Concurrencia y Paralelismo

Grado en Informática 2016

Práctica #3 (Servidor de Recursos)

Defina un proceso servidor que actúe como un gestor de recursos, es decir, que guarde una lista de recursos que podrán ser solicitados por otros procesos clientes. El servidor debe guardar la lista de recursos libres, y la lista de recursos asignados y el proceso al que se han asignado.

Este proceso servidor debe ser capaz de procesar los siguientes mensajes:

- {`alloc, From`}, que devuelve al proceso `From` el par {`ok, Recurso`}, si hay algún recurso disponible en el servidor, o bien {`error, sin recursos`} si no hay recursos disponibles.
- {`release, From, Recurso`}, que devuelve un `Recurso` previamente solicitado por `From`, devolviendo ok si la operación se lleva a cabo con éxito (`Recurso` queda disponible para nuevas reservas) o {`error, recurso no reservado`} si el recurso que intenta devolverse no fue previamente asignado al proceso `From`.
- {`avail, From`}, que devuelve un entero con el número de recursos disponibles en el servidor.

Cada recurso se va a modelar como un átomo (i.e., a), proporcionándose en el momento de iniciar el servidor una lista con todos los recursos disponibles inicialmente (i.e., `[a,b,c,d]`).

Para facilitar la interacción con el servidor a los clientes se les ofrecen las funciones `alloc/1`, `release/2` y `avail/1` que encapsulan la interacción con el mismo.

```
1> G = gestor:start([a,b,c,d,e]).
<0.33.0>
2> gestor:avail(G).
5
3> gestor:alloc(G).
{ok, a}
4> gestor:avail(G).
4
5> gestor:release(G, x).
{error, recurso_no_reservado}
6> gestor:release(G, a).
ok
7> gestor:avail(G).
5
8> gestor:release(G, a).
{error, recurso_no_reservado}
```

Nótese que el unico proceso que debe poder devolver un recurso reservado es el propio proceso que solicitó la reserva.

```
9> Pid = self().     %% Pid ligado a la identidad del shell
<0.44.0>
10> spawn(fun () -> {ok, R} = gestor:alloc(G),
                    Pid ! R
          end).
<0.34.0>
11> receive A -> gestor:release(G, A) end.
{error, recurso_no_reservado}
```

Tenga cuidado con el hecho de que el shell es un proceso Erlang que ante cualquier error muere y es creado nuevamente.

```
13> self().
<0.26.0>
14> gestor:alloc(G).
{ok, b}
15> 1/0.
=ERROR REPORT==== 22-May-2015::17:12:56 ===
Error in process <0.26.0> with exit value:
 {badarith,[{erl_eval,eval_op,3},{erl_eval,exprs,4},{shell,eval_loop,2}]}
** exited: {badarith,[{erl_eval,eval_op,3},
                      {erl_eval,exprs,4},
                      {shell,eval_loop,2}]} **
16> self().
<0.31.0>
17> gestor:release(G, b).
{error, recurso_no_reservado}
```