SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

PC3R - TD3: Canaux synchrones

Equipe Enseignante PC3R

11/02/2021

Ressources de l'UE: `Moodle`

1 [*Go*] Exercices de base

1.1 Impressions

1. Modéliser un système composé de deux processus, l'un imprime `"Ping"` sur la sortie standard et l'autre `"Pong"`. Mettre en place un mécanisme de synchronisation qui garantit que, sur la sortie standard, chaque `"Ping"` sera suivi par un `"Pong"`, et vice-versa.
2. Modéliser un système composé de cinq processus qui, en boucle et **sans mécanisme de synchronisation**, impriment chacun sur la sortie standard un morceau de la phrase *"belle marquise / vos beaux yeux / me font / mourir / d'amour"*. Prédire l'effet produit.
3. Même question en utilisant un mécanisme de synchronisation simple. Prédire l'effet produit.

1.2 Compteur

On modélise, en passage de messages, un système où `NB_T` processus incrémentent chacun *un compteur distribué*.

1. Modéliser un tel système dans lequel un processus spécial est chargé de maintenir le compteur, et reçoit des injonctions d'incrémentation des autres processus.
2. Modéliser un tel système sans utiliser de processus supplémentaire.

1.3 Asynchronie

On modélise, en passage de messages, un système composé d'un producteur et d'un consommateur:

- le producteur produit successivement plusieurs entiers pseudo-aléatoires et les envoie au consommateur.
- le consommateur prend un temps non-négligeable pour consommer chaque entier qu'il reçoit, puis il les imprime sur la sortie standard.

1. Modéliser un tel système en utilisant un canal synchrone. Décrire les comportements possibles du système.
2. Modéliser un tel système en utilisant un canal asynchrone de taille fixée (en utisant les *Buffered Channels* de *Go*). Décrire les comportements possibles du système.
3. Modéliser un tel système en utilisant un mécanisme implémentant un canal de taille fixée (sans utiliser les *Buffered Channels*). Décrire les comportements possibles du système.

1.4 Polyadicité

On modélise, en passage de messages, un système composé de n producteurs et de c consommateurs (on pourra supposer que n est un multiple de c):

- les producteur produisent successivement deux valeurs: un fruit (une chaîne de caractères) et un numéro (un entier) ; ils envoient ensuite les deux valeurs aux consommateurs.
- les consommateurs attendent en boucle de recevoir deux valeurs (un fruit et un entier) et imprime un message correspondant à ce qu'ils ont reçu sur la sortie standard.

1. Modéliser un tel système en utilisant deux canaux (un pour chaque valeur) entre les consommateurs et les producteurs. Décrire les comportements possibles du système, en discutant sur le nombre (1 ou plusieurs) de producteurs et de consommateurs.
2. Modéliser un tel système en utilisant un unique canal qui transporte des structures contenant deux valeurs (un fruit, un entier). Décrire les comportements possibles du système.
3. Modéliser un tel système en utilisant un canal qui transporte un canal polymorphe (sur lequel on peut faire passer des fruits ou des entiers). Décrire les comportements possibles du système.

2 [*Go*, *OCaml*] Exercices avancés

2.1 Sélection

1. Ecrire un programme qui lance trois processus *fournisseurs* qui mettent un temps aléatoire pour envoyer, chacun sur un canal différent, une *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.

2.2 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

2.3 *** Examen Réparti 1 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 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*.