Debugging a Release

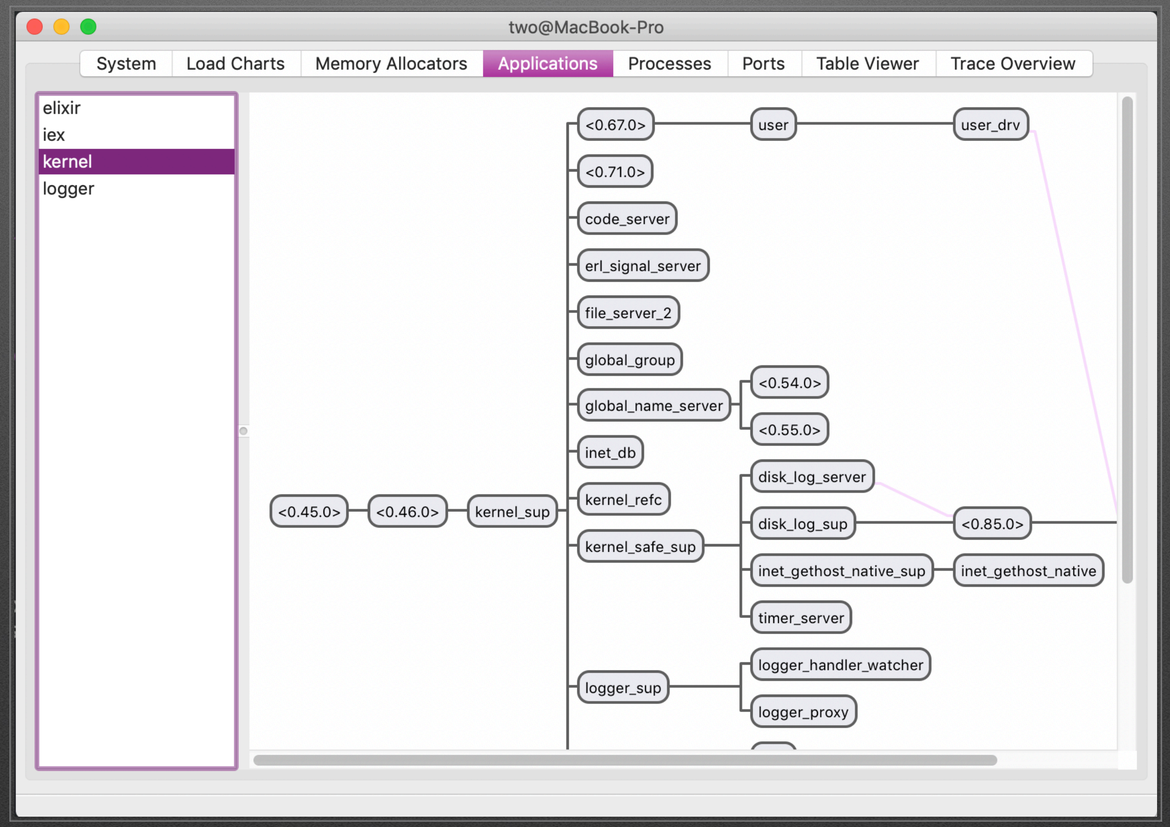
Release the Kraken… Err… Elixir!

What Do We Need to Know?

- Topology
- Events
- Metrics
- Messages Sent
- Functions Called

Topology

`:observer.start`



```elixir
def application do
  [
    extra_applications: [:logger, :observer, :runtime_tools, :wx],
    mod: {Exercise.Application, []}
  ]
end
```

`:observer` Config

`:observer_cli.start`

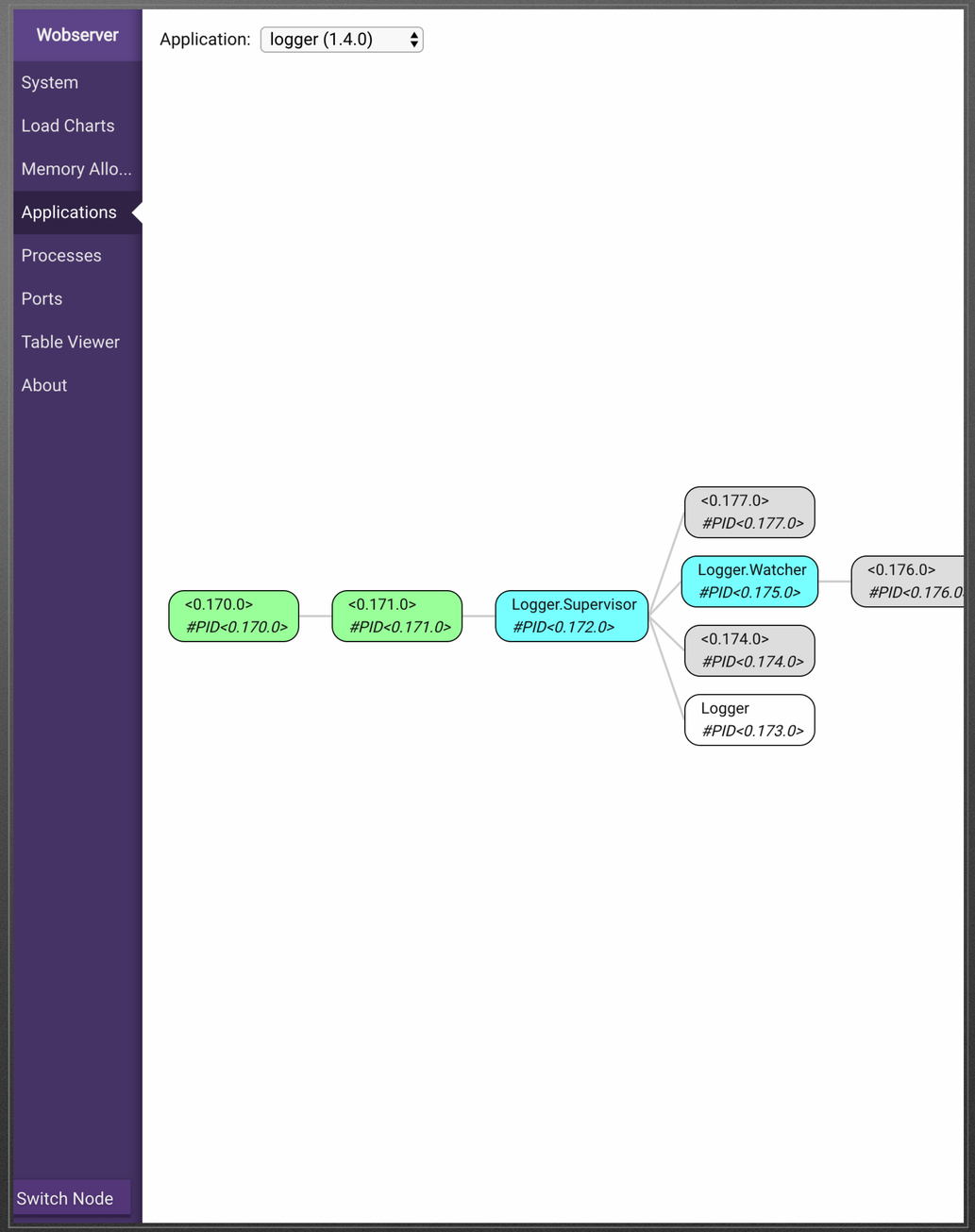
```
|Home(H)|Network(N)|System(S)|Ets(E)|Mnesia(M)|App(A)|Doc(D)|Plugin(P)|recon:proc_count(memory, 45) Interval:1500ms    | 0Days 0:0:33      |
|Erlang/OTP 22 [erts-10.4.4] [source] [64-bit] [smp:16:16] [ds:16:16:10] [async-threads:1] [hipe]                                |
|System      | Count/Limit            | System Switch            | Status                  | Memory Info             | Size                    |
|Proc Count  | 75/262144              | Smp Support              | true                    | Allocted Mem            | 102.9492 MB     | 100.0% |
|Port Count  | 6/65536                | Multi Scheduling         | enabled                 | Use Mem                 | 64.8810 MB      | 63.02% |
|Atom Count  | 29296/1048576          | Logical Processors       | 16                      | Unuse Mem               | 38.0683 MB      | 36.98% |
|Mem Type    | Size                   | Mem Type                 | Size                    | IO/GC                   | Interval: 1500ms        |
|Total       | 68.5828 MB    | 100.0% | Binary                   | 99.3516 KB    | 00.14% | IO Output               | 9.9688 KB               |
|Process     | 22.5379 MB    | 32.86% | Code                     | 22.9483 MB    | 33.46% | IO Input                | 122 B                   |
|Atom        | 790.6602 KB   | 01.13% | Reductions               | 133622                  | Gc Count                | 11                      |
|Ets         | 2.4654 MB     | 03.59% | RunQueue                 | 1                       | Gc Words Reclaimed      | 285366                  |
|1                        00.02% |9                        00.51% |17                       00.00% |25                       00.00% |
|2                        00.02% |10                       00.02% |18                       00.00% |26                       00.00% |
|3                        00.02% |11                       00.02% |19                       00.00% |27                       00.00% |
|4                        00.02% |12                       00.02% |20                       00.00% |28                       00.00% |
|5                        00.02% |13                       00.02% |21                       00.00% |29                       00.00% |
|6                        00.02% |14                       00.02% |22                       00.00% |30                       00.00% |
|7                        00.02% |15                       00.02% |23                       00.00% |31                       00.00% |
|8                        00.02% |16                       00.02% |24                       00.00% |32                       00.00% |
|No | Pid          |     Memory     |Name or Initial Call                |          Reductions| MsgQueue |Current Function          |
|1  |<0.860.0>     |     8.7840 MB  |code_server                         |            217419| 0        |code_server:loop/1        |
|2  |<0.859.0>     |     6.3570 MB  |kernel_sup                          |              5710| 0        |gen_server:loop/7         |
|3  |<0.9.0>       |     1.0634 MB  |erl_prim_loader                     |             61582| 0        |erl_prim_loader:loop/3    |
|4  |<0.854.0>     |   449.6016 KB  |application_controller              |             19128| 0        |gen_server:loop/7         |
|5  |<0.948.0>     |    33.9375 KB  |Elixir.IEx.Evaluator:init/4         |              2176| 0        |io:execute_request/2      |
|6  |<0.950.0>     |    22.8320 KB  |erlang:apply/2                      |               189| 0        |observer_cli_store:loop/1 |
|7  |<0.905.0>     |    21.1367 KB  |release_handler                     |              1679| 0        |gen_server:loop/7         |
|8  |<0.867.0>     |    13.5391 KB  |net_sup                             |               442| 0        |gen_server:loop/7         |
|9  |<0.874.0>     |    13.4961 KB  |file_server_2                       |               282| 0        |gen_server:loop/7         |
|10 |<0.878.0>     |    13.4609 KB  |supervisor_bridge:user_sup/1        |               212| 0        |gen_server:loop/7         |
|11 |<0.932.0>     |    11.7109 KB  |timer_server                        |              2135| 0        |gen_server:loop/7         |
|12 |<0.934.0>     |    11.5391 KB  |inet_tcp_dist:do_accept/7           |              1397| 0        |dist_util:con_loop/2      |
|13 |<0.870.0>     |    10.7227 KB  |net_kernel                          |              1492| 0        |gen_server:loop/7         |
|14 |<0.929.0>     |     8.7227 KB  |Elixir.Buggy.Supervisor             |               469| 0        |gen_server:loop/7         |
|15 |<0.884.0>     |     8.7227 KB  |logger_sup                          |               440| 0        |gen_server:loop/7         |
```

```elixir
defp deps do
  [
    {:observer_cli, "~> 1.5"}
  ]
end
```

`:observer_cli` Config

:wobserver

https://github.com/shinyscorpion/wobserver



Simple Functions

- `Node.list`
- `Process.registered`
- `:global.registered_names`
- `Supervisor.which_children(Supervisor.Name)`
- `Process.whereis(Local.Process.Name)`
- `:global.whereis_names(Global.Process.Name)`

Events

Simple Functions

- `:net_kernel.monitor_nodes(true)`
- `Process.alive?(pid)`
- `Process.monitor(pid)`

Metrics

Simple Functions

- `Process.info(pid, :memory)`
- `Process.info(pid, :message_queue_len)`
- `Process.info(pid, :messages)`
- `Process.info(pid, :reductions)`
- `:sys.get_state(pid)`

Live Tracing

`iex -S mix` Verses `my_release remote`

"The Heroku Problem"

Tools

- `:erlang.trace/3`: all powerful but arcane and dangerous
- `:dbg`: safer but still pretty arcane
- `:recon`: a dependency, safer, easier to understand, but for functions only

```ruby
def application do
  [
    extra_applications: [:logger, :runtime_tools],
    mod: {Exercise.Application, []}
  ]
end
```

`:dbg` Config

```
iex(1)> :dbg.h
The following help items are available:
   p, c
       - Set trace flags for processes
   tp, tpl, ctp, ctpl, ctpg, ltp, dtp, wtp, rtp
       - Manipulate trace patterns for functions
   n, cn, ln
       - Add/remove traced nodes.
   tracer, trace_port, trace_client, get_tracer, stop, stop_clear
       - Manipulate tracer process/port
   i
       - Info

call dbg:h(Item) for brief help a brief description
of one of the items above.

:ok
iex(2)> :dbg.h :p
p(Item) -> {ok, MatchDesc} | {error, term()}
 - Traces messages to and from Item.
p(Item, Flags) -> {ok, MatchDesc} | {error, term()}
 - Traces Item according to Flags.
   Flags can be one of s,r,m,c,p,sos,sol,sofs,
   sofl,all,clear or any flag accepted by erlang:trace/3

:ok
```

`:dbg` "Help"

https://zorbash.com/post/debugging-elixir-applications/

```elixir
# start tracing:
device = Process.group_leader
trace_fn = fn msg, _state -> IO.puts(device, inspect(msg)) end
:dbg.tracer(:process, {trace_fn, :initial_state})

# specify what to trace here…

# IMPORTANT, stop and clean up:
:dbg.stop_clear
```

What Tracing Looks Like

Memorize This Pattern

```elixir
# start tracing:
device = Process.group_leader
trace_fn = fn msg, _state -> IO.puts(device, inspect(msg)) end
:dbg.tracer(:process, {trace_fn, :initial_state})

# specify what to trace here…

# IMPORTANT, stop and clean up:
:dbg.stop_clear
```

What Tracing Looks Like

Memorize This Pattern

```elixir
# start tracing:
device = Process.group_leader
trace_fn = fn msg, _state -> IO.puts(device, inspect(msg)) end
:dbg.tracer(:process, {trace_fn, :initial_state})

# specify what to trace here…

# IMPORTANT, stop and clean up:
:dbg.stop_clear
```

What Tracing Looks Like

Memorize This Pattern

```elixir
# start tracing:
device = Process.group_leader
trace_fn = fn msg, _state -> IO.puts(device, inspect(msg)) end
:dbg.tracer(:process, {trace_fn, :initial_state})

# specify what to trace here…

# IMPORTANT, stop and clean up:
:dbg.stop_clear
```

What Tracing Looks Like

Memorize This Pattern

```elixir
# start tracing:
device = Process.group_leader
trace_fn = fn msg, _state -> IO.puts(device, inspect(msg)) end
:dbg.tracer(:process, {trace_fn, :initial_state})

# specify what to trace here…

# IMPORTANT, stop and clean up:
:dbg.stop_clear
```

What Tracing Looks Like

Memorize This Pattern

Messages

Process Tracing

- `:dbg.p(pid, [:r])`: messages received by `pid`
- `:dbg.p(pid, [:s])`: messages sent from `pid`
- `:dbg.p(:new_processes, [:s])`: messages from processes spawned after this
- `:dbg.p(pid, [:s, :sos])` (set on spawn): messages from `pid` or children
- And more…

Functions

`some_module.__info__(:functions)`

Finding Functions to Trace

Call Tracing

- First start call tracing one or more processes: `:dbg.p(pid, [:call])`
  - `:dbg.tp(Some.Module, :_, [])`: trace all function calls to `Some.Module`
  - `:dbg.tp(Some.Module, :handle_info, 2, [])`:
    trace `Some.Module.handle_info/2` calls
  - `:dbg.tp(Some.Module, :handle_info, 2, [{:_, [], [{:return_trace}]}])`:
    trace `Some.Module.handle_info/2` return values
  - And more…

Exercise: Learn Everything You Can From a Running Release

- What are the processes?
- What are they saying to each other?
- Why is the system buggy?
- What does it do?