Instituto Superior de Engenharia de Coimbra
Departamento de Engenharia Informática e de Sistemas

Operating Systems 20/21

Project Assignment - C Programming for UNIX

The practical assignment of operating systems consists on the implementation of a game championship management system called CHAMPION. The games involved are very simple and are not the main goal of the assignment. The intended system is in charge of bridging the gap between the players and the games, mediating the messages exchanged between them and managing the championship. Further details are given below.

This assignment must be programmed in C language for the Unix platform (Linux), using the operating system mechanisms covered in the theoretical and practical classes. Regarding the handling of system resources, priority should be given to the use of system calls[1] instead of library functions[2] (for example, *read()* and *write()* should be used instead of *fread()* and *fwrite()*). The assignment focuses on the correct use of the system's mechanisms and resources and there is no particular preference concerning aspects of peripheral nature to the discipline (for example: the question "should I use a linked list or a dynamic array?" will have a *shoulder shrug* as response). The use of libraries that do or hide part of the work is not allowed. The use of system mechanisms that have not been addressed in the classes is allowed, but they must be *very well* explained in the assignment presentation. An approach based on the mere pasting of excerpts from other programs or examples is not permitted. All the code presented must be understood and explained by the person presenting it.

The functionalities are described first, and the operating rules of the game championship management system are explained next.

[1] Documented in section 2 of the manpages
[2] Documented in section 3 of the manpages

1. General description

There are several games and several players. All games are single-player (there is no player *vs* player situation) and all exhibit similar interaction logic. A championship will be launched for a few minutes. During the championship, each player can interact with only one game, being given a score when the championship ends. Different players may possibly be playing different games. The player-game assignment is not controlled by the player, but the player knows which game has been assigned to him. Aspects of the game's difficulty are irrelevant, and if one game is more difficult than another, well, “too bad” for the player to whom it was assigned.

The purpose of the CHAMPION system is to mediate communication between the client applications used by the players and the games themselves, managing the championship and maintaining a scoreboard.

2. Concepts and entities

The work involves the following concepts:

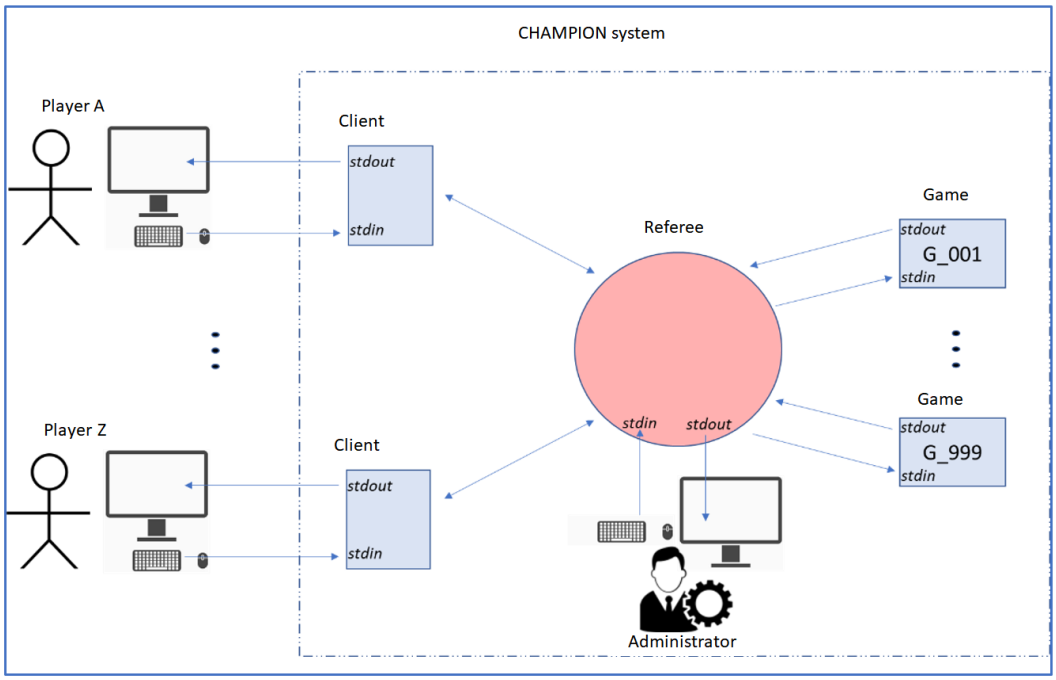
- Game
    - The "game" is of secondary relevance. It implements a game whose specific logic is not important. Examples of possible games:
        - Guess the number that the computer “thought” (ie, generated randomly). If the player gets it right, the game draws a new number and the game continues.
        - Identify the Portuguese translation of a word in another language (word drawn from some sort of list built-into the program);
        - Identify the result of a "complex" arithmetic expression;
        - Etc.
    - The game runs as a sequence of question-answers between the program and the player. So, using the example of the word translation mentioned above, the challenge is presented ("What does" XXXX "mean?"), the answer is received, which could be right or wrong, and then the game moves on to the next question (“What does“ YYYY ”mean?”).
    - The duration of the game is equal to the duration of the championship: there is no specific goal to reach that will end the game. Instead, the game will go on until the championship ends. In the example given above of guessing a number, if the player guesses the number and the championship has not yet ended, the game draws another number and the game continues.
    - There will be a score, specific to each game. In the example of word translation, it could be the number of right translations minus the number of wrong translations achieved during the game period.

- Mandatory aspects:
    - The output of the game (for example, the "challenge question") is sent to the *stdout* and the input (the player's answer) is received via *stdin*.
    - The score obtained is communicated by the game at the end through its *exit status*.
- There will be several games. The executable files that correspond to them have a name that always starts with “g_”. These are located in a directory specified by the environment variable GAMEDIR. If this variable is not defined, the system must use the current directory.

- Championship
    - A "championship" represents a set of simultaneous games (one for each user) that are played during a certain time interval. The championship is open to a maximum number of players at the same time, specified by the MAXPLAYERS environment variable and it is known in advance that it will never exceed 30 players. All the games are *single-player*. The player plays one and only one game during the championship (the game involves several interactions with the player). The player does not choose the game - it is assigned to him by the system *referee* (the referee is a program – the one that manages the championship).

- Client
    - A "client" is the program used by the player to interact with the referee of the CHAMPION system and, indirectly, with the game. The client does not implement any game: it only receives the player's input, forwarding it to the referee, and receives the information from the latter (and, indirectly, from the game), presenting it on the screen. The input received from the client by the referee may be information for the game that the player is playing or for the referee himself. The referee is responsible to distinguish them.

- Referee (the manager of the CHAMPION system)
    - The “Referee” of the CHAMPION system is the entity that is between the client and the games. This referee program is managed by an administrator. The player does not see the referee directly, he only interacts with his client, who communicates with the referee, who in turn communicates with the games.



The CHAMPION system comprises the following users:

- Players (many)
    - They are identified by a name that cannot be repeated. There is no password.
    - The players only exists in the context of the CHAMPION system, and there is no relationship with the users of the operating system.
    - There cannot be two players with the same name in the same championship.

- Administrator
    - Performs some championship management actions. Doesn't play. It has no associated username or password. It is simply the user who interacts with the process that manages the championship (the referee).

The CHAMPION system includes the following programs:

- Client (already described above)
    - Responsible for the player user interface, which follows the console (text) paradigm. The use of menus is not accepted. It won't be necessary to use any graphical interface.
    - Each player runs exactly one client.

- Referee
    - Manages the championship.
    - There is only one Referee process running at the same time and this aspect must be guaranteed by your code.

- The referee functions include
  - Creates the communication mechanisms to:
    - Receive participation requests and messages for games from clients.
    - Receive / send information to games.
  - Stores (in memory) and manages the data of users currently in the championship, including their score.
  - Interacts with the administrator and follow his instructions. This interaction is done according to the console (text) paradigm through written commands (no "menus").
  - Interacts with the games, managing their operation where necessary.
  - Interacts with the clients.
  - Manage the championship, guaranteeing its operational characteristics (duration, players admitted, etc.).

- Games
  - These are the programs described at the beginning of this section. It is mandatory to implement at least one, different from the examples mentioned above, and whose rules freely defined by each group. It is recommended that, at the beginning, all games present a welcome message to the user and a brief summary, sufficient for the player to know the name of the game and understand the rules of the game. Note that you can test the game without the CHAMPION system being completely developed since it uses your stdin/stdout to interact with the outside (i.e., the player).

3. Operational characteristics of a championship

The championship is ready as soon as the Referee is launched. The program receives some configuration data by argument from the command line and analyses the environment variables GAMEDIR and MAXPLAYERS. If something is not in accordance with the assumptions, the program must exit in an orderly manner.

The Referee waits until at least two players are ready. After two players are present, it will wait for a limited time for more players, after which the championship begins.

The start of the championship involves assigning a game (chosen by the referee at random from among the existing ones) to each player. The player is informed of the game assigned to him. The referee is responsible for preparing the execution of the various games and forwarding (intermediating) all information between the player's client and the game assigned to him in that championship. The referee keeps all the information related to the players, updating whenever there are any changes (new player, someone left, a score was received, etc.)

The championship has a maximum duration. After this time: games are warned that they must end with a SIGUSR1 signal; the games communicate the score of “its” player to the referee through the exit status, as already described; the SIGUSR1 signal is also sent to clients to inform them that the championship has ended (this does not mean that client’s processes must end); the referee will inform all clients about their player's score and will also disclose the identity of the winner (i.e., who has scored more points in that championship).

After a championship, the referee keeps running and the cycle is repeated. Players interested in playing again do not need to exit the client, nevertheless they will have to repeat the process of interacting with the referee from the beginning (identify player, etc.).

During the championship it is assumed that the information that is sent from the client to the referee is to be forwarded to the game that is assigned to him. This information is text, according to the logic of interaction with the game as described above. Any information sent by the client that is to be processed by the referee must be preceded by "#". In this situation, the request to obtain the identification of the game in in the form of “#mygame” (used by the user if he has already forgotten, as this information is given to him at the beginning of the championship), and the request to indicate that he gives up championship - "#quit". Following the second request, the game assigned to this player must also end. If the championship has only one active player, it must end immediately, with the last player taking the victory.

Games are assigned to players randomly and there may be two or more players playing the same game (but running in different processes). Games are considered to be any program that exist in the directory specified by the environment variable GAMEDIR and that starts with “g_”. The administrator may interfere with the process of assigning games to players.

The duration of the championship and the waiting time for more players, in addition to the first two, are indicated to the referee by command line arguments when it is launched.

Whenever the referee leaves, all clients must be informed. Resources that are no longer needed should not remain in memory.

4. Forms of use

Client usage

The client must support:

- Obtaining the player's identification.
- Collecting the player's orders, related to the game or others (giving up, other).
- Display data related to the course of the game and information regarding the start and end of a championship, as well as other control information.
- The interaction with the user must be carried out in “text mode” by written commands. What the player must write is dependent on the game's logic. There is no need to manage content formatting (placement, colours, etc.).

Use of the referee

The referee interacts only with the administrator, according to the logic of written commands. It must support the following commands:

- List players in championship (name and assigned game). Command "players".
- List available games. Command "games".
- Remove a player from the championship. The effect is the same as the player giving up, but in this case the client is informed of what happened. Command "k" concatenated with the player's name (e.g. "krui" - removes player "rui").
- Suspend communication between player and game. The messages of this game-player pair are temporarily no longer forwarded between them. The addressed client is informed of what is happening. Command "s" concatenated with the player's name.
- Resume communication between player and game. The messages from this game-player are forwarded again. The client in question is informed. Command "r" concatenated with the player's name.
- End the championship immediately. It has the same effect that occurs when the championship reaches the end of its time. “end” command.
- Exit, ending the referee. “exit” command.

Optionally, other commands can be implemented (no extra credits given, though), for testing and debugging purposes. For example, a command that forces a player to be assigned to a specific game.

5. Implementation restrictions

There is no direct communication between the client and the games.

Only the communication mechanisms that have been addressed in the classes can be used. Pipes and named pipes will play a leading role. The use of regular files as a communication mechanism between processes is not allowed.

The validity of the operations carried out by the programs must be enforced, according to the strategy that is implemented. This question stems from the characteristics of the solution that is chosen and not from something that is said in this document. For example, if at any time the strategy followed needs synchronization mechanisms, then this need must be assured, and its omission will carry grade penalties that cannot be avoided with excuses, such as “the practical assignment document did not directly demand this” or “that issue will rarely happen”.

Data persistence

There is no persistence of information between different executions of the referee, and the same applies to client executions. There is no permanent record of users or scores.

If you want, you can make this information persistent (store in a file), but this is optional and without any bonus.

6. Additional considerations

- All system actions described here take place on the same machine and in the same user account. Different consoles / terminals will be used for each client, plus one for the referee.
- The client must be able to handle multiple tasks simultaneously. The same for the referee. You must take this in consideration.
- There may be potential situations of conflict or error that are not explicitly mentioned in this document. These situations must be identified, and programs must deal with them in a controlled and stable manner. A program ending abruptly when facing one of these situations is not considered an appropriate way to deal with the scenario.
- If inconsistencies are detected in this statement, they will be corrected and published. There may be additional documents in the OS moodle course, namely a Frequently Asked Questions (FAQ) with the most frequently asked questions and their answers.

7. General rules of work, GOALS and IMPORTANT DATES

The following rules apply, described in the first class and in the course unit form (FUC):

- The work must be carried out in groups of two students (groups of three are not admitted and any request to that effect will always be denied or ignored).
- There is a mandatory presentation ("defense"). The presentation will be carried out in a manner to be defined and announced through the usual channels in due time. The presentation will be in person (i.e., not remote) unless the school board says otherwise.
- There are three goals in all, as described in the FUC. The target dates and requirements are shown below. In all goals, delivery is made via moodle by submitting a single zip file[3] whose name follows the pattern[4]:

  so_2021_tp_meta1_name1_number1_name2_number2.zip

  ("meta" mean goal, and, evidently, meta1, name and number will be adapted to the goal, names and numbers of the members of the group)

  Failure to adhere to the indicated file format and/or to the file name causes delays in the evaluation process and will be penalized in the final grade, and could possibly lead to the work not being evaluated.

Goals: requirements and delivery dates

Goal 1: November 15th

Requirements:

- Plan and define the data structures responsible for managing the functionalities of the referee and the client. Define the various header files that include the symbolic constants, the default values, as well as the relevant data structures, both for the client and server.
- Obtain the "championship duration" and the "waiting time", when the referee executes, by reading the command line arguments. Tip: use the `getopt()` and the `getsubopt()` functions.
- Implement a game that follows the requirements specified in this document, except for the signal handling. Something very simple will suffice as long as you follow the logic of the games to be used with the CHAMPION system.

[3] Read "zip" - it is not arj, rar, tar, or others. The use of another format may be penalized. There are many UNIX command line utilities to handle these files (zip, gzip, etc.). Use one.

[4] Failure to comply with the name format causes delays in the management of the submitted assignments and will be penalized.

- Develop the logic of reading the environment variables GAMEDIR and MAXPLAYER by the referee, reflecting their value on the data structures mentioned in the previous point. Tip: use the function `getenv()`.
- Develop and deliver a makefile that must have the compilation targets "all" (compilation of all programs), "client" (compilation of the client program), "referee" (compilation of the server program), "game" (compilation of the game program proposed) and “clean” (elimination of all temporary files that support compilation and all executables).

Delivery date: Sunday, November 15th, 2020. No possibility of late delivery.

Goal 2: December 13th

Requirements:

- Upgrade the game implemented in goal 1 by allowing it to terminate when receiving the SIGUSR1 signal and delivering the game result through the exit status.
- Implement all communication logic between the clients and the referee through the named pipes mechanism.
- Implement the commands supported by the client: `#mygame` and `#quit`.
- Start developing the processing of the referee's administration commands by implementing the reading and validation of the commands and their parameters, including the complete implementation of the commands: `players`, `games`, `k` and `exit`.

Delivery date: Sunday, December 13th, 2020. No possibility of late delivery.

Goal 3:January 24

Requirements:

- All requirements set out in the statement.

Delivery date: Sunday, January 24, 2021. Subject to adjustments if there are changes in the school calendar.

In goals 1 and 2, a two-page document describing the details of the implementation and main options taken should be delivered with the project.

In goal 3, a full report should be delivered with the project. The report will comprise the content that is relevant to justify the work done and the authorship should be exclusive to group members. If, however, a template report is released, then those instructions must be followed.

8. Work evaluation

For the evaluation of the work, the following elements will be taken into account:

- **System architecture** - There are aspects related to the interaction of the various processes that must be carefully planned in order to present an elegant, light and simple solution. The architecture must be well explained in the report so that there are no misunderstandings.
- **Implementation** - It must be rational and must not waste system resources. The solutions found for each problem must be clear and well documented in the report. The programming style should follow good practices. The code must have relevant comments. System resources must be used according to their nature.
- **Report** - The report should describe the work properly. Merely superficial or generic descriptions will do little good. In general, the report describes the strategy and models followed, the structure of implementation and the options taken. Additional information about its elaboration may eventually be given. The report must be delivered together with the code in the file submitted via moodle in each goal.
- **Presentation ("defense")** - The project is subject to individual defense, during which the author authenticity and knowledge level will be verified. There may be need for more than one defense if there are any doubt about the authorship of the work. The final grade of the work is directly proportional to the quality of the defense. Members of the same group may have different grades depending on the individual's performance and degree of participation in the defense.
  Although the defense is individual, both elements of the group must attend at the same time. Failure to defend automatically implies the loss of the entire grade of the work.
  This year, a software will be used to automate the fraud detection. The school rules describe what happens in situations of fraud (e.g., plagiarism).
- Projects that do not work will be heavily penalized regardless of the quality of the source code or architecture presented. Works that do not even compile will have an extremely low grade.
- The identification of the group elements must be clear and unambiguous (both in the file submitted and in the report). Anonymous projects will not be graded.
- Any deviation to the format and form of each submission (example, file type) will result in penalties.