Erlang. Concurrencia.

Práctica 1.

Deben entegarse via Campus Virtual solo los ejercicios 1 a 4

1. Extender la implementación del servidor de archivos presentado en clase (módulos *afile_server* y *afile_client*) de manera que pueda modificarse el directorio de trabajo una vez creado el servidor, mediante el envío de un mensaje (para simplificar, asumiremos que se trabaja solo con *rutas absolutas* como directorios de trabajo). Para ello, el proceso servidor admitirá mensajes de la forma {`change_dir,AbsolutePath`} y el cliente se extenderá con una función `cd(AbsolutePath)`.

2. Implementar un servidor de eco, i.e., un proceso que reciba un átomo Erlang y lo escriba en pantalla. El proceso debe debe terminar al recibir el mensaje *stop*. Hacer después un interface para ocultar el envío de mensajes, que contenga tres funciones:

   - *start()*: arranca el (proceso) servidor de eco y devuelve su Pid.
   - *print(Pid,Term)*: envía al servidor de eco *Pid* el término *Term* para que lo imprima en patalla.
   - *stop(Pid)*: termina el proceso servidor de eco *Pid*.

   Tal como lo has implementado, ¿*print(Pid,stop)* termina el servidor? ¿Podrías evitarlo refinando el envío de mensajes?

   ¿Se podría refinar la implementación de modo que las funciones *print* y *stop* no necesiten el parámetro *Pid*?

3. Implementar una función *reloj(N)* que escriba la hora a intervalos regulares de *N* milisegundos. Por ejemplo, *reloj(1000)* escribiría algo así:

   ```
   Hora: {14,55,55}
   Hora: {14,55,56}
   Hora: {14,55,57}
   Hora: {14,55,58}
   Hora: {14,55,59}
   Hora: {14,56,0}
   Hora: {14,56,1}
   Hora: {14,56,2}
   ```

   a intervarlos de 1 segundo.

4. Implementar un módulo *escucha* que cree un proceso que:

   - acepte un mensaje con remitente: escribe el mensaje en pantalla, envía un `ok` al remitente y queda de nuevo a la escucha
   - acepte el mensaje `stop`: informa de la finalización del proceso y termina el mismo
   - transcurridos 5 segundos, si no ha recibido ningún mensaje, recordará en pantalla que está a la espera y seguirá a la espera.

   Este módulo debe implementar también un interfaz con funciones para arrancar el proceso, mandarle un mensaje y pararlo.

No hace falta entregar los siguientes ejercicios

5. Dada la función

```
f(N) ->
    Fa=fun(_) -> io:format("a~n",[]) end,
    Fb=fun(_) -> io:format("b~n",[]) end,
    spawn(fun() -> lists:foreach(Fa,lists:seq(1,N)) end),
    spawn(fun() -> lists:foreach(Fb,lists:seq(1,N)) end).
```

   determinar qué se obtiene en pantalla al evaluar f(3).

6. Implementar una función que lance dos procesos *Ping* y *Pong* que intercambien mensajes entre ellos y simulen N jugadas de una partida de ping-pong (una jugada consiste en un pase de *Ping* y otro de *Pong*). Después ambos procesos terminarán (y morirán). Por ejemplo, para una partida de 3 pases, la salida sería:

```
> ej:pingPong(3).
recibe ping
recibe pong
<0.129.0>
recibe ping
recibe pong
recibe ping
recibe pong
ping termina
pong termina
```

   Pensar distintas posibilidades de implementación para manejar el contador de jugadas N. ¿Que ocurre si otro proceso envía un mensaje a *Ping* y se mete en la partida? ¿Cómo podría evitarse que otro proceso se *colase* en la partida enviando un mensaje a uno de nuestros procesos?