SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

PC3R - TD4: Canaux synchrones (II)

Equipe Enseignante PC3R

18/02/2021

1 [*Go*, *OCaml*] Exercices avancés

1.1 Sélection

1. Ecrire un programme qui lance trois processus *fournisseurs* qui mettent un temps aléatoire pour envoyer, chacun sur un canal différent, un *produit* différent (par exemple, des fruits) et un quatrième processus qui doit recevoir tous les produits. Les threads fournisseurs se relancent un nombre fini de fois.

1.2 Modélisation

On se propose de modéliser un service de vente en ligne impliquant un intermédiaire. Chaque élément du système sera représenté par un thread et sera associé à un canal (il y aura donc autant de canaux que d'agents dans le système). Le système sera composé:

- De deux clients, qui envoient des requêtes pour un produit spécifique (par exemple ″`cafe`″ pour l'un, ″`the`″ pour l'autre) sur le canal de l'intermédiaire. Ils attendent ensuite de recevoir un nom canal frais sur leur propre canal, puis attendent de recevoir le produit sur ce nouveau canal. Ils recommencent un nombre fini de fois.
- D'un intermédiaire qui reçoit sur son canal des requêtes des deux clients. Lors de la réception d'une requete, i) il crée un nouveau canal, ii) il envoie sur le canal du client le nom du nouveau canal, iii) il envoie sur le canal du vendeur un couple composé du produit demandé et du nom du nouveau canal. Ensuite, il se relance.
- D'un vendeur qui contient un compteur interne et qui reçoit une requête composé d'un produit et d'un canal. Il crée un "produit fini" composé du nom du produit demandé et de son compteur (comme ″`the 3`″, par exemple). Il envoie sur le canal reçu le produit fini.

1. Décrypter la spécification. Décrire les agents du systèmes, les différents canaux utilisés, et le comportement de chacun des agents.

2. Ecrire les fonctions récursives exécutées par le vendeur et l'intermédiaire, qui proposent des services (et sont donc non-terminantes). Celle du vendeur incrémentera son compteur à chaque requête traitée.

3. L'intermédiaire doit pouvoir distinguer quel client lui parle quand il reçoit une requête (afin de communiquer le nouveau canal à la bonne personne). Comment procéder ?

4. Ecrire la fonction exécutée par les deux threads clients. Afin de vérifier que les clients récupèrent bien le produit demandé, on fera en sorte que chaque client mette à jour une variable `log` (chaine de caractères) qui enregistrera les produits reçus.

5. Ecrire la fonction principale qui lance les quatre threads (deux clients, un intermédiaire, un vendeur), qui attend la terminaison des deux clients et qui affiche leurs logs.

1.3 Diffusion

On modélise un *serveur de diffusion*, créé avec comme argument une liste de canaux de sortie. A chaque fois qu'il recoit une information sur son canal d'entrée, il propage cette information sur tous les canaux de sortie. On dispose, en outre, dans le système, de plusieurs processus *écouteurs*, chacun associé à un canal de sortie. Ils attendent un temps aléatoire puis recoivent l'information sur leur canal de sortie associé.

1. Ecrire une fonction `broadcast` aux qui prend une liste de canaux et une valeur et qui envoie une fois la valeur sur chaque canal de la liste. Attention : l'ordre dans lequel les synchronisations vont s'effectuer doit pouvoir varier en fonction des disponibilité des écouteurs.
2. Ecrire le système dans son ensemble.

2 [*Go*, *OCaml*] Futures

2.1 Examen 2012 - Futures

On cherche à construire la structure de contrôle concurrente appelée "future". Ce mécanisme permet de lancer un calcul asynchrone, et de se synchroniser lors de la première utilisation de la valeur censée être retournée par ce calcul. Pour être intéressant, ce calcul peut être exécuté sur un thread particulier. On va implanter ce mécanisme en lançant un thread par "future". Si le résultat du calcul est nécessaire avant que le calcul ne soit terminé, alors l'accès à cette valeur est alors bloquant.

```
type 'a future
val spawn : ('a -> 'b) -> 'a -> 'b future
val get : 'a future -> 'a
val isDone : 'a future -> bool
```

- `get` retourne si elle est calculée la valeur de la "future" et sinon attend
- `isDone` retourne un booléen à `true` si le calcul a effectivement eu lieu, et `false` sinon
- `spawn` : lance le thread de calcul de la fonction de type `'a -> 'b` sur le paramètre de type `'a` , et construire une `'b future`.

1. Ecrire la définition du type `future` et les quatre méthodes du tableau précédent.
2. Détailler le déroulement du programme suivant, où la fonction `f` calcule le ième nombre de Fibonacci[1].

```
(* val f : int -> int *)
let x1 = spawn f 5
and x2 = spawn f 4
and x3 = spawn f 6 ;;
let result =
 ((get x1)+(get x2)+(get x3))/3;;
```

3 [*Go*] Client-Serveur TCP

On veut écrire un serveur d'écho en *Go*: le client lit des chaînes de caractères sur l'entrée standard et les envoie au serveur, le serveur capitalise les chaînes (`toUpper`) et les renvoie au client, qui les affiche sur la sortie standard.

1. Donner le code *Go* du client et du serveur.

4 [*Go*, *OCaml*] Examen Réparti 2018

On cherche à implémenter un modèle d'annuaire de services distribués à l'aide des canaux synchrones d'*OCaml* ou de *Go*. Un **client** se connecte à un **intermédiaire** pour requérir un service, l'intermédiaire

[1] où $F_0 = 0, F_1 = 1$ et $F_{n+2} = F_n + F_{n+1}$ pour $n > 1$

le met ensuite en relation avec un **serveur** qui propose le service demandé. Dans un premier temps on supposera que les différents services sont des fonctions des entiers dans les entiers.
Dans l'intermédiaire, la correspondance entre services et serveurs est donné par un annuaire (*registry*) dont on donne ici une interface en *OCaml*:

```
type 'a option = Some of 'a | None

type registry
create_registry : unit -> registry
register : registry * string * (int * (int channel)) channel -> registry
lookup : registry * string -> ((int * (int channel)) channel) option
```

et en *Go*

```
type int_et_chan struct{entier: int, canal: chan int}

type registry interface{
  register(s string, c channel int_et_chan)
  poll (s string) : bool
  get (s string) : chan int_et_chan
  }
```

Ainsi l'annuaire associe au nom du service, un canal . La fonction `register` permet d'enregistrer un nouveau service et l'appel `lookup s` renvoie un type option en *OCaml* (`None` si le service n'existe pas dans l'annuaire). En *Go*, une méthode `poll` permet de savoir si le service existe. L'implémentation de l'annuaire n'est pas demandé dans l'exercice.

Le comportement du système est décrit ainsi:

- les **serveurs** possèdent un canal privé et calculent une fonction particulière des entiers dans les entiers (leur "service"). Ils commencent par envoyer à l'intermédiaire, sur son canal d'enregistrement, un message contenant le nom du service qu'ils effectuent et leur canal privé.

  Ensuite ils bouclent: ils attendent d'être contactés sur leur canal privé. Ils reçoivent sur ce canal un message composé d'un argument entier et d'un canal. Ils calculent leur service sur l'argument, envoient le résultat sur le canal reçu, et deviennent à nouveau disponibles.

- les **clients** possèdent deux canaux privés. Ils contactent l'intermédiaire en envoyant sur son canal de recherche un nom de service et leur premier canal privé et écoutent ensuite sur ce dernier. Si le service existe dans l'annuaire de l'intermédiaire, ils reçoivent sur le canal qu'ils ont envoyé une réponse contenant le canal sur lequel contacter le serveur (sinon ils attendent pour l'éternité). Ils envoient ensuite sur ce canal un argument entier et leur deuxième canal privé et reçoivent la réponse du serveur sur ce dernier.

- l'**intermédiaire** maintient l'annuaire de services et peut être contacté sur deux canaux publics: un canal d'enregistrement et un canal de recherche. Il écoute **simultanément** sur les deux canaux.

  Sur le canal d'enregistrement `reg`, il peut recevoir des messages contenant un nom de service et un canal d'un serveur. Si c'est le cas, il ajoute cette association à son annuaire et redevient disponible.

  Sur le canal de recherche `lku`, il peut recevoir des messages contenant un nom de service et un canal de client. Si le service existe dans l'annuaire, il envoie sur le canal reçu le canal associé au nom de service et redevient disponible (sinon il redevient directement disponible).

1. Réaliser un rapide schéma d'un système avec un intermédiaire, deux clients et deux serveurs, en représentant les canaux qui lient les composants et en donnant leurs **types**.
2. Ecrire un serveur qui enregistre un service `"square"` qui calcule le carré de l'argument qu'il reçoit.
3. Ecrire un client qui requiert à l'intermédiaire le service `"square"` pour calculer le carré de 42.

4. Donner le code de l'intermédiaire. **Attention:** il doit être capable d'écouter simultanément sur `reg` et `lku`.

5. Donner le code d'une fonction initialisant un système avec trois threads effectuant les tâches des trois questions précédentes.

   On propose maintenant plusieurs modifications du système:

6. Modifier le code de l'intermédiaire pour que la gestion des enregistrements sur `ref` et celle des recherches sur `lku` soient effectuées par deux threads différents.

7. Donner différentes modifications de code à apporter aux réponses des questions 2., 3., 4. pour gérer le cas où le service requis par un client n'existe pas dans l'annuaire.

8. [*Go* uniquement] Donner différentes modifications de code à apporter aux réponses des questions 2., 3., 4. pour mettre en place un annuaire de services de types et d'arités arbitraires: c'est-à-dire que l'annuaire de l'intermédiaire peut contenir simultanément le service `"carre"` qui calcule le carré d'un entier et le service `"plus_long"` qui prend deux chaînes de caractères et décide si oui ou non la première est la longueur de la première est supérieure ou égale à celle de la seconde.

9. [*OCaml* uniquement] Expliquer pourquoi on ne peut réaliser la question précédente en *OCaml*.