Master d'informatique | **module PC2R (4I507)** | **Université Pierre et Marie Curie**
Documents autorisés | *Durée 2 heures* | **Année 2017-2018**

Examen réparti du 29 mars 2018

Exercice 1 : Livraison avec FairThread

Nous proposons un système de livraison utilisant les fairthreads composé de
- un centre où les `cnb_A` produits à destination de la région A et `cnb_B` produits à destination de la région B sont stockés,
- une chaîne de tri séparée du centre,
- un `transporteur`,
- des `livreurs[]`.

avec les contraintes suivantes :
- Le centre est automatiquement rempli lorsqu'il ne reste aucun produit, c'est-à-dire lorsque `cvide == TRUE` (et `cnb_A == 0 et cnb_B == 0`). C'est la tâche de la procédure `void *_centre (void *x)` qui va générer deux valeurs aléatoires entre 0 (compris) et 5 pour `cnb_A` et `cnb_B` (`aleatoire(5)`) et qui mettra `cvide` à `FALSE`.
- La chaîne de tri stocke les `ch_nb_A` produits pour la région A et les `ch_nb_B` produits pour la région B. Et ces deux valeurs doivent être toujours inférieures ou égales à `NB_MAX`.
- Lorsque le `transporteur` n'a plus aucun produit à livrer (`tnb_A == 0` et `tnb_B == 0`), il doit aller au centre pour prendre tous les produits disponibles pour les transporter jusqu'à la chaîne de tri aux livreurs. Si aucun produit n'est disponible, il attendra au centre.
- Une fois qu'il est dans la chaîne de tri, le `transporteur` doit y vider partiellement ou totalement tous ses produits en une ou plusieurs fois en respectant la capacité de stockage de la chaîne.
- Une fois tous ses produits vidés, le `transporteur` peut alors aller chercher d'autres produits au centre.
- Chaque livreur est dédié à une seule région (A ou B). Il prend dans la chaîne un seul produit de sa région et le livre, puis il recommence une fois le produit livré.
- Si aucun produit de sa région n'est disponible dans la chaîne, il attendra la prochaine livraison du transporteur.

Le système fonctionne de la manière suivante :
- Le transporteur et les livreurs travaillent d'une manière continue et autonome.
- La présence et la quantité des produits sont simulées par des variables de la manière suivante :
  - Pour le centre : `cnb_A`, `cnb_B` et `cvide`.
  - Pour la chaîne de tri : `chnb_A` et `chnb_B`.
  - Pour le `transporteur` : `tnb_A` et `tnb_B`.

A l'aide du canevas donné ci-dessous et en évitant toute boucle d'attente active et/ou inutile :

Question 1. Déclarer vos différents thread.

Question 2. Compléter la procédure `_centre()` pour remplir le centre.

Question 3. Compléter la procédure `_livreur_A()` pour la livraison. Inutile de définir la procédure `_livreur_B()`.

Question 4. Compléter la procédure `_transporteur()` pour transporter des produits du centre vers la chaîne de tri.

Question 5. Compléter la procédure `main()` pour faire fonctionner le système.

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include "fthread.h"

#define NB_LIVREURS      6
#define NB_MAX           5
#define min(a, b)        (((a) < (b)) ? (a) : (b))
#define max(a, b)        (((a) > (b)) ? (a) : (b))
#define TRUE             1
#define FALSE            0

/* Pour le centre */
/* ????? (1) ????? */
int cnb_A = 0, cnb_B = 0;
int cvide = TRUE;

/* Pour la chaine ch de tri */
/* ????? (2) ????? */
int ch_nb_A = 0, ch_nb_B = 0;

/* Pour le transporteur */
ft_thread_t transporteur;
int         tnb_A = 0, tnb_B= 0;

/* Pour les livreurs */
ft_thread_t livreurs[NB_LIVREURS];

int aleatoire (int max) {                    /*   int rand ();              */
  return max - (rand()*1.0 / RAND_MAX) * max; /*   0 <= rand() <= RAND_MAX   */
}

void *_centre (void *x) {

  /* ????? (3) ????? */
}

void _livreur_A (void *arg) {
  long  numero;
  int   duree;

  numero = (long)arg;

  /* ????? (4) ????? */
}

void _livreur_B (void *arg) {
  /***** meme corps que _livreur_A en echangeant A par B *****/
}

void _transporteur (void *arg) {
  int            duree;
  int            tmpA, tmpB;

  /* ????? (5) ????? */
}

int main (void)
{
  long i, numero;

  /* ????? (6) ????? */

  for (i = 0; i < NB_LIVREURS / 2; ++i) {
    numero   = i;
    livreurs[i] = ft_thread_create(ch, _livreur_A, NULL, (void *)numero);
  }
  for (i = NB_LIVREURS / 2; i < NB_LIVREURS; ++i) {
    numero   = i;
    livreurs[i] = ft_thread_create(ch, _livreur_B, NULL, (void *)numero);
  }

  transporteur = ft_thread_create(ch, _transporteur, NULL, NULL);

  ft_scheduler_start(ch);
  ft_exit();
  return 0;
}
```

Exercice 2 : Evaluation booléennes concurrentes, futures et par réseau

Certains langages de programmation (*Haskell*, *Erlang* et *Scheme*) pourraient supporter l'évaluation simultanée de toutes les parties d'une expression booléenne composée, telle que

$$a \,\mathbf{and}\, (b + c) > 0 \,\mathbf{and}\, !(isPrime(n))$$

Bien sûr, la valeur globale de l'expression booléenne est vraie si et seulement si chacune des parties est vraie. Si chacune des parties est coûteuse en termes de calcul, mais que la machine possède plusieurs processeurs, chacune de ces parties peut être évaluée simultanément pour accélérer les choses. Supposons qu'un langage de programmation modélise la notion d'une expression booléenne comme ceci

```
class BoolExpression
{
    // e'value l'expression et renvoie true ou false
    public bool evaluate ();
}
```

Nous allons implémenter des fonctions d'évaluation concurrentes des expressions booléennes composées qui acceptent un vecteur de `BoolExpression`, les évaluent et renvoient `True` si et seulement si tous les résultats sont vrais. Pour ce faire, nous considérons tout d'abord la possibilité de réaliser la fonction `concurrentOr` qui effectue un "OU" booléen sur un ensemble d'expressions et suit la signature suivante :

```
public bool concurrentOr (ArrayList<BoolExpression>);
```

Question 1. Implémenter la fonction `concurrentOr` en utilisant un système multi-thread. Vous pouvez supposer qu'aucune des expressions `BoolExpression` du vecteur fourni ne dépend récursivement de `concurrentOr`. Toutefois, il peut y avoir d'autres appels à `concurrentOr` s'exécutant simultanément.

Quelques contraintes et conseils :

- Chacune des `BoolExpression` fournies doit être évaluée à l'aide d'un ensemble de threads.
- Vous pouvez supposer que `BoolExpression.evaluate()` est thread-safe.
- Votre implémentation doit utiliser une condition_variable_any pour bloquer efficacement jusqu'à ce que toutes les expressions `BoolExpression` aient été évaluées. Cependant ceci ne doit pas bloquer la réactivité du système

Question 2. Nous allons maintenant implémenter la fonction `concurrentAnd` qui effectue le "ET" logique. Même si la signature et le fonctionnement sont les mêmes que pour "OU", notez que dans le cas d'un "ET" logique, l'expression est fausse dès que l'une est fausse. Nous pourrions donc mettre en œuvre une forme d'évaluation de court-circuit, c'est-à-dire que nous pourrons arrêter le calcul dès qu'une des expressions est fausse. Décrivez les modifications à apporter et proposez une implémentation.

Nous voulons maintenant considérer qu'il peut y avoir des appels récursifs (car une certaine expression booléenne peut contenir un nombre indéterminé de combinaisons de OU et ET logiques.

Question 3. Proposer (sans implémentation) une solution, en explicitant les contraintes éventuelles sur le système en termes de vivacité, bloquage et mémoire partagée.

Question 4. Ecrire une implémentation simplifiée de votre système.

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 quand il est utilisé avec TCP, `S` est le serveur et `C` un client :

```
S: écoute sur le  port 1037.

C: connexion sur le port 1037.

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 serveur écoute pour une connexion sur le port 1037, quand la connexion est établie, le serveur renvoie une valeur temps sous forme d'une chaîne de caractères. 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 le 1/1/1900 et le 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 du polycopié de cours en indiquant bien lesquels vous utilisez, il est inutile de les recopier.

Question 1. Ecrire le serveur OCaml répondant à ce protocole. Le serveur doit pouvoir répondre à plusieurs requêtes simultanées.

Question 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 contrebalancer les effets dus à l'encombrement du réseau. Pour cela le client lance 4 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 4 triplets sont construits, le client détermine alors la différence de date entre lui et le serveur et se met à jour en en tenant compte.

Question 3. Proposer une solution pour la synchronisation des réponses du serveur à ces quadruples requêtes et indiquer comment effectuer la mise à jour de l'heure en tenant compte des 4 triplets.

Question 4. Implanter votre solution en modifier votre client en conséquence.