



Université Toulouse III – Paul Sabatier
118 route de Narbonne
31062 Toulouse cedex 9

Travaux pratiques – n°1

Threads Posix et verrous d'exclusion mutuelle

Documentation

En TP, nous utiliserons le concept de « moniteur » afin de synchroniser des threads Posix (type `pthread_t`) en utilisant les conditions (type `pthread_cond_t`) et les verrous d'exclusion mutuelle (type `pthread_mutex_t`) proposés par Posix.

Un document présentant ces concepts est à votre disposition sous Moodle, si vous ne maîtrisez pas ces concepts, récupérez-le et lisez-le. Vous pouvez aussi consulter les cours Système de L2 (Système 2 du S4 – concept de processus Unix) et de L3 (Programmation Système du S5 – concept de thread Posix) en vous inscrivant aux espaces de cours correspondant sous Moodle.

Lors de cette séance, nous n'utiliserons que les verrous d'exclusion mutuelle pour synchroniser les exécutions de threads.

Exercice 1 – Application « multi-threadée » – Accès à une ressource partagée

Sous Moodle, récupérez le fichier `thd_afficher.c` qui permet d'exécuter en parallèle `nbThreads` (paramètre de l'application) threads chargés d'afficher des informations à l'écran. Chaque thread affiche `nbMessages` messages constitués de `nbLignes` lignes, en temporisant un peu entre chaque ligne (voir remarque en fin de cet énoncé).

`nbMessages` est une constante. `nbLignes` est choisi de manière aléatoire entre 1 et `NB_LIGNES_MAX`.

1. Objet : passer une information à un thread lors de sa création.

Modifiez le programme pour que chaque thread créé affiche un nombre de messages différent, i.e. `nbMessages` sera choisi et transmis par le thread créateur. Compilez et exécutez le programme obtenu plusieurs fois pour des valeurs de `nbThreads`, `nbMessages` et `nbLignes` différentes. Que constatez-vous ?

2. Objet : synchroniser l'accès à une ressource partagée. **[Code à déposer sous Moodle]**

Repartir de la version précédente. En utilisant les verrous d'exclusion mutuelle fournis par Posix, que pouvez-vous proposer pour que les affichages se fassent de façon à conserver l'intégrité de chaque message ?

Exemple avec 3 threads affichant respectivement 2, 2 et 3 messages de 3, 2 et 4 lignes chacun, on doit pouvoir obtenir des messages « complets » i.e. dans lesquels toutes les lignes se suivent.

Thread 30022008, Message 0 Ligne 0

Thread 30022008, Message 0 Ligne 1

Thread 30022007, Message 0 Ligne 0

Thread 30022007, Message 0 Ligne 1

Thread 30022007, Message 0 Ligne 2

Thread 30022009, Message 0 Ligne 0

Thread 30022009, Message 0 Ligne 1

Thread 30022009, Message 0 Ligne 2

Thread 30022009, Message 0 Ligne 3

Thread 30022007, Message 1 Ligne 0

Thread 30022007, Message 1 Ligne 1

Thread 30022007, Message 1 Ligne 2

Thread 30022008, Message 1 Ligne 0

Thread 30022008, Message 1 Ligne 1

Thread 30022009, Message 1 Ligne 0

Thread 30022009, Message 1 Ligne 1

Thread 30022009, Message 1 Ligne 2

Thread 30022009, Message 1 Ligne 3

Thread 30022009, Message 2 Ligne 0

Thread 30022009, Message 2 Ligne 1

Thread 30022009, Message 2 Ligne 2

Thread 30022009, Message 2 Ligne 3

3. Objet : synchroniser l'accès à une ressource partagée.

En fixant nbThreads à 2, que proposez-vous pour que les affichages de messages complets se fassent à tour de rôle (en alternance) ?

Exemple avec 2 threads affichant N messages de 3 lignes chacun

Thread 30022008, Message 0 Ligne 0

Thread 30022008, Message 0 Ligne 1

Thread 30022008, Message 0 Ligne 2

Thread 30022007, Message 0 Ligne 0

Thread 30022007, Message 0 Ligne 1

Thread 30022007, Message 0 Ligne 2

Thread 30022008, Message 1 Ligne 0

Thread 30022008, Message 1 Ligne 1

Thread 30022008, Message 1 Ligne 2

Thread 30022007, Message 2 Ligne 0

Thread 30022007, Message 2 Ligne 1

Thread 30022007, Message 2 Ligne 2

..... (même principe pour les N-2 messages restant qui seront alternés)

Pour répondre à ces deux dernières questions, vous pouvez vous inspirer des exercices sur les sémaphores que vous avez vus en L3 ou sur la modélisation par réseaux de Petri effectuée lors du TD1 (exercice 4).

4. Comment pourrait-on généraliser la solution précédente à N threads ? *[Code à déposer sous Moodle]*

Exercice 2 – Application « multi-threadée » – Accès à des variables partagées

1. Objet : partage d'une variable entre threads. *[Code à déposer sous Moodle]*

Modifiez l'application précédente pour que deux types de threads partagent une valeur entière. Le premier type de threads modifiera la valeur par des ajouts et des retraits. Le second type de threads affichera sa valeur. Que constatez-vous ? (si vous ne voyez rien d'anormal, augmentez le nombre de threads et/ou temporez l'exécution des threads – voir ci-dessous)

2. Objet : partage de plusieurs variables entre threads. *[Code à déposer sous Moodle]*

Étendre le partage à un tableau d'entiers. Le premier type de threads modifiera la valeur des cases du tableau. Le second type de threads affichera le contenu du tableau. Que constatez-vous ? (si vous ne voyez rien d'anormal, temporez les exécutions – voir ci-dessous)

Temporiser les exécutions

Si les affichages sont trop rapides – ou si les threads ne rendent pas l'unité centrale assez souvent pour que leurs instructions d'entrelacent bien – il est possible de temporiser l'exécution d'un thread pendant quelques microsecondes ou nanosecondes à l'aide des primitives :

```
int usleep (useconds_t usec) ;
```

```
int nanosleep(const struct timespec *req, struct timespec *rem) ;
```

Voir le manuel en ligne pour leur utilisation (man 3 usleep ou man 2 nanosleep).

On peut utiliser une valeur générée aléatoirement (voir les fonctions srand et rand) pour varier les délais d'attente d'un thread à un autre.

Mais, **attention**, toute exécution de l'application doit donner un résultat cohérent **sans** temporisation !