Programming Distributed Systems

Erlang OTP

Annette Bieniusa, Albert Schimpf

AG Softech
FB Informatik
TU Kaiserslautern

Summer Term 2020

Error handling in Erlang

Two kinds of errors:

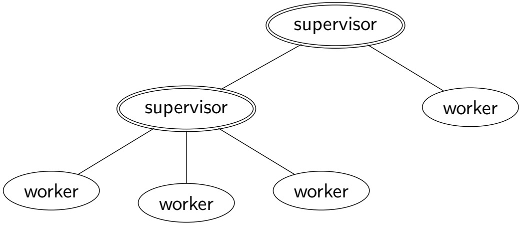
- Predictable errors
  - Wrong user input, connection problem, error reading file
  - Often handled with special return values, e.g. `read_file(Filename) -> {ok, Binary} | {error, Reason}`
  - Sometimes handled with exceptions
- Unpredictable errors
  - Software bugs, corrupt state, system resources exhausted
  - Handled by monitoring whole processes ($\Rightarrow$ supervisors)

Linked processes and monitoring

- Processes can be linked
    - A link has no direction
    - `spawn_link` spawns a new process and links it to the current
        - Also: `link` and `unlink` functions
    - If a process terminates, all linked processed are notified:
        - by default linked process terminates as well (with same reason)
        - if `process_flag(trap_exit, true)` is set, a special message `{'EXIT', Pid, Reason}` is sent instead
- Processes can be monitored
    - Only one direction
    - If monitored process terminates, monitoring process receives message `{'DOWN', MonitorRef, Type, Object, Info}`

Supervisors

- Start child processes (with link)
- Trap exits
- Handle termination of child processes (e.g. restart)
- Cleanly terminate applications
- Usually organized hierarchical



Generic Supervisor

Just implement callback `init/1` to specify the supervisor.

```
{ok, {SupFlags,[ChildSpec]}}.
```

SupFlags is a map with the following keys:

- `strategy`: Strategy for restarting children
  - `one_for_one`: Restart only terminated process (default value)
  - `one_for_all`: Restart all child processes
  - `rest_for_one`: Restart all processes, that were started after the terminating process
  - `simple_one_for_one`: Like `one_for_one`, but all children run the same code
- `intensity` (`MaxR`) and `period` (`MaxT`)
  - If more than `MaxR` number of restarts occur in the last `MaxT` seconds, the supervisor terminates all the child processes and then itself.

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Supervisor Children

ChildSpec is a a map with the following keys:

- `id`: Name of the child
- `start`: Tuple `{Module, Func, Args}` to call for initialization
- `restart`:
  - `permanent`: always restart
  - `temporary`: never restart
  - `transient`: restart only after crash
- `shutdown`: How long to terminate children
- `type`: `worker` or `supervisor`
- `modules`: `[ModuleName]` or `dynamic` (used for managing releases)

Children can be dynamically added and removed:

- `start_child(SupRef, ChildSpec)`
- `delete_child(SupRef, Id)`

Supervisor example

```
-module(example_sup).
-behaviour(supervisor).
-export([start_link/0, init/1]).
-export([stop/0]).

start_link() ->
    supervisor:start_link(?MODULE, []).

init(_) ->
    ChildSpecList = [child(service1), child(service2)],
    {ok,{{intensity => 2, period => 3600}, ChildSpecList}}.

child(Module) ->
    {id => Module, start => {Module, start_link, []},
     restart => permanent, shutdown => 2000}.
```

Erlang OTP

- Generic servers (`gen_server`)
- Generic Supervisors (`supervisor`)

More features:

- Generic state machine behavior `gen_statem` (different states accept different messages)
- Generic event handling behavior `gen_event` (multiple event handlers receive notification for one event)
- Applications, releases and release handling