Errors in Concurrency

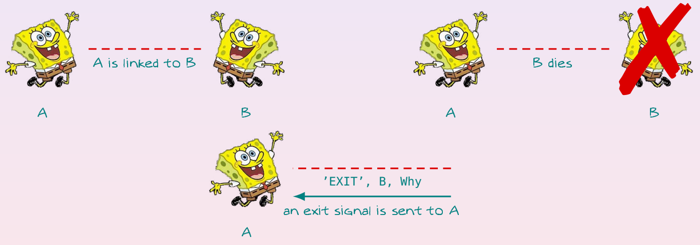
Walter Cazzola

Dipartimento di Informatica
Università degli Studi di Milano
e-mail: cazzola@di.unimi.it
twitter: @w_cazzola

Errors in Concurrent Programs
Error Handling on Exit

When two processes are related

- the errors of one affect the behavior of the other process;
- the BIF link function helps to monitor.



If A is linked to B

- when B dies an exit signal is sent to A;
- the signal is a message like {'EXIT', Pid, _}.

Errors in Concurrent Programs
Error Handling on Exit

```
-module(dies).
-export([on_exit/2]).

on_exit(Pid, Fun) ->
  spawn(fun() ->
      process_flag(trap_exit, true),
      link(Pid),
      receive
        {'EXIT', Pid, Why} -> Fun(Why)
      end
  end).
```

```
1> F = fun() -> receive X -> list_to_atom(X) end end.
#Fun<erl_eval.20.67289768>
2> Pid = spawn(F).
<0.35.0>
3> dies:on_exit(Pid, fun(Why) -> io:format("~p died with:~p~n",[Pid, Why]) end).
<0.37.0>
4> Pid ! hello.
<0.35.0> died with:{badarg,[{erlang,list_to_atom,[hello]}]}

=ERROR REPORT==== 9-Nov-2011::17:50:20 ===
Error in process <0.35.0> with exit value:  {badarg,[{erlang,list_to_atom,[hello]}]}
hello
```

Errors in Concurrent Programs
Details of Error Handling

Links

- defines an error propagation path between two processes;
- if a process dies an exit signal is sent to the other process;
- the set of processes linked to a given process is called link set.

Exit Signals

- they are generated by a process when it dies;
- signals are broadcast to all processes in the link set of the dying process;
- the exit signal contains an argument explaining why the process died (exit(Reason) or implicitly set).
- when a process «naturally dies» the exit reason is normal;
- exit signals can be explicitly sent via exit(Pid, X): the sender does not die («fake death»).

System Processes

- a non system process that receives a exit signal dies too;
- a system process receives the signal as a normal message in its mailbox;
- process_flag(trap_exit, true) transform a process into a system process.

Errors in Concurrent Programs
Details of Error Handling (Cont'd)

Receiver's Behavior

trap_exit	Exit Signal	Action
true	kill	dies & broadcasts it to its link set
true	X	adds {'EXIT', Pid, X} to the mailbox
false	normal	continues & the signal vanishes
false	kill	dies & broadcasts it to its link set
false	X	dies & broadcasts it to its link set

Alternatives

- I don't care if a process I create crashes.
  `Pid = spawn(fun() -> ... end)`
- I want to die if a process I create crashes.
  `Pid = spawn_link(fun() -> ... end)`
- I want to handle errors if a process I create crashes

```
process_flag(trap_exits, true),
Pid = spawn_link(fun() -> ... end).
```

Errors in Concurrent Programs
Going into Details of Error Handling

```
-module(edemo1).
-export([start/2]).

start(Bool, M) ->
  A = spawn(fun() -> a() end),
  B = spawn(fun() -> b(A, Bool) end),
  C = spawn(fun() -> c(B, M) end),
  sleep(1000), status(b, B), status(c, C).

a() -> process_flag(trap_exit, true), wait(a).
b(A, Bool) -> process_flag(trap_exit, Bool), link(A), wait(b).
c(B, M) -> link(B),
  case M of
    {die, Reason} -> exit(Reason);
    {divide, N} -> 1/N, wait(c);
    normal -> true
  end.
```

This starts 3 processes: A, B and C

- A will trap exits and watch for exits from B;
- B will trap exits if Bool is true and
- C will die with exit reason M.

Errors in Concurrent Programs
Going into Details of Error Handling (Cont'd)

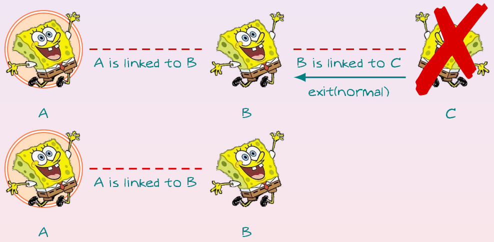
```
wait(Prog) ->
  receive
    Any ->
       io:format("Process ~p received ~p~n", [Prog, Any]),
       wait(Prog)
  end.

sleep(T) ->
  receive
  after T -> true
  end.

status(Name, Pid) ->
  case erlang:is_process_alive(Pid) of
    true -> io:format("process ~p (~p) is alive~n", [Name, Pid]);
    false -> io:format("process ~p (~p) is dead~n", [Name, Pid])
  end.
```

This starts 3 processes: A, B and C

- wait/1 just prints any message it receives;
- sleep/1 awakes the invoking process after a period of time;
- status/2 prints the aliveness of the invoking process.

Errors in Concurrent Programs
Going into Details of Error Handling (Cont'd)

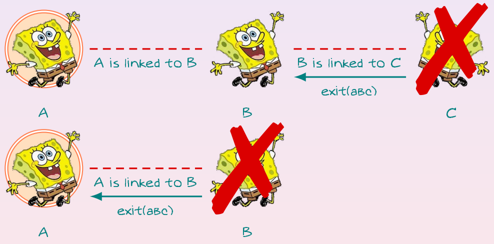
```
1> edemo1:start(false, {die,normal}).
process b (<0.48.0>) is alive
process c (<0.49.0>) is dead
ok
```



- B is not a system process;
- when C dies with normal signal, B doesn't die.

Errors in Concurrent Programs
Going into Details of Error Handling (Cont'd)

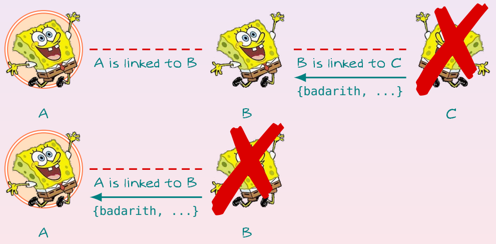
```
1> edemo1:start(false, {die, abc}).
Process a received {'EXIT',<0.40.0>,abc}
process b (<0.40.0>) is dead
process c (<0.41.0>) is dead
ok
```



- B is not a system process;
- when C evaluates exit(abc), process B dies;
- when B exits rebroadcasts the unmodified exit signal to its link set
- A traps the exit signal and convert it to the error message

Errors in Concurrent Programs
Going into Details of Error Handling (Cont'd)

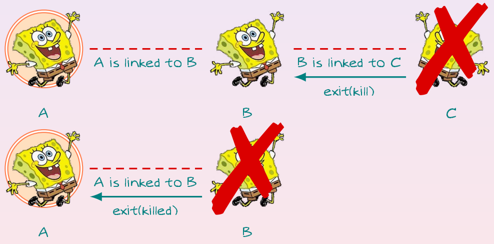
```
6> edemo1:start(false, {divide,0}).
Process a received {'EXIT',<0.56.0>,{badarith,[{edemo1,c,2}]}}
=ERROR REPORT==== 11-Nov-2011::18:03:29 ===
Error in process <0.57.0> with exit value: {badarith,[{edemo1,c,2}]}
process b (<0.56.0>) is dead
process c (<0.57.0>) is dead
ok
```



- B is not a system process;
- when C tries to divide by zero an error occurs and C dies with a {badarith, ...} error;
- B receives this and dies and the error is propagated to A

Errors in Concurrent Programs
Going into Details of Error Handling (Cont'd)

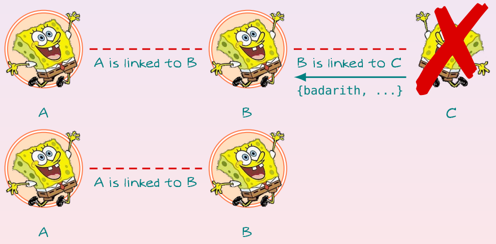
```
1> edemo1:start(false, {die, kill}).
Process a received {'EXIT',<0.60.0>,killed}
process b (<0.60.0>) is dead
process c (<0.61.0>) is dead
ok
```



- B is not a system process;
- the exit reason kill causes B to die, and the error is propagated to its link set.

Errors in Concurrent Programs
Going into Details of Error Handling (Cont'd)

```
8> edemo1:start(true, {divide,0}).
Process b received {'EXIT',<0.65.0>,{badarith,[{edemo1,c,2}]}}
=ERROR REPORT==== 11-Nov-2011::18:16:47 ===
Error in process <0.65.0> with exit value: {badarith,[{edemo1,c,2}]}
process b (<0.64.0>) is alive
process c (<0.65.0>) is dead
ok
```



- B is a system process;
- in all cases, B traps the error;
- the error is never propagated to A

Errors in Concurrent Programs
Monitors: Unidirectional Links

Links are symmetric

- i.e., if A dies, B will sent an exit signal and vice versa;
- to prevent a process from dying, we have to make it a system process that is not alway desirable.

A monitor is an asymmetric link

- if A monitors B and B dies A will be sent an exit signal but
- if A dies B will not be sent a signal.

A can create a monitor for B calling `erlang:monitor(process, B)`

- if B dies with exit reason `Reason` a `'DOWN'` message

  `{'DOWN', Ref, process, B, Reason}`

  is sent to A (`Ref` is the reference to the monitor).
- the monitor is unidirectional:
  - to repeat the above call will create several, independent monitors and each one will send a `'DOWN'` message when B terminates.

References

- Gul Agha.
  Actors: A Model of Concurrent Computation in Distributed Systems.

  MIT Press, Cambridge, 1986.
- Joe Armstrong.
  Programming Erlang: Software for a Concurrent World.
  The Pragmatic Bookshelf, fifth edition, 2007.
- Francesco Cesarini and Simon J. Thompson.
  Erlang Programming: A Concurrent Approach to Software Development.
  O'Reilly, June 2009.