Universitat Pompeu Fabra Barcelona

Lab 04: Delivery Exercise

Understanding threads and their management and synchronization.

In this lab we will revisit the game of Nim. However, in this case, we will use threads to simulate each player, with their appropriate synchronization mechanisms to avoid race conditions.

You have to implement the nim game with the same behaviour as in lab 3. Your program will receive a parameter from the terminal, and this will set the initial number of tokens that you will need to generate inside the file game.dat.
The game will be played by player A and B, each represented by 3 threads. Each thread will output a message indicating to which player it belongs and how much it will decrease the counter. To make it simpler, for each player there will be a thread that decreases the counter by 1, another by 2, and another one by 3.
Each turn, starting player A, all three threads of player A will try to get the lock of the file simultaneously. But only one of them (which, from our perspective, will be random) will succeed, and will decrease the number of tokens in game.dat according to its value. Then, it will pass the turn to the other player. Make proper use of synchronization to ensure that only one thread of one player modifies the file at a time.
Together with the instructions, you have a file, named P4_no_syncrho.c containing an implementation of the threads functions (missing the part where it reads their parameter) as well as some parts of the main. This file is to help you to implement the exercise solution, but it requires all the synchronization mechanisms to make the game work as described. To obtain the maximum grade, make sure to avoid active waiting and starvation (HINT: use condition variables).
When the game ends, i.e., the last token is taken from the file, all threads should be woken up and declare whether player A or B has won. Then, make sure that the program ends correctly.

Delivery Instructions:
For this laboratory we will use the platform gradescope (www.gradescope.com) to submit the exercise. You should have received an email allowing you to create an account there using the university email address. Upload in practice 4 all documents associated with the practice as usual:

- Submit the following files using the same specified naming conventions below:
  - Nim_threads.c
  - integrants.txt include all participants of the group, separated by lines, in the format *NIA Name Lastname1 LastName2 , and the practice group number.*
- Ensure each program compiles successfully and adheres to the CLI argument structure provided. Include the instruction you used for compiling.
- Take a screenshot showing your program successfully compiled without errors.

- Submit the programs as a compressed file named through Gradescope `practice4_<student1_student2>.zip`, replacing `<student1>` and `<student2>` with your full names (first and last names). You should have received an invitation to the OS course to your UPF email account. Along with the compressed file, include screenshots showing the program successfully compiled and implemented without errors.
- Only 1 submission per pair is required. You need to indicate, when doing the submission in gradescope, the members of the group.
- Be prepared to explain the implementation and use of each program to your peers in the next class. Ensure that all programs are written in the C programming language and follow the provided naming conventions.
- Allowed function calls: open, close, read, write, dup2, fork, wait, waitpid, pipe, getpid, getppid, printf, scanf, malloc, free, signal, kill, and these seen in class for thread management and synchronization. You are not allowed to use **any other functions** that directly or indirectly use a system call (i.e. fwrite).