Birzeit University - Faculty of Engineering and Technology
Electrical & Computer Engineering Department - ENCS4330
Real-Time Applications & Embedded Systems - 2^{nd} semester - 2024/2025

Project #1
Signals, pipes & fifos under Linux
Due: March 29, 2025

Instructor: Dr. Hanna Bullata

Rope pulling game Simulation

We would like to build a multi-processing application that simulates the the behavior of 2 teams playing the rope pulling game (see Figure 1). The simulated game can be explained as follows:

Figure 1: A rope pulling game.

1. We'll call the two teams **team_1** and **team_2** consecutively.

2. Each team is composed of 4 players numbered 0 to 3. Initially when the game starts, all players have a random high level of energy that is bound to a user-defined range. Of course that energy decreases with time and effort each player puts while playing and the rate of decrease is on per-player basis (consider the rate of decrease to be random but belongs to a user-defined range).

   In addition, a referee is needed as well to keep the score of each team and to coordinate actions as described below. We'll assume that the parent of all players is the referee for the game.

3. Upon a signal from the referee (the parent), the players of `team_1` and `team_2` will get ready, meaning get aligned and the rope grabbed in the hand. The players' alignment for each team depends on the current effort level of each team member: The player with the lowest effort level is placed in the middle and the player with the highest effort level is placed towards the end of the rope (to maximize the chances of winning for each team).

4. Upon another signal from the referee, the players of `team_1` will start pulling the rope in their direction while the players of `team_2` will start pulling the rope in their direction as well (opposite direction).

5. Every second, the parent will request from each player to send its current energy level (effort level). Before sending its energy level, each player will multiply it by a factor that depends on its location. Thus the first player next the middle of the rope

multiplies its effort by 1, the player next to it multiplies its effort level by 2, and so on until the last player at the end of the rope multiplies its effort level by 4.

6. Upon receiving these values, the parent will sum the effort values of `team_1` and the effort values of `team_2` and then subtract the totals.

   If the subtraction result is positive, then `team_1` is heading towards winning the round. Otherwise, if the subtraction result is negative, then `team_2` is heading towards winning the round.

7. While playing, a player may accidentally fall and thus its effort level becomes zero momentarily. It'll take that player a random amount of time (that belongs to a user-defined range) to re-join the pulling activity of its team.

8. When one of the teams gets a total effort value that exceeds a user-defined threshold, the parent will decide that team as the round winner. The parent will afterwards inform each team member if it won or lost the round.

9. Go back to step **3** above to start a new round.

10. The simulation ends if any of the following is true:

    - The user-defined amount of time allocated to the game is over.
    - One of the teams has got a score that exceeds a user-defined number provided by the user.
    - One of the two teams has won 2 rounds in a row (consecutively). You can make the number of rounds user-defined if you wish.

What you should do

- Implement the above problem on your Linux machines using a multi-processing approach. Make sure the created processes consume CPU time when needed only.
- Compile and test your program.
- Check that your program is bug-free. Use the `gdb` debugger in case you are having problems during writing the code (and most probably you will :-). In such a case, compile your code using the `-g` option of the `gcc`.
- In order to avoid hard-coding user-defined values or ranges in your programs, think of creating a text file that contains all the values that should be user-defined and give the file name as an argument to the main program. That will spare you from having to change your code permanently and re-compile.
- Use graphics elements from opengl library in order to best illustrate the application. Nothing fancy, just simple and elegant elements are enough.
- Be realistic in the choices that you make!
- Send the zipped folder that contains your source code and your executable before the deadline. If the deadline is reached and you are still having problems with your code, just send it as is!