Sistemas Operativos I - LCC
Práctica 3
Erlang

Introducción a Erlang

Erlang es un lenguaje de programación concurrente y un sistema de ejecución que incluye una máquina virtual y bibliotecas. La creación y gestión de procesos es trivial en Erlang, mientras que, en muchos lenguajes, los hilos se consideran un apartado complicado y propenso a errores. En Erlang toda concurrencia es explícita. Los programas se guardan en archivos con extensión `.erl` y se los compila desde el shell de Erlang (`erl`) con la función `c/1`. Los módulos deben exportar alguna función (con la cláusula `-export`) para poder invocarla desde el shell.

1. Ejercicio Introductorio

Dado el siguiente código (fuente aquí):

```
-module(intro).
-export([init/0]).

match_test () ->
  {A,B} = {5,4},
  {C,C} = {5,4},
  {B,A} = {4,5},
  {D,D} = {5,5}.

string_test () -> [
  helloworld == 'helloworld',
  "helloworld" < 'helloworld',
  helloworld == "helloworld",
  [$h,$e,$l,$l,$o,$w,$o,$r,$l,$d] == "helloworld",
  [104,101,108,108,111,119,111,114,108,100] < {104,101,108,108,111,119,111,114,108,100},
  [104,101,108,108,111,119,111,114,108,100] > 1,
  [104,101,108,108,111,119,111,114,108,100] == "helloworld"].
```

```
tuple_test (P1,P2) ->
  io:format("El nombre de P1 es ~p y el apellido de P2 es ~p~n",[nombre(P1),apellido(P2)]).

apellido (P) -> ok.
nombre (P) -> ok.

filtrar_por_apellido(Personas,Apellido) -> ok.

init () ->
  P1 = {persona,{nombre,"Juan"},{apellido, "Gomez"}},
  P2 = {persona,{nombre,"Carlos"},{apellido, "Garcia"}},
  P3 = {persona,{nombre,"Javier"},{apellido, "Garcia"}},
  P4 = {persona,{nombre,"Rolando"},{apellido, "Garcia"}},
  match_test(),
  tuple_test(P1,P2),
  string_test(),
  Garcias = filtrar_por_apellido([P4,P3,P2,P1],"Garcia").
```

- Explique justificando cuáles match's de la función `match_test/0` deberían ser válidos y cuáles no.
- Implemente las funciones `nombre/1` y `apellido/1` para que devuelvan esos campos de las tuplas que obtienen como argumento utilizando pattern matching.
- Explique el resultado de cada una de las comparaciones de la función `string_test/0` (es decir por qué dan true o false).
- Implemente la función `filtrar_por_apellido/2` para que devuelva los nombres (sin el apellido) de las personas de la lista `Personas` cuyo apellido coincide con `Apellido` utilizando comprensión de listas.

2. Temporización en Erlang

- Implemente una función `wait/1` que tome como argumento una cantidad de milisegundos y espere ese tiempo.
- Implemente un cronómetro que reciba tres argumentos, `Fun`, `Hasta` y `Periodo` y ejecute `Fun/0` cada `Periodo` milisegundos hasta que hayan pasado `Hasta` milisegundos **sin bloquear el intérprete**.

  Un caso de prueba sería:

  ```
  cronometro(fun () -> io:format("Tick~n") end,60000,5000).
  ```

  que imprimiría `"Tick~n"` cada 5 segundos durante un minuto.

3. Servidor de Eco Revisitado

Reimplemente el servidor de eco de la práctica 2 en Erlang. Aquí puede encontrar un esqueleto para manejar conexiones TCP-IP en Erlang.

- Compare el servidor en PThreads y el actual con el cliente dado anteriormente, para 200, 2000 y 20000 conexiones simultáneas. Puede usar el cliente que se encuentra aquí.
- ¿Ve una diferencia importante en el consumo de memoria de los dos servidores? ¿A qué cree que se puede deber?
- ¿Puede cada servidor aceptar 50000 conexiones simultáneas?

4. Lanzar Procesos en Anillos

Escriba un programa que lance N procesos en anillos. Cada proceso recibirá dos clases de mensajes:

- `{msg,N}` donde `N` es un entero. Deberá decrementarlo y enviarlo al siguiente proceso en el anillo si `N` es mayor que cero. En caso contrario deberá enviar un mensaje `exit` y terminar cuando todos los demás lo hayan hecho.
- `exit` cuando el proceso debe terminar

Modifique el programa para que el mensaje enviado gire una vez alrededor del anillo y sea descartado por el que inició el envío.

5. Broadcaster

Implemente un proceso servidor que distribuya los mensajes que recibe entre todos sus subscriptores. El servidor tiene las siguientes operaciones:

Subscribirse. El proceso llamado es incluido en la lista de subscriptores.

Enviar mensaje. El mensaje recibido debe ser reenviado a todos los subscriptores en la lista.

Desubscribirse. El proceso llamado es eliminado de la lista de subscriptores.

6. Sincronización en Erlang

Complete el código (fuente aquí) siguiente para implementar Locks y Semáforos en Erlang usando paso de mensajes. Las funciones `testLock/0` y `testSem/0` son casos de uso.

```
-module(synch).
-export([testLock/0,testSem/0]).

%internal
-export([f/2,waiter/2]).
-export([waiter_sem/2,sem/2]).


lock (L) -> ok.
unlock (L) -> ok.
createLock () -> ok.
destroyLock (L) -> ok.

createSem (N) -> ok.
destroySem (S) -> ok.
semP (S) -> ok.
semV (S) -> ok.

f (L,W) -> lock(L),
  %   regioncritica(),
  io:format("uno ~p~n",[self()]),
  io:format("dos ~p~n",[self()]),
  io:format("tre ~p~n",[self()]),
  io:format("cua ~p~n",[self()]),
  unlock(L),
  W!finished.

waiter (L,0)  -> destroyLock(L);
waiter (L,N)  -> receive finished -> waiter(L,N-1) end.

waiter_sem (S,0)  -> destroySem(S);
waiter_sem (S,N)  -> receive finished -> waiter_sem(S,N-1) end.


testLock () -> L = createLock(),
  W=spawn(?MODULE,waiter,[L,3]),
  spawn(?MODULE,f,[L,W]),
  spawn(?MODULE,f,[L,W]),
  spawn(?MODULE,f,[L,W]),
  ok.

sem (S,W) ->
  semP(S),
  %regioncritica(), bueno, casi....
  io:format("uno ~p~n",[self()]),
  io:format("dos ~p~n",[self()]),
  semV(S),
  W!finished.

testSem () -> S = createSem(2), % a lo sumo dos usando io al mismo tiempo
```

```
  W=spawn(?MODULE,waiter_sem,[S,5]),
  spawn(?MODULE,sem,[S,W]),
  spawn(?MODULE,sem,[S,W]),
  spawn(?MODULE,sem,[S,W]),
  spawn(?MODULE,sem,[S,W]),
  spawn(?MODULE,sem,[S,W]).
```

7. Hello Tolerante a Fallas

El siguiente programa (fuente aquí) crea un proceso que imprime "Hello" a intervalos regulares. Por una falla desconocida termina al poco tiempo con un error.

```
-module(hello).
-export([init/0, hello/0]).

hello() ->
    {A1,A2,A3} = now(),
    random:seed(A1,A2,A3),
    helloloop().

helloloop() ->
    receive
        after 1000 -> ok
    end,
    io:format("Hello ~p~n",
        [case random:uniform(10) of 10 -> 1/uno; _ -> self() end]),
    helloloop().

init() -> spawn(?MODULE, hello, []).
```

Reemplace este proceso por dos, donde el segundo deba levantar al proceso que imprime "Hello" cada vez que se caiga.

Nota: puede ser de ayuda utilizar `process_flag(trap_exit, true)`.

8. Reemplazando Módulos

El cliente está satisfecho con el servicio de salutación, pero le gustaría que lo salude en castellano y no en inglés. Modifique el código de manera que, una vez levantado el servicio, se pueda cambiar el mensaje por "Hola" sin darlo de baja. Es decir, se pueda hacer lo siguiente:

```
2> hello:init().
...
Hello <0.XX.0>
Hello <0.XX.0>
```

Después reemplazar "Hello" por "Hola" en `hello.erl`, y

```
3> c(hello).
Hello <0.XX.0>
Hola <0.XX.0>
...
```