Master d'informatique | **module PC2R** | **Université Pierre et Marie Curie**
Documents autorisés | *Durée 2 heures* | **Année 2012-2013**

Examen de rattrapage du 14 mai 2013

Exercice 1 : Usine robotisée (Fair Threads)

Une usine est composée de 2 chaînes (C1 et C2) qui fabriquent chacune une pièce spécifique et un transporteur.
La chaîne C1 fonctionne en continu et est composée de :

- 1 panier P1 qui contient à chaque instant au plus une pièce,
- 3 robots (R1A, R1B et R1C) qui usinent chacun une pièce identique en prenant un certain temps (aléatoire, voir la fonction `alea()`) et tentent de ranger chacun leur pièce usinée dans le panier P1. Chaque robot ne doit en aucun cas bloquer les autres robots de la chaîne sauf pour ranger sa propre pièce.

La même chose pour la chaîne C2 avec son panier P2 et ses 3 robots R2A, R2B et R2C.
Le transporteur passe en continu par le panier P1 pour le vider puis par le panier P2 pour le vider. Si l'un des paniers est vide, il attendra. Une fois les 2 pièces prises, il les range à l'extérieur de l'usine et recommence le cycle.

On donne un squelette d'un programme en C mais vous pouvez aussi répondre en Java ou OCaml.
Vous pouvez et/ou devez ajouter les élements nécessaires pour le fonctionnement de l'usine. On évitera de faire des attentes actives.

Question 1 Compléter la procédure `r1()` pour les robots R1A, R1B et R1C.

Question 2 Compléter la procédure `t()` pour le transporteur.

Question 3 Compléter la procédure `main()` pour faire fonctionner l'usine.

Extrait de programme `usine.c`

```
#include "fthread.h"
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
ft_thread_t    R1A, R1B, R1C, R2A, R2B, R2C, T;
ft_scheduler_t C1, C2;
...
char *P1, *P2;
int alea (int n) {
  return (int)(1.0 * random() / RAND_MAX * n) + 1;
}
void r1 (void *arg) {
  char *nom = (char *)arg;
  int   i;
  char piece[200];
  ...
}
void r2 (void *arg) {
  char *nom = (char *)arg;
  int   i;
  char piece[200];
  ...
}
void t (void *arg) {
  ...
}
```

```
int main (void) {
  P1            = "";     /* panier P1 vide */
  C1            = ft_scheduler_create ();
  R1A           = ft_thread_create(C1, r1, NULL, "R1A");
  R1B           = ft_thread_create(C1, r1, NULL, "R1B");
  R1C           = ft_thread_create(C1, r1, NULL, "R1C");

  ...

  P2            = "";     /* panier P2 vide */
  C2            = ft_scheduler_create ();
  R2A           = ft_thread_create(C2, r2, NULL, "R2A");
  R2B           = ft_thread_create(C2, r2, NULL, "R2B");
  R2C           = ft_thread_create(C2, r2, NULL, "R2C");

  ...

  T             = ft_thread_create(C1, t, NULL, NULL);

  ft_scheduler_start(C1);
  ft_scheduler_start(C2);
  ft_exit();
  fprintf(stdout, "***** Fin de programme. *****\n");
  return 0;
}
```

Exécution du programme usine (exercice FT)

```
$ ./usine

Le robot R1A fabrique la piece PIECE_R1A_0 avec un certain temps
Le robot R2C fabrique la piece PIECE_R2C_0 avec un certain temps
Le robot R2A fabrique la piece PIECE_R2A_0 avec un certain temps
Le transporteur T attend dans C1 pour prendre une piece.
Le robot R1C fabrique la piece PIECE_R1C_0 avec un certain temps
Le robot R2B fabrique la piece PIECE_R2B_0 avec un certain temps
Le robot R1B fabrique la piece PIECE_R1B_0 avec un certain temps
Le robot R1B depose la piece PIECE_R1B_0 dans le panier P1.
Le transporteur T prend dans P1 la piece PIECE_R1B_0.
Le robot R1B fabrique la piece PIECE_R1B_1 avec un certain temps
Le transporteur T attend dans C2 pour prendre une piece.
Le robot R2C depose la piece PIECE_R2C_0 dans le panier P2.
Le transporteur T prend dans P2 la piece PIECE_R2C_0.
Le transporteur T range les 2 pieces a l'exterieur.
Le robot R2C fabrique la piece PIECE_R2C_1 avec un certain temps
Le transporteur T attend dans C1 pour prendre une piece.
Le robot R1B depose la piece PIECE_R1B_1 dans le panier P1.
Le transporteur T prend dans P1 la piece PIECE_R1B_1.
Le robot R1B fabrique la piece PIECE_R1B_2 avec un certain temps
Le transporteur T attend dans C2 pour prendre une piece.
Le robot R2A depose la piece PIECE_R2A_0 dans le panier P2.
Le robot R1C depose la piece PIECE_R1C_0 dans le panier P1.
Le robot R1C fabrique la piece PIECE_R1C_1 avec un certain temps
Le transporteur T prend dans P2 la piece PIECE_R2A_0.
Le transporteur T range les 2 pieces a l'exterieur.
Le robot R2A fabrique la piece PIECE_R2A_1 avec un certain temps
Le transporteur T prend dans P1 la piece PIECE_R1C_0.
Le transporteur T attend dans C2 pour prendre une piece.
Le robot R1A depose la piece PIECE_R1A_0 dans le panier P1.
Le robot R1A fabrique la piece PIECE_R1A_1 avec un certain temps
Le robot R1B attend pour deposer la piece PIECE_R1B_2
   dans le panier P1
```

Exercice 2 : Serrue digitale (Esterel)

On cherche à simuler une serrure digitale ayant le principe de fonctionnement suivant :

- Au départ, elle émet à chaque `tick` le signal `ROUGE`.
- Pour passer au `VERT`, l'utilisateur doit rentrer successivement 4 codes valués (un code valué par `tick`). Leurs valeurs doivent être respectivement `code1`, `code2`, `code3` et `code4`,
- Une fois le premier code rentré, l'utilisateur doit absolument rentrer dans une durée de 10 `tick` les 3 codes restants (voir l'exemple d'exécution ci-dessous). Chaque code peut être séparé du prochain par un ou plusieurs `tick`.
- Si l'utilisateur n'a pas pu rentrer ces 4 codes à temps, il doit tout recommencer à partir du début.
- S'il a rentré un mauvais code (voir l'exemple d'exécution ci-dessous), le nombre de tentatives (`nb_tentatives`) est augmenté de 1 et il doit attendre `duree * nb_tentatives` fois `tick` avant de pouvoir recommencer à partir du début. Pendant cette attente, le signal `ATTENTE` est émis à chaque `tick` et bien sûr toute tentative de rentrer des codes est ignorée.
- S'il a réussi, l'émission du signal `ROUGE` est interrompue et celle du signal `VERT` commence pour une durée de 10 `tick`. Pendant cette période, toute tentative de rentrer des codes est ignorée. On évitera d'émettre dans un même `tick` les signaux `ROUGE` et `VERT`.
- Après l'émission des signaux `VERT`, celle du `ROUGE` reprend comme au départ.

1. Compléter le programme ci-dessous pour cette simulation.

```
module serrure :

input  CODE : integer;
output ROUGE, VERT, ATTENTE;

constant code1 = 1, code2 = 2,
         code3 = 3, code4 = 4, duree = 10 : integer;
  var nb_tentatives := 0 : integer in

   ...

  end var

end module

******************************************

$ serrure
serrure> ;;;
--- Output: ROUGE
--- Output: ROUGE
--- Output: ROUGE
serrure> CODE(1);;;CODE(2);;CODE(3);CODE(4);;;;;;;
--- Output: ROUGE
--- Output: ROUGE
--- Output: ROUGE
--- Output: ROUGE
--- Output: ROUGE
--- Output: ROUGE
--- Output: ROUGE
--- Output: VERT
--- Output: VERT
--- Output: VERT
--- Output: VERT
--- Output: VERT
serrure> ;;;;
--- Output: VERT
--- Output: VERT
--- Output: VERT
--- Output: VERT
serrure> ;;;
--- Output: ROUGE
--- Output: ROUGE
--- Output: ROUGE
serrure> CODE(1);CODE(2);CODE(4);CODE(3);;;;;;;;;;
--- Output: ROUGE
--- Output: ROUGE
--- Output: ROUGE
--- Output: ATTENTE
--- Output: ATTENTE
--- Output: ATTENTE
--- Output: ATTENTE
--- Output: ATTENTE
--- Output: ATTENTE
--- Output: ATTENTE
--- Output: ATTENTE
--- Output: ATTENTE
--- Output: ATTENTE
serrure> ;;
--- Output: ROUGE
--- Output: ROUGE
serrure>
```

Exercice 3 : Client-serveur pour la synchronisation de date (OCaml et Java)

Le but de cet exercice est d'implanter un service de mise à jour des dates, à la manière de `rdate` (RFC 868). Le serveur sera écrit en OCaml et les clients en Java. Voici le schéma du service de date, `S` est le serveur et `C` un client :

```
S: écoute sur le  port 37.

C: connexion sur le port 37.

S: envoi du temps comme un nombre entier en format texte.

C: réception du temps.

C: fermeture de la connexion.

S: fermeture de la connexion.
```

Le temps de ce protocole est compté en secondes écoulées depuis le premier janvier 1900 (GMT). En OCaml la fonction `Unix.time` retourne le temps écoulé depuis le 1er janvier 1970 en secondes. En Java on utilisera la classe `java.util.Date` dont le constructeur sans argument initialise l'instance avec la date courante. La méthode `long getTime()` de la classe `Date` récupère le nombre de millisecondes écoulée depuis une date fixe à la date actuelle et la méthode `void setTime(long)` la modifie. Les entiers java (de type *long*) argument ou résultat de ces méthodes correspondent au nombre de millisecondes écoulées depuis le 1er janvier 1970. On appellera `decal` la constante du nombre de secondes entre 1/1/1900 et 1/1/1970. On utilisera la méthode statique (de la classe `Long`) `public static long valueOf(String s) throws NumberFormatException` pour convertir une chaîne en *long* ;
Dans les différentes questions, vous pouvez supposer connus différents fragments de code issus des polycopiées de cours en indiquant bien lesquels vous utilisez.

1. Ecrire le serveur OCaml répondant à ce protocole. Le serveur doit pouvoir répondre à plusieurs requêtes simultanées.
2. Ecrire un client Java simple, demandant une date et l'affichant. Il n'est pas demandé d'écrire la fonction de conversion de secondes en date.

On cherche à améliorer ce protocole client-serveur pour contre balancer les effets dus à l'encombrement du réseau. Pour cela le client lance 3 requêtes distinctes simultanément en conservant la date de l'émission. Quand le serveur répond à une requête, le client construit 1 triplet : (date émission, date envoyée du serveur, date de réception). Quand les 3 triplets sont construits, le client détermine alors la différence de date entre lui et le serveur et se met à jour.

3. Proposer une solution pour la synchronisation des réponses du serveur à ces triples requêtes et indiquer comment effectuer la mise à jour de l'heure en tenant compte des 3 triplets.
4. Implanter votre solution et modifier votre client en conséquence.

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>` :

- `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 2013.
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.