Abstractions and Types for Concurrent Programming

Yaron Minsky
Jane Street
Programming Models
Programming Models

Event Loop
Programming Models

Event Loop

Threads
Programming Models

Async

Event Loop

Threads
No new ideas

- 1999: “A Poor Man’s Concurrency Monad”, Ken Claessen
- 2002: LWT, Jerome Vouillon
- 2006: Async for MLton, Stephen Weeks
- 2007: Async for F#, Don Syme
Async Design Principles
Async Design Principles

- No preemption
Async Design Principles

• No preemption
• No inversion-of-control
Async Design Principles

• No preemption
• No inversion-of-control
• Make blocking explicit in the type system
  ‘a -> ‘b Def.t vs ‘a -> ‘b
module Def : sig

  type 'a t

  val return : 'a -> 'a t
  val map    : 'a t -> ('a -> 'b)    -> 'b t
  val bind   : 'a t -> ('a -> 'b t) -> 'b t

  val all    : 'a t list -> 'a list t

  ...

end
let file_length filename =
    Def.map (read_file filename) String.length

let sum_of_file_lengths files =
    let all_lengths =
        Def.all (List.map ~f:file_length files)
    in
    let sum l = List.fold ~init:0 ~f:(+) l in
    Def.map all_lengths sum

let manifest_length manifest =
    Def.bind (read_file manifest) (fun manifest ->
        let files = String.split ~on:'\n' manifest in
        sum_of_file_lengths files)
What's a monad?

```ocaml
let handle_message conn =
  Def.bind (Conn.read_message conn) (fun message ->
    Def.bind (process_message message) (fun reply ->
      Conn.send_message conn reply))
```
What’s a monad?

let handle_message conn =
    Conn.read_message conn >>= fun message ->
    process_message message >>= fun reply ->
    Conn.send_message conn reply
What’s a monad?

```ocaml
let handle_message conn =
  let! message = Conn.read_message conn in
  let! reply    = process_message message in
  Conn.send_message conn reply
```
Implementation
Implementation

• Deferred is (almost) a ref option with a place to install callbacks
Implementation

- Deferred is (almost) a ref option with a place to install callbacks
- Simple scheduler on top of select or epoll
Implementation

• Deferred is (almost) a ref option with a place to install callbacks
• Simple scheduler on top of select or epoll
• Lots of tricky corner cases!
Implementation

- Deferred is (almost) a ref option with a place to install callbacks
- Simple scheduler on top of select or epoll
- Lots of tricky corner cases!
- UNIX nonblocking API blocks
Implementation

• Deferred is (almost) a ref option with a place to install callbacks
• Simple scheduler on top of select or epoll
• Lots of tricky corner cases!
  • UNIX nonblocking API blocks
• Tail recursive bind
Implementation

- Deferred is (almost) a `ref` option with a place to install callbacks
- Simple scheduler on top of select or epoll
- Lots of tricky corner cases!
  - UNIX nonblocking API blocks
  - Tail recursive bind
- Pure OCaml (2.5kloc core, 10kloc total)
Streams
Streams

- Streams are pure and eager
Streams

- Streams are pure and eager
- Space leaks + No pushback
Streams

- Streams are pure and eager
- Space leaks + No pushback
- Now, Pipes: mutable, buffered channels
Error Handling
Error Handling

- Trickiest (remaining) part of concurrent programming
Error Handling

• Trickiest (remaining) part of concurrent programming

• Monitors: a place for uncaught exceptions
Error Handling

- Trickiest (remaining) part of concurrent programming
- Monitors: a place for uncaught exceptions
- try-with and protect built on monitors
Error Handling

• Trickiest (remaining) part of concurrent programming
• Monitors: a place for uncaught exceptions
• try-with and protect built on monitors
• No terminate operation
Async Lessons
Async Lessons

- Avoiding preemption is a win
Async Lessons

- Avoiding preemption is a win
- Fit in, but don’t be invisible
Async Lessons

• Avoiding preemption is a win
• Fit in, but don’t be invisible
• Beware of purity
Async Lessons

• Avoiding preemption is a win
• Fit in, but don’t be invisible
• Beware of purity
• Error handling is hard
Modeling RPCs

type request = | Listdir of path
               | Read_file of path
               | Move of path * path
               | Put_file of path * string
               | File_size of path
               | File_exists of path
               with sexp

    type response = | Ok
                    | Error of string
                    | File_size of int
                    | Contents of string list
                    | File_exists of bool
    with sexp
RPC Types
RPC Types

serialization

type 'a embedding = {
  inj : 'a -> Sexp.t;
  prj : Sexp.t -> 'a;
}
RPC Types

**Serialization function**

```ocaml
type 'a embedding = {
inj : 'a -> Sexp.t;
prj : Sexp.t -> 'a;
}

type ('a,'b) rpc = {
tag    : string;
query  : 'a embedding;
resp   : 'b embedding;
}
```
let delete_file = {
    tag = "delete_file";
    query = <:embedding<path>>;
    resp = <:embedding<[`Ok | `Error of string]>>>;
}
RPC Interface

```javascript
let delete_file = {
  tag = "delete_file";
  query = <:embedding<path>>;
  resp = <:embedding<[`Ok | `Error of string]>>;
  version = 0;
}
```
RPC Interface

let delete_file = {
  tag = "delete_file";
  query = <:embedding<path>>;
  resp = <:embedding<[`Ok | `Error of string]>>;
  version = 0;
  help = "Deletes the specified file, returning [`Ok] if successful, [`Error msg] otherwise";
}
RPC Interface

let delete_file = {
    tag = "delete_file";
    query = <:embedding<path>>;
    resp = <:embedding<[`Ok | `Error of string]>>;
    version = 0;
    help = "Deletes the specified file, returning [`Ok] 
          if successful, [`Error msg] otherwise";
    examples =
          [ Path.of_string "./etc/bashrc", Error "permission denied"
            ; Path.of_string "./home/yminsky/.bashrc", Ok ];
};
Client Side
val exec_rpc :
  ('a,'b) rpc
  -> (Conn.t -> 'a -> 'b Deferred.t)
val exec_rpc : 
  ('a,'b) rpc
  -> (Conn.t -> 'a -> 'b Deferred.t)

let delete_file = exec_rpc Rpc_specs.delete_file
let listdir = exec_rpc Rpc_specs.listdir
Server Side
Server Side

type rpc_impl
val implement_rpc : ('a, 'b) rpc -> ('a -> 'b) -> rpc_impl
val start_server : rpc_impl list -> port:int -> unit
Server Side

type rpc_impl
val implement_rpc : ('a, 'b) rpc -> ('a -> 'b) -> rpc_impl
val start_server : rpc_impl list -> port:int -> unit

let filesystem_server () =
  let rpcs = [
    implement_rpc Rpc_specs.delete_file Sys.unlink;
    implement_rpc Rpc_specs.listdir Sys.listdir;
  ]
in
start_server rpcs ~port:8080
RPC Lessons
RPC Lessons

- Type precision eases programmer’s lives
RPC Lessons

- Type precision eases programmer’s lives
- Fit in, but don’t be invisible

Friday, November 23, 12
RPC Lessons

• Type precision eases programmer’s lives
• Fit in, but don’t be invisible
• Fine-grained protocol versioning is a win
RPC Lessons

- Type precision eases programmer’s lives
- Fit in, but don’t be invisible
- Fine-grained protocol versioning is a win
- Polytypic programming matters
RPC Lessons

• Type precision eases programmer’s lives
• Fit in, but don’t be invisible
• Fine-grained protocol versioning is a win
• Polytypic programming matters
• Not just about RPC!
Interested?

http://opam.ocamlpro.com

opam install async