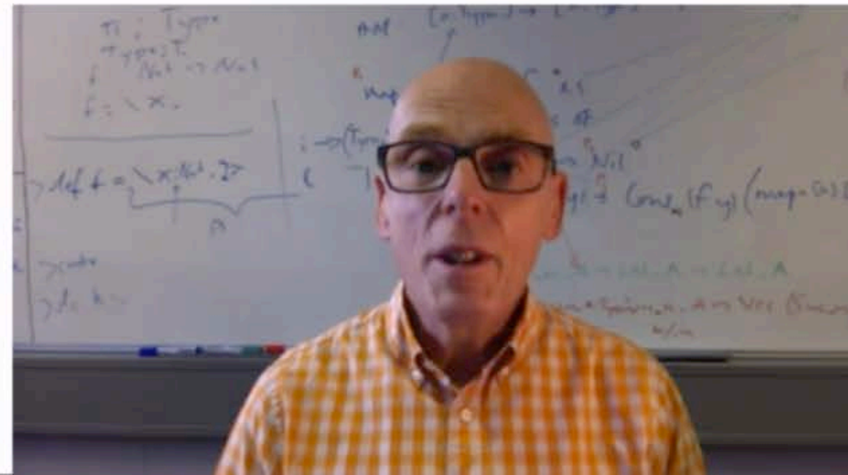

University of
Kent

OTP – gen_server



University of Kent

The generic server

Generic

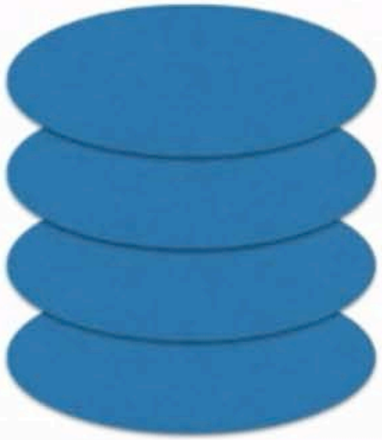
Concurrency

Error recovery

Supervision protocols

Timeouts

Setup and tear-down



Specific

Specific setup and tear-down

… including the server state

Handling specific requests …

… a/synchronously

Supervision policy

An echo server

The job of the echo server is to **echo** messages that it receives, and count the number of messages received.

It can also be asked to
- return the current `count`,
- `reset` the count to a new value, `X`,
- `stop` the server.

Here are the internals making it work.

```
init() ->
    State = 0,
    loop(State).

loop(N) ->
    receive
       {count, From} ->
          From ! N,
          loop(N);
       {{reset,X}, _From} ->
          loop(X);
       {stop, _From} ->
          ok;
       {Msg, From} ->
          From ! Msg,
          loop(N+1)
    end.
```

```
start_link() ->
    Pid = spawn_link(?MODULE,init,[]),
    register(?MODULE,Pid).

count() ->
    ?MODULE ! {count, self()},
    receive
       Reply -> Reply
    end.

echo(X) ->
    ?MODULE ! {X, self()},
    receive
       Reply -> Reply
    end.

reset(N) ->
    ?MODULE ! {{reset,N}, self()},
    ok.

stop() ->
    ?MODULE ! {stop, self()},
    ok.
```

University of Kent

The job of the echo server is to **echo** messages that it receives, and count the number of messages received.

It can also be asked to
- return the current `count`,
- `reset` the count to a new value, `X`,
- `stop` the server.

Here we see the function to start the server, and the functional interface to the services.

```
start_link() ->
    Pid = spawn_link(?MODULE,init,[]),
    register(?MODULE,Pid).

count() ->
    ?MODULE ! {count, self()},
    receive
       Reply -> Reply
    end.

echo(X) ->
    ?MODULE ! {X, self()},
    receive
       Reply -> Reply
    end.

reset(N) ->
    ?MODULE ! {{reset,N}, self()},
    ok.

stop() ->
    ?MODULE ! {stop, self()},
    ok.
```

```
init() ->
    State = 0,
    loop(State).

loop(N) ->
    receive
       {count, From} ->
          From ! N,
          loop(N);
       {{reset,X}, _From} ->
          loop(X);
       {stop, _From} ->
          ok;
       {Msg, From} ->
          From ! Msg,
          loop(N+1)
    end.
```

Terminal Shell Edit View Window Help

93% 11:38

esp.erl

New Open Recent Revert Save Print Undo Redo Cut Copy Paste Search Preferences

es.erl esp.erl

```
-module(esp).

%% API
-export([start_link/0,echo/1,count/0,reset/1,stop/0]).
-export([init/0,loop/1]).

%%%==========================================================
==================
%%% API
%%%==========================================================
==================

start_link() ->
    Pid = spawn_link(?MODULE, init, []),
    register(?MODULE,Pid).

count() ->
    ?MODULE ! {count, self()},
    receive
        Reply -> Reply
    end.

echo(X) ->
    ?MODULE ! {X, self()},
    receive
        Reply -> Reply
    end.

reset(N) ->
```

-:--- esp.erl Top (22,0) (Erlang EXT Flymake)

simonthompson — xterm — beam.smp -- -root /usr/local/lib/erlang -progname erl -- -home ~ -- ▸ erl_child_setup

```
% erl
Erlang/OTP 19 [erts-8.0] [source-6dc93c1] [64-bit] [smp:8:8] [async-threads:1
0] [hipe] [kernel-poll:false]

Eshell V8.0  (abort with ^G)
1> c(esp).
{ok,esp}
2> esp:start_link().
true
3> esp:count().
0
4> esp:echo(foo).
foo
5> echo(foot).
** exception error: undefined shell command echo/1
6> esp:echo(foot).
** exception error: bad argument
     in function  esp:echo/1 (esp.erl, line 22)
7> esp:start_link().
true
8> esp:echo(foo).
foo
9> esp:echo(foot).
foot
10> esp:count().
2
11> esp:reset(34).
ok
12> esp:echo(foot).
foot
13> esp:count().
35
14> esp:stop().
ok
15>
```

Rely/guarantee contract

A `behaviour` relies on someone providing an implementation of the callback functions `lt` and `eq` …

… given these, it will supply implementations of set functions, namely `insert` and `nil`.

```
-module(set).
-export([behaviour_info/1]).
-export([insert/3,nil/1]).

behaviour_info(callbacks) ->
  [ {lt,2}, {eq,2} … ];

insert(Mod,X,[Y|Ys]) ->
  case Mod:lt(X,Y) of …

nil(_Mod) -> [].
```

```
-module(nums).
-behaviour(set).
-export([lt/2,eq/2,]).
-export([insert/2,nil/0]).

lt(A,B) -> A<B.
eq(A,B) -> A==B.

insert(A,As) -> set:insert(?MODULE,A,As).
nil()        -> set:nil(?MODULE).
```

University of Kent

Rely/guarantee contract

A `behaviour` relies on someone providing an implementation of the callback functions `lt` and `eq` …

… given these, it will supply implementations of set functions, namely `insert` and `nil`.

A `gen_server` relies on having the callback functions

- `init` (initial state)
- `handle_call` (synchronous msg)
- `handle_cast` (asynchronous msg)

Given these, it implements

- `start_link` (start the server)
- `call` (make synchronous call)
- `cast` (send asynchronous msg)
- `stop` (stop the server)

```
init([]) ->
    {ok, 0}.


handle_call(count, _From, State) ->
    {reply, State, State};

handle_call(Msg, _From, State) ->
    {reply, Msg, State+1}.


handle_cast({reset,N}, _State) ->
    {noreply, N}.
```

```
init() ->
    State = 0,
    loop(State).

loop(N) ->
    receive
        {count, From} ->
            From ! N,
            loop(N);
        {{reset,X}, _From} ->
            loop(X);
        {stop, _From} ->
            ok;
        {Msg, From} ->
            From ! Msg,
            loop(N+1)
    end.
```

```
start_link() ->
    Pid = spawn_link(?MODULE,init,[]),
    register(?MODULE,Pid).

count() ->
    ?MODULE ! {count, self()},
    receive
       Reply -> Reply
    end.

echo(X) ->
    ?MODULE ! {X, self()},
    receive
       Reply -> Reply
    end.

reset(N) ->
    ?MODULE ! {{reset,N}, self()},
    ok.

stop() ->
    ?MODULE ! {stop, self()},
    ok.
```



```
start_link() ->
    gen_server:start_link(
       {local, ?MODULE},
       ?MODULE, [], []).

count() ->
    gen_server:call(?MODULE,count).

echo(X) ->
    gen_server:call(?MODULE,X).

reset(N) ->
    gen_server:cast(?MODULE,{reset,N}).

stop() ->
    gen_server:stop(?MODULE).
```

```
-module(es).

-behaviour(gen_server).

%% API
-export([start_link/0,echo/1,…]).

%% gen_server callbacks
-export([init/1,handle_call/3,…]).

init([]) ->
    {ok, 0}.

handle_call(count, _From, State) ->
    {reply, State, State};

handle_call(Msg, _From, State) ->
    {reply, Msg, State+1}.


handle_cast({reset,N}, _State) ->
    {noreply, N}.
```

```
start_link() ->
    gen_server:start_link(
       {local, ?MODULE},
       ?MODULE, [], []).

count() ->
    gen_server:call(?MODULE,count).

echo(X) ->
    gen_server:call(?MODULE,X).

reset(N) ->
    gen_server:cast(?MODULE,{reset,N}).

stop() ->
    gen_server:stop(?MODULE).
```

Terminal Shell Edit View Window Help

84% 11:46

es.erl

New Open Recent Revert Save Print Undo Redo Cut Copy Paste Search Preferences

es.erl esp.erl

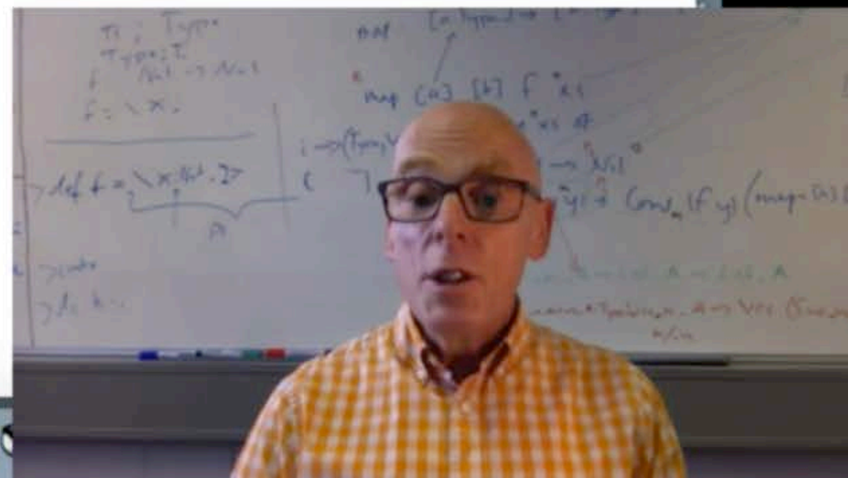
```
-module(es).

-behaviour(gen_server).

%% API
-export([start_link/0,echo/1,count/0,reset/1,stop/0]).

%% gen_server callbacks
-export([init/1, handle_call/3, handle_cast/2,
         handle_info/2, terminate/2, code_change/3]).

-define(SERVER, ?MODULE).


%%%==========================================================
================
%%% API
%%%==========================================================
================

%%--------------------------------------------------------
---------------
%% @doc
%% Starts the server
%%
%% @spec start_link() -> {ok, Pid} | ignore | {error,
Error}
%% @end
```

-:--- es.erl Top (21,0) (Erlang EXT Flymake)

simonthompson — xterm — beam.smp -- -root /usr/local/lib/erlang -progname erl -- -home ~ -- ▸ erl_child_setup

```
foo
9> esp:echo(foot).
foot
10> esp:count().
2
11> esp:reset(34).
ok
12> esp:echo(foot).
foot
13> esp:count().
35
14> esp:stop().
ok
15> c(es).
{ok,es}
16> es:start_link().
{ok,<0.86.0>}
17> es:count().
0
18> es:echo(foo).
foo
19> es:echo(foot).
foot
20> es:echo(food).
food
21> es:count:().
* 1: syntax error before: ':'
21> es:count().
3
22> es:reset(0).
ok
23> es:echo(food).
food
24> es:count().
1
25> es:stop().
ok
26>
```

University of Kent



University of Kent

The small print

The full description of `gen_server` facilities is given in the online documentation. It includes facilities for

- timeouts,
- supervision,
- multicast,
- non-local servers, …

A `gen_server` relies on having the callback functions

- `init` (initial state)
- `handle_call` (synchronous msg)
- `handle_cast` (asynchronous msg)

Also should implement

- `handle_info` (non call/cast msgs)
- `terminate` (tidy up on close)
- `code_change` (upgrade state)

But these have default implementations in the template.


University of
Kent