set d=2*d
   b. If d>2, raise an exception and stop!
   c. Otherwise go to step 2
5. If receive a reply before timeout, check the status code and the result from the server.
6. If the status code is OK (200), display the result. And if the status code is anything else (say 300), warn the user of the failure.
7. Go to step 1

General:

You should program each process (client and server) to print out informative messages along the way. This helps you to follow your program and debug it and me and the grader to verify that your program is working fine.

Write your code generically, i.e. use 127.0.0.1 as your server address (assuming your client and server are on the same computer) so your code can be used on any computer.

You can use any programming language of your choice, C, C++, Java or Python. You can use any environment of your choice, Unix, Linux, or Windows.

Should you decide to use Java, Edition 5 and below of our textbook has examples of Java Socket Programming. If you need them, please see me to give you a copy of the programs.

A good reference for Python socket programming is
http://docs.python.org/howto/sockets.html

Another good book for Python network programming is
"Foundation of Python Network Programming", 3rd Ed. By Brandon Rhodes and John Goerzen, Apress, 2014.

There are many similar resources available for Java and C

If you use C or C++, you need to submit your executables (for either Linux Windows) as well as source so we can test them.

You can use any reference material that you like, work together (which I encourage), but PLEASE make sure your submission is yours. I count on everyone's honesty. Use this as an opportunity to learn.

Tips:

- Please check Moodle to find out what to hand in.

- You must choose a server port number greater than 1023 (to be safe, choose a server port number larger than 50000).

- I would strongly suggest that everyone begin by writing one client and one server first, i.e., just getting the two of them to interoperate corrections.

- You will need to know your machine's IP address, when one process connects to another. You can use "ipconfig /all" command on Windows or similar commands on Linux/Unix. Alternatively, you can use the name of your server. But remember, the name should be resolvable to an IP address.

- Many of you will be running the clients and senders on the same UNIX/Linux machine (e.g., by starting up the receiver and running it in the background, then starting up the relay in the background, and then starting up the sender. This is fine; since your using sockets for communication these processes can run on the same machine or different machines without modification. Recall the use of the ampersand to start a process in the background. If you need to kill a process after you have started it, you can use the UNIX kill command. Use the UNIX ps command to find the process id of your server

- Make sure you close every socket that you use in your program. If you abort your program, the socket may still hang around and the next time you try and bind a new socket to the port ID you previously used (but never closed), you may get an error. Also, please be aware that port ID's, when bound to sockets, are system-wide values and thus other students may be using the port number you are trying to use.

- I strongly suggest that, once your program is running, use 2 computers so that you can use Wireshark to see the flow, it is fun!

Good luck to you all …. PK