Extended Process Registry for Erlang

Ulf Wiger
Ericsson AB

ERICSSON
TAKING YOU FORWARD

Problems

- How to debug a system with many processes?
- How to represent dependencies between processes?
- How to find dependencies between processes and ”things”?

Debugging a live system

(c.erl)

```
i() ->
    Ps = processes(),
    i(Ps, length(Ps)).

i(Ps, N) when N =< 100 ->
    iformat("Pid", "Initial Call", "Heap", "Reds", "Msgs"),
    iformat("Registered", "Current Function", "Stack", "", ""),
    {R,M,H,S} = foldl(fun(Pid, {R0,M0,H0,S0}) ->
                              {A,B,C,D} = display_info(Pid),
                              {R0+A,M0+B,H0+C,S0+D}
                      end, {0,0,0,0}, Ps),
    iformat("Total", "", w(H), w(R), w(M)),
    iformat("", "", w(S), "", "");
```

- Starts by building a list of all Pids in the system
- Erlang supports 134 million concurrent processes...
- No convenient means of selecting a subset

Debugging a live system (2)

```
<0.15.0>            inet_db:init/1                233      108    0
inet_db             gen_server:loop/6              12
<0.17.0>            global_group:init/1           233       73    0
global_group        gen_server:loop/6              12
<0.18.0>            file_server:init/1          10946    32183    0
file_server_2       gen_server:loop/6              12
<0.22.0>            group:server/3                4181     3724    0
user                group:server_loop/3              4
<0.23.0>            group:server/3                610     9806    0
                    group:server_loop/3              4
<0.24.0>            kernel_config:init/1          233       49    0
                    gen_server:loop/6              12
<0.25.0>            supervisor:kernel/1           233       61    0
kernel_safe_sup     gen_server:loop/6              12
```

- How to spot reliably what behaviour a process uses? (without looking into the source)
- Lots of obscure dependencies
  - which processes to suspend before upgrading a module?
  - which processes subscribe to certain events?
  - which open files does a process have?
  - ...

Finding the Right Process

(dbg_ieval.erl)

```
eval(Mod, Func, Args) ->
    Debugged = self(),
    Int = dbg_iserver:find(),
    case dbg_iserver:call(Int, {get_meta,Debugged}) of
        {ok,Meta} ->
            Meta ! {re_entry, Debugged, {eval,{Mod,Func,Args}}},
            Meta;
        {error,not_interpreted} ->
            spawn(fun() ->
                        meta(Int, Debugged, Mod, Func, Args)
                end)
    end.
```

- Here, a central server which keeps track of Module <-> Pid mapping
- Other applications might use an ets table
- Lots of similar code fragments here and there

Which Process ”owns” a File?

```
[{file, N, O} ||
    {N, {monitors,[{process,O}]}} <-
          [{N, process_info(IOS,monitors)} ||
              {IOS,N} <- ets:tab2list(file_io_servers)].
```

- Requires knowledge of the file_server.erl source
- Only works for some types of file
- File_server keeps a private File <-> Pid mapping

Naming processes

(observer_backend.erl)

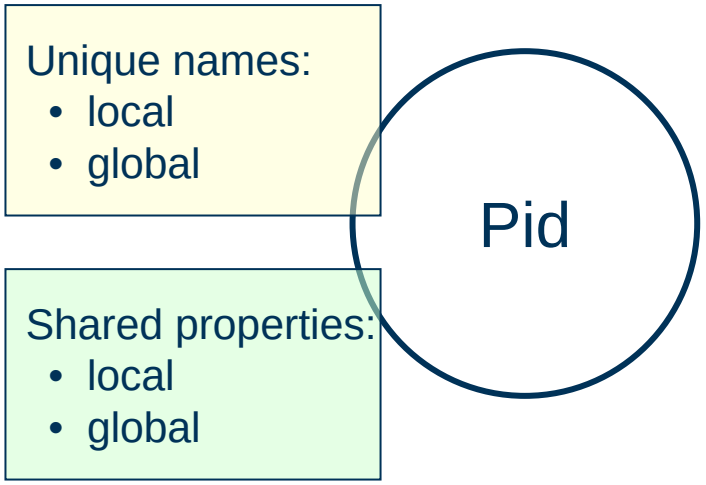
```
pinfo(P,Globals) ->
    case process_info(P,registered_name) of
        [] -> case lists:keysearch(P,1,Globals) of
                  {value,{P,G}} -> {pid,{P,{global,G}}};
                  false ->
                      case process_info(P,initial_call) of
                          {_,I} -> {pid,{P,I}};
                          undefined -> [] % the process has terminated
                      end
              end;
        {_,R} -> {pid,{P,R}};
        undefined -> [] % the process has terminated
    end.
```

- Local registry: only atoms (no structured names)
  - Each process can only have one name
- Global registry: an add-on; structured names
  - Allows multiple names/process, but prints a warning
- Different API and semantics for global & local scope
- Neither has a good search facility

gproc: Extended Process Registry

- A common registry for publishing the ”footprint” of a process
- Ordered set semantics – searchable with QLC
- Symmetrical support of both global and local scope

- Shared properties enable grouping related processes in a uniform way



Logical Data Structure

```
{Key, Pid, Value}
Key :: {Type, Context, Name}
Type :: n | p | c | a
Context :: g | l
```

Type:

```
n = name
p = property
c = counter
a = aggregated counter
```

Context:

```
g = global
l = local
```

Abbreviated for compact shell printouts

Value is only sometimes useful, but always present for symmetry

Examples (patched OTP)

```
=PROGRESS REPORT==== 5-Jul-2007…
application: sasl
started_at: nonode@nohost
Eshell V5.5.5 (abort with ^G)
1> Q1 = qlc:q([{P,Fs} ||
        {{p,l,supflags},P,Fs} <-
        gproc:table(props)]).
{qlc_handle,{qlc_lc,…}}
2> qlc:eval(Q1).
[{<0.10.0>,{one_for_all,0,1}},
{<0.27.0>,{one_for_one,4,3600}},
{<0.32.0>,{one_for_one,0,1}},
{<0.33.0>,{one_for_one,4,3600}}]
```

```
3> Q2 = qlc:q([P ||
{{p,l,behaviour},P,supervisor} <-
gproc:table(props)]).
{qlc_handle,…}
4> qlc:eval(Q2).
[<0.10.0>,<0.27.0>,<0.32.0>,<0.33.0>]
```

(gen.erl)

```
reg_behaviour(B) ->
    catch begin
              Key = {p,l,behaviour},
              try gproc:reg(Key, B)
              catch
                  error:badarg ->
                      gproc:set_value(Key, B)
              end
          end.
```

(supervisor.erl)

```
init({SupName, Mod, Args}) ->
    process_flag(trap_exit, true),
    gen:reg_behaviour(?MODULE),
    case Mod:init(Args) of
        {ok, {SupFlags, StartSpec}} ->
            gproc:reg({p,l,supflags}, SupFlags),
            case init_state(...) of
                ...
            end;
        ignore ->
            ignore;
        Error ->
            {stop, {bad_return, {...}}}
    end.
```

Publish/Subscribe with gproc

```
subscribe(Event) ->
    gproc:reg({p, l, {?MODULE,subs,Event}}, []).

notify(Event, Info) ->
    Q = qlc:q([P ! {self(), {?MODULE, Event, Info}} ||
          {{p,l,{?MODULE,subs,Event}},P,_} <- gproc:table(props)]),
    qlc:eval(Q).

list_subscribers(Event) ->
    Q = qlc:q([P ||
          {{p,l,{?MODULE,subs,Event}},P,_} <- gproc:table(props)]),
    qlc:eval(Q).
```

- gproc:info(Pid, gproc) will list all published info for process Pid, including all events it subscribes to.
- No special process or handler needed to provide a notification service.
- One pattern, one place to look during debugging.
- (Note: simple iterators could be added on top of QLC).

Performance Comparison

Mapping + Reverse mapping

Ets	gproc
1	3

[Ets] Assumes local op within existing process
*[gproc] local property registration < 15 us**

Local name registration

Reg BIF	gproc
1	68.8

gproc is still pretty fast:
local name registration < 40 us

Global name registration (4 nodes)

global	gproc
1	0.16

global is really very slow.

** on a 2 GHz Pentium/Linux*

Experiences

- We have used a gproc predecessor in product development.
- Significant reduction in code volume.
- Developers have re-written more and more code to use the process registry.

- Some modifications to gen_leader yet unverified. One gen_leader problem remains.

Questions?

SVN: http://svn.ulf.wiger.net/gproc