Errors in Concurrency

Walter Cazzola

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

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.

A is linked to B

If A is linked to B
- when B dies an exit signal is sent to A;
- the signal is a message like ("EXIT", Pid, _).

- links
- monitors

References

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

<table>
<thead>
<tr>
<th>trap_exit</th>
<th>Exit Signal</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>kill</td>
<td>dies &amp; broadcasts it to its link set</td>
</tr>
<tr>
<td>false</td>
<td>normal</td>
<td>continues &amp; the signal vanishes</td>
</tr>
<tr>
<td>false</td>
<td>kill</td>
<td>dies &amp; broadcasts it to its link set</td>
</tr>
</tbody>
</table>

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:
  we use spawn_link and trap_exits

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 invoing process.

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

- 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

- A is linked to B
- 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 converts it to the error message

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.

1> 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 concurrency
walter cazzola

error handling
links
monitors
references
Errors in Concurrent Programs
Monitors: Unidirectional Links

Links are symmetric
- i.e., if A dies, B will send an exit signal and vice versa;
- to prevent a process from dying, we have to make it a system process that is not always 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 \texttt{erlang:monitor(process, B)}
- if B dies with exit reason \texttt{Reason} a 'DOWN' message
  \[
  \{'DOWN', \text{Ref}, \text{process, B, Reason}\}
  \]
  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.
- Joe Armstrong.
- Francesco Cesarini and Simon Thompson.
  O'Reilly, June 2009.