INFO-F201 – Systèmes d'exploitation
Séance 11
Programmation de *threads*

Naïm Qachri et Stéphane Fernandes Medeiros

Année académique 2011–2012

1 Rappels

Avant de commencer la séance, je vous rappelle que vous devez inclure `pthread.h` dans votre code source. De plus, vous devrez rajouter à la compilation l'argument suivant :

```
gcc code.c -pthread -o test
```

2 Exercices

2.1 Les *thread*s

exercice 1 : Ecrivez un programme qui crée un *thread* simple. Le code principale devra afficher 100 fois la valeur "1" et le code appelé dans le*thread* devra afficher 100 fois la valeur "2". Que constatez-vous lorsque vous exécutez le programme plusieurs fois d'affilé ?

exercice 2 : Vous allez ajouter dans votre programme le passage d'un paramètre à la fonction utilisée pour le *thread* (par exemple un string). Ce paramètre devra être affiché avec `printf` dans la fonction du *thread*.

exercice 3 : Si vous voulez passer plusieurs arguments à un *thread*, vous devez utiliser une `struct` défini spécifiquement pour votre programme (et dans sa zone globale). Vous devez, dans cet exercice, modifier le programme précédent pour lui passer une `struct` en paramètre composée d'un entier et d'un string. Dans le code de la fonction du *thread*, affichez les variables de la `struct` passée en paramètre.

exercice 4 : Maintenant créez plusieurs *thread*s (5 pour fixer les idées) au sein du programme de base. Vous forcerez le programme principale à attendre la fin des 5 *threads* avant d'afficher le message `"fin du programme"`.

2.2 Les mutex

exercice 5 : Vous allez écrire un code qui lancera trois *threads*. Le programme possédera une variable globale `resultat` initialisée à 4 et qui sera manipulée en parallèle par les trois *threads*. Le premier devra multiplier `resultat` par 2 et itérativement 26 fois. Le second *thread* devra sommer 5 à `resultat` 170 fois (toujours de manière itérative). Le troisième soustraira 168 fois la valeur 3 à `resultat`. Vous devrez afficher les résultats intermédiaires à chaque itération dans le code des *threads*. Que constatez-vous lorsque vous exécutez le programme plusieurs fois ?

exercice 6 : Ajoutez maintenant un `mutex` afin de verrouiller chaque série d'itération afin d'afficher un résultat plus cohérent et stable durant les itérations. Pourquoi ce genre de solutions ne suffisent-elles pas toujours ?

3 Remarques finales

Vous avez vu comment créer, attendre, ou sortir d'un *thread*, et comment lui passer des paramètres divers. Vous avez appris que les mutex permettent d'exécuter des zones critiques de code en se synchronisant sur l'accès au variables.

Pour aller plus loin, il est intéressant de savoir qu'un *thread* peut récupérer son numéro d'identifiant avec l'appel système `pthread_self()`. Il est également possible de contrôler plus finement les *threads* avec des variables conditions. Ces variables permettent ainsi de se synchroniser sur les valeurs des variables plutôt que leur accès. En effet, si certaines de vos variables ou zones d'exécution sont soumises à un *lock* au moment de certains tests de conditions, vous pouvez demander à attendre jusqu'à ce que les *thread* utilisant la variable (qui pourrait potentiellement changer) effectue une mise à jour et tout cela sans bloquer tout le système. Les variables conditions sont utilisées conjointement avec les mutex et leur mécanisme peut s'apparenter à l'envoi de signaux entre les processus.

Les `pthdread_cond` sont donc un bon point de départ pour en apprendre plus sur les mécanismes les plus fins de la gestion de synchronisation de *thread*.

Un autre exercice intéressant est de créer et de lancer un *thread* et de demander aux deux traitements parallèles le PID de leur processus et que vous compariez leur affichage.