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

Basics

Process

- Actor with separated memory space (own heap and stack)
- Do not share memory
- Own state
- Communication with message passing

Basics

Communication models

- Shared memory + lock
- Software transactional memory (STM)
- Futures and Promises
- Message passing: synchronous, asynchronous

Basics

Ping-pong

```
run() ->
  Pid = spawn(fun ping/0),
  Pid ! {ping, self()},
  receive
    pong -> ok
  after
    1000 -> nok
  end.

ping() ->
  receive
    {ping, From} -> From ! pong
  end.
```

Erlang Processes

- Creating processes:
  `spawn/3, spawn_link/3, spawn/1`, etc.
  `Pid = spawn(Mod, Fun, [Arg1, ..., ArgN])`
  `Pid = spawn(fun Mod:Fun/0)`
- Erlang VM processes: `processes/1, i/0`
- BIFs: `self/0, pid/3`

Message Sending

- Message sending:
  ```
  Pid ! Msg
  Pid ! {msg, "Final message''}
  ```
- Emptying the mailbox: `flush/0`
- BIFs: `send/2`

Examples

```
Pid1 = spawn(lists, seq, [1, 100000])

Pid2 = spawn(fun() ->
                  apply(lists, seq, [1,100000])
             end)

self() ! apple.
flush().
```

Process links and error handling

- `link/1, spawn_link/3`
- exit signal, if process terminates– normal or non-normal
- `process_flag(trap_exit, true)`
- `{'EXIT', Pid, Reason}` message
- `unlink/1`
- `exit(Reason), exit(Pid, Reason)` – normal, kill, other
- supervision

Receive Expressions

```
receive
    Pattern1 [when Guard1] -> ExprList1;
      ...
    PatternN [when GuardN] -> ExprListN
end
```

Timeout

```
receive
    Pattern1 [when Guard1] -> ExprList1;
     ...
    PatternN [when GuardN] -> ExprListN
after
    Milliseconds -> ExprListN
end
```

Default timeout: `infinity`

Example

```
self() ! {msg, apple},
receive
    {msg, Data} when Data = apple ->
        fruit_arrived;
    Other ->
        {something_else, Other}
after
    100 -> nothing_happened
end
```

On the Next Lecture ...

- Process Registration
- Distributed Erlang Processes and Nodes