Redes de Computadores

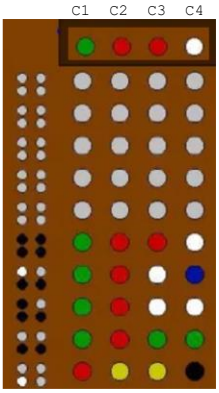
1st Semestre, P2, 2024/2025
LEIC Alameda

Programming using the *Sockets* interface

"RC Master Mind"

1. Introduction

The goal of this project is to develop a simple version of the *Master Mind* game.



The development of the project requires implementing a *Game Server* (*GS*) and a *Player Application* (*Player*). The *GS* and multiple *Player* application instances are intended to operate simultaneously on different machines connected to the Internet.
The *GS* will be running on a machine with known IP address and ports.

The interface with the user of the *Player* application supports a set of actions to control the actions to take:

- Start new game. The *Player* application starts a new game and indicates the *max_playtime* value, in seconds, in which the player proposes to complete the game. Each player is identified by a *player ID* `PLID`, a 6-digit IST student number. When the *GS* receives a request to start a new game it checks if this player already has any ongoing game (each player can only play one game at any given instant). If a new game can be started the *GS* generates a 4-colour secret key to be guessed by the player: `C1 C2 C3 C4`. The valid colours are: {red (`R`), green (`G`), blue (`B`), yellow (`Y`), orange (`O`) and purple (`P`)}. Colours can be repeated in the key. The *Player* application is informed that it can start playing.
- Try. For an ongoing game the *Player* application sends the *GS* server a trial to guess the 4-colour secret key. The *GS* replies with two numbers, `nB nW`, where `nB` (number of black) indicates the number of `Ci` guesses that are correct in both colour and position, while `nW` (number of white) indicates the number of `Ci` guesses that belong to the secret key but are incorrectly positioned.
  If the reply is `4 0` (4 black, 0 white) the secret code has been correctly guessed and the player wins the game.
  After the maximum number of trials (8), or if the maximum playtime (*max_playtime* seconds) is reached, the player loses the game. The number of trials is increased with every play.

- Show trials. The player can request to see the list of previously made trials and the respective results. The *GS* replies with a text file containing the requested list (including for each trial: `C1 C2 C3 C4 – nB nW`) and the remaining playing time.
- View scoreboard. The player can request to see the scoreboard, to which the *GS* replies with a text file containing the top 10 scores (including: *`PLID`*, number of plays to win, secret key).
- Quit. The player can ask to terminate the game at any moment. If a game was under way, the *GS* is informed.
- Exit. The player can ask to exit the *Player* application. If a game was under way, the *GS* is informed.
- Debug. The *Player* starts a new game in debug mode. It indicates the `max_playtime` value, in seconds, to complete the game and specifies the secret key `C1 C2 C3 C4`. When the *GS* receives this request, it checks if this player already has any ongoing game, and if a new game can be started the *GS* uses the secret key provided in the message. The *Player* application is informed that it can start playing.

The implementation uses the application layer protocols operating according to the client-server paradigm, using the transport layer services made available by the socket interface. The applications to develop and the supporting communication protocols are specified in the following.

2. Project Specification

2.1 Player Application (*Player*)

The program implementing the game playing application (*Player*) is invoked using:

`./player [-n GSIP] [-p GSport]`,

where:

GSIP is the IP address of the machine where the game server (*GS*) runs. This is an optional argument. If this argument is omitted, the *GS* should be running on the same machine.

GSport is the well-known port (TCP and UDP) where the *GS* accepts requests. This is an optional argument. If omitted, it assumes the value 58000+GN, where GN is the group number.

Once the *Player* application is running, it can start or end a game, as well as exchange messages related to game playing or visualization of the game history.

The commands supported by the *Player* user interface (using the keyboard for input) are:

- ***start*** *PLID max_playtime* – following this command the *Player* application sends a message to the *GS*, using the UDP protocol, asking to start a new game, provides the player identification *PLID* and indicates the *max_playtime* value, in seconds, in which the player proposes to complete the game (it cannot exceed 600 seconds).
  The *GS* randomly selects a 4 colour key: `C1 C2 C3 C4` and informs the player that it can start playing. The *Player* application displays this information.
- ***try*** *C1 C2 C3 C4* – the *Player* application sends a message to the *GS*, using the UDP protocol, asks to check if `C1 C2 C3 C4` is the secret key to be guessed. If the maximum number of trials has been exceeded the player loses the game, otherwise the number of trials is increased. If the maximum playtime (*max_playtime*) has been reached the player loses the game.
  Otherwise, the *GS* replies informing the number of `Ci` guesses that are correct in both colour and position (`nB`), and the number of `Ci` guesses that belong to the secret key but are incorrectly positioned (`nW`). If `nB = 4` the secret code has been correctly guessed and the player wins the game. The *Player* application displays the received information.
- ***show_trials*** or ***st*** – following this command the *Player* establishes a TCP session with the *GS* and sends a message asking to receive a list of previously made trials and the respective results. In reply, the *GS* sends a text file containing the requested list (including a line for each trial: `C1 C2 C3 C4 nB nW`) and the remaining playing time. After receiving the reply from the *GS*, the list of trials and the corresponding results is displayed by the *Player* application.
- ***scoreboard*** or ***sb*** – following this command the *Player* establishes a TCP session with the *GS* and sends a message asking to receive an updated scoreboard. In reply, the *GS* sends a text file containing the top 10 scores (including for each: *PLID*, number of plays to win, secret key) – the file only contains scores for games where the user won the game and discovered the secret key. After receiving the reply from the *GS*, the scoreboard is displayed as a numbered list.

- ***quit*** – the player can ask to terminate the game at any moment. If a game was under way, the *GS* server should be informed, by sending a message using the UDP protocol.
- ***exit*** – the player asks to exit the *Player* application. If a game was under way, the *GS* server should be informed, by sending a message using the UDP protocol.
- ***debug*** *PLID max_playtime C1 C2 C3 C4* – following this command the *Player* starts a game in debug mode. It sends a message to the *GS*, using the UDP protocol, asking to start a new game, providing the player identification *PLID*, indicating the *max_playtime* value, and specifying the secret key to be used: C1 C2 C3 C4.
  When the *GS* receives this request, it checks if this player already has any ongoing game, and if a new game can be started the *GS* uses the secret key provided in the message. The *Player* application is informed that it can start playing.

Only one messaging command can be issued at a given time.
The result of each interaction with the *GS* should be displayed to the user.

2.3 *Game Server* (*GS*)

The program implementing the Game Server (*GS*) is invoked with the command:

`./GS [-p GSport] [-v],`

where:

GSport is the well-known port where the *GS* server accepts requests, both in UDP and TCP. This is an optional argument. If omitted, it assumes the value 58000+GN, where GN is the number of the group.

The *GS* makes available two server applications, both with well-known port *GSport*, one in UDP, used for playing the game, and the other in TCP, used to transfer the trials text file to the *Player* application.
If the *–v* option is set when invoking the program, it operates in *verbose* mode, meaning that the *GS* outputs to the screen a short description of the received requests (*PLID*, type of request) and the IP and port originating those requests.
Each received request should start being processed once it is received.

3. Communication Protocols Specification

The communication protocols to implement the game playing and list of trials and scoreboard file transfer functionalities are described in this section.
For the communication protocols *PLID* is always sent using 6 digits.

3.1 *Player–GS* Game Protocol (in UDP)

The interaction between the player application (*Player*) and the game server (*GS*) for playing the game is supported by the UDP protocol.
The request and reply protocol messages to consider are:

a) SNG *PLID time*
Following the ***start*** command, the *Player* requests to start a new game and sends the *player ID*, *PLID*, and the maximum play time in seconds, *time*, to the *GS* server. *time* is represented using 3 digits.

b) RSG *status*
In reply to a SNG request the *GS* server checks if player *PLID* has any ongoing game, in which case the reply *status* is NOK. If the game can be started the *status* is OK and the *GS* server randomly selects a 4 colour key, C1 C2 C3 C4, informing the player that it can start playing. The status is ERR, if the syntax of the SNG was incorrect, the PLID is invalid, or the *time* is incorrect (it cannot exceed 600 seconds).

c) TRY *PLID C1 C2 C3 C4 nT*
Following the ***try*** command, the *Player* sends the *GS* a request to check if C1 C2 C3 C4 (separated by single spaces) is the secret key to be guessed. Also the trial number *nT* and the *player ID* *PLID* are sent.

d) RTR *status [nT nB nW][C1 C2 C3 C4]*
In reply to a TRY request the *GS* replies informing whether the secret key guess was successful or not. *status* can take the following values:

- OK, if the trial is valid, in which case the trial number *nT* is returned, along with the number of Ci guesses that are correct in both colour and position (nB), and the number of Ci guesses that belong to the secret key but are incorrectly positioned (nW) are sent. If nB = 4 the secret code has been correctly guessed and the player wins the game. The same reply is provided if the trial number *nT* is the expected value minus 1, and the secret key guess repeats the one of the previous message (it is a resend) – in this case the number of trials is not increased;
- DUP, if the secret key guess repeats a previous trial's guess. In this case the number of trials is not increased;
- INV, if the trial number *nT* is the expected value minus 1, but the secret key guess is different from the one of the previous message, or if the trial number *nT* is not the expected value;
- NOK, if the trial is out of context, like the player *PLID* not having an ongoing game;
- ENT, if no more attempts available. In this case the secret key is revealed, sending the corresponding colour code: *C1 C2 C3 C4*;

- ETM, if the maximum play time has been exceeded. In this case the secret key is revealed, sending the corresponding colour code: *C1 C2 C3 C4*;
- ERR, if the syntax of the TRY message was incorrect, the *PLID* is invalid, or the colour code received *C1 C2 C3 C4* is not valid.

The *Player* application should display this information.

e) QUT *PLID*
Following the **quit** or **exit** commands, and if there is an ongoing game, the *Player* application sends the *GS* a message with the player *PLID* requesting to terminate the game.

f) RQT *status [C1 C2 C3 C4]*
In reply to a QUT request the *GS* server sends a status message. If player *PLID* had an ongoing game, the *GS* replies with *status* = OK, and the secret key is revealed, sending the corresponding colour code: *C1 C2 C3 C4*. If *PLID* did not have an ongoing game, the *GS* replies with *status* = NOK. Otherwise the *status* is ERR.

g) DBG *PLID time C1 C2 C3 C4*
Following the ***debug*** command, the *Player* requests to start a new game, sending the *player ID*, *PLID*, the maximum play time in seconds, *time*, and the secret key to be guessed, *C1 C2 C3 C4*, to the *GS* server.

h) RDB *status*
In reply to a DBG request the *GS* server checks if player *PLID* has any ongoing game, in which case the reply *status* is NOK. If the game can be started the *status* is OK and the *GS* server uses as secret key the C1 C2 C3 C4 values provided by the *Player*. The status is ERR, if the syntax of the DBG message was incorrect, the PLID is invalid, or the colour code received *C1 C2 C3 C4* is not valid, or the time is incorrect.

If an unexpected protocol message is received, the reply is ERR.
In the above messages the separation between any two items consists of a single space.
Each request or reply message ends with the character "\n".

3.2 *Player–GS* Messaging Protocol (in TCP)

The interaction between the player application (*Player*) and the game server (*GS*) for messaging related to the visualization of the scoreboard and of the list of trials made so far is supported by the TCP protocol.
The request and reply protocol messages to consider are:

a) `STR` *`PLID`*
Following the ***`show_trials`*** command, the *Player* application opens a TCP connection with the *GS* and asks to receive a list of previously made trials and the respective results.

b) `RST` *`status [Fname Fsize Fdata]`*
In reply to a `STR` request the *GS* replies depending on whether player *`PLID`* has an ongoing game or not.
If there is an ongoing game, then *`status`* = `ACT` and the *GS* server sends a text file containing the summary game so far, with one line for each trial made and the corresponding result: `C1 C2 C3 C4 nB nW`. A final line includes the remaining playing time in seconds. Upon receiving this summary, the player displays the information and continues with the game.
If there is no ongoing game for this player then *`status`* = `FIN` and the *GS* server responds with a file containing the summary of the most recently finished game for this player. Upon receiving this summary, the player displays the information and considers the current game as terminated.
If the *GS* server finds no games (active or finished) for this player, or if there is some other problem, then *`status`* = `NOK`.
When replying with the game state information, a text file is sent. The reply message then includes:

- the filename *`Fname`*;
- the file size *`Fsize`*, in bytes;
- the contents of the selected file (*`Fdata`*).

When receiving the text file including the game summary a local copy is stored by the *Player* using the filename *`Fname`*. The name and size of the stored file are displayed.
After receiving the reply message, the *Player* closes the TCP connection with the *GS*.

c) `SSB`
Following the ***`scoreboard`*** command, the *Player* application opens a TCP connection with the *GS* and asks to receive the top-10 scoreboard.

d) `RSS` *`status`* `[`*`Fname Fsize Fdata`*`]`
In reply to a `SSB` request the *GS* sends a status response. If the scoreboard is still empty (no game was yet won by any player) the *GS* replies with *`status`* = `EMPTY`, otherwise the *GS* replies with *`status`* = `OK`, and sends a text file containing the top-10 scores of the game. The information sent includes:

- the filename *`Fname`*;
- the file size *`Fsize`*, in bytes;
- the contents of the selected file (*`Fdata`*).

The text file contains one score per line consisting of the player ID *`PLID`*, followed by the **secret key** guessed, and the **total number of plays** needed to

win; these fields are separated by spaces. The file starts with the top score (less plays needed to guess the secret key) and can contain up to 10 lines, each line terminated with a '`\n`'. These scores are displayed by the *Player*.
A local copy of the score board is stored using the filename *`Fname`*.
After receiving the reply message, the *Player* closes the TCP connection with the *GS*.

The filenames *`Fname`*, are limited to a total of 24 alphanumerical characters (plus '-', '_' and '.'), including the separating dot and the 3-letter extension: "*nnn…nnnn.xxx*". The file size *`Fsize`* is limited to 2 kiB (1024 B), being transmitted using a maximum of 4 digits.

If an unexpected protocol message is received, the reply will be `ERR`.
In the above messages the separation between any two items consists of a single space.
Each request or reply message ends with the character "`\n`".

4. Development

4.1 Development and test environment

Make sure your code compiles and executes correctly in the development environment available in the labs *LT4* or *LT5*.

4.2 Programming

The operation of your program, developed in *C* or *C++*, may need to use the following set of system calls:

- Reading user information into the application: `fgets()`;
- Manipulation of strings: `sscanf()`, `sprintf()`;
- UDP client management: `socket()`, `close()`;
- UDP server management: `socket()`, `bind()`, `close()`;
- UDP communication: `sendto()`, `recvfrom()`;
- TCP client management: `socket()`, `connect()`, `close()`;
- TCP server management: `socket()`, `bind()`, `listen()`, `accept()`, `close()`;
- TCP communication: `write()`, `read()`;
- Multiple inputs multiplexing: `select()`.

4.3 Implementation notes

Developed code should be adequately structured and commented.

The `read()` and `write()` system calls may read and write, respectively, a smaller number of bytes than solicited – you need to ensure that your implementation still works correctly.

The client and the server processes should not terminate abruptly in failure situations, such as these:

- wrong protocol messages received by the server: an error message should be returned, as specified by the protocol;
- wrong protocol messages received by the client: the interaction with the server is not continued and the user is informed;
- error conditions from system calls: the programs should not terminate abruptly, avoiding cases of "segmentation fault" or "core dump".

5 Bibliography

- W. Richard Stevens, Unix Network Programming: Networking APIs: Sockets and XTI (Volume 1), 2nd edition, Prentice-Hall PTR, 1998, ISBN 0-13-490012-X, chap. 5.
- D. E. Comer, Computer Networks and Internets, 2nd edition, Prentice Hall, Inc, 1999, ISBN 0-13-084222-2, chap. 24.
- Michael J. Donahoo, Kenneth L. Calvert, TCP/IP Sockets in C: Practical Guide for Programmers, Morgan Kaufmann, ISBN 1558608265, 2000
- On-line manual, `man` command
- Code Complete - http://www.cc2e.com/
- http://developerweb.net/viewforum.php?id=70

6 Project Submission

6.1 Code

The project submission should include the source code of the programs implementing the *Player* and the *GS server*, as well as the corresponding *Makefile* and the *auto-avaliação* excel file.
The *makefile* should compile the code and place the executables in the current directory.

6.2 Auxiliary Files

Together with the project submission you should also include any auxiliary files needed for the project operation together with a *readme.txt* file.

6.3 Submission

The project submission is done by e-mail to the lab teacher, **no later than December 20, 2024, at 23:59 PM**.
You should create a single `zip` archive containing all the source code, makefile, all auxiliary files required for executing the project and the *auto-avaliação* excel file. The archive should be prepared to be opened to the current directory and compiled with the command `make`.
The name of the archive should follow the format: **`proj_`*`group number`*`.zip`**

7 Open Issues

You are encouraged to think about how to extend this protocol in order to make it more generic.