Concurrencia y Paralelismo

Grado en Informática 2025

Práctica 3 – Paso de mensajes

Ejercicio 1 (**Stack**) Implemente un módulo `stack` que proporcione una pila en Erlang. No es necesario crear un proceso para guardar la pila. El API debe ser el siguiente:

- `stack:empty()`, devuelve una pila vacía.
- `stack:push(Stack, Elem)`, donde stack es una pila, y Elem un elemento, inserta el elemento en la pila, y devuelve la pila resultante.
- `stack:pop(Stack)` elimina el elemento de la cima de la pila. Si la pila está vacía, devuelve una pila vacía.
- `stack:peek(Stack)` devuelve `{ok, Elem}`, donde `Elem` es el elemento en la cima. Si la pila está vacía devuelve el átomo `empty`.

Por ejemplo:

```
1> S0 = stack:empty().
[]
2> stack:peek(S0).
empty
3> S1 = stack:push(S0, a).
[a]
4> S2 = stack:push(S1, b).
[b, a]
5> stack:peek(S2).
{ok, b}
6> stack:pop(S2).
[a]
```

Ejercicio 2 (**Procesos en línea**) Implemente un módulo `line`, que cree una secuencia de procesos, donde cada proceso conoce el PID del siquiente en la secuencia. El módulo debe proporcionar las siguientes funciones:

- `line:start(N)`, donde N es el número de procesos de la secuencia, y devuelve el PID del primer proceso.
- `line:send(Pid, Msg)`, donde Pid es el Pid del primer proceso en una secuencia y Msg es un término Erlang cualquiera. Cada proceso debe imprimir su número en la secuencia, el mensaje recibido, y pasar el mensaje al siguiente proceso. Cuando el mensaje llegue al último se imprime pero se para la propagación.
- `line:stop(Pid)`, que para todos los procesos de la secuencia.

Por ejemplo:

```
1> P = line:start(3).
<0.84.0>
2> line:send(P, hola).
0 received mesage hola
```

```
ok
1 received mesage hola
2 received mesage hola
```

El átomo ok que devuelve como resultado line:send puede aparecer en cualquier posición dentro de la secuencia de mensajes impresos.

Ejercicio 3 (**Anillo de procesos**) Implemente un módulo `ring`, que cree un anillo de procesos, donde cada proceso conoce el PID del siquiente en el anillo. El módulo debe proporcionar las siguientes funciones:

- `ring:start(N)`, donde N es el número de procesos en el anillo, y devuelve el PID del primer proceso.
- `ring:send(Pid, N, Msg)`, donde Pid es el Pid del primer proceso en un anillo, N el número de nodos que procesarán el mensaje, y Msg un término Erlang cualquiera. Cada proceso debe imprimir su número en el anillo, el número de mensajes por imprimir y el mensaje recibido. Si el número de mensajes por imprimir es mayor que 0 también enviará el mensaje al siguiente proceso. El mensaje puede dar más de una vuelta al anillo si N es mayor que el número de procesos en el anillo.
- `ring:stop(Pid)`, donde Pid es el primer proceso de un anillo, para todos los procesos de un anillo.

Por ejemplo:

```
1> R = ring:start(3).
<0.84.0>
2> ring:send(R, 5, hola).
0 receiving message hola with 4 left
ok
1 receiving message hola with 3 left
2 receiving message hola with 2 left
0 receiving message hola with 1 left
1 receiving message hola with 0 left
```

El átomo ok que devuelve como resultado ring:send puede aparecer en cualquier posición dentro de la secuencia de mensajes impresos.

Entrega

La fecha límite de entrega es el 19 de marzo. Para crear el respositorio y acceder al código inicial, acceda a github classroom en https://classroom.github.com/a/Pn_GoSod. El nombre de los grupos de prácticas debe ser el login de ambas personas, separados por un guión.