UNIVERSITY OF DELAWARE
DEPARTMENT OF COMPUTER & INFORMATION SCIENCES

CISC 650 / CPEG 651 / ELEG 651: Computer Networks II

Fall Semester, 2015 Professor: Adarsh Sethi

PROGRAMMING PROJECT 1

This project is an individual assignment and must be done independently by each student. The course late submission policy will apply to this project.

Your task in this project is to design and implement a server which manages a bank account and a client that interacts with the server using TCP to perform transactions on the account. You may use either C or C++ for your programming. You may also use the example client-server programs supplied in the class as a starting point for your programs.

In addition to the project description that follows, there are useful hints available in the accompanying document called “Project 1 Hints”. Be sure to read through that document as well.

Server

Each server maintains one checking account and one savings account both of which will be accessed by a single client. When the server starts, the balance in the accounts is initialized to 0. The client can ask the server to perform any of the following transactions:

- check balance in an account
- deposit a specified amount to an account
- withdraw a specified amount from an account
- transfer a specified amount from one account to the other

In the client-server transactions, you may assume that the balance in the accounts as well as the amounts to be deposited, withdrawn, or transferred are in whole dollars (with 0 cents), and also that these amounts will never exceed one million dollars. The amount to be withdrawn should only be in multiples of 20 dollars, and should not be greater than the current balance in the account. The client may request a withdrawal from any account, but the server should only permit a withdrawal from the checking account. All of the conditions above should be checked by the server, and if any of them are violated, the transaction should not be executed and an error should be returned.

The message from the client to the server will specify the type of transaction requested, the amount (if applicable), and the account on which the transaction is requested. The server’s response contains the type of transaction, an error code if any, the account (savings or checking), the balance before the transaction, and the balance after the transaction (in case of a deposit or withdrawal that was successful). For a transfer, the response should only give information about the account into which the transfer took place. The server must display on its standard output the details of the message received from the client and the response returned. This should include the size (number of bytes) of the message and response.

Client

The client always waits for a response to a transaction request before sending the next request. After the client connects to the server, it may send many transaction requests over the same connection before disconnecting. It may then connect again to send more requests.

The client should prompt the user to enter a transaction type, and the account and amount (if applicable). The client displays on its standard output both the message sent and the response from the server. This should include the size (number of bytes) of the message and response.

Message Formats:

Design the message formats for messages in both directions. Each message will consist of a number of fields, which may depend on the message type. Each field must be of a well-defined data type. The only data types permissible are integers, enumerated integers, characters, and fixed-length strings. Floats and doubles are not allowed. Strings can only be used if they are fixed length. In your submission, describe each of the message types and the message formats with a diagrammatic description of the message format, clearly specifying the fields in the messages, the data types to be carried in the fields, and the lengths in bytes of the fields. This should include a specification of error codes and what they mean.

In your program, be sure to convert numbers between the host and network byte orders before transmission and after reception in either direction. Use only integers to store and represent the account balance and withdrawal/deposit/transfer amounts. The amounts included in the messages only need to specify dollars and no cents.

Submission:

Test the functionality of the client and the server by giving various commands through the user interface. Submit two scripts (using the unix *script* command), one for the client and one for the server. Each script should contain at the very least:

- an initial long listing (*ls -l*) of the files in your directory
- delete the object files/executables
- a listing of the source code
- show the program being compiled
- another long listing of the directory files
- execution of the program with a listing of the output

The test session should include at least two different connections between the client and the server *in the same run* of the programs, and at least three transactions in each connection. Show transactions that exercise the various error cases.

Also prepare a typed description of your message formats, supported by diagrams, including all the information described in the Message Format section above. This description should be stored in a PDF file which should be in the same directory as your programs.

Submit a zipped copy of your project directory. This directory should include all your original source files, the executables, the input and output data files, the scripts generated by you for your test run, and the message format description file. **The file submitted by you must have an extension .zip. No other format is acceptable.**

On the Assignments tab in Sakai, provide the following information as a cover page for your submission: The name of your project directory, the names of the source files, the names of the files that contain the executables, the names of the files containing the scripts with the test runs, and the name of the file that contains the message formats.

Grading:

Your programs will be graded on the following criteria:

Correctness: 60 %
Testing and Proper Output: 20 %
Readability and Documentation: 20 %

Points for documentation will not be awarded lightly; we will be looking for meaningful variable and function names, good use of comments, good modular structure with appropriate use of functions, good programming style, and proper indentation.

Deadline for submission: 11 pm, Tuesday October 20, 2015.

The Late Policy for assignments stated in the Course Policies handout will apply. It is to be understood that computers and computing facilities are subject to failure, overload, unavailability, etc., so it is your responsibility to plan and execute your project work sufficiently in advance so as to meet the due date, and these will not serve as excuses for making exceptions to that policy.