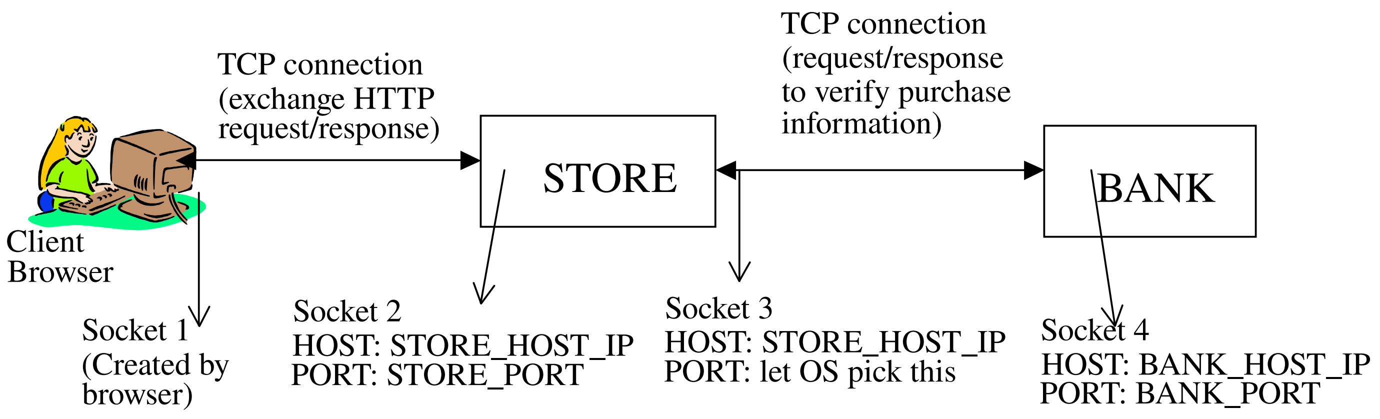
PROGRAMMING ASSIGNMENT 3: Online Purchase System

Course Number: CSE 3101
Course Title: Computer Networking
Submission Deadline: 24th April 2019, 11:59 p.m
Maximum points: 40
START EARLY

Goal: In this assignment your task is to design and implement a simple Online Purchase System, which will consist of an on-line music CD store, and a banking system. A user should be able to visit your music store website using a regular web browser (Firefox). The user can then purchase CDs to his/her liking and pay for them using his/her credit card. You have to design a simple web server for this store and also another independent program that implements the banking system, for verifying the user credit card information before allowing the purchase to be approved.

Specification:
The following illustration will clarify the whole system:You will implement the following two programs:



Bank

- The bank program will accept one argument: BANK_PORT, which is the port number on which it has to open a listening TCP socket. Note that this argument will have to be greater than 1023 (recall that 0 to 1023 are the well-known ports).
- This program is a simple server that verifies if the current transaction should be processed. The server maintains a database, which contains the following information about each user: first name, family name, postcode, credit card number, balance and available credit (in this exact order). The database will be maintained as a simple text file. The following shows an example entry of the database:

First	Family	Postcode	Credit Card	Balance	Credit
Jim	Morrison	2052	12345678	100	1000
Eric	Clapton	2019	53466207	40	500

You have to maintain a simple database file , called “database.txt”. Make sure that your server can read and write into this file. Note that, the format of each entry in the database is exactly similar to the above. The postcode and credit card numbers will be 4 and 8 digit numbers respectively. Also the balance and credit will be specified as whole integers (no fractional part). We will use this database.txt file while marking your assignment.

The bank server will open a listening socket on the specified port (BANK_PORT) and wait for a TCP connection from the store. When a user initiates a purchase, the store program opens a TCP connection with this listening socket. The store program will then send all the user information obtained from the web form (see the description of the Store program for more details about the form) over this connection. There is no fixed format for the messages that will be exchanged between the bank and the store. You may choose any format that you think is suitable for this.

- The bank server then has to verify the transaction and send an appropriate response message to the store. There can be three possible responses:
  (a) The bank has to first check if the first four fields (first and family name, postcode and credit card number) match any of the entries in the database. If no correct match is found for all the four fields, an error indication is sent back to the store. The store should send back an HTML message to the client browser indicating a message along the following lines: "The user information entered is invalid". The wordings of the message could be different but it should convey that the account data did not generate a match.
  (b) If the above check is OK, the bank server next checks if the user has sufficient credit available to pay for the current purchase. If this is not true, i.e. for example the user is requesting to purchase items worth 100$ when he/she only has credit for 50$, then an appropriate error message is sent back to the store. On receiving this error message, the store should send back an appropriate HTML message to the client browser stating that "Your account does not have sufficient credit for the requested transaction" or something along these lines.
  (c) If the user has available credit for the purchase, a message indicating purchase approval is sent back to the store. The store will then send back a "Transaction Approved" HTML message (or something similar) back to the client browser. Note that if the purchase has been approved, the bank program should also modify the available credit and balance for the user accordingly. For example, if the current purchase was worth 20$, then 20$ should be added to the balance and 20$ should be subtracted from the available credit for that user.

  To summarize the response message sent from the bank to store must belong to one of the above 3 categories. The store on receiving this message should inform the user about the status of the transaction (The message sent by the store to the user's browser needs to be in the form of an HTML page, which can be displayed by the browser).
- Once the bank server has sent back one of the above messages to the store, the on-going TCP connection between the bank and the store should be closed. The bank should then open another listening socket and wait for a new TCP connection from the store.
- Note that, the bank program need only handle one TCP connection from the store at any given time (i.e. the bank does not have to deal with multiple connections at the same time). This simplifies your task considerably.
- To terminate the bank server, the user should type CTRL-C at the command prompt to kill the process.

Store

- The store program will accept 3 command-line arguments: (i) STORE_PORT- the port number that the store web server will listen for an incoming connection. (ii) BANK_HOST_IP – the IP address of the host machine for the bank server and (iii) BANK_PORT – the port number at which the bank server will be listening for a new connection. (This must correspond to the BANK_PORT command line argument used with the bank program)
- This program will primarily implement the web server for your Internet CD store. The store server will open a listening TCP socket on port number STORE_PORT and wait for a connection from the user browser. Your server may implement either version 1.0 or 1.1 HTTP (i.e. non-persistent HTTP or persistent HTTP). (Version 1.0 is easier)

- The user will open a web browser (Firefox) and type in the following URL: http://STORE_HOST_IP:STORE_PORT/index.html into the URL field, where STORE_HOST_IP is the IP address of the machine on which the store server is currently running and STORE_PORT is the first command-line argument. This will result in a TCP connection being established with the web server and a GET request for the index.html will be sent to the store server over this connection.
- The default web page (index.html) that you have to use for the web server must be designed by you. This page must be in the same directory, which contains the store program. When we mark your assignment, we will use your index.html page to run all the tests.
- Upon receiving the GET HTTP request (as discussed above), your store server should send back a HTTP response message containing the index.html page to the browser. If you are implementing version 1.1 of HTTP then the connection between the store and bank must be closed following the transfer of this response message.
- The index page provides the user with the information about available CDs, cost, etc. There is also a form included, which allows the user to enter the desired item number, the quantity, first and last name, postcode and credit card information.
- Once this information has been entered, the user will hit the submit button (If implementing HTTP version 1.0, a new TCP connection will be established between the browser and the store server). Following this, the POST method is executed, which will send all the user information to the store server. Your server needs to implement a procedure to process the POST message received from the browser.
- You may assume that all information entered by the user is in the correct format i.e. the name fields will always contain alphabets; item number, quantity, postcode and credit card number are all numbers. You do not have to worry about handling these errors. However, note that the bank still has to verify that all the four fields (first and last name, postcode and credit card number) for the current user match an entry in the database.
- Upon receiving the information about the current transaction, the store needs to verify with the bank if the transaction can be processed. The store establishes a TCP connection with the bank (recall that the bank upon start-up opens a listening socket and awaits a TCP connection request) Note that, the information about the bank socket is available as the second and third command-line argument: BANK_HOST and BANK_PORT.
- After establishing the connection, the store will send all the user information to the bank, which will then verify the authenticity of the user information and also if there is sufficient credit available. The bank will respond with an appropriate response to the store. Recall from the specification of the bank program that there are 3 possible responses. Once the message is received from the bank the connection between the bank and the store should be closed.
- On receiving the bank response, the store web server has to send back a HTTP response message to the user browser containing an HTML page with a simple text message. The specification of the bank discusses this in further detail. The exact text to be included within the HTML page is for you to decide. This text should convey the appropriate information to the customer (e.g: “Transaction Approved”, “Insufficient Credit”, “User Information Invalid”).
- Following this, the TCP connection with the browser should be closed (for both HTTP version 1.0 and 1.1). The store should then open a new listening TCP socket on port number STORE_PORT.
- Your store server should only serve one client (browser) at any given time. You do not have to worry about dealing with multiple clients.
- As explained in the bank specification, you are free to define the message formats of the messages exchanged between the store and the bank.
- To terminate the store server one should type CTRL-C to kill the process.

Programming Notes

The programs will be tested on CSE Linux machines. So please make sure that your entire application runs correctly on these machines. This is especially important if you plan to develop and test the programs on your personal computers (which may possibly use a different OS or version). We will use the Firefox browser during marking. Please ensure that your program is compatible with Firefox. The bank and store programs could be run either on separate hosts or on the same host by choosing different port numbers (there is no real difference between these scenarios from the point of view of writing the code - if your code can handle one scenario it should work correctly for the other as well). See the sequence of operations listed below for details.

Additional Notes

- This is a group assignment. You are expected to work on this with your partner.
- **Where to Start:** The textbook contains code for a simple Web server. Your store program has to implement the functionality of a Web server. Consequently, this code will prove quite useful. There are web resources to the RFC's for HTTP version 1.0 and 1.1. You may browse through these to understand how to handle the POST method.
- Language and Platform: You are free to use either C, C++ or JAVA to implement this assignment. Please choose a language that you are comfortable with. Your assignment will be tested on the Linux Platform. Make sure you develop your code under Linux.

Assignment Submission

- We will inform you about the details of submission before the deadline . You really only need two files: bank.c (or bank.java) and store.c (or store.java). If you are using C and if you are going to use any other files besides these two such as header files or other .c files then you will have to submit a Makefile along with your code. This is because we need to know how to resolve the dependencies among all the files that you have provided. If you are using Java and have multiple files, a makefile will not be necessary (javac *.java should work here). In addition you should submit a small report, report.pdf (no more than 2 pages) describing how you have implemented the communication between the bank and the server and in particular, discuss the format of the messages that are exchanged between them. The report should also include a brief description of how your store program handles the POST command. If your program does not work under any particular circumstances please report this here. Also indicate any segments of code that you have borrowed from the Web or other books.

Late Submission Penalty: Late penalty will be applied as follows:

- 1 day after deadline: 10% reduction
- 2 days after deadline: 20% reduction
- 3 days after deadline: 30% reduction
- 4 days after deadline: 40% reduction
- 5 or more days late: NOT accepted

Plagiarism

- You are to write all of the code for this assignment yourself. All source codes are subject to strict checks for plagiarism, via highly sophisticated plagiarism detection software. These checks may include comparison with available code from Internet sites and assignments from previous semesters. In addition, each submission will be checked against all other submissions of the current semester. Please note that we take this matter quite seriously. The LIC will decide on appropriate penalty for detected cases of plagiarism. The most likely penalty would be to reduce the assignment mark to ZERO. We are aware that a lot of learning takes place in student conversations, and don't wish to discourage those. However, it is important, for both those helping others and those being helped, not to provide/accept any programming language code in writing, as this is apt to be used exactly as is, and lead to plagiarism penalties for both the supplier and the copier of the codes. Write something on a piece of paper, by all means, but tear it up/take it away when the discussion is over. It is OK to borrow bits and pieces of code from sample socket code out on the Web and in books. You MUST however acknowledge the source of any borrowed code. This means providing a reference to a book or a URL when the code appears (as comments). Also indicate in your report the portions of your code that were borrowed. Explain any modifications you have made (if any) to the borrowed code.

Sequence of Operation for Marking

The following shows the sequence of events that will be involved in marking your assignment:

1) We will first run your bank program on a certain machine: BANK_HOST, and provide the BANK_PORT as an argument as shown below:

```
> bank BANK_PORT (for a complied C program)
> java bank BANK_PORT (for a compiled JAVA program)
```

2) The next step involves starting your store server on another machine STORE_HOST_IP, which may or may not be different from the BANK_HOST_IP. The store program accepts three arguments as follows:

```
> store STORE_PORT BANK_HOST_IP BANK_PORT (for C)
> java store STORE_PORT BANK_HOST_IP BANK_PORT (JAVA)
```

3) The user will open a Firefox web browser and type the following url: http://STORE_HOST_IP:STORE_PORT/index.html, The browser should now display your CD Store index page.
4) The user will enter the required information and his/her purchase details and press the submit button.
5) Your store program must check with the bank about the validity of the user data and respond with an appropriate HTML page to the browser. As discussed in the specification there could be 3 possible messages.
6) We will test your program multiple times (by reloading your index page) to check if your program can handle all the 3 combinations of inputs discussed above.

Marking Policy: The marking will be based on the following:

- Test 1: Correct Compilation
  Correct compilation of all files: 1 mark
- Test 2: Design of the web page: 5 marks
- Test 3: Store Web Server Test
  The index.html page is loaded in the web browser when the appropriate store URL is typed in to the URL window of the browser: 7 marks
- Test 4: Customer Interaction Test
  We will test for all the three possible input scenarios:
  1) Incorrect User Data: In this case there will be no match generated for the user in the bank database.
  2) Unavailable Credit: In this scenario the user does not have sufficient credit for the current purchase.
  3) Transaction Approved: The purchase is validated by the bank and the database file is updated accordingly.
  Correct functioning of each test case with an appropriate HTML page being displayed in the client browser (and appropriate modification of the database file in case the purchase is approved): 6 marks for the first two and 8 marks for the last (Total = 20 marks)
- Test 5: Connection Closure Test
  1) The TCP connection between the bank and the store is closed following the request/response exchange: 0.5 mark
  2) The TCP connection between the browser and the store is closed following the interaction between the user and the store. i.e. when the store has sent the final response HTML page to the browser: 0.5 mark
  Test 6: Proper comment on your code: 2 marks
- Test 7: Makefile: 1 mark
- Test 8: Report
  Implementation of the communication between the store server and the bank and handling of the POST command: 3 marks (Note that, we will verify that the description in your report confirms with the actual implementations in the programs)
  IMPORTANT NOTE: We will not read through your code to evaluate your coding style, etc. For assignments that fail to execute all of the above tests, we will be unable to award you a substantial mark.