TD5-TME5 : Events et Canaux Synchrones en CaML

Exercice 1 – Mobilité : Vente en ligne

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.

Question 1

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

Question 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.

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 ?

Question 3

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.

Question 4

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.

Exercice 2 – Sélection

Question 1

Ecrire un programme qui lance trois threads fournisseurs qui mettent un temps aléatoire pour envoyer, chacun sur un canal différent, une produit différent (par exemple les chaines de caractères ″`pommme`″, ″`orange`″ et ″`banane`″) et un quatrième thread qui doit recevoir tous les produits. Les threads fournisseurs se relancent un nombre fini de fois.

Exercice 3 – Broadcast

On se propose d'écrire un serveur de broadcast, 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 threads écouteurs, chacun associé à un canal de sortie. Ils attendent un temps aléatoire puis recoivent l'information sur leur canal de sortie associé.

Question 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 ; on utilisera wrap.

Question 2
Ecrire le programme entier. Utiliser un mutex pour afficher les sorties des threads écouteurs.

Exercice 4 – Monte-Carlo

Dans cet exercice on se propose de calculer π par la méthode de MonteCarlo !

On tire aléatoirement un point $M(x, y)$ avec $0 < x < 1$ et $0 < y < 1$.
Il appartient au disque (plus exactement au quart supérieur droit du disque) de centre $(0, 0)$ et de rayon 1 si et seulement si $x^2 + y^2 \leq 1$.
Tout se base sur le fait simple que la probabilité que ce point $M(x, y)$ appartient au disque est $\pi/4$.
Pour calculer π, il suffit donc de tirer aléatoirement des points $M(x, y)$ avec $-1 < x < 1$ et $-1 < y < 1$ et de calculer le rapport entre le nombre de points tirés appartenant au disque et le nombre total de points tirés.

Question 1
Ecrire un programme mettant en jeu :
- deux `channel`
- de ref sur des grands entiers
- 4 threads

On souhaite avoir deux threads qui testents des points et deux threads comptant respectivemnt les point dans le cercle et les points hors du cercle.

Exercice 5 – Examen 2012

Les étudiants de l'UE MI019 PC2R jouent au jeu PPNU (plus petit nombre unique) qui consiste à deviner le plus petit nombre qui ne sera choisi par aucun autre joueur. En pratique, chaque étudiant écrit une fonction (de type `unit -> int` en OCaml) qui rend un nombre (les fonctions peuvent faire des opérations d'entrées/sorties avec le monde extérieur, mais on suppose qu'elles finissent toutes par terminer en temps raisonnable).
Soit `jVec` le vecteur qui contient toutes les fonctions des joueurs. En OCaml, `jVec` est de type `(unit -> int) array`.
Attention à bien gérer la synchronisation de manière à ne pas obtenir un « simple programme séquentiel » !

Question 1
Créer un canal d'événements nommé `cRes` qui vous servira à envoyer les résultats des fonctions des joueurs (donc des paires d'entiers : numéro du joueur $\times$ nombre choisi) a un thread jouant le rôle d'arbitre.

Question 2
Donner une définition du thread arbitre qui se charge de garder en mémoire le plus petit nombre unique qu'il a reçu ainsi que le numéro du joueur bien sûr. À la fin du jeu, le thread arbitre affiche le numéro du gagnant ainsi que le nombre qu'il a choisit, et meurt.

Question 3
Créer un canal d'événements nommé `cFun` qui vous servira à envoyer les fonctions des joueurs aux threads chargés de les appliquer. Chaque fonction sera envoyée avec son numéro.

Question 4
Créer un thread qui envoie toutes les fonctions du vecteur `jVec` dans le canal d'événement `cFun`.

Question 5
Créer un lot de `nTh` threads, chacun se chargeant, **tant qu'il est nécessaire de le faire**, de choisir une fonction dans le canal d'événements `cFun`, de l'exécuter, et d'envoyer le résultat de la fonction sur le canal `cRes`.

Question 6
Écrire le code qui manque pour lancer le programme et afficher le résultat (une seule fois, et si on veut rejouer, on relance le programme). Si aucun nombre choisi n'est unique, il n'y a aucun gagnant, et on pourra déclarer vainqueur le joueur `-1`.

Question 7
Expliquer (en le moins de mots possible) pourquoi vous n'avez pas eu besoin d'utiliser des *mutex*. (Toutefois, si vous avez vraiment dû en utiliser, expliquer pourquoi ils vous ont été indispensables.)

Question 8
Votre programme est-il robuste aux limitations des ressources du système ? Expliquer pourquoi (en le moins de mots possible). (Selon vos choix d'implantation, la réponse peut être oui comme elle peut être non, ce qui compte ici est donc l'explication.)

Question 9
Bonus : expliquer comment gérer une durée de vie limitée (*timeout*) pour chaque fonction des joueurs, ou bien implantez-le.

Voici les signatures des fonctions que vous pouvez utiliser si vous choisissez C en langage d'implantation.

```
typedef struct channel* channel_t;
typedef struct event* event_t;
channel_t new_channel ();
event_t send (channel_t, void*);
event_t receive (channel_t);
event_t always (void*);
event_t choose2 (event_t, event_t);
event_t choose3
  (event_t, event_t, event_t);

event_t chooseN (event_t*);
void* sync (event_t);
void* select2 (event_t, event_t);
void* select3 (event_t, event_t, event_t);
void* selectN (event_t*);
int poll (event_t, void**);
/* renvoie TRUE si une valeur est disponible,
auquel cas elle est stockée dans le second paramètre.
 */
```

Exercice 6 – 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.

Voici une interface simplifiée de “future” en Java et un équivalent OCaml :

Java	OCaml

`public interface Future<V>`
`  extends Runnable {`
`  public void run()`
`  public V call()`
`  public V get();`
`  public boolean isDone();`
`}`
	`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
— `call` ou `spwan`
  — `run` : lance le thread de calcul d'une valeur de type `V` en appelant la méthode `call`, la valeur retournée qui pourra être accédée ensuite par `get`
  — `spawn` : lance le thread de calcul de la fonction de type `'a -> 'b` sur le paramètre de type `'a` , et construire une `'b future`.

Les questions peuvent être traitées en Java ou en OCaml.

Question 1

Indiquer le langage que vous prenez, et donner les mécanismes de base que vous allez utiliser pour implanter les “futures”. Dans un premier temps on considère que le calcul demandé n'est jamais interrompu.

Question 2

Ecrire :
— soit en Java une classe abstraite `MyFuture` qui implante l'interface `Future<V>` en implantant le mécanisme de base des `future`, c'est-à-dire toutes les méthodes sauf `call`,
— soit en OCaml la définition du type `future` et les quatre méthodes du tableau précédent.

Question 3

Détailler le déroulement d'un des deux programmes suivants, où la classe `f`(ou la fonction `f`) calcule le ième nombre de Fibonacci [1], en fonction de votre implantation des “futures”.

Java	OCaml

`/* class f extends MyFuture<Integer> {`
`  ...} */`
`{ MyFuture<Integer> x1 = new f(5);`
`  MyFuture<Integer> x2 = new f(4);`
`  MyFuture<Integer> x3 = new f(6);`
`  int result =`
`  (x1.get()+x2.get()+x3.get())/3;`
`}`
	`(* 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;;`

Question 4

On cherche maintenant à traiter les cas où le calcul est interrompu par une exception pendant le calcul.

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

Java	OCaml
`public interface FutureExc<V> extends Future<V> { public boolean isCancelled(); }`	`val isCancelled : 'a future -> bool`

— `isCancelled` retourne un booléen à `true` si le calcul a été interrompu avant d'avoir retourné un résultat.
Attention cette prise en compte modifie le comportement de `get` qui lors d'un accès redéclenche l'exception (au sens `Runtime Exception` de Java) qui a provoqué l'arrêt du calcul. Si vous modifier la création de "futures" vous pouvez indiquer uniquement les différences par rapport à la question précédente. Implanter ce nouveau comportement.

Question 5

On cherche à définir un itérateur à base de "futures" qui applique un même traitement sur l'ensemble des éléments d'une structure linéaire classique (à la `ArrayList<T>` ou à la `'a array` ou `'a list` et qui retourne la structure linéaire des (futurs) résultats (à la `map`). Chaque calcul intermédiaire est lui-même un "future". Proposer une architecture permettant de lancer de tels calculs puis l'implanter.

Question 6

On cherche à ajouter un future de synchronisation sur l'ensemble des calculs de la structure indiquant que tous les calculs intermédiaires sont finis. Cela permet d'avoir d'une part accès au ième calcul sans avoir à attendre la fin de tous les autres et d'autre part avoir accès à l'ensemble de la structure alors complètement calculée. Indiquer comment implanter ce mécanisme à partir de la question précédente.