

Erlang

Ejercicio 4: Concurrencia en Erlang

1 Servidor con servicios dinámicos

Implemente un servidor que tenga por estado una lista de la forma `[{Servicio,Funcion}]`, donde `Servicio` es un átomo que identifica un determinado servicio del servidor y `Funcion` es una función de aridad 1 que implementa dicho servicio. El servidor proporciona las siguientes funcionalidades:

- `{establecer, Servicio, Funcion}`, añade al estado del servidor un nuevo servicio, identificado por el átomo `Servicio`; si `Servicio` ya existe, el nuevo servicio reemplaza al anterior. Devuelve `{resultado, ok}`.
- `servicios`, devuelve los servicios disponibles `{resultado, [Servicio]}`.
- `{peticion, Servicio, Param}`, que solicita la invocación del servicio con el parámetro dado. Devuelve `{resultado, {ok, Resultado}}`, con `Resultado=Funcion(Param)`, o bien `{resultado, {error, servicio_desconocido}}` si el servidor no conoce el servicio solicitado.
- `stop`, que detiene el servidor. Devuelve `{resultado, ok}`.

El módulo `servidor` que implementa este proceso debe incorporar el siguiente API:

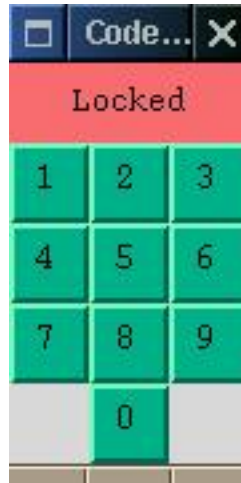
- `start/0`, arranca el servidor sin servicios y devuelve el `Pid` del servidor. `stop/1` para el servidor.
- `establecer(Pid, Servicio, Funcion)`, que dado un `Pid`, un `Servicio`, y una `Funcion`, establece un nuevo servicio.
- `servicios(Pid)`, que dado un `Pid`, devuelve los servicios disponibles en el servidor.
- `peticion(Pid, Servicio, Param)`, que dado un `Pid`, un `Servicio`, y el parámetro del servicio `Param`, invoca un determinado servicio.

Por ejemplo,

```
1> S = servidor:start().
<0.33.0>
2> servidor:establecer(S, suma, fun ({X,Y}) -> X+Y end).
ok
3> servidor:establecer(S, ordenar, fun (L) -> ordena:ord_insercion(L) end).
ok
4> servidor:servicios(S).
[suma, ordenar]
5> servidor:peticion(S, suma, {2,3}).
5
6> servidor:peticion(S, ordenar, [2,3,1,6,2,0]).
[0,1,2,2,3,6]
7> servidor:peticion(S, mult, {4,5}).
{error, servicio_desconocido}
```

2 Controlador de un *Code Lock*

Implemente un proceso controlador para el *Code Lock* simulado por el módulo `cl_sim`.



El *Code Lock*, inicialmente en estado *Cerrado (Locked)*, debería abrirse al introducir, dentro de un tiempo razonable, los cuatro dígitos de la clave. Transcurrido un cierto tiempo en estado abierto, el *Code Lock* cerrarse automáticamente.

El módulo `cl_sim`, que sólo simula los botones y el *display* de la cerradura, se inicializa con la identidad del proceso que implementa la lógica. Por ejemplo, el controlador podría inicializar el simulador utilizando:

```
...
SimPid = cl_sim:start(self()),
...
```

Tras inicializar el simulador, éste le enviará al controlador mensajes de la forma `{button, Button}`, donde `Button` es un valor entre 0 y 9. Para cambiar el *display*, el controlador le envía al simulador el mensaje `{display, String}`. Por ejemplo,

```
...
SimPid ! {display, "Abierto"}
...
```

Se pide:

1. Construya un diagrama de estados del controlador que modele su lógica
2. A partir del diagrama de estados, implemente el controlador como un proceso Erlang