Projeto de PLP

Tema: Incluir mecanismo de concorrência, inspirado no de Erlang, na linguagem Funcional 3

Equipe: Gabriela Cunha Sampaio (gcs)

Roberto Souto Maior

(rsmbf)

Contexto

- Linguagem Funcional 3 (LF3)
  - Lista, compreensão de lista, alta ordem, etc.
  - Sequencial apenas
- Erlang
  - Telecomunicações (inicialmente)
  - Funcional
  - Bom suporte a concorrência
  - Processos leves

Motivação


T · · Mobile ·
stick together


ERICSSON

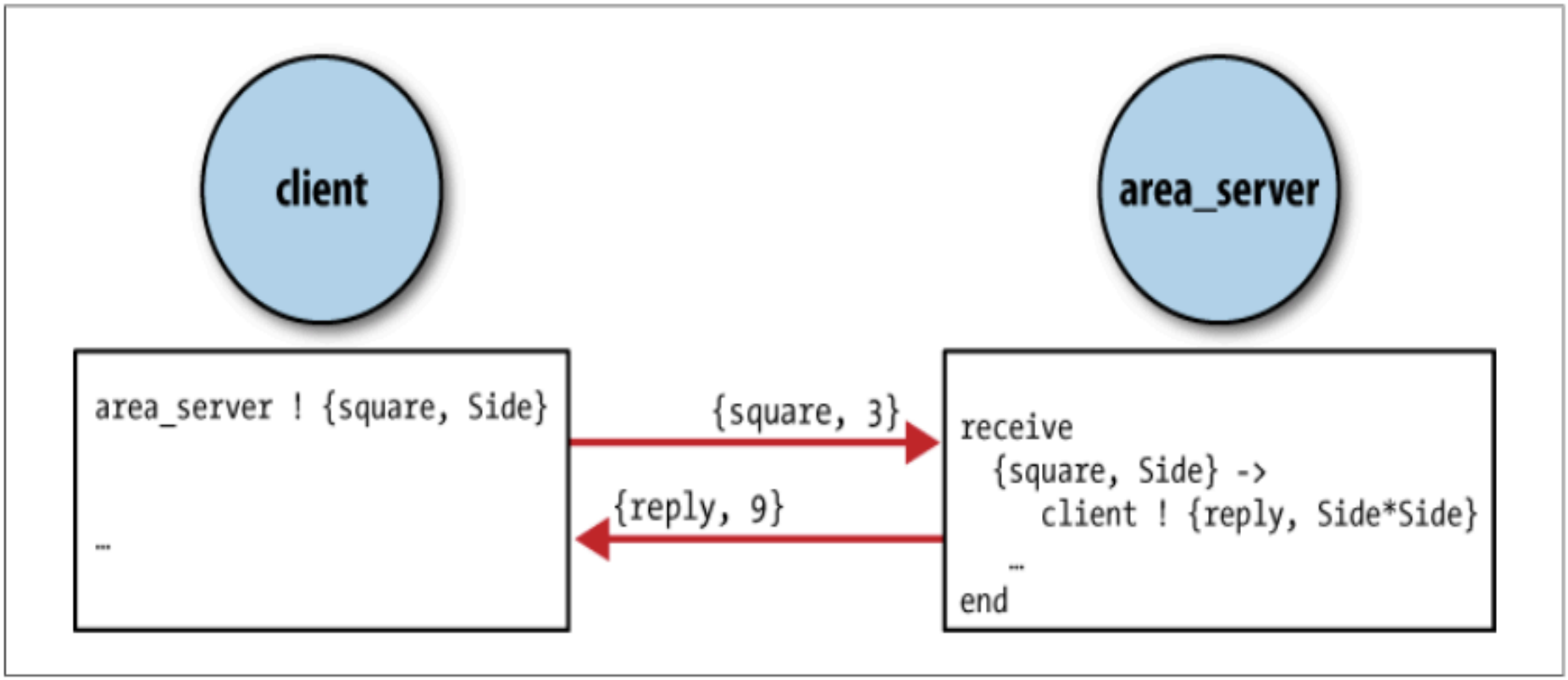

ERLANG


MOTOROLA


YAHOO!

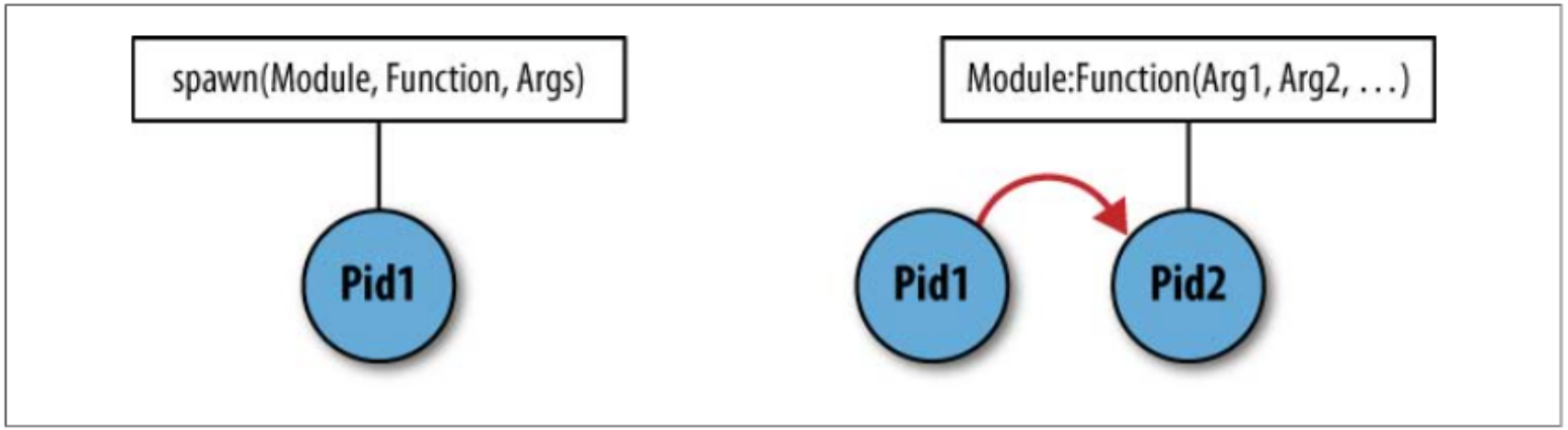

amazon

Erlang – Mecanismo de Troca de Mensagens



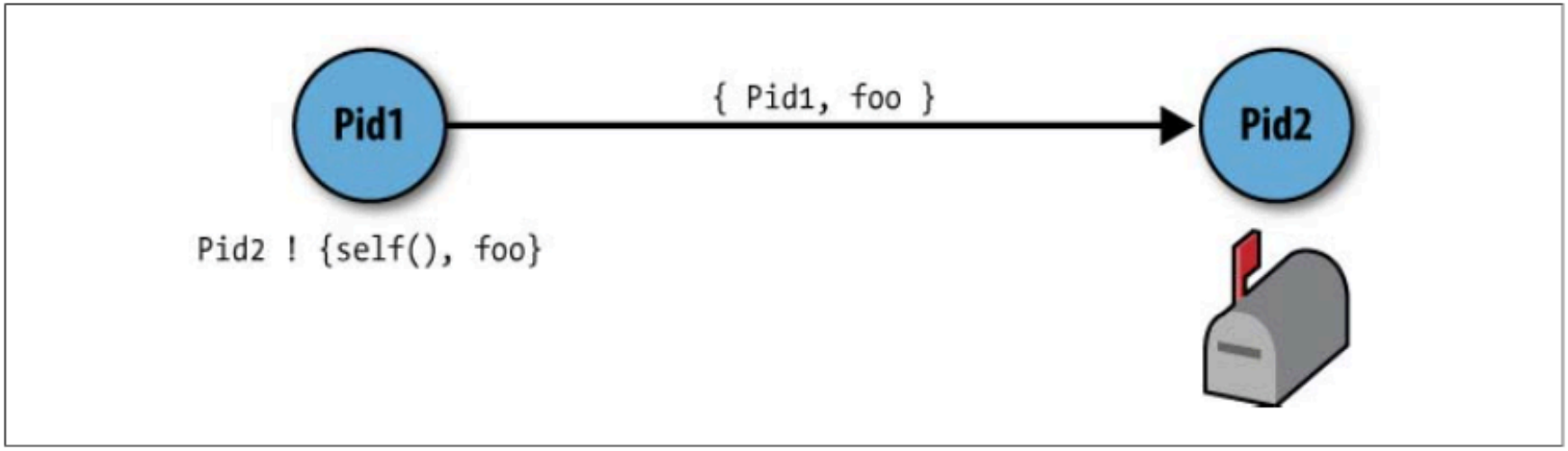
Erlang – Sintaxe de processos

- Criação através da função spawn:
  - Spawn(Fun)
  - Spawn(Mod,Fun,Params)
  - Retorna Id do processo criado (Pid)



Erlang – Sintaxe de processos

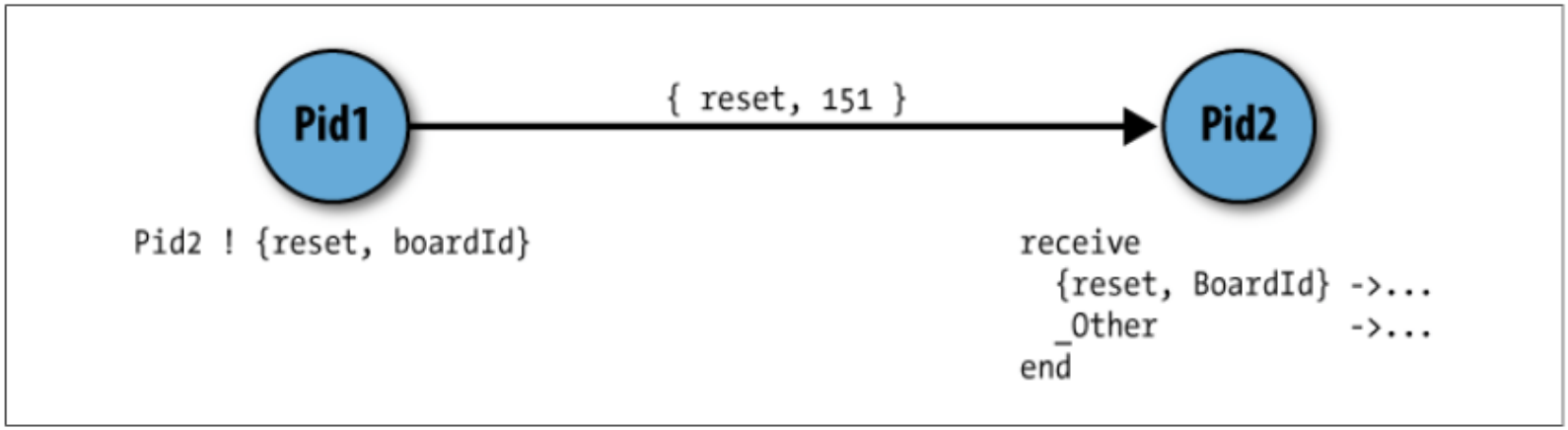
- Envio de mensagens através do operador "!":
  - Pid ! Mensagem
  - Pid ! Pid2 ! Pid3 ! Mensagem



Erlang – Sintaxe de processos

- Recebimento de mensagens através do comando receive:

```
receive
    Pattern1 when Guard1 -> exp11, .., exp1n;
    Pattern2 when Guard2 -> exp21, .., exp2n;
    Other -> expn1, .., expnn
end
```



Exemplos de concorrência em Erlang

```erlang
-module(echo).
-export([go/0, loop/0]).

go() ->
    Pid = spawn(echo, loop, []),
    Pid ! {self(), "Hello"},
    receive
      {Pid, Msg} ->
        io:format("~p~n",[Msg])
    end,
    Pid ! false.


loop() ->
    receive
      {From, Msg} ->
         From ! {self(), Msg ++ " world!"},
         loop();
      false ->
        true
    end.
```

```
Eshell V5.10.3  (abort with ^G)
1> c(echo).
{ok,echo}
2> echo:go().
"Hello world!"
false
3>
```

Exemplos de concorrência em Erlang

```erlang
-module(example2).
-compile([export_all]).

main() ->
    PidC = spawn(fun example2:funC/0),
    PidB = spawn(example2, funB, [PidC]),
    PidB ! {self(),["Hello from A"]},
    receive
        Msg ->
            Msg
    end.

funB(PidC) ->
    receive
        {PidA,Msg} ->
            PidC ! {PidA, Msg++["Hello from B"]}
    end.

funC() ->
    receive
        {PidA,Msg} ->
            PidA ! (Msg ++ ["Hello from C"])
    end.
```

```
Eshell V5.10.3  (abort with ^G)
1> c(example2).
{ok,example2}
2> example2:main().
["Hello from A","Hello from B","Hello from C"]
3>
```

Gramática

- Programa ::= Expressao
- Expressao ::= Valor| ExpUnaria
  | ExpBinaria | ExpDeclaracao | Id
  | Aplicacao | IfThenElse
  | "self()" | ExpressaoLista
  | ExpressaoTupla
- Valor ::= ValorConcreto
  | ValorAbstrato
- ValorAbstrato ::= ValorFuncao
- ValorConcreto ::= ValorInteiro
  | ValorBooleano | ValorString
  | ValorLista | ValorTupla
- ValorFuncao ::=
  "fn" ListId "." CompExp
- ExpUnaria ::= "-" Expressao
  | "not" Expressao
  | "length" Expressao
  | head(Expressao)
  | tail(Expressao)
  |ExpCompreensaoLista
- ExpCompreensaoLista ::=
  Expressao Gerador
  | Expressao Gerador Filtro
- Gerador ::=
  "for" Id "in" Expressao
  | "for" Id
  "in" Expressao [", "] Gerador
- Filtro ::= "if" Expressao

Gramática

- ExpBinaria ::=
  Expressao "+" Expressao
  | Expressao "-" Expressao
  | Expressao "*" Expressao
  | Expressao ">" Expressao
  | Expressao "<" Expressao
  | Expressao "and" Expressao
  | Expressao "or" Expressao
  | Expressao "==" Expressao
  | Expressao "++" Expressao
  | Expressao ".. " Expressao
  | Expressao ":" Expressao
  | Expressao "^^" Expressao

- ExpDeclaracao ::=
  "let" DeclaracaoFuncional
  "in" CompExp

- DeclaracaoFuncional ::=
  DecVariavel | DecFuncao
  | DeclaracaoFuncional ","
  DeclaracaoFuncional

- DecVariavel ::=
  "var" Id "=" CompExp

- DecFuncao ::=
  "fun" ListId "=" CompExp

- ListId ::= Id | Id ListId

- Aplicacao:=
  Expressao"(" ListExp ")"

- ListExp ::= Expressao
  | Expressao, ListExp

- IfThenElse ::= "if" Expressao
  "then" CompExp "else" CompExp

Gramática

- Spawn := "spawn""("SpawnFunArg, ExpressaoLista")"
- SpawnFunArg := Id | ValorFuncao
- ExpProc := Send | Receive | Spawn
- Receive := "receive" ClauseList "end"
- ClauseList := Clause | Clause";" ClauseList
- Clause := Pattern "->" CompExp
- Pattern := Id | ValorConcreto | Pattern ":" Pattern | "{"ListPattern"}"
- ListPattern := Pattern | Pattern","ListPattern
- Send := Id "!" Message
- Message := Expressao | Send
- ExpressaoLista := "[]" | "["ListExp"]"
- ExpressaoTupla := "{"ListExp"}"
- CompExp := ListExpProc | [ListExpProc","] Expressao [","ListExpProc]
- ListExpProc := ExpProc | ExpProc","ListExpProc

Programas

```
let fun loop words =
        receive
                {from, msg} ->
                        from ! {self(), msg ++ words}, loop(words);
                false -> false
        end
In let var pid = spawn(loop, [" world!"]) in
        let fun go words =
                pid ! {self(), words},
                receive
                        {pid2, msg} ->
                                pid2 ! false, msg
                end
        in go("Hello")
```

Saída: "Hello world!"

Programas

```
let fun proB pidC text =
        receive
            {pidA, msg} -> pidC ! {pidA, msg ^^ [text]}
        end,
    fun proC text =
        receive
            {pidA, msg} -> pidA ! (msg ^^ [text])
        end
In let fun main textA textB textC =
        let var pidC = spawn(proC,[textC]) in
            let var pidB = spawn(proB,[pidC,textB]) in
                pidB ! {self(), [textA]},
                receive msg -> msg end
    in main("Hello from A", "Hello from B", "Hello from C")
```

Saída:
["Hello from A",
"Hello from B",
"Hello from C"]

Programas

```
let fun conta saldo =
        receive
                {"debitar", valor} -> conta(saldo - valor);
                {"creditar", valor} -> conta(saldo + valor);
                {"transferir", valor, contaDestino} ->
                        contaDestino ! {"creditar", valor},
                        conta(saldo - valor);
                {"consultar", pid} ->
                        pid ! {self(), saldo},
                        conta(saldo);
                "encerrar" -> true
        end
      …
```

Programas

```
… , fun consultarAll contas saldos =
        if contas == [] then saldos
        else
            let var conta = head(contas) in
                conta ! {"consultar", self()},
                conta ! "encerrar",
                receive
                    x -> consultarAll(tail(contas), saldos ^^ [x])
                end
in
    …
```

Programas

```
... let fun banco saldo = let
                    var c1 = spawn(conta,[saldo]),
                    var c2 = spawn(conta,[saldo]),
                    var c3 = spawn(conta,[saldo]),
                    var c4 = spawn(conta,[saldo]),
                    var c5 = spawn(conta,[saldo])
                in c1 ! {"creditar",10},
                    c2 ! {"creditar", 50},
                    c2 ! {"transferir", 20, c3},
                    c3 ! {"transferir", 5, C4},
                    c5 ! {"creditar", 10},
                    c5 ! {"debitar", 9},
                    consultarAll([c1,c2,c3,c4,c5],[])
in banco(0)
```

Possível Saída:

[{<0.58.0>,10},
{<0.59.0>,30},
{<0.60.0>,15},
{<0.61.0>,5},
{<0.62.0>,1}]

Projeto de PLP

Tema: Incluir mecanismo de concorrência, inspirado no de Erlang, na linguagem Funcional 3

Equipe: Gabriela Cunha Sampaio (gcs)

Roberto Souto Maior

(rsmbf)