Concurrencia y Paralelismo

Grado en Ingeniería Informática

1. **Notificador**

   Implemente en Erlang un servicio de propagación de mensajes. El API de este servicio tiene tres funciones:

   - `notifier:start()`, que arranca un servicio de propagación y devuelve su `PID`.
   - `notifier:register(N)`, que registra al proceso que la llama en el notificador `N`
   - `notifier:send(N, Msg)`, que envía el mensaje `Msg` a todos los procesos registrados en el notificador `N`.

   ```
   -module(notifier).

   -export([start/0, register/1, send/2]).
   -export([loop/...]).

   %% API

   start() -> spawn(?MODULE, loop, []).

   register(N) -> ...
   send(N, Msg) -> ...

   %% Internal Functions

   loop() ->
   ...
   ```

   Implemente las funciones `send/2`, `register/1` y `loop`. Puede añadir parámetros a loop si lo necesita.

2. **Almacén de Productos**

   Se desea implementar un almacén donde se puedan guardar y recuperar productos. El API es el siguiente:

   - `store:start()`, que arranca un almacén y devuelve su pid.
   - `store:store(S, P)`, que guarda el producto P en el almacén con pid S.
   - `store:get(S, F)`, donde S es un almacén, y F un predicado sobre productos. La función devuelve {`error, no_product`} si no hay ningún producto que cumpla F en el almacén, y {`ok, P`} (siendo P un producto que cumple `F(P)`) si lo hay. Al devolver un producto se elimina del almacén.

   ```
   -module(store).

   -export([start/0, store/2, get/2]).
   -export([loop/0]).

   %% API

   start() -> spawn(?MODULE, loop, []).

   store(S, P) -> ...
   get(S, F) -> ...
   ```

```
%% Internal Functions

loop() ->
  ...
```

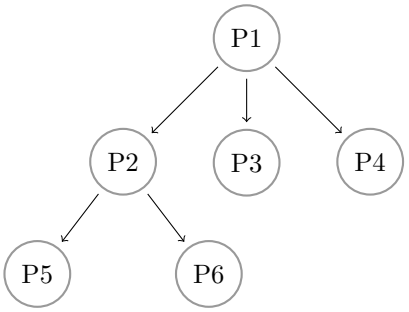
Implemente las funciones `store/2`, `get/2` y `loop/0`. Puede añadir funciones auxiliares y parámetros a `loop` si lo necesita.

3. **Árbol de Procesos**

Tenemos un sistema donde hay un árbol n-ario de procesos, donde cada nodo es un proceso que guarda una lista con los pids de sus hijos. Los nodos se crean llamando a `start_node/0`, y añadimos hijos a un nodo existente con `add_child/2`.

```
-module(tree).

-export([start_node/0, add_child/2, height/1]).

%% API

start_node() ->
    spawn(?MODULE, init_node, []).

add_child(Tree, Child_Tree) ->
    Tree ! {add_child, Child_Tree}.

height(Tree) ->
    ...

%% Internal functions

init_node() ->
    node_loop([]).

node_loop(Children) ->
    receive
        {add_child, Child_Tree} ->
            node_loop(Children ++ [Child_Tree]);
        ...
    end.
```

Por ejemplo, el árbol:



Se crearía de la siguiente forma:

```
P1 = start_node().
P2 = start_node().
```

```
P3 = start_node().
P4 = start_node().
P5 = start_node().
P6 = start_node().
add_child(P2, P5).
add_child(P2, P6).
add_child(P1, P2).
add_child(P1, P3).
add_child(P1, P4).
```

Implemente la función `height/1` que calcula la altura de un árbol de procesos.