Erlang. Concurrencia.

Práctica 4

En esta práctica vamos a implementar un sencillo juego distribuido en el que asumimos un conjunto de jugadores conectados a un servidor. El servidor enviará una palabra cualquiera a los jugadores y estos deben reenviarla al servidor. Gana el que primero envíe de vuelta esa misma palabra, es decir, el más rápido con el teclado.

En el campus están los módulos (compilados) **serv** y **player** que implementan el servidor y el jugador respectivamente, para probar el juego en el laboratorio y asimilar su funcionamiento. Para arrancarlo:

- el profesor arranca un proceso servidor en su máquina informando a los alumnos del nombre del nodo y la cookie utilizada y arrancará el proceso servidor que es un proceso registrado con el nombre `serv`:

  ```
  erl -name jaime@147.96.84.122 -setcookie abc
  > serv:start().
  ```

- el alumno a su vez arrancará un nodo con la IP de su ordenador, un identificador y la misma cookie:

  ```
  erl -name alumno1@147.96.84.133 -setcookie abc
  ```

  Y después el proceso cliente, con la información del nodo servidor de la forma:

  ```
  > player:start('jaime@147.96.80.131').
  ```

- Una vez establecida la conexión, el servidor puede arrancar el juego con:

  ```
  > serv ! play.
  ```

Tras jugar unas partidas con el juego y entender la mecánica del mismo, la práctica consistirá en implementar los módulos cliente y servidor. El servidor llevará cuenta de los clientes (PIDs) conectados al sistema así como la palabra en curso y podrá recibir los siguiente mensajes:

- **players**: muestra la lista de jugadores conectados (los PIDs)
- **play**: arranca el juego, enviando una palabra a los jugadores y esperando respuesta. Esta espera no será bloqueante, es decir, entre tanto el servidor puede responder a otras peticiones, como añadir jugador al juego, etc.
- **{From,{word,W}}**: el jugador `From` envía la respuesta **W**. El servidor comprobará si es correcta, en cuyo caso informa a los jugadores de quién ha ganado y empezará otra partida con otra palabra.
- **stop**: informa a los jugadores del fin del juego y termina el proceso servidor.

Por su parte, el proceso cliente (jugador) debe responder a los siguientes mensajes:

- **{From,{word,W}}**: recibe una palabra del servidor, la muestra en pantalla (del jugador) y solicita que la teclee. Esto puede hacerse de manera sencilla con el siguiente código:

  ```
  io:format("Write ~p",[W]),
  W0 = io:get_line("> "),
  % quitamos la "\n" del final
  W1 = string:substr(W0,1,string:len(W0)-1)
  ```

  Después enviará esa palabra al servidor y esperará la corrección (correcta o incorrecta), informando al jugador. Si es incorrecta el jugador podrá seguir intentándolo mientras otro jugador no se adelante.

  Importante: la interacción con el usuario (entrada/salida) no puede ser bloqueante, i.e., mientras el jugador está tecleando, el proceso cliente puede recibir la notificación de que otro jugador ha ganado la partida, un mensaje de terminación, etc.

- **{From,{Player,wins}}**: el servidor (`From`) informa de que el jugador **Player** ha ganado la partida.
- **stop**: se informa al servidor de que este jugador abandona el juego y termina el proceso asociado.

Esta sería la implementación básica del juego. Es posible implementar infinidad de mejoras y extensiones: identificar a los jugadores por su nombre, incorporar **timeouts** para controlar los tiempos de respuesta tanto del servidor como de los jugadores, llevar cuenta de los aciertos y fallos de cada jugador, y hacer estadísticas.... Pueden hacerse también diversas variantes: descentralizar la función del servidor y en cada ronda, el jugador que ha ganado la anterior sea el que plantee la palabra al resto; utilizar preguntas estilo trivial en vez de teclear palabras sin más; hacer equipos; ...