INFO0012-2/3 : Project 2 (Parallel programming)

Deadline December 7th, 2015

```
 --- --- --- ---
| X | X | O | X |
 --- --- --- ---
| X | O | O |   |
 --- --- --- ---
| X | O | O |   |
 --- --- --- ---
| O |   | O | X |
 --- --- --- ---
```

Figure 1: Sample display

For this project, you are asked to develop a program that will let the user play tic-tac-toe against the computer. We will use the following terminology:

- **Grid:** The representation of the play-board. It is a square of $N \times N$ tiles on which players can write 'X's and 'O's.
- **Tile:** A space on the grid that can hold an 'X' or an 'O'.
- **Player:** A human or a computer program playing the game.

1 Commands

Commands are taken from the console (keyboard) and are of three types. The user can:

1. Select, at the beginning of a game, whether he wants to play 'X's or 'O's;
2. During the game, select a tile (described by its coordinates on the grid) on which to write his symbol;
3. At the end of the game, restart a new game or exit the program.

2 Game rules

- There are two players : the (human) user and the automated player.
- The player writing 'O's always starts first.
- The players move alternatively.
- A player can only write on an empty tile.
- On a grid of size $N \times N$, the first player that fills a horizontal, vertical or diagonal line of size N wins.
- If all tiles have been used and no player has been able to fill a line of size N, the game ends in a draw.
- A win is preferable to a draw, which is preferable to a loss.

3 The automated player

The automated player is organized as set of processes: one master process and a set of worker processes. It proceeds according to the algorithm described below (for the game to be fun to play, we don't want it to be too smart).

- The master process creates P worker processes (the number P being a parameter of the program). This is done only once.
- At each round, the automated player selects its move as follows.
  - For each free tile, the master process assigns it to one of the worker processes by sending a message to that process;
  - When receiving a move to be examined, a worker process does T (T is another parameter of the program) random simulations of the rest of the game, i.e. it chooses each of its opponent's and of its own moves using a uniform distribution until the game ends (by a win, a draw or a loss) and records that result;
  - The worker process then computes a value for its assigned position by adding twice the number of wins to the number of draws and sends this result to the master process;
  - it then waits for another request from the master process, or for a signal to quit (when the program is being stopped)
  - The master process then selects the move with the highest score (if several moves has the same score, it can choose any of these).

4 Implementation

You are asked to write in C a parallel program that plays the game as described above.

- The size $N \in [3, 6]$ of the grid, the number $P \in [1, 10]$ of processes and the number $T \in [1, 100]$ of loops should be parameters of your program.
- The grid is displayed on the console and represented with '-', '`|`', 'X' and 'O' characters, as in the example given in Figure 1.
- The grid is stored in shared memory; an additional "game manager" process is used to handle the console inputs and outputs. All processes have access to the data stored in the shared memory.
- Access to the shared memory will be controlled with semaphores, if needed. You are not allowed to use active waits (i.e., repeatedly testing a condition in a loop).
- The user commands (selecting 'X's or 'O's, selecting a move, game restart) are transmitted to "game manager" using terminal I/O.
- Communication between the "game manager" and the "master process" of the automated player is handled using only shared memory and semaphores for synchronization.
- Communication between the master process and the worker processes is handled using blocking message queues.

5 Submission procedure

- This project must be coded in C using the `System V IPCs` for the shared memory, the semaphores and the message queues. The display will be performed on the console.
- You must write a report describing how you implemented your program, and in particular how the synchronization is performed.
- The project has to be done in teams of 2 students and completed before before December 7th at 23h59. The completed program and report (PDF only) will be included in a ZIP archive named `sXXXXXX_NAME1_sYYYYYY_NAME2.zip` where `sXXXXXX`, `NAME1`, `sYYYYYY`, and `NAME2`, are the student IDs and uppercase surnames of the team members.

Submit your archive to the **Montefiore Submission Platform**[1], after having created an account if necessary. If you encounter any problem with the platform, let me know (S.Hiard@ulg.ac.be). However problems that unexpectedly and mysteriously appear five minutes before the deadline will not be considered. **Do not send your work by E-mail; it will not be read**.

Good programming...

[1]http://submit.run.montefiore.ulg.ac.be/