ERLANG / OTP

Paul Valckenaers

Processes and Messages

- spawn => een process aanmaken (= lightweight thread)
- ! => een boodschap sturen
- self() => de Pid (process-identifier) van het uitvoerend process
- receive => boodschappen uit de postbus halen en verwerken

spawn – maakt een proces aan

- spawn(Mod, Fun, Args)

- Mod is de module,
  Fun is een functie die Mod exporteert,
  Args is een **lijst** met de parameters die aan Fun worden meegegeven.

- **(Slecht) voorbeeld dat wel werkt:**
  spawn(teller, begin(), [123]).

```
-module(teller).
-export([loop/1, start/0, begin]).
start() -> loop(0).
begin() -> loop.
loop(N) ->
        receive
                inc -> loop(N+1);
                ...
        end.
```

Voorbeeld van een fout:

```
spawn(teller, start(), [123])
```

spawn_link

- Verbind het lot van de betrokken processen
- Pid_3 = spawn_link(wx, demo, []).
- Pid_3 crashes >> oproeper crashes (en omgekeerd)
- Ter informatie, dit kan worden omgezet naar het ontvangen van een boodschap.
- Ter informatie, proclib:spawn geeft meer debugging, tracing informatie (advanced).

spawn – geeft de proces identifier terug

- De functie-oproep

  Pid = spawn(Mod, Fun, Args)

  geeft de identifier (Pid) terug
  van het proces dat werd aangemaakt

- Normaliter vangt de aanmaker dit op:

  Teller_1 = teller:create(),
  Teller_2 = spawn(teller, loop, [0]).

```
-module(teller).
-export([create/0, ...]).
-export([loop/1]).

create() ->
      spawn(?MODULE, loop, [0]).

loop(N) -> ...
```

N.B. ?MODULE is een macro-oproep;
de compiler vervangt ?MODULE door teller.

Verbinding maken

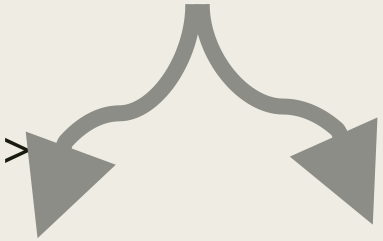
- Dienstverlener_Pid = spawn(M,F,[]),
  register(dienst_123, Dienstverlener_Pid),
  dienst_123 ! {self(), Msg}.

- register(dienst_abc, spawn(M,F,[]).

- Boodschap naar Dienstverlener_Pid die niet meer bestaat >> NOP
  Boodschap naar dienst_123 die niet meer bestaat >> crash

Verbinding maken

- Dienstverlener_Pid ! {self(), Msg}



- loop(MyState) ->

```
        receive
                {Pid, Msg}   ->     Pid ! Antwoord(Msg),
                                    NewState = ... ,
                                    loop(NewState);
                ...
        end.
```

! – stuurt een boodschap

- Pid ! Msg

  stuurt de boodschap Msg naar het process met identifier Pid …

- Dit plaatst Msg in de postbus van het Pid process.
- De zender blokkeert niet; de ontvanger beslist hoe/wanneer de postbus wordt geledigd.
- Voorbeeld:

  teller:incr(Teller_1).

```
-module(teller).
-export([create/0, incr/1, reset/1, ...]).
-export([loop/1]).

create() ->
    spawn(?MODULE, loop, [0]).

incr(Teller_Pid) ->
    Teller_Pid ! incr.

reset(Teller_Pid) ->
    Teller_Pid ! reset.

loop(N) -> ...
```

N.B. in java-stijl: Teller_1.incr()

receive – haalt boodschap uit postbus

```
-module(teller).
-export([create/0, reset/1, incr/1, get/1]).
-export([loop/1]). % only for create/0

reset(TellerPid) -> TellerPid ! reset.

incr(TellerPid) -> TellerPid ! incr.

get(TellerPid) ->
        TellerPid ! {get, self()},
        receive
                N -> N
        end.
```

```
loop(N) ->
        receive
                {get, Pid} ->
                                Pid ! N,
                                loop(N);
                reset ->
                                loop(0);
                incr ->
                                loop(N+1)
        end.
```

make_ref() –> verwarring voorkomen

De gebruiker maakt een antwoordfunctie aan:

- Sender = self(), Ref = make_ref(),
  F1 = fun(Msg) ->
  Sender ! {Ref, Msg} end.
- dienstverlener ! {F1, Verzoek},
  receive
  {Ref, Antw} -> {ok, Antw}
  end.

De dienstverlener:

- loop(Toestand) ->
  receive
  {Antw, Verzoek} ->
  X = doe(Verzoek),
  Antw(X), loop(...);
  ....
  end.

make_ref() –> verwarring voorkomen

De gebruiker maakt een antwoordfunctie aan:

- Sender = self(), Ref = make_ref(),
  F1 = fun(Msg) ->
  Sender ! {Ref, Msg} end.
- F2 = fun(Msg) ->
  self() ! {Ref = make_ref(), Msg} end.

  Waarom werkt F2 niet zoals gewenst/bedoeld?

- dienstverlener ! {F1, Verzoek},
  receive
  {Ref, Antw} -> {ok, Antw}
  end.

De dienstverlener:

- loop(Toestand) ->
  receive
  {Antw, Verzoek} ->
  X = doe(Verzoek),
  Antw(X), loop(...);
  ....
  end.

receive – time-out

```
get(TellerPid) ->
        TellerPid ! {get, self()},
        receive
                N -> {ok, N}
        after 1000 ->
                not_ok
        end.
```

```
loop(N) ->

        receive

                {get, Pid} ->
                                Pid ! N,
                                loop(N);

                ...
        end.
```

receive – postbus leegmaken

```
loop(State) ->
        receive
                _Any -> loop(State)
        after 0 ->
                doe_iets(State)
        end.
```

ets - tabellen

ets:new(Name, Options) -> tid() | atom()

Options >> Type, Access, named_table, ...

Default:
[**set**, **protected**, {keypos,1}, {heir,none}, {write_concurrency,false}, {read_concurrency,false}].

set,	public	named_table
ordered_set,	protected	
bag,	private	
duplicate_bag.		

ets - tabellen

ets:insert(Tab, ObjectOrObjects)

Objects moeten tuples zijn.

Andere functies:

delete/2,
lookup/2, first/1, next/2,
give_away/3,
to_dets/2, from_dets/2, ...