Registered Processes

- A perennial problem in client-server computing is “How do clients find the server?”
- Two possible answers:
  - Clients are told about the servers as parameters when the clients are started
    - Lots of parameters; startup order dependency
  - Clients use a *name service* to locate the servers
- Erlang answer: registered processes
  - `register(AnAtom, Pid)`
  - Can then send to the atom as if it were a Pid.
- Note: mutable global state!

Concurrency and Error Handling

- Linked processes
- Monitored processes
- System processes
- Error handling idioms
- Fault tolerance 101
- Keep-alive process

Linked Processes

- link(Pid)
- spawn_link(Fun)
  - Not the same as `Pid = spawn(Fun), link(Pid)`
- The set of links partitions the set of processes into a collection of equivalence classes
  - Equiv. classes == symmetric, transitive, and reflexive
  - If any process in a class exits abnormally (other than by returning from its top function or by explicit normal exit) all the processes exit
  - Mechanism is called *exit signals* {'EXIT', Pid, Reason}

Monitored Processes

- Not symmetric or transitive
- Instance of a more general mechanism for notifying processes of system state changes
- Not considered further

System Processes

- I lied: if all linked processes died together there'd be no opportunity to recover – no process that continued to live would know about the failure
- `process_flag(trap_exit, true)` turns the calling process into a *system process* that instead of dying, receives a message in its mailbox when a linked sibling dies
  - `{'EXIT', Pid, Reason }` is the message format

Explicit exit

- exit/1 – `exit(Reason)` – a process is exiting and telling its linked siblings why
- exit/2 – `exit(Pid2, Reason)` – fake death: if caller has Pid1, this sends an exit signal to Pid2 as if Pid1 had died (but it doesn't die)
- `kill` as Reason: die and send `killed` as reason to link set. `kill` cannot be caught, even by system processes

Error Handling Idioms

- Don’t care about child’s fate
  - `Pid = spawn(Fun)`
- Die with child
  - `Pid = spawn_link(Fun)`
- Be notified if child dies

```
process_flag(trap_exit, true),
Pid = spawn_link(Fun),
loop() ->
   receive
      {‘EXIT’, SomePid, Reason} -> % handle it
      …
   end
```

Fault tolerance 101

- Key idea: redundancy – some resource must take over for another one that fails
  - Taking over means you have to know about the failure
    - Language-level link and monitor mechanisms
  - Being ready to take over means knowing something about the state of the failed resource as of the time of failure
    - Parallel execution, periodic state update, …

Keep-alive

- Common problem: ensure that a service is "continuously" available, even if it sometimes crashes
- An Erlang service is typically provided by a process whose body is a function and is registered under some name
- Thus, we'd like an encapsulated solution to the problem: spawn an arbitrary function, register it, monitor its existence, and respawn it if it crashes `keep_alive(Name, Fun)`

Proposed solution, Part 1

```
on_exit(Pid, Fun) ->
    spawn(fun() ->
        process_flag(trap_exit, true),
        link(Pid),
        receive
            {'EXIT', Pid, Why } ->
                Fun(Why)
        end
    end).
```

- A new process linked to Pid
- Fun(Why) is called in that new process

Proposed solution, Part 2

```
keep_alive(Name, Fun) ->
    register(Name, Pid = spawn(Fun)),
    on_exit(Pid, fun(_Why) ->
        keep_alive(Name, Fun) end).
```

- Shared state rears its ugly head
  - Race between multiple registerers
  - Race between register and link
  - Race between dying spawned process and link operation

How can it be fixed

- **Can’t use on_exit**
- **spawn_link apparently has to occur *in the process that will do the monitoring***

A better solution

```
keep_alive(Name, Fun) ->
   spawn(fun () ->
      process_flag(trap_exit, true),
      ka_helper(Name, Fun) end).
ka_helper(Name, Fun) ->
   register(Name, Pid = spawn_link(Fun)),
   receive
      {‘EXIT’, Pid, _Why } ->
         ka_helper(Name, Fun)
   end.
```

- Still have the registration race but the link race is gone.