Industrial Functional Programming [1]

Melinda Tóth, István Bozó

Dept. Programming Languages and Compilers
Eötvös Loránd University, Budapest, Hungary

[1] Supported by TÁMOP-4.1.2.A/1-11/1-2011-0052

Contents

Servers step-by-step

- Starting and initialising
- $\downarrow$ Waiting for clients' requests $\downarrow$
- $\uparrow$ Serving clients $\uparrow$
- Terminating

Starting the server

```
start(Args) ->
    register(server,
             spawn_link(?MODULE, init, [Args])).

init(Args)->
    InitState = initialize_state(Args),
    loop(InitState).

loop(State)->
    receive
        ...
    end.
```

Handling requests

```
loop(State)->
    receive
        ...
        {handle, Msg} ->
            NewState = handle_req(Msg, State),
            loop(NewState);
        _Other ->
            an_unhandled_message
    end.

handle_req(_Msg, _State)->
    do_sth.
```

Stopping the server

```
loop(State)->
    receive
        stop ->
            terminate(State);
        ...
    end.

stop() ->
    server ! stop.

terminate(_State)->
    do_cleanup.
```

Server Skeleton

```
-module(server_skeleton).
-export([start/1, stop/0]).
-export([init/1, loop/1, terminate/1]).

start(Args) ->
    register(server, spawn_link(?MODULE, init, [Args])).

stop() ->
    server ! stop.

init(Args)->
    InitState = initialize_state(Args),
    loop(InitState).

loop(State)->
    receive
        stop ->
            terminate(State);
        {handle, Msg} ->
            NewState = handle_req(Msg, State),
            loop(NewState);
        _Other ->
            an_unhandled_message
    end.

initialize_state(_Args) ->
    do_init.
terminate(_State)->
    do_cleanup.
handle_req(_Msg, _State)->
    do_sth.
```

Software Upgrade

- Upgrading the code of the running application after compiling
- The old version is available only if there is “reference” to it
- Qualified function applications have to be used
- `code:load_file(Module)`
- Code Server

Code Server

- `code:purge(Module)`
- `code:soft_purge(Module)`
- `code:get_path()`
- `code:add_path*(Path)`

ETS

- Erlang Term Storage
- %% Shared memory
- Key-Value storage for large quantities of data
- Constant time access
- Not a KV Database, no transactions

ETS operations

- Creating tables:
  `TableId = ets:new(TableName, [Options])`
- Options: `named_table, set, bag, ordered_set, duplicate_bag, private, protected, public, {keypos, Key}, read/write_concurrency`
- Deleting tables: `ets:delete(TableId)`
- Inserting new elements:
  `ets:insert(TableId, Key, Value)`
- Finding elements by key: `ets:lookup(TableId, Key)`
- Deleting elements: `ets:delete(TableId, Key)`

ETS Advanced Search

- `ets:first(TableId),`
  `ets:next(TableId),`
  `'$end_of_table'`
- `ets:match(TableId, Pattern) - $1, $0`
- `ets:match_object(TableId, Pattern)`
- `ets:delete_object(TableId, Pattern)`
- `ets:select(TableId, MatchSpec)`

DETS

- Disk based ETS
- No transactions
- Similar interface to ETS

Tool to Use

- `tv:start()`

On the Next Lecture ...

- Erlang/OTP
- OTP behaviours
- Generic Servers