Réseaux de Kahn

Mihai Dusmanu Clément Pascutto

Systèmes et réseaux
École normale supérieure

Lundi 30 Mai 2016

Sommaire

Implémentation avec des pipes et processus Unix

```
type 'a channel = {
    in_ch: Pervasives.in_channel;
    out_ch: Pervasives.out_channel;
}
```

Implémentation avec des pipes et processus Unix

```
type 'a channel = {
    in_ch: Pervasives.in_channel;
    out_ch: Pervasives.out_channel;
}
```

```
1  let doco l () =
2      let ths = List.map
3          (fun f ->
4              let pid = Unix.fork () in
5              if pid = 0 then (
6                  f ();
7                  exit 0;
8              );
9              pid)
10         l in
11     List.iter
12         (fun pid -> let _ = Unix.waitpid [] pid in ())
13         ths
```

Implémentation séquentielle

```
type 'a channel = 'a Queue.t
```

Implémentation séquentielle

```
type 'a channel = 'a Queue.t
```

Pour simuler le CPS, on utilise le type `process` suivant :

```
type 'a process = ('a -> unit) -> unit
```

Implémentation séquentielle

```
type 'a channel = 'a Queue.t
```

Pour simuler le CPS, on utilise le type `process` suivant :

```
type 'a process = ('a -> unit) -> unit
```

Hormis la fonction `run`, les fonctions sont inspirées de l'article *A poor man's concurrency monad*, K. Classen.

```
1 let run e =
2     let res = ref None in
3     bind
4         e
5         (fun x -> res := Some x; (fun f -> f ()))
6         (fun () -> ());
7     match !res with
8     | None -> assert false
9     | Some x -> x
```

Implémentation séquentielle

Le scheduler éxécute les processus dans la file tant que celle-ci n'est pas vide.

```
1  let proc_q = Queue.create ()
2  let sch_state = ref false
3
4  let rec scheduler () =
5      if !sch_state then ()
6      else begin
7          sch_state := true;
8          while not (Queue.is_empty proc_q) do
9              let f = Queue.pop proc_q in f ()
10         done;
11         sch_state := false
12     end
```

Implémentation séquentielle

```
1  let pid = ref 0
2  let pidStatus = Hashtbl.create 1000
3
4  let doco l f =
5      let q = Queue.create () in
6      List.iter
7          (fun p ->
8              let aux = !pid in
9              Queue.add
10                 (fun () ->
11                     p
12                     (fun () ->
13                         Hashtbl.add pidStatus aux true))
14                 proc_q;
15             Queue.push aux q;
16             incr pid)
17         l;
18     Queue.add
19         (fun () -> doco_finish q f)
20         proc_q;
21     scheduler ()
```

Implémentation en réseau

Architecture

On choisit une architecture server/workers.

Implémentation en réseau

Architecture

On choisit une architecture server/workers.

On utilise le port 1043 pour les communications.

```
type 'a in_port = int
type 'a out_port = int

type 'a communication =
    | Put of 'a out_port * 'a
    | Get of 'a in_port
```

Implémentation en réseau

Architecture

On choisit une architecture server/workers.

On utilise le port 1043 pour les communications.

```
type 'a in_port = int
type 'a out_port = int

type 'a communication =
    | Put of 'a out_port * 'a
    | Get of 'a in_port
```

Et le port 1042 pour les demandes.

```
type query =
    | NewChannel
    | Doco of string list
    | Finished of int * string
```

Implémentation en réseau

Mini système de priorités pour les threads OCaml

Problème : La fonction get peut être bloquante pour un processus (côté worker).

Implémentation en réseau

Mini système de priorités pour les threads OCaml

Problème : La fonction get peut être bloquante pour un processus (côté worker).
On aimerait qu'elle soit prise en compte le plus tôt possible sur le serveur.

Implémentation en réseau

Mini système de priorités pour les threads OCaml

Problème : La fonction get peut être bloquante pour un processus (côté worker).
On aimerait qu'elle soit prise en compte le plus tôt possible sur le serveur.
Tous les threads commencent avec 50 jokers. Ils en reçoivent 1 à chaque prise en compte d'un get.

Implémentation en réseau

Mini système de priorités pour les threads OCaml

Problème : La fonction get peut être bloquante pour un processus (côté worker).
On aimerait qu'elle soit prise en compte le plus tôt possible sur le serveur.
Tous les threads commencent avec 50 jokers. Ils en reçoivent 1 à chaque prise en compte d'un get.

```
1 ||    if !jokers = 0 then
2 ||        Thread.yield ()
3 ||    else
4 ||        decr jokers
```

Implémentation en réseau

Lecture non bloquante (lock-free)

Problème : La fonction `Marshal.from_channel` est bloquante.

Implémentation en réseau

Lecture non bloquante (lock-free)

Problème : La fonction `Marshal.from_channel` est bloquante.
Supposons que l'on veuille lire `size` caractères depuis `inChannel`.

```
1 let non_blocking_read size inChannel =
2     let buf = Bytes.create size in
3     let pos = ref 0 in
4     while pos <> size do
5         pos := !pos +
6             input inChannel buf !pos (size - !pos);
7         Thread.yield ()
8     done
```

Démo

- Exemple de base : compteur.
- Recherche de cycle hamiltonien.
- Simulation d'automates non déterministes.