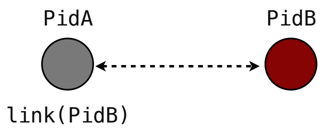
Erlang Solutions Ltd.

Process Error Handling

Overview: process error handling

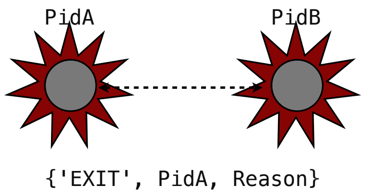
- Process Error Handling I
  - Links
  - Exit Signals
  - Definitions
  - Propagation Semantics
- Process Error Handling II

Links



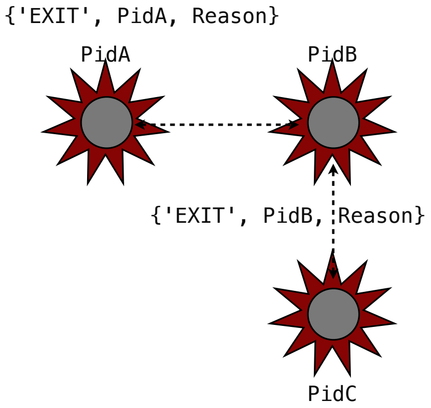
- **link/1** will create a bi-directional link between the process calling the BIF and the process **PidB**
- **spawn_link/3** will yield the same result as calling **spawn/3** followed by **link/1**, only that it will do it **atomically**

Links



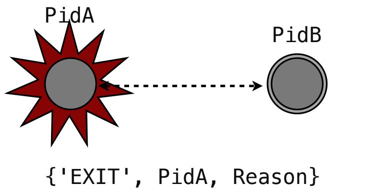
- **Exit Signals** are sent when processes terminate abnormally
- They are sent to all processes to which the failing process is currently linked to
- The process receiving the signal will exit, then propagate a new signal to the processes it is linked to

Links



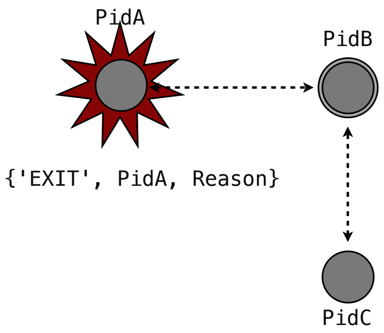
- When process **PidA** fails, the exit signals propagate to **PidB**
- From **PidB**, it propagates to **PidC**.

Exit Signals



- Processes can trap exit signals by calling the BIF **process_flag(trap_exit, true)**
- Exit signals will be converted to messages of the format **{'EXIT', Pid, Reason}**
- They are saved in the process mailbox
- If an exit signal is trapped, it does not propagate further

Exit Signals



- Process B marked with a double ring is trapping EXITs
- If an error occurs in A or C, then they will terminate.
- Process B will receive the **{'EXIT', Pid, Reason}** message
- The process that did not terminate will not be affected.

Definitions: terminology

Link

A bi-directional propagation path for exit signals set up between processes

Exit Signal

A signal transmitted by a process upon exiting. It contains termination information

Error Trapping

The ability of a process to handle exit signals as if they were messages

Definitions: built-in functions

link(Pid)

Set a link between the calling process and **Pid**

unlink(Pid)

Removes a link to **Pid**

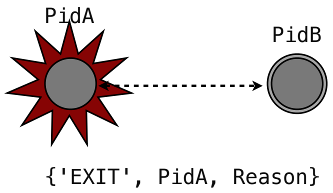
spawn_link(M,F,A)

Atomically spawns and sets a link between the calling and the spawned processes.

process_flag(trap_exit, Bool)

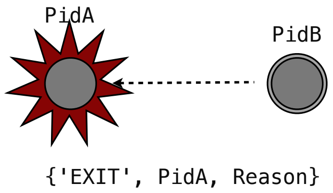
Sets the current process to convert exit signals into exit messages

Definitions: built-in functions



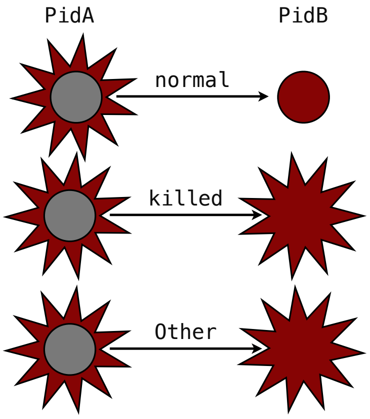
- the BIF **exit/1** terminates the process which calls it
- It generates an exit signal sent to linked processes
- The BIF **exit/1** can be caught in a **catch**.

Definitions: built-in functions



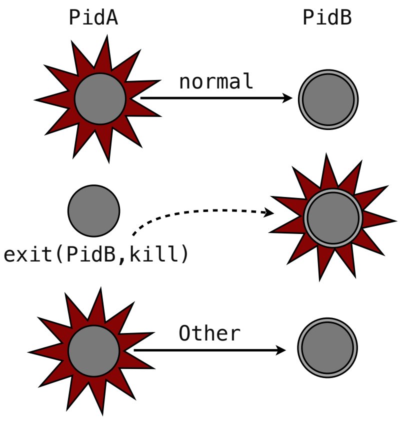
- **exit(Pid, Reason)** sends an exit signal containing **Reason** to the process **Pid**
- If trapping exits, the signal is converted to an exit message

Propagation Semantics: no trapping



- Nothing happens to PidB
- PidB terminates with reason '**killed**'
- PidB terminates with reason '**Other**'

Propagation Semantics: **trapping exits**



- PidB receives **{'EXIT', PidA, normal}**
- PidB terminates with reason '**killed**'
- PidB receives **{'EXIT', PidA, Other}**

Propagation Semantics

- When a process terminates, it sends an exit signal to the processes in its link set
- Exit signals can be **normal** or **non-normal**
- A process not trapping exits dies if it receives a non-normal one. Normal signals are ignored.
- A process which is trapping exit signals converts all incoming exit signals to conventional messages handled in a receive statement
- If the reason is **kill**, the process is terminated unconditionally

Summary: **process error handling I**

- Process Error Handling I
  - Links
  - Exit Signals
  - Definitions
  - Propagation Semantics
- Process Error Handling II

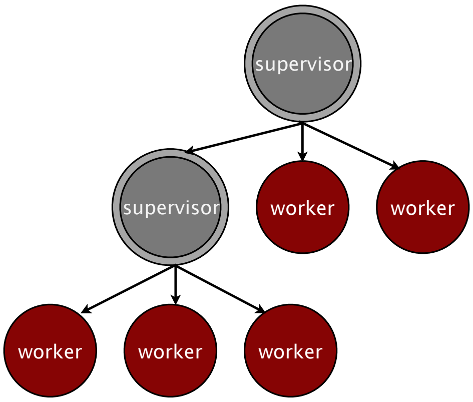
Overview: process error handling II

- Process Error Handling I
- Process Error Handling II
  - Robust Systems
  - A Robust Server

Robust Systems

- Building a system in layers can make it robust
  - Level N-1 traps and fixes errors occurring in level N
  - The leaves of the tree are workers
- In well designed systems, application programmers will not have to worry about error handling code
  - Error handling will be isolated by higher levels of the system, managed uniformly across processes
- Processes whose only task is to supervise children are called supervisors

Robust Systems



- Robust systems can be designed by layering

A Robust Server

- Remember the server example from the process design patterns section?
- The Server is unreliable!
  - What happens if the client crashes before it sends the release message?
- Let's rewrite the server making it reliable by monitoring the clients
  - If a client terminates before deallocating a frequency, the server will deallocate it automatically

A Robust Server

```
-module(frequency).
-export([start/0, stop/0,  allocate/0, deallocate/1]).
-export([init/0]).

start() ->
    register(frequency, spawn(frequency, init, [])).

init() ->
    process_flag(trap_exit, true),
    Frequencies = {get_frequencies(), []},
    loop(Frequencies).

get_frequencies() -> [10,11,12,13,14, 15].
```

A Robust Server

```
allocate({[], Allocated}, Pid) ->
    {{[], Allocated}, {error, no_frequencies}};
allocate({[Freq|Frequencies], Allocated}, Pid) ->
    link(Pid),
    {{Frequencies, [{Freq, Pid}|Allocated]}, {ok, Freq}}.

deallocate({Free, Allocated}, Freq) ->
    {value, {Freq, Pid}} =
      lists:keysearch(Freq, 1, Allocated),
    unlink(Pid),
    NewAllocated = lists:keydelete(Freq, 1, Allocated),
    {[Freq|Free],  NewAllocated}.
```

A Robust Server

```
loop(Frequencies) ->
  receive
    {request, Pid, allocate} ->
      {NewFreqs, Reply} = allocate(Freqs, Pid),
      reply(Pid, Reply),
      loop(NewFrequencies);
    ...
    {'EXIT', Pid, Reason} ->
      NewFrequencies = exited(Frequencies, Pid),
      loop(NewFrequencies);
    {request, Pid, stop} ->
      reply(Pid, ok)
  end.
```

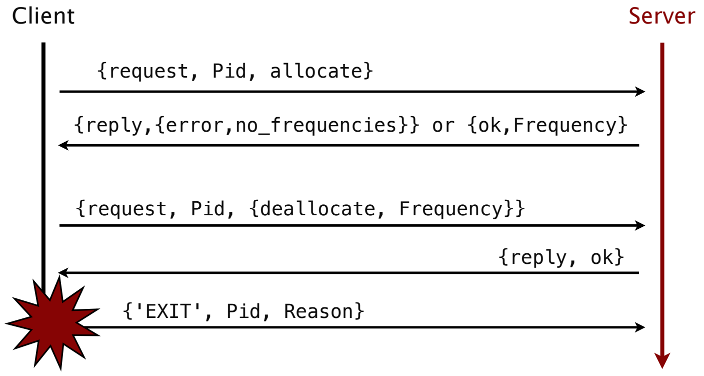
A Robust Server

```
%% Help functions used when a client crashes.

exited({Free, Allocated}, Pid) ->
  case  lists:keysearch(Pid, 2, Allocated) of
    {value, {Freq, Pid}} ->
      NewAllocated = lists:keydelete(Freq, 1, Allocated),
      {[Freq|Free],  NewAllocated};
    false ->
      {Free, Allocated}
  end.
```

The EXIT message was sent before the server unlinked, but after it released the frequency

A Server Example



Client

Server

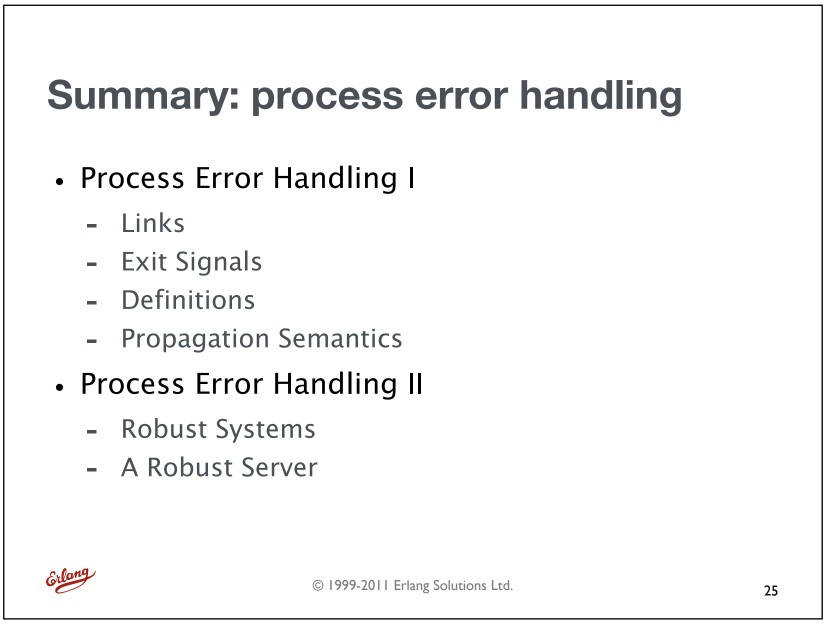
{request, Pid, allocate}

{reply,{error,no_frequencies}} or {ok,Frequency}

{request, Pid, {deallocate, Frequency}}

{reply, ok}

{'EXIT', Pid, Reason}



Summary: process error handling

- Process Error Handling I
  - Links
  - Exit Signals
  - Definitions
  - Propagation Semantics
- Process Error Handling II
  - Robust Systems
  - A Robust Server