UNIVERSITÉ CAEN NORMANDIE

TD/TP : Thread

25 novembre 2018

1 Introduction

Dans ce TD/TP nous allons (ou nous aurions dû ?)
— Apprendre à créer, travailler avec et arrêter des threads.
— Gérer les signaux
— Savoir reconnaitre les données partagées entre différents threads.
— Être capable d'orchestrer la synchronisation de threads au moyen des primitives de terminaison ou de sémaphores d'exclusion mutuelle.

2 Exercice 2 : Les signaux

1. **Expliquez** rapidement la différence entre les 2 codes ***signal-thread-1.c*** et ***signal-thread-2.c*** que vous pouvez récupérer sur ecampus
2. **Question** : un thread doit-il gérer un signal ?
3. **Concluez**.

3 Exercice 4 : Synchronisation et données partagées

Dans cet exercice vous n'allez pas coder... c'est bien trop long... Vous allez juste compléter le code déjà donné sur ecampus, le fichier **Rappels**.
Les threads disposent des mêmes outils de synchronisation que les processus.
— Le premier d'entre eux est l'attente passive de la fin d'une autre activité qui rappelle les primitives ***wait***() ou ***waitpid***() liées aux processus.
La primitive permettant ce comportement est la suivante :

```
#include <pthread.h>
int pthread_join(pthread_t tid, void **status);
```

— Le second est le sémaphore d'exclusion mutuelle.
On manipule les sémaphores pour les threads à l'aide des 4 primitives fondamentales suivantes :

```
#include <pthread.h>
int pthread_mutex_init(pthread_mutex_t *p_mutex, NULL);
int pthread_mutex_lock(pthread_mutex_t *p_mutex);
int pthread_mutex_unlock(pthread_mutex_t *p_mutex);
int pthread_mutex_destroy(pthread_mutex_t *p_mutex);
```

dont les définitions sont :
— **pthread_mutex_init(p_mutex,NULL)** réalise l'initialisation d'un sémaphore d'exclusion mutuelle de type pthread_mutex_t dont on aura passé l'adresse en premier argument.
Le second argument indique les attributs du sémaphore, on indiquera NULL pour avoir les attributs par défaut qui initialisent le sémaphore avec un droit (le sémaphore n'est pas bloqué).
— **pthread_mutex_lock(p_mutex)** réalise l'opération P (demande d'un droit) sur le sémaphore dont l'adresse est passée en argument.
— **pthread_mutex_unlock(p_mutex)** réalise l'opération V (restitution d'un droit) sur le sémaphore dont l'adresse est passée en argument.
— **pthread_mutex_destroy(p_mutex)** rend le sémaphore dont l'adresse est passée en argument inutilisable à moins d'un nouvel appel à pthread_mutex_init(p_mutex, NULL).

1. **Récupérez** le codes sur ecampus dans le répertoire thread :
2. **Ouvrez** le fichier C et **analysez** le.
3. **Compilez** et **lancez** et **expliquez, tout semble aller pour le mieux dans le meilleur des mondes ?**
Allez ne soyons pas si candides ☺
4. **Modifiez** le code en dé-commentant la ligne :

```
//for (j=0; j<0xFFFFFFF; j++); // un peu de travail
```

5. **Compilez, lancez** et **expliquez** !

6. **Modifiez** le code de manière à ajouter le mécanisme des ***mutex*** en vous inspirant du code de l'exemple du cours. Si vous n'y arrivez pas, alors prenez le code sur ecampus
7. **Compilez** et **lancez** et **expliquez**
8. **Concluez** !

4 Concluez