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, récupérez-le.

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 nbFois (une constante) un message constitué de nbLignes (choisi de manière aléatoire entre 0 et nbFois) lignes, en temporisant un peu entre chaque ligne (voir remarque en fin de cet énoncé).

1. Modifiez le programme pour que chaque thread créé affiche son message un nombre de fois différent, i.e. nbFois sera choisi et transmis par le thread qui créateur. Compilez-le et exécutez le programme obtenu plusieurs fois pour des valeurs de nbThreads, nbFois et nbLignes différentes. Que constatez-vous ?
2. En utilisant les verrous d'exclusion mutuelle fournis par Posix, que pouvez-vous proposer pour que les affichages se fassent correctement ?
3. En fixant nbThreads à 2, que proposez-vous pour que les affichages se fassent à tour de rôle (en alternance) ?

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 ?

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

1. 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, temporisez l'exécution des threads, voir ci-dessous)
2. 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, temporisez 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 !