UNIVERSIDADE DE ÉVORA
ESCOLA DE CIÊNCIAS E TECNOLOGIA
Departamento de Informática

`Computer Networks`

Final Project (v.1) - 2024/2025

1 Introduction

The aim is to develop a real-time communication system that allows company employees to quickly exchange information.

Unlike typical "chat" programs, this system does not operate based on the concept of a "message" (using a "send" button to dispatch a pre-written message), but rather on a letter-by-letter system (each "keypress" is sent immediately). Thus, users receive the messages as they are being typed!

Because the system must function in real time, we opt for an implementation based on the UDP protocol; however, it is necessary to define an application-level protocol so that the data exchange occurs seamlessly.

1.1 Work Teams

- The project must be carried out in teams of 4 students, with the suggestion that they work together in defining the protocol and then split into two groups, one implementing the client and the other the server.
- The teams must be registered on Moodle, and a unique identification number will be subsequently assigned to each team.

2 Base Project (8 points)

2.1 Protocol

The protocol must specify how the interaction between the server and clients is processed, describing all types of requests, responses, and other actions to be executed by both the clients and the server, ensuring that communication can occur without issues.

It should also specify the appropriate actions for the server when unforeseen situations occur (e.g., when "rogue" or badly implemented clients appear, etc.)[1]

2.2 Server and Client

The server accepts UDP datagrams on port 12345.

Each client sends messages to the server, which can be of various types:

- Authentication (extra 3.1);
- Warnings (if implemented);

[1]For inspiration, one may refer to RFC 1459 [`https://www.rfc-editor.org/rfc/rfc1459.html`] which defines the IRC protocol.

- Characters (letters) of the messages;
- Others...

In most cases, there should be no response from the server to client messages, except when strictly necessary (e.g., when the client fails to properly authenticate, when the client specifically requests a response, etc.).

In the base implementation, the following functionalities must be implemented:

1. The server must be able to handle multiple clients simultaneously.
2. Clients can send messages (text) to the server, which forwards them to all other clients.
3. Clients can receive messages from the server, which may correspond to messages (text) from other clients or other types of messages from the server.
4. The server can send messages to the clients indicating that a new client has "joined" or "left" the service.
5. The client must send a "farewell" message when leaving the service.
6. The server must detect when a client has "left" the service without sending the farewell message.

3 Extras

3.1 Authentication (3 points)

A basic authentication system must be created using `username` and `password`, so that clients are uniquely identified.

Thus, messages can be directed to specific users by their username, as well as being marked with the sender's username. This implies that clients can indicate to whom the message is addressed (whether to a specific user or to everyone) and that the server handles them appropriately, sending them to the correct recipient(s).

Usernames and passwords must be stored in a persistent structure (e.g., in a file) so that they are not lost when the server restarts.

3.2 Error Control (3 points)

Since we are using UDP, it is possible that some messages may be lost in the network. Therefore, there must be a way to periodically verify the number of messages sent since the last "checkpoint", allowing any errors that may have occurred to be corrected.

3.3 File Transfer (4 points)

It should be possible for a system user to initiate a file transfer process to one or more users. This process works as follows:

1. The client notifies the recipients that a file is about to be sent;
2. The client opens a TCP server on a port;
3. Recipients who agree to receive the file connect (`connect()`) to that port and receive the file;
4. After a specified period (defined in the protocol), the process ends and the file is no longer available.

3.4 Groups (3 points)

Similar to modern chat applications, it should be possible for users to create groups in which they add other users, in order to send messages to the group (all group members receive the messages).

Groups must have a name (functioning like a username) and must persist if the server is restarted. It should also be possible to remove members from a group and to delete groups (only the group creator can delete a group).

4 Problems, Choices, and Omissions in the Statement

It is normal that issues arise when developing the protocol for this system. Whenever there is no "official" recommendation on how to resolve a problem, it will be up to the team to propose a compromise solution for that issue, and this solution will be part of the final evaluation.

An example is the handling of simultaneous messages from multiple users and their treatment by the client (it is not very useful to display letters from various messages from multiple clients "mixed" together in the output…). Should we "block" clients when another is "speaking", or use the "walkie-talkie" technique, where the speaker indicates the end by saying "over"?

It is up to each team to resolve these types of situations efficiently and effectively.

5 Submissions

The project is evaluated through a final submission. However, there will be two intermediate submissions ("milestones") during practical classes, where teams must demonstrate their progress, clarify doubts, and receive useful guidance on how to proceed. Although these do not count towards the final evaluation, missing the *milestones* presentations will result in a 20% deduction (per absence) from the final project grade.

The final submission will be subject to presentation/discussion and each team member will be evaluated on their individual performance.

5.1 Milestone 1: Protocol

At this stage, teams must present the protocol they have developed for their application. It is advisable that the protocol covers all parts (base + extras), even if the extras are not implemented.

Deadline: **week of April 7 to 11**.

5.2 Milestone 2: Base Project

At this stage, teams must present a functional prototype that implements the base part of the project (client + server).

Deadline: **week of May 5 to 9**.

5.3 Final Submission: Base Project + Extras

Here, teams must deliver the final project, accompanied by an implementation report presenting the protocol, the implementation choices, and a critical and objective evaluation of the application's performance.

Deadline: **June 9**.

5.3.1 Final Submission Format

The final submission must be made via Moodle, in the designated area, through a file named `teamTT.tar.gz` or `teamTT.zip`, where `TT` is the number assigned to the team. Upon extraction, the project should generate a directory `teamTT/` which in turn contains 3 directories:

```
teamTT/
 ├─ client    # contains the client code
 ├─ report    # contains the report in .txt or .pdf format (never .doc or .docx!)
 └─ server    # contains the server code
```

Good luck!