Project 01

Simple Application Protocol Design
(Implementation Using Socket Programming)

Fall 2023 – CMSC 481

1 Overview

This exercise provides hands-on experience in designing a simple application protocol and implementing the same using socket programming. This task is to be done in a team of two though you can do it individually as well.

1.1 Submission time: 4 weeks.

2 Learning Objectives

- Develop familiarity with Application Protocol design and implementation.
- Develop basic knowledge of TCP socket programming.
- Working as a team and develop communication skills.

3 Learning Resources

1. Kurose, Ross: "Computer Networking: 8th edition – A Top Down Approach", Pearson, Chapter 02.
2. Socket Programming HowTo (Python3): https://docs.python.org/3/howto/sockets.html
3. Netcat: Test and establish TCP/UDP connections. https://www.digitalocean.com/community/tutorials/how-to-use-netcat-to-establish-and-test-tcp-and-udp-connections

4 Prerequisites and environment familiarity

4.1 Familiarization

i. Student should be familiar with Linux and command line terminal usage.
ii. Student should be familiar with tcpdump and wireshark.

4.2 Creating the setup

For this exercise, Ubuntu-20.04 LTS Linux VM, with the IP Address 130.85.121.14, in the CyberRange will be used. Connect using the GlobalProtect VPN and access this VM using your assigned account.

5 Assignment Description

5.1 Simple Protocol Design

Design a simple Application Layer protocol to accomplish some basic service functionality.. Choose your own application task to accomplish. For example, this task could be maintaining your To-Do list (e.g., Date: Task Description), managing your calendar activities (date/time: description), or maintaining your class schedule for the week (day/time: course: room number) etc. The protocol should have 3 phases of communication: For simplicity of debugging and testing, design the protocol in clear text i.e., all keywords, etc. should be in clear ASCII text and capital letters. Implement using your preferred programming language e.g., python3, C, Java, etc., for implementation of the protocol.

i. Phase 1: Identification. The client should be able to identify to the Application server using some identity. For example, after the client establishes a TCP Connection with the server, it should identify itself using some keyword e.g.

    IDENTITIFER: <user identity>

    Choose your own keyword instead of IDENTIFIER.

- If this is not the first message (before you get the session token), the server should reject this message and return ERROR (choose your keyword for returning the error) along with some error code.
- The server should maintain a predefined set of identifiers, e.g., these could be a set of UMBC IDs, phone numbers, or any other values and if the identifier matches among the known values, it returns a success message with a session token, e.g.
    - SUCCESS: <token id>
    - Generate your own token id choose your own keyword for success identifier.

This session token should be used in all subsequent communications.

- (Optional): Add authorization to identification e.g. using password along with Identification.

ii. Phase 2: Perform few (or even none) transactions. These would correspond to retrieving information about your current task or adding/deleting information about your current task. Keywords identifying such messages from clients could be as below. Again, define your own keywords.

a. RETRIEVE : obtain the task information already stored with server
b. ADD: Create new task info and store with server
c. DELETE: Removed the specified task info

Along with the message keyword, you need to define the required syntax and semantics for the messages. The server should respond according to a message received from the client. For example, RETRIEVE should provide details of all (default) or specific requested items stored in the server repository, whereas ADD should create a new task, etc. Make your own decision whether you would like to permit duplicate items.

For any unrecognized keyword, the server should return an error. Define your own error codes or messages.Similarly, requesting a non-existing specific item should return a corresponding error.

Implement a session time out, that is, if for some time (duration chosen by you), there is no communication from the client, the server should simply close the session. Any communication received from the client after time out should result in an error response.

iii. Phase 3: The last phase of this task management is logout. This could be a BYE (or EXIT/LOGOUT) message and the server would respond accordingly.

Note:

- Ensure your programs are robust and neither client nor server should crash for any invalid or unexepcted input.
- The protocol communication should be in clear text (ASCII) thus one should be able to mimic both client and server using Netcat. The use of Netcat (nc) would be a helpful tool to debug the implementation of client and server. Netcat works as both client and server, your client and server should be able to communicate with Netcat as server/client respectively.

5.2 Bonus work

i. Implement concurrency of tasks i.e., multiple clients can communicate with the server at the same time. (can use multithreading/multiprocessing, etc.)
ii. Implement communication over SSL. Thus, all communication should occur in a secure way.
iii. Implementation of client and server in different programming languages. This demonstrate clarity of protocol communication.
iv. Implement the Wireshark plugin for your protocol. This plugin should add the capability to wireshark so as to decode the application layer protocol and show messages as per protocol format. (Explore lua)

6 Assessment

1. A Readme.txt file provides the following:
    a. Team details: UMBC ID, Name of both the team partners.
    b. Brief description of application task and protocol being implemented.
    c. Summary of your learning
    d. Challenges faced in doing the assignment and how did you address it.
2. Approximate duration (in days) spent in completing the assignment.
3. A document (.pdf or Word) providing detailed protocol design and implementation i.e. messages/requests being sent by the client and corresponding responses from the server. Provide format details of both request and response.

4. Design FSM for both client and server and work out timeline sequence diagrams covering various success and error cases. Show timeline sequence diagram fo one success case and at least 2 error cases
5. Both the client and server programs.

7 Evaluation:

i. Total marks for assignment: 40 marks
   a. 10 marks for the protocol design document
   b. 5 marks for FSM design
   c. 5 marks for Timeline Sequence diagrams.
   d. 8 marks each for client and server-side implementation
   e. 4 marks for Summary of learning and challenges faced.

ii. Bonus marks
   a. 5 marks for concurrency i.e. server should support multiple clients concurrently.
   b. 5 marks for SSL-based implementation.
   c. 5 marks of implementing client and server in different programming languages.
   d. 5 marks for the Wireshark plugin/addon. This should show protocol level formatting in wirehsark protocol pane.

8 Note

Any plagiarism activity will result in penalties of being awarded 0 marks.

<end of Project 01>