Lab 3 - UDP Socket

Part A: *UDPSocket*

In this lab you need to build a *UDPSocket* class that wraps the "C" UDP socket implementation. The new class will implement both client and server socket implementation

The *UDPSocket* interface is specified in the "UDPSocket.h" file and will have the following functionality:

- *UDPSocket(int port)* - initialize a UDP socket and binds it on the given port. if no port is specified a default port 9999 is set. This UDP socket can be used both for server socket and client socket
- *int recv(char* buffer, int length)* - reads an incoming message from the UDP socket. the message is copied into the given buffer up to the specified length.
- *int sendTo(string msg, string ip, int port)* - sends the given message as a UDP message to the given address specified by IP and port.
- *int reply(string msg)* - reply to an incoming message, this method will send the given message as a UDP message to the peer from which the last message was received.
- *void close()* - close the UDP socket.
- *string fromAddr()* - return the sender IP of the last received message.

The *UDPSocket class and the MThread class from the previous class should be compiled into a static library (not an exe).*

In addition to the *UDPSocket* class you need to implement a test project with a main function that tests the new *UDPSocket* class.

To assist you in performing the lab you should use the code provided in "Lab3…".

The zip also contains an exe program of the lab solution so you can view the required behaviour.

Part B: *UDPMessenger*

In this lab you need to build a UDP based messenger that enables the user to do the following:

1. send messages to remote peer by specifying their IP address.
2. receive a message from remote peers.
3. reply to a received message from a remote peer.

In order to achieve that you need to use the *UDPSocket* class you created in part A and create a new class named *UDPMessenger (in a new project)* that implement the required functionality.

The *UDPMessenger* interface is specified in the "UDPMessenger.h" file and will have the following functionality:

- ***UDPMessenger()*** - empty constructor that will start a thread that waits for incoming messages on the socket.
- *void* ***sendTo****(string msg,string ip)* - sends the given message to the given peer specified by IP
- *void* ***reply****(string msg)* - reply to an incoming message, this method will send the given message the peer from which the last message was received.
- *void* ***close()*** - close the messenger and all related objects (socket).

The following method is not part of the interface, its the method that will be executed by the secondary thread that will perform the read from the socket.

- *void** ***receiver****(void* args)* - This method runs in a separate thread, it reads the incoming messages from the socket and prints the content of the messages to the terminal. This method should exit when the socket is closed.

In addition to the *UDPMessenger* class you need to implement a main function that reads the user input and acts accordingly.

To assist you in performing the lab you should use the code provided in "Lab3…".

The zip also contains an exe program of the lab solution so you can view the required behaviour.

Each pair of students need to submit a single zip file of the complete eclipse project.

The submitted file must be named in the following format:
lab3_<student 1 id>_<student 2 id>.zip
All code must be well written and commented and any warning in the code should be avoided.

UDP Programming in C / C++

```
int port = 10456;  //port number
/*System Call Create a new socket (file descriptor)
AF_INET - IPv4
SOCK_DGRAM - type for UDP
0 - UDP Protocol */
int socket_fd = socket (AF_INET, SOCK_DGRAM, 0);
//socket_fd <0 fail
struct sockaddr_in  s_in; //structure containing an internet address. defined in
netinet/in.h.
// clear the s_in struct - you must do that!
bzero((char *) &s_in, sizeof(s_in));

s_in.sin_family = (short)AF_INET;
s_in.sin_addr.s_addr = htonl(INADDR_ANY);    /* WILDCARD any system ip*/
s_in.sin_port = htons(port); //network byte order

//system call binds a socket to an address
if (bind(socket_fd, (struct sockaddr *)&s_in, sizeof(s_in))<0)
     error("Error naming channel");//the most obvious being that this socket is
already in use.

//send a short message
string sMsg = "testing 1 2 3 ...";
int n=sendto(socket_fd,sMsg.data(), sMsg.length(),0,(struct sockaddr
*)&s_in,sizeof(s_in));
if (n < 0)
         error("Sendto");
//Read N bytes into BUF through socket FD
n=recvfrom(socket_fd,buffer,100,0,(struct sockaddr *)&from,&fsize);
/*n - number of bytes read. buffer char array, 100 - maximum 100 char to read,  from
the remote address, fsize - form size in int.*/
if (n < 0)
         error("recvfrom");

//Shut down all or part of the connection open on socket FD, receptions or
transmissions.
shutdown(socket_fd,SHUT_RDWR);
```

```
//close the file descriptor
close(socket_fd);
```