Erlang
- Programação concorrente -

Luís Nogueira

luis@dei.isep.ipp.pt

Departamento Engenharia Informática

Instituto Superior de Engenharia do Porto

Introdução

- Processos totalmente independentes
- Não existe partilha de dados
  - Não pode ser feita em paralelo
  - Exclusão mútua
- Cada processo tem o seu próprio PID
- Comunicam apenas por passagem de mensagens
  - Usando o pid do destinatário
  - Comunicação assíncrona

Criar um novo processo

```
spawn(Módulo, Função, Lista de argumentos)
```

- Cria um novo processo e retorna o seu PID
- Novo processo executa função indicada
- Novo processo existe enquanto possui código para executar
- Função usada em `spawn/3` tem de ser exportada
- Chamada a `spawn/3` não bloqueia

```
cria_proc_soma(A,B) ->
   spawn(mat,soma,[A,B]).
```

Envio de mensagens

```
erlang:send(PID_destino, Msg)

PID_destino ! Msg
```

- Única forma de comunicação entre processos
- Necessário conhecer PID do destinatário
- `Msg` pode ser qualquer termo válido
  - Normalmente é um tuplo com uma etiqueta

```
cria_proc_soma(A,B) ->
   Pid = spawn(mat,soma,[]),
   Pid ! {operandos, A, B}.
```

Recepção de mensagens

```
receive
   Padrao_1 [when Guard_1]-> processar mensagem;
   ...
   Padrao_n [when Guard_n]-> processar mensagem
end.
```

- Processo tem uma *mailbox* FIFO
- Se a mensagem recebida validar algum padrão é processada (avaliados por ordem)
- Se não validar nenhum padrão permanece na *mailbox*
- Padrões podem ter *guards* opcionais
- `receive` bloqueia até à recepção de uma mensagem válida

Recepção de mensagens

```
cria_proc_soma(A,B) ->
   Pid = spawn(mat,soma,[]),
   Pid ! {operandos, A, B},
   receive
      {resultado, X} -> X;
      {erro, Erro} -> Erro
   end.
```

- Nunca vai obter resposta!
- Processo `soma` não pode responder
  - Não conhece PID do remetente

Envio de PIDs nas mensagens

- `self()` retorna o PID do próprio processo
- Processo remetente envia o seu PID junto com o pedido
- Processo destino pode agora responder
- Formato das mensagens tem de ser igual nos dois processos!

```
cria_proc_soma(A,B) ->
   Pid = spawn(mat,soma,[]),
   Pid ! {operandos, A, B, self()},
   receive
      {resultado, X} -> X;
      {erro, Erro} -> Erro
   end.
```

Modelo cliente/servidor

1. Servidor aguarda pedido
2. Ciente conhece PID do servidor
3. Cliente envia o seu PID junto com o pedido
4. Servidor processa o pedido
5. Servidor responde ao PID recebido na mensagem

```
servidor() ->
  receive
    {pedido,From,P} ->
      R = processar(P),
      From!{resposta,R}
  end,
  servidor().
```

```
cliente(PidS,P)->
  PidS!{pedido,self(),P},
  receive
    {resposta,R}->R
  end.
```

Registo de processos

`register(Nome, Pid)`

- Regista o processo `Pid` com o nome global `Nome`
- `Nome` tem que ser um átomo
- Envio de mensagens através do nome registado

  `nome_registado ! Msg`
- Algumas primitivas
  - `unregister(Nome)` - Remove a associação
  - `whereis(Nome)` - devolve Pid associado a `Nome` ou `undefined`
  - `registered()` - devolve lista dos processos registados

Tempo máximo de recepção

```
receive
   Padrao_1 [when Guard_1]-> processar mensagem;
   ...
   Padrao_n [when Guard_n]-> processar mensagem
   after Timeout -> AccaoT
end.
```

- Tempo máximo para a recepção de uma mensagem
- `Timeout` é um inteiro em milisegundos
- Se nenhuma mensagem for recebida durante `Timeout` ocorre `AccaoT`

Tempo máximo de recepção - Exemplos

- Suspender um processo durante `T` milisegundos

```
sleep(T) ->
   receive
      after T -> ok
   end.
```

- Definir um alarme

```
set_alarm(T,Msg) ->
   spawn(timer,alarm,[self(),T,Msg]).

alarm(Pid,Msg) ->
   receive
      after T -> Pid ! Msg
   end.
```

Exemplo - Servidor ping

```
start_server() ->
    Pid = spawn(?MODULE,server_loop,[]),
    register(server, Pid).

server_loop() ->
   receive
      {msg,From,ping} ->
         From!{reply,pong};
      {msg,From,Msg} ->
         From!{error,'msg invalida'}
   end,
   server_loop().
```

Exemplo - Cliente ping

```
ping(Msg,Timeout) ->
   server!{msg,self(),Msg},
   receive
      {reply,Reply} ->
         Reply;
      {error,Error} ->
         Error
      after Timeout ->
         'sem resposta do servidor'
    end.
```