Master d'informatique | **module PC2R (4I507)** | **Université Pierre et Marie Curie**
Documents autorisés | *Durée 2 heures* | **Année 2016-2017**

Examen de rattrapage du 15 juin 2017

Exercice 1 : Générateur de nombres aléatoires (OCaml, Java ou C)

On cherche à définir un générateur de nombres aléatoires en utilisant le non-déterminisme des processus légers pour calculer une racine (*seed*) qui sera passée à l'initialiseur `Random.init` (Random.setSeed()) du générateur de nombres aléatoires de la bibliothèque `Random`. Les fonctions `Random.int` et `Random.float` retournent respectivement un entier (un flottant) entre 0 (0.0) et la borne passée en argument.

On commence par définir la classe abstraite suivante :

pseudo code OCaml

```
class virtual gen_alea_abs n  =
object(self)
  val mutable racine = 0
  method get_racine = racine
  method virtual init :  int
  method int i = Random.int i
  method float f = Random.float f
  initializer
     racine <- self#init n;
     Random.init racine
end;;
```

pseudo code Java

```
abstract class GenAlea {
  int racine = 0;
  GenAlea(int n) {
    this.init(n);
    Random.init(racine);
  }
  int get_racine() { return racine;}
  abstract int init(int x);
  void int(int i){Random.int(i);}
  void float(float f){Random.float(f);}
}
```

On cherche à écrire la sous-classe concrète `gen_alea` (GenAlea) qui implante la méthode `init`. Celle-ci lance deux threads qui tirent l'une à pile (1) et l'autre à face (0). Chacun de ces threads stocke son résultat dans une même case mémoire, ainsi seulement un des deux résultats est conservé. À partir de `n` doubles tirages, on obtient une liste (de longueur n) de `bits` qui correspond à un nombre entier positif. On se limitera à une longueur maximale de 30.

1. Ecrivez la classe `gen_alea` qui hérite de la classe `gen_alea_abs` et définit la méthode `init` comme précédemment décrite. Comme il y a plusieurs implantations possibles, justifiez vos choix techniques.

2. Indiquez les calculs effectués lors de l'appel de `new gen_alea 16;;`. Pouvez-vous prévoir le résultat ?

3. Que se passe-t-il si à la place des deux threads, on choisit deux fair threads contrôlés par un seul *scheduler* ? ou deux fair threads où chacun est contrôlé par un *scheduler* séparé ? Justifiez vos réponses.

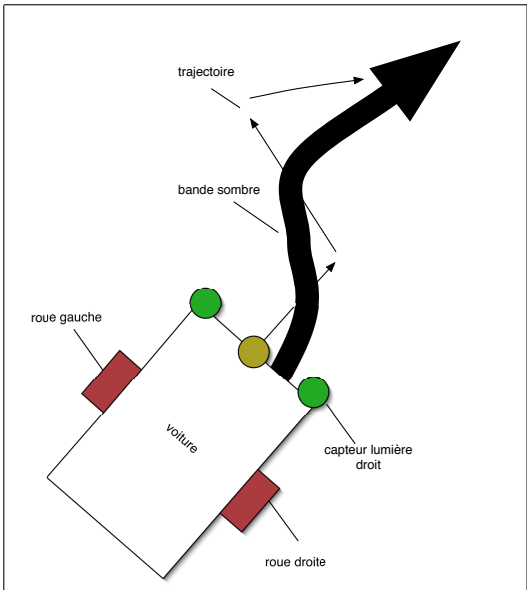
Exercice 2 : Déplacements d'un robot (Esterel)

On cherche à programmer les déplacements d'un petit robot en Esterel. Chaque robot peut avancer grâce à deux moteurs (un pour la roue gauche et un pour la roue droite) qui sont connectés sur les ports de sortie G et D. Chaque robot possède trois capteurs de lumière : un au dessus de chaque roue et le dernier placé au centre du capot avant. Ces différents composants sont nommés de la manière suivante : `MOTEUR_G, MOTEUR_D, LUMIERE_G, LUMIERE_C, LUMIERE_D`.

Pour ce robot, l'Api de programmation Esterel définit les signaux et constantes suivants :

— pour les capteurs de lumière, les signaux LUMIERE_FAIBLE_GAUCHE (resp. LUMIERE_FAIBLE_CENTRE, LUMIERE_FAIBLE_DROITE) et LUMIERE_FORTE_GAUCHE (resp. LUMIERE_FORTE_CENTRE, LUMIERE_FORTE_DROITE) sont déclenchés quand le capteur considère (par rapport à un seuil donné) que la lumière est faible ou forte. Pour notre exemple, lorsque le capteur droit passe d'une zone blanche à la bande sombre, le signal LUMIERE_FAIBLE_DROIT est émis. Lorsqu'il passe de la bande sombre à la zone blanche, le signal LUMIERE_FORTE_DROITE est émis.
— pour les moteurs le signal VITESSE_MOTEUR_GAUCHE(integer) (resp. VITESSE_MOTEUR_DROITE) définit une vitesse à appliquer sur le moteur. Les vitesses sont entières et comprises entre 0 et 255. Enfin le signal de direction du moteur DIR_MOTEUR_GAUCHE(integer) (resp. DIR_MOTEUR_DROIT) définit le sens de la rotation du moteur GAUCHE (resp. DROIT). Les contantes MOTEUR_AVANCE et MOTEUR_RECULE indiquent le sens positif et négatif de la rotation du moteur (et donc des roues).

Voici un exemple de suivi d'une bande sombre par un robot utilisant son capteur de lumière central (le robot tourne quand les deux vitesses des roues sont différentes ou en sens contraire) :



trajectoire

```
module suivi1 :
  input LUMIERE_FAIBLE_CENTRE, LUMIERE_FORTE_CENTRE;
  output VITESSE_MOTEUR_GAUCHE : integer,
         VITESSE_MOTEUR_DROITE : integer,
         DIR_MOTEUR_GAUCHE : integer,
         DIR_MOTEUR_DROITE : integer;
  constant MOTEUR_AVANCE, MOTEUR_RECULE : integer;

  emit DIR_MOTEUR_GAUCHE(MOTEUR_AVANCE);
  emit DIR_MOTEUR_DROITE(MOTEUR_AVANCE);

  loop
    emit VITESSE_MOTEUR_GAUCHE(150);
    emit VITESSE_MOTEUR_DROIT(50);
    await LUMIERE_FAIBLE_CENTRE;
    await LUMIERE_FORTE_CENTRE;
    emit VITESSE_MOTEUR_GAUCHE(50);
    emit VITESSE_MOTEUR_DROIT(150);
    await LUMIERE_FAIBLE_CENTRE;
    await LUMIERE_FORTE_CENTRE;
  end loop.
```

programme Esterel

1. Expliquer la trajectoire du robot de la figure précédente en vous basant sur la position initiale et le programme donné.
2. On cherche maintenant à décrire le comportement d'un robot roulant à l'intérieur d'une route (définie par une zone blanche encadrée par 2 lignes noires) et utilisant les capteurs de lumière gauche et droite. Comme chaque roue est autonome et chaque capteur aussi, proposez un algorithme pour le suivi d'une bande en parallélisant le comportement de chaque couple (moteur, capteur) d'un côté. On suppose que la route est suffisamment large pour permettre aux deux capteurs d'être sur la zone blanche. Dans la position initiale, le robot est entre 2 lignes noires et chaque capteur de lumière est sur l'intérieur de la route (zone blanche).
   Implanter votre solution en Esterel. On supposera tous les signaux et constantes de l'énoncé prédéfinis.

Exercice 3 : Pierre-Ciseaux-Puit-Feuille réseau (OCaml, Java ou C)

Le but de ce problème est de définir le cadre pour des jeux en réseau à plusieurs joueurs. On fera particulièrement attention à la généricité du code développé. On se servira de ce cadre pour définir un serveur d'un jeu classique de cours d'école : Pierre-Ciseaux-Puit-Feuille ou Chi-Fou-Mi. Les règles du jeu sont les suivantes : il y a `n` joueurs, à chaque tour chaque joueur propose un des coups possibles (Pierre, Ciseaux, Puit ou Feuille) sachant que les gains ou pertes sont les suivants :

$\rightarrow$	Pierre	Ciseaux	Puit	Feuille
Pierre		G	P	P
Ciseaux	P		**P**	G
Puit	G	G		P
Feuille	G	P	G	

Les Ciseaux tombent dans le Puit ou se cassent sur la Pierre, dans ce cas ils perdent le point, par contre ils coupent la feuille et gagnent le point.

Le serveur doit donc gérer les tâches suivantes :
- Attendre que tous les joueurs (pour un nombre fixé) soient connectés et leur indiquer le début de partie ;
- Faire pour un nombre fixé de coups :
  - Attendre les coups de tous les joueurs ;
  - Indiquer les coups joués à tous les joueurs ;
  - Calculer les scores de tous les joueurs ;
  - Envoyer ces scores à tous les joueurs.

Voici quelques indications pour réaliser les questions : chaque client est géré par un *thread* différent sur le serveur. Tous ces *thread*s partagent le tableau des derniers coups joués et le tableau des scores. Quand un coup est reçu par un *thread* gérant un client, son coup est stocké dans le tableau des coups et le *thread* attend que tous les coups soient joués pour les envoyer à son client puis calcule son score, attend que tous les scores soient calculés et les envoie à son client. Chaque *thread* calcule le score du joueur qu'il traite par rapport à tous les autres joueurs.

On définit le protocole de communication suivant :
- du client vers le serveur : la chaine de caractères correspondant à un coup ("FEUILLE") ;
- du serveur vers le client :
  - "IDENT n" pour donner un numéro au joueur (ce qui permet de retrouver son coup et son score dans les envois futurs) ;
  - "DEBUT" pour le démarrage de la partie ;
  - "FIN" pour la fin de partie ;
  - "COUPS n" suivi de n lignes pour l'envoi des coups joués (incluant le sien) ;
  - "SCORE n" suivi de n lignes pour l'envoi des scores (incluant le sien).

Vous pouvez vous inspirer des différents exemples donnés en cours et en TD, et même citer des parties des polycopiés. On supposera connu la fonction `compte` de type OCaml `string -> string -> int` (ou son équivalent C ou Java) qui prend deux coups joués et retourne $+1$, 0 ou -1 si le premier argument gagne, est identique ou perd, vis-à-vis du second selon les règles énoncées précédemment.

1. Donnez une hiérarchie de classes ou une organisation de fonctions pour implanter un tel serveur en indiquant en particulier les variables partagées par les différents processus. Présentez l'organisation générale de votre serveur.
2. Ecrivez la partie attente de connexion de tous les joueurs du serveur.
3. Ecrivez la partie traitement des réponses d'un joueur en suivant le protocole indiqué. Il est nécessaire dans les réponses à un joueur que celui-ci ait tous les éléments pour vérifier que le calcul de son score est correct, vous précisez (sans l'impanter) le calcul effectué alors du côté du client.
4. Ecrivez le lancement de votre serveur pour des parties de 5 joueurs en dix coups.
5. Expliquez comment tester le serveur avec `telnet`.
6. Décrivez sans les implanter les modifications à apporter à votre architecture pour autoriser plusieurs parties simultanées.

Exercice 4 : Dictionnaire en RMI (Java)

On cherche à implanter un service de dictionnaire en utilisant le mécanisme d'appel distant RMI de la plate-forme d'exécution Java.
Pour implanter un dictionnaire, on utilisera l'interface `Map<K,V>` et son implantation `TreeMap<K,V>` où le paramètre de type `K` correspond au type de la clé et le paramètre de type `V` au type de la valeur associée à la clé. On utilisera principalement les deux méthodes suivantes de la classe `TreeMap<K,V>` :

```
import java.rmi.*;

public interface RDictInt extends Remote   {
    public String get (String o) throws RemoteException;
    public String put (String c, String v) throws RemoteException;
}
```

- `V get(Object c)` : retourne la valeur associée à la clé `c` ou `null` s'il n'y en a pas ;
- `V put(K c, V v)` : associe la valeur `v` à la clé `c` ; s'il y a déjà une liaison la nouvelle valeur remplace l'ancienne qui est alors retournée si elle existe et `null` sinon.

On définit alors l'interface distante d'un dictionnaire de la manière suivante :

```
import java.rmi.*;

public interface RDictInt extends Remote   {
    public String get (String o) throws RemoteException;
    public String put (String c, String v) throws RemoteException;
}
```

1. Ecrire une la classe `RDict` implantant cette interface.
2. Ecrire un serveur qui crée deux objets de la classe `RDict` et les expose sous les noms `"dico1"` et `"dico2"`. Ecrire les commandes pour lancer le serveur sur la machine `exam.upmc.fr` (sur laquelle vous êtes connecté) sur le port 2017.
3. Ecrire un client simple qui recherche le nom `"univers"` d'abord dans `"dico1"`, puis si celui-ci n'existe pas recherche dans `"dico2"`. En cas d'échec dans la deuxième recherche le client déclenche l'exception `Not_found`, préalablement définie, sur le client.
4. Les méthodes `get` et `put` de la classe `TreeMap` ne sont pas synchronisées (au sens du mot clé `synchronized`). Donner un exemple où la recherche d'un mot dans un dictionnaire distant ne retourne pas la valeur attendue par rapport à l'état du dictionnaire au début de la recherche. Indiquer ensuite comment modifier votre programme pour éviter ce comportement et implanter ces indications.
5. On veut maintenant rechercher dans les 2 (potentiellement n) dictionnaires sans blocage en implantant un mécanisme de rappel RMI. Dans l'exemple, l'idée est de lancer la recherche sur les 2 dictionnaires et de pouvoir utiliser le résultat dès qu'une requête a retourné un résultat différent de `null` en faisant attention que la deuxième requête n'efface pas ce résultat.
   (a) Indiquer la synchronisation souhaitée du côté du client et du côté serveur si nécessaire.
   (b) Modifier l'interface, la classe, le serveur puis le client en fonction de la synchronisation indiquée.