MANAGING PROCESSES WITHOUT OTP

(and how to make them OTP-compliant)

Jay Nelson
@duomark

ERLANG PROCESSES

- Code only executes in a process
- Each process has its own memory space
- Serial execution logic
- Separate garbage collector for each

CREATING A PROCESS

- Spawn for independent process
- Spawn_link for linked process
  - Abnormal exit takes down all linked processes
- Spawn_monitor for monitored process
  - Parent receives a message when it goes down

NEW PROCESS CREATES

- Encapsulated empty memory space
- Local process dictionary of Key/Value pairs
- Execution begins with passed in function
- Returns a Pid reference to the process
- Format is <N.PPP.I> (node, process, int)

EXAMPLE SPAWNED PROCESSES

```
-module(procs).
-export([get_dict/0]).

get_dict() ->
    Dict = process_info(self(), dictionary)
    error_logger:info_msg("dict: ~p~n", [Dict]).
```

```
Eshell
1> spawn(procs, get_dict, []).

     =INFO REPORT===
     dict: {dictionary, []}

<0.150.0>
```

EXAMPLE SPAWNED PROCESSES

```
2> F1 = fun() ->
           error_logger:info_msg("dict: ~p~n",
              [process_info(self(), dictionary)])
end.

Eshell
3> spawn(F1).

    =INFO REPORT===
    dict: {dictionary, []}

<0.151.0>
```

COMMUNICATION BETWEEN PROCESSES

- Sending
  - Pid ! Msg
  - erlang:send(Pid, Msg)
  - erlang:send(Pid, Msg, Options)
- Message mailbox
  - maintains list of messages in order arrived
  - guaranteed in order sent if no middle process

IN-ORDER RECEIVE

```
receive
  Any_Msg -> handle(Any_Msg)
end.
```

- Dequeues all messages
- Synchronously handles each one as it is dequeued

OUT-OF-ORDER (SELECTIVE) RECEIVE

```
receive
 {my_msg, Data} -> handle_my_way
after 100 -> continue
end.
```

- Only messages tagged my_msg are handled
- Other messages left in mailbox in original order

GEN_SERVER/GEN_FSM ET AL

- start_link to spawn a new process
- Pushes messages (no receive in user code)
- Supports sync, async erlang messages
- Supports TCP raw data as info messages
- Allows for controlled code change
- Cleanup of state on shutdown

SUPERVISORS

- Link gen servers into a hierarchy
- Allows controlled startup
- Can shutdown branches of an application
- Purpose is automated restart on failure

4 REQUIREMENTS TO BE OTP-COMPATIBLE

- Use proc_lib to spawn processes
- Handle {system, From, Msg} messages
  - plain system messages
  - `sys:handle_system_msg/6` implements
- Respond properly to shutdown messages
  - `{'EXIT', Parent, Reason} -> exit(Reason)`
- Handle `{get_modules, From}` for code upgrade

SYSTEM-LEVEL TOOLS FOR MANAGING PROCESSES

- proc_lib for creating
- Sys for controlling and debugging
- Erlang for tracing
  - VM capability, not discussed further here

PROC_LIB

```
3> proc_lib:spawn(F1)
        =INFO REPORT===
        dict: {dictionary,[{'$ancestors'[<0.156.0>]},
                           {'$initial_call',
                            {erl_eval,'-expr/5-fun-1-',0}}]}
<0.158.0>
```

- Allows tracking the spawn hierarchy
- Reports crashes with linked process context

HIBERNATING

- VM capability to shrink a process
- Removes call stack (including pending try/catch)
- Full garbage collect
- Resizes process to smallest possible
- Even if smaller than minimum process size
- gen_server/gen_fsm support this in API

HIBERNATING RAW PROCESSES

- proc_lib:hibernate(M,F,A)
  - invokes erlang:hibernate/3
  - identifies wake call
  - reinstalls proc_lib crash reporting
- When message received
  - uncompress, resize to minimum if too small
  - wake function is called

EXAMPLE: PAUSING MESSAGE RECEIVE

- jump to a selective receive loop
- resume back to original loop later

```
loop() ->
    receive
      {pause, Amt} -> wait(Amt);
      Msg -> handle(Msg)
    end.

wait(Amt) ->
    receive
    after Amt -> loop()
    end.
```

OTP APPROACH

- sys:suspend transfers loop control to OTP sys code
- Never returns (directly)
- OTP receive loop only recognizes system messages
- sys:resume transfers back to user's receive loop

OTP APPROACH (CONT.)

- sys:resume/2,3 calls user's code
- User provided callback functions required:
  - ?MODULE:system_continue/3
  - ?MODULE:system_terminate/4
  - ?MODULE:system_code_change/4

WHEN A SYSTEM MSG ARRIVES:

- Call sys:handle_system_msg/6

```
loop(Debug_Opts) ->
      receive
        {system, From, Msg} ->
          sys:handle_system_msg(Msg, From, Parent,
                                ?MODULE, Debug_Opts, Extra);
        Msg ->
          handle_msg(Msg),
          loop(Debug_Opts).
      end.
```

3 RETURNS FROM SYS ALTERNATE RECEIVE LOOP

- ?MODULE:system_continue/3
  - user function jumps to user receive loop
    - never returns (counterpart to sys:suspend)
    - called by sys:resume
- ?MODULE:system_terminate/4
  - supervisor issues terminate request
  - user function should exit with same reason

3 RETURNS FROM SYS ALT RECEIVE LOOP (CONT)

- ?MODULE:system_code_change/4
  - user function migrates any data structures
  - then returns the Extra data
  - sequence: suspend, code_change, resume
  - implemented by reltool on app upgrade/downgrade

DEBUG

- Manually insert trace/logging code into actual logic
- Initialize multiple options with sys:debug_options/ I
- Sets types of debugging to enable
- OR, call a function for each type of debug option
- Trace, log, statistics, install custom function

RECORDING DEBUG INFORMATION

- Low overhead method to debug manually chosen events
- Events are written to circular queue (in Debug_Opts)
- Defaults to 10 events, can be overridden
- sys:handle_debug/4 called to create an event
- User provides Format_Func(Device, Event, Extra)
- Custom format function for logging event

VIEWING LOGGED DEBUG INFORMATION

- `sys:print_log(Debug_Opts)` => prints debug queue

- `sys:log(Pid, print)` => prints debug queue

- `sys:log(Pid, get)` => gets the debug queue

- `sys:log_to_file(Pid, Filename)` => log to a file (not RAM)

INSTALLING TRACE FUNCTIONS

- sys:install(Pid, {Function_Name, Init_State})
- Installed function inspects current state
- returns new state, reinstalls function
- returns done, uninstalls function
- sys:remove(Pid, Function_Name)
- allows manual turn off

STATISTICS COLLECTION

- sys:statistics/2,3: elapsed time, reductions and message counts
- Uses install/remove to invoke sys implemented capability
- Requires manually inserting sys:handle_debug/4 in your code
- Options to start or end collection, and report statistics

REPORT COMPACT SUMMARY OF PROCESS STATE

- sys:get_status/1,2 returns a formatted summary
- User provides ?MODULE:format_status/2
- Added to gen_server/gen_fsm when crash reports killed VM

DYNAMIC MODULE CODE CHANGE

- Release handler requests dynamic modules for code change
- Sends `{get_module, From}`
- Process should respond `From ! {modules, Modules}`
- Lists all currently executing Modules

DO IT YOURSELF!

- http://www.erlang.org/doc/design_principles/spec_proc.html
- http://www.erlang.org/doc/man/proc_lib.html
- http://www.erlang.org/doc/man/sys.html
- Example usage inside Erlang/SP library:
  - https://github.com/duomark/erlangsp
  - /apps/coop/src/coop_node_data_rcv.erl

QUESTIONS