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

Travaux pratiques – n°2
Threads Posix et synchronisation de type « moniteur »

Documentation

Le concept de « moniteur » peut être utilisé pour synchroniser des threads Posix en utilisant des conditions Posix (type pthread_cond_t). Voir la documentation à votre disposition sous Moodle.

Exercice 1 – Lecteurs-Rédacteurs

On considère des activités parallèles (threads) qui simulent des lecteurs et des rédacteurs ayant accès à un fichier commun en lecture ou en écriture. Les lectures peuvent se faire en parallèle mais les écritures ne peuvent se faire qu'en exclusion mutuelle.

Les comportements des threads sont les suivants :

Un lecteur	**Un rédacteur**

Boucler sur {
...
demanderAccesEnLecture() ;
Lire le fichier partagé;
libererAccesEnLecture();
...
}
	Boucler sur {
	...
	demanderAccesEnEcriture() ;
	Modifier le fichier partagé;
	libererAccesEnEcriture();
	...
	}

En assurant une synchronisation de type moniteur, écrire les opérations demanderAcces* et libererAcces* de manière à ce qu'un rédacteur donne la priorité à un autre rédacteur et soit prioritaire sur les demandes de lectures en attente (V2 du TD).

Tester ces opérations en écrivant une application dans laquelle L lecteurs et R rédacteurs coexistent (L et R peuvent être paramètres de l'application).

Remarque : Un squelette de code est fourni dans le fichier tp2_lectred_base.c

[Code à déposer sous Moodle]

Exercice 2 – Gestion d'une voie unique

On considère des activités parallèles (threads) qui simulent le comportement de véhicules circulant dans un certain sens. On considère aussi une portion de voie unique sur laquelle, afin d'éviter des collisions, ne peuvent circuler que des véhicules allant dans le même sens.

Le comportement des véhicules est le suivant :

```
Vehicule (monSens)
Boucle nbFois sur {
        Rouler normalement sur la voie à double sens, dans monSens
        demanderAccesVU (monSens)
        Rouler sur la voie unique dans monSens
        libererAccesVU()
}
```

Écrire les opérations demanderAccesVU et libererAccesVU pour synchroniser les accès à la voie unique de façon à permettre à un nombre illimité de véhicules de circuler sur la voie unique à condition qu'ils aillent dans le même sens

Tester ces opérations en écrivant une application dans laquelle N1 véhicules allant dans un sens et N2 véhicules allant dans l'autre coexistent (N1 et N2 peuvent être paramètres de l'application).

Exemples de tests possibles :

N1 = 2, N2 = 5, nbFois = 2

N1 = 0, N2 = 3, nbFois = 2

N1 = 4, N2 = 0, nbFois = 3

Remarque : Un squelette de code est fourni dans le fichier tp2_vu_base.c

[Code à déposer sous Moodle]

Rappel : Si les affichages sont trop rapides, 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**, la temporisation n'est pas là pour résoudre les problèmes d'accès concurrents à des variables partagée. En d'autres termes : toute exécution d'une application parallèle doit donner un résultat cohérent **sans** temporisation !