Objective

The objective of this session is to carry out an extension on sockets, a communication mechanism between processes that may be running on the same or different systems.

Motivation

More specifically, with this session, the student will exercise:

- Creation of sockets (socket)
- Association (bind)
- Receiving connections (listen)
- Accepting connections (accept)
- Closing the connection (close)
- Monitoring connections (select)
- Passing parameters to an executable (argc, argv)

Previous documentation

To complete this session, it is recommended to read the following references:

SALVADOR J., CANALETA X. (2014). *Programació en UNIX per a pràctiques de Sistemes Operatius*, Publicacions d'Enginyeria i Arquitectura La Salle (Edició PDF). Capítulo 7. Pág. 97.

Dictionary Server

In the Operating Systems team, we value knowledge and continuous learning. Therefore, in this session, we propose **creating a dictionary server** that allows multiple clients to connect and query word definitions, add new definitions, and list all available words.

For this session, the client is already implemented, so you only need to implement the server. The server will receive three arguments:

1. The IP on which you want to set up this server (e.g., localhost).
2. The port where the server is to be deployed (provided in the list of ports for the practice).
3. The name of the file where the dictionary data will be found. This file will have the following format:

```
<num_words>\n
<word>:<definition>\n
…
```

The server must stay updated with the data of new words added by users.

Client Interaction

A client will be able to perform a total of 3 actions:

1. Query the definition of a word:

To request the definition of a word, the client will send the following frame: `C*<word>\n`

The server will search for the requested word and respond with one of the following two options:

a) **If the word is NOT found**, it should respond with the following frame: `"E* The word <word> has not been found in the dictionary.\n"`

b) **If the word is found**, it should respond with the following frame: `D*<word>*<definition>\n`

2. Add a new word and its definition:

To add a new word to the dictionary, the client will send the following frame:

`A*<word>*<definition>\n`

The server must add the new word to the dictionary and update the file.

a) If the word already exists, it should respond with: *"E* The word <word> is already in the dictionary.\n"*
b) If the word was added successfully, the server can confirm the operation with:
"OK The word <word> has been added to the dictionary.\n"*

3. List all available words:

The client will send the following frame:

L\n*

The server should respond with a single frame containing all the available words, separated by line breaks:

L<num_words>*<word1>\n<word2>\n<word3>\n...*

Execution Examples:

Client

```
alumne@matagalls:~/SO/Practiques20242025/ICE/S7$ ./client.exe
Error: Invalid number of arguments
 Usage: ./client.exe <server_ip> <server_port>
alumne@matagalls:~/SO/Practiques20242025/ICE/S7$ ./client.exe 127.0.0.1
Error: Invalid number of arguments
 Usage: ./client.exe <server_ip> <server_port>
alumne@matagalls:~/SO/Practiques20242025/ICE/S7$ ./client.exe 127.0.0.1 3000
```

Image that reflects how the client should be executed

```
Welcome to the client of Dictionary Server
Please enter your username: Ferran
```

Entry to the program where the user is asked for his name

```
1. Search a word
2. Add a word
3. List Words
4. Exit
Choose an option:
```

Image showing the main menu of the program

```
Enter the word you want to search: hola
The word hola has not been found in the dictionary.
```

Image showing what should happen if we press option 1 and search for a word that does not exist

```
Enter the word you want to search: queue
Word: queue
Definition: A data structure in which elements are added at one end and removed from the other (FIFO).
```

Image showing what happens if we press option 1 in the menu and search for a word that does exist

```
List of words
1: framework
2: function     3: hash
4: hostname     5: JSON
6: kernel       7: latency
8: library      9: object
10: queue       11: token
12: variable
```

Image showing what is displayed on the screen when pressing option 3 of the menu

```
Enter the word you want to add: test
Enter the definition of the word: hola
The word test is already in the dictionary.
```

Image showing us how we can add a word with its definition using option 2 of the menu

```
List of words
1: framework
2: function     3: hash
4: hostname     5: JSON
6: kernel       7: latency
8: library      9: object
10: queue       11: token
12: variable    13: test
```

```
Enter the word you want to search: test
Word: test
Definition: hola
```

Images illustrating how the word test has been added correctly

Server

```
alumne@matagalls:~/SO/Practiques20242025/ICE/S7$ ./S7.exe
ERROR: Number of args is not 2. Is necessary to provide the IP, the port and the filenime
alumne@matagalls:~/SO/Practiques20242025/ICE/S7$ ./S7.exe 0.0.0.0
ERROR: Number of args is not 2. Is necessary to provide the IP, the port and the filenime
alumne@matagalls:~/SO/Practiques20242025/ICE/S7$ ./S7.exe 0.0.0.0 3000
ERROR: Number of args is not 2. Is necessary to provide the IP, the port and the filenime
alumne@matagalls:~/SO/Practiques20242025/ICE/S7$ ./S7.exe 0.0.0.0 3000 dictionary.txt
```

Image reflecting how the server should be run

```
Dictionary server started
Opening connections...
Waiting for connection...

New user connected: Ferran
User Ferran has requested search word command
Searching for word -> queue
User Ferran has requested add word command
User Ferran has requested list words command
User Ferran has requested add word command
User Ferran has requested search word command
Searching for word -> hola
User Ferran has requested add word command
User Ferran has requested list words command
User Ferran has requested search word command
Searching for word -> test2
New exit petition: Ferran has left the server
```

Image illustrating the logs that should be displayed on the server

Considerations

- Client monitoring must be done with select()
- All shared resources must be protected
- It can be assumed that the server program will be executed before the client program.
- It can be assumed that the client will always respect the frames and that the input file format will not contain errors.
- The use of “system” or “popen” functions or similar from the same family is not allowed.
- All input and output must be done with file descriptors; the use of printf, scanf, FILE*, getchar, or similar is not allowed.
- All resources must be freed, otherwise the practice will not be considered unsuitable.
- Connections must be closed as well as possible, also considering unexpected closures. Hanging connections will not be accepted.
- The output of the program must be identical to that of the statement (see examples of execution).
- A single S7.c file must be delivered with the names and logins of the group members commented. Otherwise, the session will not be corrected.
- It must be compiled using the flags –Wall, –Wextra and -lpthread.
- Any session containing warnings or errors will be unsuitable.