BTN415 Lab 5

Introduction to Sockets – UDP/IP Communications

In this lab, you will learn how to create an object-oriented library to handle socket communications via UDP/IP using winsock tools.

LEARNING OUTCOMES

Upon successful completion of this lab, you will have demonstrated the ability to:

- Create a base class to handle UDP/IP sockets
- Create a class to handle UDP/IP client sockets
- Create a class to handle UDP/IP server sockets
- Implement client and server applications to send data back and forward using objects from the implemented classes.

SPECIFICATIONS

In this lab, we will re-write the UDP/IP client and server applications presented in our previous class. In this new version, these applications will make sure of an object-oriented paradigm to shift most common socket functionality to methods defined for three classes of objects as described shortly. These classes should be declared in a file called **oop_udp_winsock.h** and defined in a file called **oop_udp_winsock.cpp**.

Download/Clone the Visual Studio solution from the class Github and perform the following:

Class udp_winsock_server

```
class udp_winsock_server : public udp_winsock {
protected:
        SOCKET server_socket;
        std::ofstream *ofs;

        void Print(std::string msg)
        {
                std::cout << msg << std::endl;
                *ofs << msg << std::endl;
        }

public:
        void bind_socket();
        char * receive_message_from(int, std::string);
        void send_message_to(char *, int, std::string);
        udp_winsock_server(int, std::string, std::ofstream *);
        ~udp_winsock_server();
};
```

For this class, you need to define five methods: **bind_socket()**, **receive_message_from()**, **send_message_to()**, a constructor, **udp_winsock_server()**, and a destructor, **~udp_winsock_server()**, method. These methods are explained in what follows:

bind_socket()

This method is almost identical to the **bind_socket()** method created for TCP sockets. It calls the **bind()** winsock method to bind the server to an IP address (provided by the **this->IP** property) and a port (provided by the **this->port** property). In case the **bind()** method fails, this method should call **WSACleanup()**, warn the user that it could not bind the socket to the specified address, wait for the user to press enter, and then exit the application.

*receive_message_from(char *, int, std::string):*

This method calls the **recvfrom()** winsock method. It takes as arguments the port and IP address of the client who is sending the message in integer and string format, respectively. Then, it saves the received message in **this->rx_buffer**, prints the message using **Print()** and returns it. Note that the **recvfrom()** method should use, as its first argument, this->server_socket. Make sure to create a proper **sockaddr_in** structure to pass it to **recvfrom()**.

*send_message_to(char *, int, std::string):*

This method calls the **sendto()** winsock method. It takes as arguments the message to be send, the port, and the IP address of the client who will receive the message in char *, integer, and string format, respectively. Before sending the message it will be printed using the **Print()** method. Note that the **sendto()** method should use, as its first argument, **this->server_socket**. Make sure to create a proper **sockaddr_in** structure to pass it to **sendto()**.

constructor: udp_winsock_server(int, std::string):

This method takes two arguments, and save them as **this->port**, and **this->ip**, respectively. Following, this method should call the **initialize_udp_socket()** method and save its returned value into **this->server_socket**. Finally, this method should call the **bind_socket()** method.

destructor: ~udp_winsock_server():

This method should simply use the **closesocket()** winsock method to close the **this->server_socket**.

Class udp_winsock_client

```
class udp_winsock_client : public udp_winsock {
protected:
        SOCKET client_socket;
        std::ofstream *ofs;

        void Print(std::string msg)
        {
                std::cout << msg << std::endl;
                *ofs << msg << std::endl;
        }
public:
        char * receive_message_from(int, std::string);
        void send_message_to(char *, int, std::string);
        udp_winsock_client(int, std::string, std::ofstream *);
        ~udp_winsock_client();
};
```

For this base class, you need to define four methods: **receive_message_from()**, **send_message_to()**, a constructor, **udp_winsock_client()**, and a destructor, **~usp_winsock_client()**. These methods are explained in what follows.

> *Hint: Note that most of these methods are very similar to the ones defined for the **udp_winsock_server** class. The major difference is that they use **client_socket**, instead of **server_socket**.*

*receive_message_from(char *, int, std::string):*

This method calls the **recvfrom()** winsock method. It takes as arguments the port and IP address of the client who is sending the message in integer and string format, respectively. Then, it saves the received message in **this->rx_buffer**, prints the message using **Print()** and returns it. Note that the **recvfrom()** method should use, as its first argument, **this->client_socket**. Make sure to create a proper **sockaddr_in** structure to pass it to **recvfrom()**.

*send_message_to(char *, int, std::string):*

This method calls the **sendto()** winsock method. It takes as arguments the message to be send, the port, and the IP address of the client who will receive the message in char *, integer, and string format, respectively. Before sending the message it will be printed using the **Print()** method. Note that the **sendto()** method should use, as its first argument, **this->client_socket**. Make sure to create a proper **sockaddr_in** structure to pass it to **sendto()**.

constructor: udp_winsock_client(int, std::string):

This method takes two arguments, and save them as **this->port**, and **this->ip**, respectively. Following, this method should call the **initialize_udp_socket()** method and save its returned value into **this->client_socket**.

destructor: ~udp_winsock_server():

This method should simply use the **closesocket()** winsock method to close the **this->client_socket**.

Take Home

Change your code so that, instead of initializing the server with “A cool message to be transmitted”, your server should get the contents of a text file. Also, change the **send_packet()** method so that it sends

SUBMISSION REQUIREMENTS

Once you have completed your lab create and upload the following files:

- Create a single ZIP file that contains all your source code files (*.h and *.cpp)
- The output.txt file generated by the lab
- Any additional information you feel necessary for me to mark your lab