Exercise Sheet 2

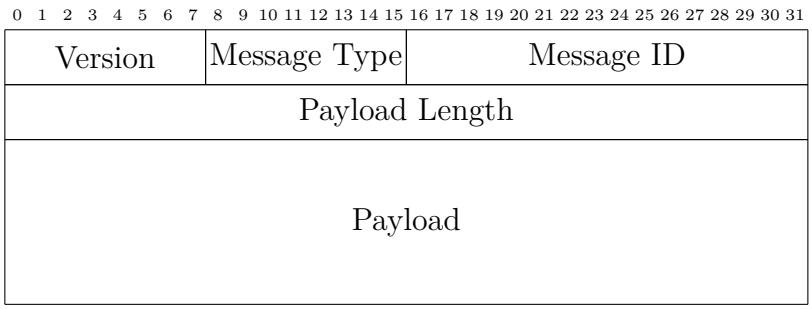
Deadline: May 31, 2022 – 04:00 am CEST

Exercise 1 (UDP chat server and client)

In this exercise you will develop a chat server and the corresponding client application using UDP as transport layer protocol.

Protocol Format

All messages follow the following protocol format:



All protocol fields are sent in **network byte order**. The `Version` field of client and server *shall* match. If they do not match the receiver *may* discard the packet and send a `NAK`. The `Message Type` can contain the following values:

- `0x01 - LOGIN`
- `0x02 - MESSAGE`
- `0x03 - ACK`
- `0x04 - NAK`
- `0x05 - LOGOUT`

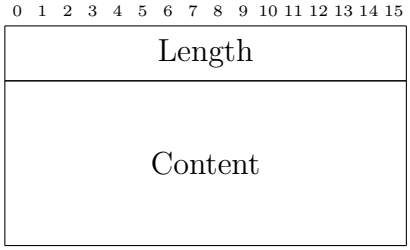
The `Message ID` of a response *must* be identical to the `Message ID` of the corresponding request. Requests may pick a random number for the `Message ID`. The field `Payload Length` specifies the entire size of the following `Payload`. The content of `Payload` depends on the used `Message Type`:

LOGIN and LOGOUT: The payload contains a string to identify the client as `CLIENT_ID`.

MESSAGE: The payload may contain up to two strings where the first one contains the receiver `CLIENT_ID` and the second optionally contains the message to be sent.

ACK and NAK: The payload must be empty.

Each string is represented by its `Length` and `Content`:



Protocol Flow

Messages of type `LOGIN` and `LOGOUT` *must not* be accepted by a client. Messages of type `ACK` and `NAK` *must* be discarded if no corresponding request using the same `MESSAGE ID` has been sent before. All messages of type `LOGIN`, `LOGOUT`, and `MESSAGE` *must* be answered with a message of type `ACK` or `NAK`. All erroneous are answered with a message of type `NAK`, all valid messages return an `ACK` message. If a server receives a message of type `MESSAGE` which contains two strings, it *must* store the content of the messages.

Implementation hint: You can store the messages in a file with the `CLIENT_ID` as filename. The function `getline()` can be used to read from a file line by line. This function can also be used to read a line from `stdin`. Alternatively, you can write messages bytewise into the file using `write`.

If a server receives a message of type `MESSAGE` which contains only one string, it *must* check whether a message for the requesting client has been stored and send this message back to the requesting client as message type `MESSAGE`. Messages *may* be delivered in a reversed order. The server *must* discard all messages of type `MESSAGE` if the given `CLIENT_ID` is not currently logged in (i.e., send a message of type `LOGIN` before, after the last `LOGOUT` message). The server *may* remove existing messages for the given `CLIENT_ID` upon reception of a `LOGOUT` message. A client *must* discard any unsolicited message of type `MESSAGE`.

1. Implement a server application which is expects the UDP port to listen on as command line parameter, e.g., `./udpchatserver <PORT>`.

2. Implement a client application which expects three command line parameters: the address of the server, the port of the server, and a command. The command can be either *login*, *logout*, *send*, or *receive*.

   - If the client application gets called with *login* or *logout* it shall request the `CLIENT ID` from the user via `stdin`.

   - If the client application gets called with *send* it shall request the `CLIENT ID` from the user via `stdin` first and the message itself next.

   - If the client application gets called with *receive* it shall send the message to server immediately.