SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

PC3R - TD2: Préemption et Coopération

Equipe Enseignante PC2R

04/02/2021

Lien vers les ressources de l'UE: `https://www-master.ufr-info-p6.jussieu.fr/2019/PC2R`

1 Préemption

1.1 [Sémantique] Trains

On modélise un problème de circulation de trains:

- on dispose d'une voie v_1 sur laquelle les trains circulent d'Est en Ouest et, juste à côté, une voie v_2 sur laquelle les trains circulent d'Ouest en Est.
- les voies partagent un segment s surlequel, de fait, des trains peuvent rouler dans les deux directions.
- pour éviter un accident (deux trains roulant dans des directions opposées qui empruntent simultanément le segment s), des feux bicolores f_1 et f_2 sont placés sur v_1 et v_2 avant (dans la direction de la voie) le segment s.
- les feux sont reliés à un système de quatre capteurs c_1^e, c_2^e, c_1^o, c_2^o détectant la présence (ou l'absence) d'un train sur chacune des voies, avant et après s?

1. Faire un schéma du système.
2. Enoncer deux propriétés de sûreté (*safety*: le système n'arrive pas dans un état dangereux) et deux propriétés de vivacité (*liveness*: le système finit par arriver dans un état désirable) pour ce système.
3. Décrire l'ensemble des états possibles du système (en considérant qu'il s'agit du produit cartésient des ensembles des états possibles de ses composants).
4. Dessiner l'automate de contrôle du système, ses états internes sont les états de la question précédente, il réagit aux capteurs et agit sur les feux.
5. Les propriétés énoncées plus haut sont-elles vérifiées ?
6. En déduire un pseudocode pour l'automate de contrôle.

1.2 [OCaml, Java, Rust] Comptage au Musée

On modélise un système de comptage d'entrées/sorties dans un musée:

- un compteur partagé compte le nombre de visiteurs actuellement en train de visiter le musée,
- deux processus "entrée" sont initialisés avec une valeur représentant le nombre total de visiteurs entrant ce jour par cette entrée, et font rentrer les visiteurs (incrémentent le compteur) un par un,
- deux processus "sortie" font sortir les visiteurs (décrémentent le compteur) un par un (il ne peuvent pas faire sortir un visiteur d'un musée vide).
- le processus principal lance les quatre processus, dort un peu, puis attend que le compteur arrive à 0 puis s'arrête (arrêtant ainsi les processus qu'il a lancés).

1. Donner une implémentation naïve (directe, sans utiliser de mécanismes spécifiques) de ce modèle avec les threads OCaml et les threads Rust et les Threads Java.
2. Identifier les problèmes engendrés par de telles implémentations.
3. Corriger les implémentations.
4. On suppose maintenant que le musée a une capacité limitée, réécrire les implémentations.

1.3 [OCaml, C, Rust] Ordonnanceur Explicite

On souhaite modéliser un système ordonnancé explicitement, composé de:

- plusieurs processus *travailleurs* qui effectuent une même *tâche* atomique (typiquement, incrémenter un compteur local et écrire un message sur la sortie standard) un nombre aléatoire de fois (entre 1 et 5).
- un processus *ordonnanceur* qui s'occupe de distribuer la priorité aux travailleurs
- Implémenter un tel système (dans un langage au choix) en commençant par un ordonnanceur aléatoire (à la fin de chaque tâche, il sélectionne au hasard un travailleur pour qu'il effectue sa prochaine tâche).
- Changer l'implémentation pour que l'ordonnanceur donne la priorité à chaque travailleur pour exactement une tâche chacun.
- Changer l'implémentation pour inclure un système de priorités explicites.
- Quelles implémentations sont équitables.

2 Coopération

2.1 [C, FT] Belle marquise

Le système est composé de cinq threads qui, en boucle, écrivent chacun un morceau de la phrase *"Belle marquise / vos beaux yeux / me font / mourir / d'amour"* et rendent la main à l'ordonnanceur.

1. Ecrire une implémentation du système avec des threads POSIX et une implémentation avec des fair threads.

2. Décrire les résultats obtenus.

2.2 [FT] Envoi / Attente

On veut modéliser le système suivant:
1. des processus *travailleurs* bouclent un nombre de fois fixé: ils attendent un **évènement**, effectuent une tâche (imprime quelque chose sur la sortie standard), puis passent 3 tours.
2. un *générateur* génère un évènement tous les sept tours.
3. un *compteur* égrène les tours sur la sortie standard.
1. Implémenter le système en utilisant les Fair Threads.
2. Prédire les impressions de la sortie standard lors de l'execution de l'implémentation

2.3 [FT] *** Examen Réparti 1 2017 ***

Nous proposons un système utilisant les fairthreads composé de:
- 10 robots (`ft_thread_t robot[10]`),
- 3 chaînes de montage (`ft_scheduler_t montage[3]`),
- 1 chaîne de repos des robots (`ft_scheduler_t repos`),
- 1 chef (`ft_thread_t chef`),
- et 1 secrétaire (`ft_thread_t secretaire`).

avec les contraintes suivantes :

- dans chaque chaîne de montage, il y a au maximum 2 robots qui travaillent en coopération.

- le nombre de robots dans ces chaînes de montage est géré par un tableau `int nb_robots[3]` initialisé à 0 (`nb_robots[i]` pour `montage[i]`).

- un tableau `int affectation[10]` initialisé à -1 (libre), indique à chaque instant l'affectation de chaque robot à une chaîne de montage. Par exemple : si `affectation[7]` vaut 2, cela indique que le `robot[7]` est affecté à la chaîne `montage[2]`.

- le nombre de robots est ici choisi pour éviter toute gestion de pénurie. Le système fonctionne de la manière suivante :

- Les 10 robots, le chef et le(a) secrétaire sont liés au départ à l'ordonnaceur `ft_scheduler_t` repos.

- Le chef passe vérifier dans chaque chaîne de `montage[i]` si le nombre de robots y est au maximum (c'est-à-dire 2). Si ce n'est pas le cas, il informe le(a) secrétaire pour faire venir dans cette chaîne un robot. Il doit informer une deuxième fois si un deuxième robot est nécessaire.
- Informé par le chef, le(a) secrétaire choisit dans le tableau `affectation[]` un robot i non encore affecté , affecte `affectation[i]` avec le numéro de la chaîne de montage indiqué par le chef et fait le nécessaire pour permettre au robot i de s'installer dans cette chaîne pour travailler.
- Chaque robot i dans la chaîne repos vérifie dans `affectation[i]` s'il a une affectation. Dans ce cas, il doit intégrer la chaîne indiquée pour y travailler.
- Le travail d'un robot i dans une chaîne `montage[j]` consiste à :
- incrémenter `nb_robots[j]`,
- boucler 5 fois en coopérant et en travaillant pendant n secondes (simuler avec un sleep(n) où `n = (rand() * 10 / RAND_MAX) + 1`),
- décrémenter `nb_robots[j]`,
- retourner vers la chaîne repos et remettre `affectation[i]` à -1 pour indiquer qu'il est libre.

1. Ajouter les éléments nécessaires pour répondre aux comportements décrits ci-dessus.
2. Compléter la procédure `_robot()`.
3. Compléter la procédure `_chef()`.
4. Compléter la procédure `_secretaire()`.

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

ft_thread_t     robot[10], chef, secretaire;
ft_scheduler_t montage[3], repos;
int             nb_robots[3], affectation[10];

/********** ?????????? **********/

void _robot (void *arg) {
  /********** ?????????? **********/
}

void _chef (void *arg) {
  /********** ?????????? **********/
}

void _secretaire (void *arg) {
  /********** ?????????? **********/
}


int main (void)
{
  int i;
```

```
  for (i = 0; i < 3; ++i) {
    nb_robots[i] = 0;
    montage[i] = ft_scheduler_create();
    ft_thread_create(montage[i], traceinstants, NULL, (void *)50);
  }

  repos = ft_scheduler_create();
  ft_thread_create(repos, traceinstants, NULL, (void *)50);
  for (i = 0; i < 10; ++i) {
    robot[i] = ft_thread_create(repos, _robot, NULL, (void *)i);
    affectation[i] = -1;
  }
  chef       = ft_thread_create(repos, _chef, NULL, (void *)0);
  secretaire = ft_thread_create(repos, _secretaire, NULL, (void *)0);

  /********** ?????????? **********/

  for (i = 0; i < 3; ++i) {
    ft_scheduler_start(montage[i]);
  }

  ft_scheduler_start(repos);

  ft_exit();

  return 0;
}
```