CNT4007C Computer Network Fundamentals, Spring 2016

Programming Assignment 1

mithunpaul@ufl.edu

Date assigned : Friday, Feb 5, 2016
Date due : Wednesday, Feb 19, 2016 (10:40 am EST)

NO LATE submission will be accepted for grading!
How to submit: Please submit via SAKAI following the guidelines

Introduction

In this programming assignment, you will develop a simple client/server implementation for socket communication. It is intended for the students to experience socket programming with proper exception handlings.

The implementation will have a server program and a client program running on two CISE machines. They communicate with each other using socket communication on a TCP/IP protocol suite. The process is like this: After initialization, the server process listens and responds to the clients' requests. A client will try to connect to the server and send out a message containing a command and inputs. The command will be arithmetic calculation commands of adding, subtracting and multiplying and is followed by inputs of one or more numbers (e.g. add 12 7). Upon receiving the command and inputs, the server will respond to the client with the calculating result (e.g. 19). More details will be explained later.

Prerequisite

Socket
A socket is one endpoint of a two-way communication link between two programs running on the network. It is bound to a port number so that the TCP layer can identify the application that data is destined to. The socket API allows application programs to communicate remotely over the Internet. A socket is analogous to a file (an example of transparency) and is used very much like a file. Its send/receive primitives correspond to the write/read operations in a file system. In this project, we will use sockets over TCP/IP (not UDP datagram) in order ensure the reliability of the communication.

More about socket communication can be found from the following links (checking out the books in library and doing some internet search are also encouraged):
[Java]
http://www.oracle.com/technetwork/java/socket-140484.html
http://en.wikipedia.org/wiki/Berkeley_sockets

Specification

Server Program

A server program is an always-running program that accepts the clients' requests. This program starts earlier than all clients and waits for the in-coming connection. After accepting an incoming request from a client, it responds with a message "Hello!" to the client. Then whenever it receives an operation command from the client, it prints out the command on screen and then returns with the calculation result (e.g. for the message "add 4 5" from client, the server will send a message "9" to the client). Maximum number of inputs following the operator should be four, and the server returns an error code for exceeding or insufficient number of inputs. Note that the returning message to the client should only include a number (calculation result or error code. If the server receives a command "bye" from a client, it closes the corresponding socket, but can still be communicating with other clients. However, if the server receives command "terminate" from any client, it closes all the sockets and exit. And all clients exit too.

Client Program

Clients are started on remote machines after a server is running. After connecting to the server and receiving "Hello!" message, client program prints it out and lets the user input a command line. Then it will send the command to the server (clients **do not** handle invalid input) and receive a response. After printing out the result or error message (to be specified later), it will let the user input again. The following are the operation commands used in this project:

add number1 number2 …

subtract number1 number2 …

multiply number1 number2 …

Please note that there should be 2 to 4 input parameters after each operator.

Exception Handling

Your **server** program should be able to handle unexpected inputs. For example, consider a command "add d 7". This operation cannot be done. In this program assignment, the server needs to send an error code instead of the wrong result. Error codes are negative numbers. To avoid confusion, TA will only test your program with **non-negative natural numbers**, and the result should also be non-negative (e.g. we will not test "subtract 3 5"). Therefore, negative numbers are reserved for error code.

Your server program (NOT your client!) needs to generate an error code for incorrect command (e.g. "add 4.2" or "3 2"). The following is the code list:

-1: incorrect operation command.
-2: number of inputs is less than two.
-3: number of inputs is more than four.
-4: one or more of the inputs contain(s) non-number(s).
-5: exit.

When there is more than one error in the message, the error code on the top of the error codes list precedes other error codes. (e.g. "5 4" will have two errors: incorrect operation command & number of inputs is less than two. As the error code -1 precedes the other error codes, the server should return -1). The client program should print out the corresponding error message after receiving an error code.

Termination

Graceful termination is required in this project. If there are runaway processes, points will be deducted. The detailed process is: when user types the command "**bye**" and the client send it to the server, server will reply with error code "-5". Upon receiving "-5", the client process will print out "exit" on screen and exit. But the server program will keep accepting other connection requests. If user types "**terminate**" command and sends to server, server will also reply "-5" but both client and server will exit after that. After termination, there should not be any dangling thread, i.e. no zombie thread showing in the operating system.

Note on Run-away Processes for the Graceful Termination:

Your program should terminate gracefully. While testing your programs, run-away processes might exist. However, these run-away processes should be killed. Please check frequently if there are any remaining processes after termination. CISE department has a policy regarding this issue and your access to the department machines might be restricted if you do not clean these processes properly.

Some useful Linux/Unix commands:

To check your running processes: ps -u <your-username>
To kill a process: kill -9 pid
To kill all Java processes: killall java
To check processes on remote hosts: ssh <host-name> ps -u <your-username>
To clean Java: ssh <host-name> killall java

Execution format

Server:
[Java]
java server [port_number]

Client:
[Java]
java client [serverURL] [port_number]

Examples of compilation and execution (do not use the same port number as shown in here)
[Java]
sand> javac *.java
rain> javac *.java
sand> java server 21234
rain> java client sand.cise.ufl.edu 21234

Program should follow the sample output format. Example:
S> ./server 21234
S> get connection from … (IP)
S> get: add 5 2, return: 7
S> get: multiply 3, return -2

S> get: bye, return -5
C> ./client sand.cise.ufl.edu 21234
C> receive: Hello! (user input: add 5 2)
C> receive: 7 (user input: multiply 3)
C> receive: number of inputs is less than two.
(user input: bye)
C> receive: exit.
Your screen should show results like this, except the user input part.

Port number
You have to be careful when using a port number, i.e., some port numbers are already reserved for special purposes, e.g., 20:FTP, 23:Telnet, 80:HTTP, etc. We suggest you test your program with your last 4-digit of UFID as your port number in order to avoid conflicts with other students. Keep in mind that the port number should be less than 2^{16} (=65,536).

Getting the most points

1. The source codes need to be tested on CISE Linux/Unix machines, because we are going to grade on Linux system. Please state your testing machines and results in your report.

2. Please try to complete the socket communication part first. After testing that, your program will be able to correctly send a message to the server and receive a response, and then you can move on to the next step.

3. Try to finish the simplest operations first (e.g. add 3 4). Exception handling can follow next.

4. Have backups whenever your program can do more. That way, you can avoid the situation where your program used to work but not anymore (e.g. after trying to implement exception handling). If your program cannot be compiled, your will receive zero point. Therefore, it is very important for you to keep the working versions for submission. If your program does not handle exceptions properly, points will be deducted but this is much better than receiving no point.

Reminder

1. For your programming conveniences, your server program **does not** need to be implemented by Thread functionalities (multi-threading) in this assignment. In other words, your server program doesn't need to handle concurrent requests by several clients, i.e., we will test your server program by sequential client requests one at a time. However, you should know that a server program is originally supposed to handle simultaneous requests, which might be asked in later assignments.

2. All the coding must be done individually. Using blocks of code from other people or any other resources is strictly prohibited and is regarded as a violating of UF honor code! Please refer to the UF honor code if you are unfamiliar with it (http://itl.chem.ufl.edu/honor.html).

Report

Submitted report file should be named report.pdf or .txt and include the following:
• Your personal information: Full name, UF ID, and Email
• How to compile and run your code under which environment.
- You will receive zero points if the submitted program cannot be compiled.
• Description of your code structure.
• Show some of the execution results.
- Discuss the results you got with your program.
- Discuss any abnormal results.
• Explain about the bugs, missing items and limitations of the program if there is any.
- Honesty is valued here.
• Any additional comments.

Submission Guidelines:

1. The name of a source code for the server program should be named "server.java" and the client program "client.java".
2. Only submit your Java source code files and report, do not include .obj/.class or executable files in your submission.
3. Include "makefile" if you have one. (Not required)
4. Include the report in .pdf or .txt format. Name it report.pdf/txt.
5. Zip all your files into a packet: Firstname_Lastname_ID.zip
6. Upload the zip packet as attachment in SAKAI before deadline.

Grading Criteria:

Correct Implementation / Outputs	60%
Graceful Termination / Exception handling	20%
Report	15%
Readability / Comments / Code structure	5%
Total:	100%