CSCN72020 – Assignment 3 [Group]

TCP/IP Socket Programming C/C++

Overview

In this assignment you will work in groups of 4 broken into 2 subgroups. A Client group and a Server group. The objective is for you group to develop a very basic discussion board application for multiple clients to collaborate on. The server will allow a single client connection at any time (no multi-threading attempts please) Transmit/Receive data and disconnect allowing for another client to connect. The application layer protocol will be string based and will be of your own design. HINT: Look back at some of the protocols we have talked about to get ideas.

What to Do:

Step #1: Form Groups

This project will be completed in groups of four (4). Two members will be responsible for the Client application and the other two members will be responsible for the Server application. Select your group members and registered for a group number on eConestoga by selecting a group with no registered participants. If you are unable to find a group, one will be assigned to you by the end of the class.

Once you have your group selected, decide who will be doing what on the project. Write the names in the text boxes blow:

Client development will be completed by Saba Berenji and Sierra Erb

Server development will be completed by Hangsihak Sin, Michelle Gordon and Sudhan Dahake

Step #2: Application Layer Protocol

The next step is to define the protocol. Remember, the protocol governs the rules of the link. You need to have a very clear description of the protocol so that both the Client and Server application developers know how to Transmit and Receive data correct.

Your protocol should meet all the requirements specified below and be formatted in a single “string” to/from the applications. Review the requirements as a group of 4, and use the textbox below to define the application layer protocol your collaborative tool will use.

When you are finished designing your protocol, show it to your professor and then submit it to eConestoga as per the instructions below.

Your Application Layer Protocol Design

protocol.h:

```
#pragma once
#include <iostream>
#include <vector>
#include <cstring>
#include <sstream>
#include <memory>

#define PAYLOAD_SIZE 600


enum MessageType
{
    POST_REQUEST = 1,     // client sends a request to post a message
    POST_RESPONSE = 2,    // server responds to the post request by the client
    GET_REQUEST = 3,     // client requests to access posts
    GET_RESPONSE = 4,    // server responds to client with a list of posts
    FILTER_REQUEST = 5,  //client send a request to filter the posts
    FILTER_RESPONSE = 6,  //server responds to client with the filtered posts
    REPLY_REQUEST = 7,    // client sends a request to reply to a post
    REPLY_RESPONSE = 8, // server responds to the reply request by the client
    QUIT_REQUEST = 9     // client is done and wants to close the application

};


struct Data
{
    std::string title;
    std::string author;
    std::string content;
};


struct Packet
{
    MessageType type;
    char payload[PAYLOAD_SIZE];
};

struct PostSubmissionResponse
{
    bool success;
};



std::string serializePosts(const std::vector<Data>& posts);
std::string serializePostSubmissionResponse(const PostSubmissionResponse& response);
std::vector<Data> deserializePosts(const char* payload);
```

protocol.cpp:

```
#include "protocol.h"

// Function to serialize a collection of posts to a custom delimiter-separated string


std::string serializePosts(const std::vector<Data>& posts) {
    std::ostringstream oss;

    for (const auto& post : posts) {
        oss << post.title << ',' << post.author << ',' << post.content << ';';
    }

    std::string result = oss.str();

    // Remove the trailing semicolon if the string is not empty
    if (!result.empty()) {
        result.pop_back();
    }

    return result;
}


// Function to serialize a post submission response to a custom string
std::string serializePostSubmissionResponse(const PostSubmissionResponse& response) {
    return response.success ? "success" : "failure";
}


std::vector<Data> deserializePosts(const char* payload) {
    std::vector<Data> posts;

    std::string payloadString(payload);

    std::istringstream iss(payloadString);
    std::string postString;

    while (std::getline(iss, postString, ';')) {
        std::istringstream postStream(postString);

        std::string title, author, content;


        std::getline(postStream, title, ',');
        std::getline(postStream, author, ',');
        std::getline(postStream, content);


        posts.push_back({ title, author, content });
    }
    return posts;
}
```

Step #3: Implementation & Unit Testing
With your protocol defined above, each subgroup now has the task to design and implement your assigned application (the Client or the Server).

At this point each sub group should work independently. No sharing code, no discussions about how to implement something. Build and test an application and implement the data communications part based on the description of the protocol above. If there is something you are not sure about (as a sub group) make an assumption. Make sure you document any assumptions you make as comments embedded in the source code of your application.

You are responsible for testing and verifying your Client/Server application meets all the requirements. Since you don't have the other end of your application you will need to use Unit Testing and "Stubs" to send data in and out of your application. DO NOT use a copy of the other side of the project. That's cheating.

Step #4: Integration
In Week 13 you will come together an integrate your Client/Server applications together. This will be the first time your two applications will have talked to each other. Let's see if it works. Document your findings in the text box below. Did it work? Did it not work? What failed? What was overlooked? What assumptions were made? How could you have defined your application layer protocol better?

When first integrating the client and the server the application did not run which was due to several problems which you can find a summary of them below:

1. We faced some errors in terms of the file module after integrating the client and the server. Even though the file module was saving the posts sent to the server but it was not in the correct structure. We were getting garbage values and repeated saves in the file. This problem was solved after trouble shooting and changing some functions.

2. There were some assumptions made in terms of the filtering feature of the system. The filtering was first designed to just get a string from the user and send it to the server and this string could have been either for the author or the topic. But after integrating it was discovered that the server side had a different design on how to find the posts based on the user's input. The server was assuming there was an indicator for the input determining whether the filtering is for the author or the topic. After discussing the matter this feature was updated in the client side, a menu was added for filtering as well indicators (as concatenated strings) were added to the user input.

3. Another problem that we experienced when implementing was related to the payload on the client side. Initially, when integrating, the server was not receiving an empty payload from the client. After troubleshooting, we realized, although the client was encoding the posts, the encoded string was not put into the payload of the packet. This issue was fixed with a minor addition of the memcpy function to copy the encoded string to the payload. After this, the server was successfully receiving the posts from the client

4. Another problem we faced when integrating was getting garbage values on the server side when sending the posts from the client to the server. We fixed this issues using a NULL terminator on the client side.

5. There were initially two additional message types named REPLY_REQUEST and REPLY_RESPONSE in the protocol design (protocol.h) which we eventually removed due to the lack of time in implementing them (they were additional features which were not asked in the requirements).

The Requirements

Design, Implement and Test a Client/Server set of applications for a collaborative discussion board using the following requirements:

The Server:

SVR1	The server shall be able to accept a single post from the client
SVR2	The server shall be able to accept a collection of posts from the client
SVR3	The server shall be able to respond to the client with all posts in the discussion
SVR4	A post can be anonymous or contain one or more of the following attributes a) Author b) Topic
SVR5	The server should keep a record of all postings in a file a) Loaded into memory at startup b) Saved/Updated at shutdown
SVR6	The server shall support a single client connection at any given time. **NO THREADS**
SVR7	The server shall transmit and receive packets as a single string item
SVR8	The server shall run on a Virtual Machine with an IP Address of 172.16.5.12/16

The Client:

CLT1	The client shall be a console mode program that is menu driven to exercise all the methods on the server
CLT2	The client shall transmit and receive packets as a single string item
CLT3	The client shall run on a Virtual Machine with an IP Address of 172.16.5.50/16

Implementing only the requirements listed above will result in a maximum grade of 90%. Each of the following advanced features (requirements) can be added to your project (at 5% max each), up to a final grade of 100%.

ADV1	The Server should support multiplexing allowing 2 or more clients to connect simultaneously (**Threads**)
ADV2	The Server/Client should support requesting a set of filtered messages based on Author or Topic (**All filtered messages are in a single transmission**)
ADV4	The client should be designed as a GUI application.
ADV5	The Server/Client discussion board supports Authentication

When you have completed your Client/Server application, record a short 1-2 minute video showing your client server application running. Make sure your video demonstrates all the required functionality and any advanced features you have implemented. Use your microphone for commentary, explaining what you are showing. DO NOT just hand in a video of a mouse moving, clicking, etc

Rubric

See eConestoga for details.

What to Hand In – Protocol Definition

Once you have completed your protocol definition, submit a copy of this PDF with your protocol definition information filled out to the eConestoga Assignment3_Submission link.

What to Hand In – Final Submission

Once you have completed your assignment upload the following files ***INDIVIDUALLY*** to the Assignment3_Submission link on eConestoga:

- An updated (and completed) copy of this PDF form
- ZIP file containing your Client source code (unless it's a single file design)
- ZIP file containing your Server source code (unless it's a single file design)
- Demonstration video.
    - Once screen both virtual machines visible, running your client/server system

ZIP ONLY! Do not compress these files using Tar, 7z, etc… Non ZIP compression will result in a 10% penalty for not following instructions.