

Eio 1.0 – Effects-based IO for OCaml 5

Thomas Leonard	Patrick Ferris	Christiano Haesbaert	
Lucas Pluinage	Vesa Karvonen	Sudha Parimala	KC
Sivaramakrishnan	Vincent Balat	Anil Madhavapeddy	

Tarides

The OCaml Users and Developers Workshop, Sep 2023

Overview

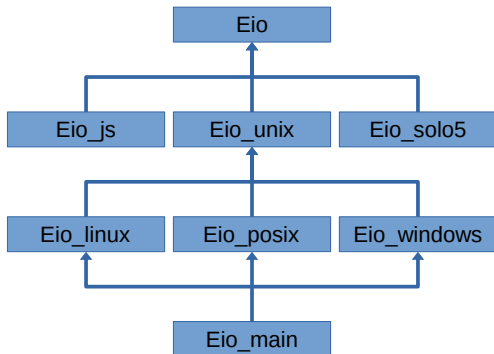
- ▶ Motivation and design
- ▶ Interoperability (Lwt, Async, Kcas, Domainslib)
- ▶ Comparison with Lwt
- ▶ Experiences porting software

Motivation

- ▶ Support effects
 - ▶ No difference between sequential and concurrent code
 - ▶ No special monad syntax
 - ▶ Can use `try`, `match`, `while`, etc
 - ▶ No separate Lwt or Async versions of code
 - ▶ No heap allocations needed to simulate a stack
 - ▶ A real stack means backtraces and profiling tools work
- ▶ Support multiple cores
- ▶ Fix some annoyances with Lwt

Eio packages

- ▶ Eio defines:
 - ▶ 3 effects (Suspend, Fork, Get_context)
 - ▶ Generic cross-platform APIs
- ▶ Backends for various platforms
- ▶ `eio_main` chooses the best backend



Performance : single core

Eio (0.38 s):

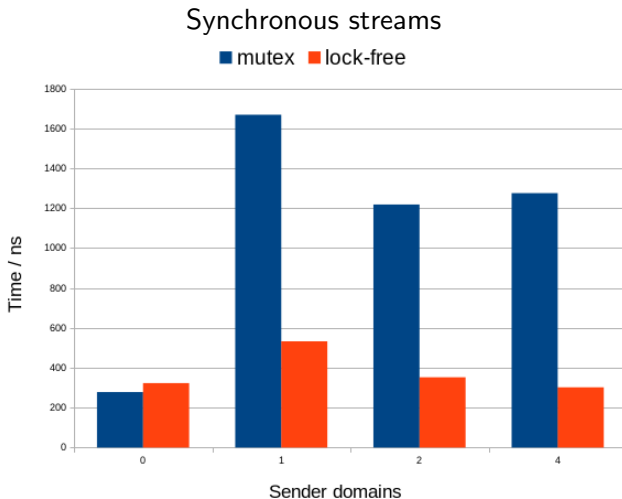
```
let parse r =  
  for _ = 1 to n_bytes do  
    let r = Eio.Buf_read.any_char r in  
    ignore (r : char)  
  done
```

Lwt (1.49 s):

```
let parse r =  
  let rec aux = function  
    | 0 → Lwt.return_unit  
    | i →  
      let* r = Lwt_io.read_char r in  
      ignore (r : char);  
      aux (i - 1)  
  in  
  aux n_bytes
```

Performance : multi-core

- ▶ Many data-structures are now lock-free
- ▶ Better performance with multiple domains



Interoperability : Lwt

To run Lwt programs under Eio, replace `Lwt_main.run` with:

```
Eio_main.run @@ fun env →  
  Lwt_eio.with_event_loop ~clock:env#clock @@ fun _ →  
  Lwt_eio.run_lwt @@ fun () →  
  ...
```

`run_lwt` and `run_eio` switch between Lwt and Eio code:

```
val run_lwt : (unit → 'a Lwt.t) → 'a  
val run_eio : (unit → 'a) → 'a Lwt.t
```

https://github.com/ocaml-multicore/lwt_eio

Interoperability : Async

Async_eio does the same for async:

```
val run_eio :  
  (unit → 'a) → 'a Async_kernel.Deferred.t
```

```
val run_async :  
  (unit → 'a Async_kernel.Deferred.t) → 'a
```

https://github.com/talex5/async_eio

Interoperability : Async, Eio and Lwt

You can even use all three libraries together in a single domain!

```
Eio_main.run @@ fun env →  
Lwt_eio.with_event_loop ~clock:env#clock @@ fun _ →  
Async_eio.with_event_loop @@ fun _ →  
...
```

<https://github.com/talex5/async-eio-lwt-chimera>

Interoperability : Domainslib and Kcas

Eio, Domainslib and Kcas all use `domain-local-await`, allowing e.g. Domainslib to add items to a Kcas queue, which is being read from an Eio domain.

perf : test code

Eio

```
let run_task1 () =  
  for _ = 1 to 2000 do  
    do_work ()  
  done  
  
let run_task2 () =  
  for _ = 1 to 2000 do  
    do_work ()  
  done  
  
let run () =  
  Fiber.both run_task1 run_task2
```

Lwt

```
let run_task1 () =  
  let rec outer = function  
    | 0 → Lwt.return_unit  
    | i →  
      let* () = do_work () in  
      outer (i - 1)  
  in  
  outer 2000  
  
let run_task2 () = ...  
  
let run () =  
  Lwt.join [  
    run_task1 ();  
    run_task2 ();  
  ]
```

perf : results

perf shows task1 vs task2 for Eio part:

- 49.94% Lwt_main.run_495
 - Lwt_main.run_loop_435
 - 49.83% Lwt_sequence.loop_346
 - Lwt.callback_1373
 - 49.77% Dune.exe.Perf.fun_967
 - + 49.77% Dune.exe.Perf.use_cpu_273
- 49.90% Eio_linux.Sched.with_sched_inner_3088
 - 49.89% Eio_linux.Sched.with_eventfd_1738
 - Stdlib.Fun.protect_320
 - 49.86% caml_runstack
 - Eio.core.Fiber.fun_1369
 - 25.07% Dune.exe.Perf.run_task2_425
 - + Dune.exe.Perf.use_cpu_273
 - 24.78% Dune.exe.Perf.run_task1_421
 - + 24.77% Dune.exe.Perf.use_cpu_273

Resource leaks

- ▶ Resources are attached to switches
- ▶ When the switch finishes, the resource is freed

Eio:

```
let accept socket =  
  Switch.run @@ fun sw →  
    let conn, _addr = Eio.Net.accept ~sw socket in  
    ...  
    Eio.Net.close conn      (* Optional *)
```

Lwt (leaks conn if cancelled):

```
let accept socket =  
  let* conn, _addr = Lwt_unix.accept socket in  
  ...  
  Lwt_unix.close conn
```

Bounds on behaviour : Lwt

```
let () =  
  Lwt_main.run (main ())
```

- ▶ What does this program do?
- ▶ What firewall rules should we set?
- ▶ Global state is hard to reason about

Bounds on behaviour : Eio

```
let () =  
  Eio_main.run @@ fun env →  
  Switch.run @@ fun sw →  
  let addr = 'Tcp (Eio.Net.Ipaddr.V4.any, 8080) in  
  let socket =  
    Eio.Net.listen ~sw env #net addr  
    ~backlog:5  
    ~reuse_addr:true  
  in  
  let dir = Eio.Path.open_dir ~sw (env #fs / "/srv/htdocs") in  
  main ~socket dir
```

- ▶ Listens on port 8080 (no other network use)
- ▶ Uses /srv/htdocs (no other file-system use)

<https://roscidus.com/blog/blog/2023/04/26/lambda-capabilities/>

Experiences porting software

- ▶ Solver service (cache-dir bug)
- ▶ Wayland proxy
- ▶ Libraries: ocaml-tls, cohttp, dream, capnp-rpc, ...

<https://github.com/ocaml-multicore/awesome-multicore-ocaml>

Future

Eio 1.0:

- ▶ Finish file-system APIs
- ▶ OCaml 5.1 events

Get involved:

- ▶ Chat on #eio (<https://matrix.to/#/#eio:roscidus.com>)
- ▶ Developer video call every two weeks

<https://github.com/ocaml-multicore/eio>