Week12

IN0003

Jigao

TUM

30. Januar 2019

What about the last weeks?

- A proving tool **Big Step** and **Induction**
- Today we will have Parallel Programmimg, that is last topic in this lecture.

Thread

- `module Thread`
- `create: ('a -> 'b) -> 'a -> t`
- This function returns the thread id. The created thread evaluates the function for its argument
- `self : unit -> t`
- `id: t -> int`
- `join: t -> unit` blocks the current thread until the evaluation of the given thread has terminated.
- `kill: t -> unit`
- `delay: float -> unit`
- `exit: unit -> unit`

Thread

- ocamlc -I +threads unix.cma threads.cma program.ml
- Example about `join, create`
- For exam: Remember to write down
  `open Thread, open Event, open List`

Channel

- Threads communicate via channels. (Maybe new to you)
- `module Event`
- `new_channel : unit -> 'a channel`
- `always : 'a -> 'a event` wraps a value into an `event`.
- `sync : 'a event -> 'a`: Synchronization on event returns their values.
- `send : 'a channel -> 'a -> unit event`
- `receive : 'a channel -> 'a event`

Channel

- `sync (send ch str)`: exposes the event of sending to the outside world and blocks the sender, until another thread has read the value from the channel .
- `sync (receive ch)`: blocks the receiver, until a value has been made available on the channel. Then this value is returned as the result.

Thread and Channel

- Try to get familiar with these functions.
- Most of them are needed 100% in the exam.
- My advice: ẅrite code downẗhis time or when you prepare for the exam.
- Example.

Thread and Channel

- Try to get familiar with these functions.
- Most of them are needed 100% in the exam.
- My advice: ẅrite code downẗhis time or when you prepare for the exam.
- Example.
- Main spawns a thread. Then it sends it a string and waits for the answer.
- The new thread waits for the transfer of a string value over the channel.
- As soon as the string is received, an answer is sent on the same channel.

Thread and Channel

- Any idea how to build a deadlock?
- Maybe base on the example before.

Thread and Channel

- Any idea how to build a deadlock?
- Maybe base on the example before.
- When we join the new threads, but it needs response from old thread.
- Both of them do not proceed to continue.

- 1. As a first step, implement a function `spawn_counter : int -> Thread.t` that spawns a new thread. This thread should then print all numbers from 0 to the passed argument to the standard output. Print the thread's id in addition to the current number, so that you can identify who is responsible for the output.

- 2. Write a function `run_counters : int -> int -> unit` that, when called with `run_counters m n`, spawns `m` counters, each counting to `n`. Make sure `run_counters` does not return before all the counters have stopped.
- `List.init: int -> (int -> 'a) -> 'a list`
- `List.iter: ('a -> unit) -> 'a list -> unit`

- 2. Write a function `run_counters : int -> int -> unit` that, when called with `run_counters m n`, spawns `m` counters, each counting to `n`. Make sure `run_counters` does not return before all the counters have stopped.
- `List.init: int -> (int -> 'a) -> 'a list`
- `List.iter: ('a -> unit) -> 'a list -> unit`
- `let counters = List.init m (fun _ -> spawn_counter n)` creats a list with `m` elements.
- `List.iter join counters`: let these thread joined before main.

- 3.As a next step, the threads shall now be synchronized, such that all threads take turns with their output. First all threads print 0, then all threads print 1 and so on. Use channels for communication between the threads. Make sure they shutdown correctly and are joined by the main thread.
- 4. Implement a new version of `spawn_counter` and `run_counters` such that the counters take turns counting.
- Hint: `spawn_counterL int -> channel -> bool`: true for print all numbers
- `let channels = List.init m (fun _ -> new_channel ()) in`
- `let counters = List.map (spawn_counter n) channels`
- Work around: first thread print $n \rightarrow$ send (also be blocked) $\rightarrow$ next thread print n and send $\rightarrow$ ... $\rightarrow$ the first thread print $n + 1$

- From last year's Exam. 20 Points.
- `post : t -> user -> pass -> string -> unit` publishes a new blog post (last argument) in the given users blog.
- `read : t -> user -> blog` requests a user's blog from the server.
- Try 10 min. They are very easy, within 5 lines together.

- Go with such structure.
- Keep this framework in mind.

```
let start_server users =
        let c = new_channel ()
        in
        let rec server_fun blogs =
        match sync (receive c) with
                | Post (user, pass, text) ->
                        (** impl **)
                | Read (user, answer_c) ->
                        (** impl **)
        in
        let _ = create server_fun []
        in
        c
```

Summary

- Hope you get feeling how would the last assignment in exam be like?
- Thread assignment has a framework.
- But the problem statement is always a great part. Take time with it.
- I think only 1% can have all the points, but everyone should have 50% of the points.

Summary

- Prepared?
- Next time maybe some exam part from me.
- (Hmm they are originally German)
- But what do you need? Depends on you.
- Any Questions?