# ComNet 2nd Assignment

Tamara Awadieh
tamara.awadieh@gmail.com
ID : 311166409
Itay Vegh
itay.vegh@gmail.com
ID : 313245110

## ComNet protocol description

The protocol operates over TCP and is constructed with a basic TLV structure.

| **Message structure** | Protocol ID | Type | *Length | Value |
|---|---|---|---|---|
| **Size** | 2 | 1 | 2 | L |
| **Example value** | 0x221E | 0xAB | 0x0014 = L | “This is my message.” |

*Length is the length of the value

# Msg Types:

## hello==>

This message is sent by the server right after a user connected to it.

| **Message structure** | Protocol ID | Type | Length | Value |
|---|---|---|---|---|
| **Size** | 2 | 1 | 2 | L |
| **Example value** | 0x221E | 0x00 | 0xLL | “Welcome! Please log in.” |

## login<==

The user specifies its username and password when trying to log into the system. Both values are sent with a NULL char separating between the two.

| **Message structure** | Protocol ID | Type | Length | Value |
|---|---|---|---|---|
| **Size** | 2 | 1 | 2 | L |
| **Example value** | 0x221E | 0x01 | 0xLL | “myname””mypass” |

We check that the user is inserting the details in the specified format, In case the username and password don’t match or the format doesn’t match the correct one we print “Authentication failed” and give a chance to insert username and password again.

## status==>

The server can accept the login attempt for authorized users, and responds with the following message.

| **Message structure** | Protocol ID | Type | Length | Value |
|---|---|---|---|---|
| **Size** | 2 | 1 | 2 | L |
| **Example value** | 0x221E | 0x02 | 0xLL | “Hi Bob, you have 8 files stored.” |

## list_of_files <==

| **Message structure** | Protocol ID | Type | Length | Value |
|---|---|---|---|---|
| **Size** | 2 | 1 | 2 | L |
| **Example value** | 0x221E | 0x03 | 0x00 | EMPTY Value |

## list_of_files_res ==>

| **Message structure** | Protocol ID | Type | Length | Value |
|---|---|---|---|---|
| **Size** | 2 | 1 | 2 | L |
| **Example value** | 0x221E | 0x04 | 0xLL | File names separated by \n |

In case user doesn't have any files, nothing is printed.

## delete_file <==

| **Message structure** | Protocol ID | Type | Length | Value |
|---|---|---|---|---|
| **Size** | 2 | 1 | 2 | L |
| **Example value** | 0x221E | 0x05 | 0xLL | "myfile.txt" |

In case of trying to delete a file that doesn't exist we print an error message "No such file exists"

## success ==>

| **Message structure** | Protocol ID | Type | Length | Value |
|---|---|---|---|---|
| **Size** | 2 | 1 | 2 | L |
| **Example value** | 0x221E | 0x06 | 0xLL | "File was written." |

## failure ==>

| **Message structure** | Protocol ID | Type | Length | Value |
|---|---|---|---|---|
| **Size** | 2 | 1 | 2 | L |
| **Example value** | 0x221E | 0x07 | 0xLL | “File was not written.” |

## transfer_file ⇐>

| **Message structure** | Protocol ID | Type | Length | Value |
|---|---|---|---|---|

This message is sent by the user and by the server when a file has to be transferred, when get_file is called the server transfers the file, when add_file is called the client transfers the file. The value of this message is the name of the file, followed by the file contents.

| **Size** | 2 | 1 | 2 | L |
|---|---|---|---|---|
| **Example value** | 0x221E | 0x08 | 0xLL | “Myfile.txt” [File Content] |

In case of add_file for a file that already exists we overwrite the existing file.
In case of get_file for a file that doesn’t exist we print an error message “Error getting file”

## get_file <==

| **Message structure** | Protocol ID | Type | Length | Value |
|---|---|---|---|---|
| **Size** | 2 | 1 | 2 | L |
| **Example value** | 0x221E | 0x09 | 0xLL | “myfile.txt” |

## quit

| **Message structure** | Protocol ID | Type | Length | Value |
|---|---|---|---|---|

User disconnects from server by sending this message.

| **Size** | 2 | 1 | 2 | L |
|---|---|---|---|---|
| **Example value** | 0x221E | 0x0a | 0xLL | “” |

## sendMsg

| **Message structure** | Protocol ID | Type | Length | Value |
|---|---|---|---|---|

User can send simple messages to other users.

| **Size** | 2 | 1 | 2 | L |
|---|---|---|---|---|
| **Example value** | 0x221E | 0x0b | 0xLL | “Recipient””DATA” |

## readMsg

| **Message structure** | Protocol ID | Type | Length | Value |
|---|---|---|---|---|

User disconnects from server by sending this message.

| **Size** | 2 | 1 | 2 | L |
|---|---|---|---|---|
| **Example value** | 0x221E | 0x0c | 0xLL | “” |

## usersOnlineReqMsg

| **Message structure** | Protocol ID | Type | Length | Value |
|---|---|---|---|---|

User disconnects from server by sending this message.

| **Size** | 2 | 1 | 2 | L |
|---|---|---|---|---|
| **Example value** | 0x221E | 0x0d | 0xLL | “” |

## usersOnlineResMsg

| **Message structure** | Protocol ID | Type | Length | Value |
|---|---|---|---|---|

User disconnects from server by sending this message.

| **Size** | 2 | 1 | 2 | L |
|---|---|---|---|---|
| **Example value** | 0x221E | 0x0e | 0xLL | “Online Users: ...” |

### Program structure description:

User functionality is implemented in user.c, as well as server functionality is implemented in server.c, a shared aux.h file that contains shared functions which both of the server and the client use and include.
The aux.h file contains the message structure and an enum of the message types, a function that creates the message structure which we defined from a string, a function that sends a message, a function that receives a message and a function that prints a message.
The user.c file has a function which initialises the connection with the server, a parser function gets the needed strings and calls the appropriate function which is implemented in user.c as well. The main function calls the initializer, gets username and password and loops in a while loop to get continuous input until quit is inserted, when data from socket is received (another user send a message) it is displayed immediately .
The server.c is divided into functions according to the functionality that it has to serve the client, it includes a control loop which parses the requests which are received from the client and deals with them.

### Running the program:

First the server should run only then the user(s) can run and connect.