Lwt: a Cooperative Thread Library

Jérémie Dimino

April 16, 2010

Concepts

Lwt threads are:

- *Cooperative*:
  - one running thread at a time
  - context switches are explicit
- *Light*: threads creation and context switches are very cheap

Threads everywhere

A thread for every function that may block.

```
1 | val input_char : in_channel -> char
2 | val Unix.sleep : int -> unit
```

become:

```
1 | val Lwt_io.read_char : Lwt_io.input_channel -> char Lwt.t
2 | val Lwt_unix.sleep : float -> unit Lwt.t
```

Thread states

A thread can in one of the following states:

- successfully terminated: `Return x`
- failed with an exception: `Fail exn`
- sleeping: `Sleep`

One can get the state of a thread with `Lwt.state`.

Thread creation

- Terminated threads:

```
1 | val Lwt.return : 'a -> 'a Lwt.t
2 | val Lwt.fail : exn -> 'a Lwt.t
```

- Sleeping threads:

```
1 | val Lwt.wait : unit -> 'a Lwt.t * 'a Lwt.u
2 | val Lwt.task : unit -> 'a Lwt.t * 'a Lwt.u
```

where:

- `'a Lwt.t` is the type of `Lwt` threads
- `'a Lwt.u` is the type of `Lwt` thread wakeners

Thread compisition

- Binding the result of a thread:

```
1 | val Lwt.bind : 'a t -> ('a -> 'b t) -> 'b t
```

- Handle exceptions in threads:

```
1 | val Lwt.catch :
2 |   (unit -> 'a Lwt.t) ->
3 |   (exn -> 'a Lwt.t) -> 'a Lwt.t
```

Multi-threads compisition

- Wait for all threads to terminate:

```
1 | val Lwt.join: unit Lwt.t list -> unit Lwt.t
```

- Wait for at least one thread to terminate:

```
1 | val Lwt.choose : 'a Lwt.t list -> 'a Lwt.t
2 | val Lwt.pick : 'a Lwt.t list -> 'a Lwt.t
```

Syntax extensions

- Parallel let-binding:

```
1 lwt x = f () and y = g () in
2 expr
```

- Errors catching:

```
1 try_lwt
2   expr
3 with
4   | pattern -> expr
5   ...
6 finally
7   expr
```

- For-loops:

```
1 for_lwt i = expr to expr do
2   expr
3 done
```

Cooperative system calls

`Lwt_unix`: cooperative version of `Unix`.

```
1 | val read : file_descr -> string -> int -> int -> int Lwt.t
2 | val write : file_descr -> string -> int -> int -> int Lwt.t
3 | val sleep : float -> unit Lwt.t
```

Cooperative buffered channels

`Lwt_io`: cooperative version of buffered byte channels.

```
1 | val write : output_channel -> string -> unit Lwt.t
2 | val read_lines : input_channel -> string Lwt_stream.t
```

`Lwt` specific functions:

- atomic uses of channels:

```
1 | val atomic :
2 |   ('a channel -> 'b Lwt.t) ->
3 |   ('a channel -> 'b Lwt.t)
```

- auto-flushing: after each write, `Lwt` launch a thread which will eventually flush the channel before the program goes into idle.

Example

We want to run 3 asynchronous functions `get1`, `get2` and `get3`, use the result of the first which terminates, then cancel the others, with a timeout.

Detaching computation to a preemptive thread

You can detach to a preemptive thread a computation that may take times to complete without cooperating:

```
1 | val Lwt_preemptive.detach : ('a -> 'b) -> 'a -> 'b Lwt.t
```

For example:

- libc calls such as `gethostbyname`
- breaking a password
- ...

`Lwt` libraries

- `lwt.unix`
- `Lwt.preemptive`
- `lwt.glib`: integration of the `Lwt` scheduler into the `GLib` main loop
- `lwt.react`: threaded version of React's primitives
- `lwt.text`: text mode utilities (terminal control, cooperative `read_line` with line editing support, ...)

Applications/libraries using `Lwt`

- unison: a file-synchronization tool
- ocsigen: web server and programming framework in OCaml
- obus: pure OCaml implementation of D-Bus
- Krobot: a robot controlled with ocaml programs
- ...