ESIEE AMIENS	TRAVAUX PRATIQUES N°3 Matière : Programmation multitâches Remis par : S.Pomportes / Y. Guerroudj J. Olive / C. Sarroche	Année : I4 Temps : 4h

OBJECTIFS

Acquérir les bases en programmation multi-threadée et dans l'utilisation des sémaphores.
Savoir mettre en pratique une partie des paradigmes de la concurrence (exclusion mutuelle, passage de témoin, synchronisation).

REMARQUES

Vous placerez l'ensemble de vos fichiers sources dans le répertoire PMTP3 de votre répertoire personnel (commande mkdir $HOME/PMTP3).
Vous rédigerez un compte rendu au format openoffice ou txt.
Chaque fichier (compte rendu ou code) doit comporter vos noms au début.

Threads et sémaphores

I. Implémentation de threads

Le but de cet exercice est d'écrire un programme créant 3 nouveaux threads. Le processus « père » doit afficher son PID, créer les trois threads puis attendre leurs morts. Les threads créés doivent afficher leurs PID à leur tour, puis attendre chacun 10 secondes avant de mourir en envoyant un code de retour à leur père. Le père devra finalement afficher ces codes de retour avant de mourir.
Astuce : Les fonctions pthread_exit et pthread_join fonctionnent avec des pointeurs, qui ne sont autres que des variables codées sur un certain nombre d'octets. Plutôt que de passer des « vrais » pointeurs, vous pouvez tricher en passant directement des valeurs en les transtypant en (void*).
Sur combien d'octets est codé un pointeur de type (void *) ?
Si on utilise la technique du transtypage, combien de valeurs de retour différentes peuvent être transmises à la mort d'un thread ?

Pendant l'attente de 10 secondes, effectuez des « ps -L H », observez et commentez.

NB : Pour compiler le programme avec les threads, il faut ajouter l'option *-lpthread* à la ligne de commande permettant de compiler. A quoi sert cette option ?

Remarque : Sur un noyau 2.4.X, la gestion des threads n'est pas la même. Notamment, il y a un thread_manager, et les PID des threads d'un même processus sont différents.

II. Section critique

Remarque générale :
Pour initialiser et contrôler les sémaphores, nous utiliserons les fonctions *sem_init*, *sem_wait* et *sem_post* (voir le man).

Le processus « père » doit afficher son PID, créer les trois threads puis attendre leurs morts.
Le père initialisera deux variables globales int a=1 et int b=1.

Dans un premier temps, nous n'utiliserons pas les sémaphores.

La thread1 réalisera cet algorithme :
- Attendre entre 0 et 1 seconde (temps aléatoire)
- Ajouter 1 à a,
- Attendre entre 0 et 1 seconde (temps aléatoire)
- Ajouter 1 à b,
- Attendre entre 0 et 1 seconde (temps aléatoire)
- Afficher le résultat sous la forme : « Thread1 : a / b »

La thread2 réalisera cet algorithme :
- Attendre entre 0 et 1 seconde (temps aléatoire)
- Multiplier par 2 la variable a,
- Attendre entre 0 et 1 seconde (temps aléatoire)
- Multiplier par 2 la variable b,
- Attendre entre 0 et 1 seconde (temps aléatoire)
- Afficher le résultat sous la forme : « Thread2 : a / b »

La thread3 réalisera cet algorithme :
- Attendre entre 0 et 1 seconde (temps aléatoire)
- soustraire 1 à a,
- Attendre entre 0 et 1 seconde (temps aléatoire)
- soustraire 1 à b,
- Attendre entre 0 et 1 seconde (temps aléatoire)
- Afficher le résultat sous la forme : « Thread3 : a / b »

a- Lancer votre programme plusieurs fois d'affilées (n'oubliez pas d'initialisé votre graine pour avoir des résultats aléatoires différents à chaque démarrage du programme). Citez quelques résultats obtenus.
b- Mettre en place un sémaphore d'exclusion mutuelle permettant d'obtenir un résultat où a sera toujours égal à b.

III. Passage de témoin

Reprendre votre programme précédant. Nous voulons maintenant visualiser à l'écran :
- 0 / 0
- 1 / 1
- 2 / 2

Penser à une stratégie permettant d'organiser un passage de témoin entre les threads dans le but d'afficher les résultats, en permanence , dans cet ordre .

III. Synchronisation - Le coiffeur (voir TD)

Dans un salon de coiffure, il y a un coiffeur C, un fauteuil F dans lequel se met le client pour être coiffé et N sièges pour attendre.

- Quand le coiffeur C a fini de coiffer un client, il appelle le client suivant dans la salle d'attente.
- S'il n'y a pas de clients, le coiffeur C somnole dans le fauteuil F.
- Quand un client arrive et que le coiffeur C dort, il le réveille ; C se lève alors. Le client s'assied dans F et se fait coiffer.
- Lorsqu'un client arrive pendant que le coiffeur travaille :
    - Si un des N sièges est libre, il s'assied et attend,
    - Sinon il ressort.

Il s'agit de synchroniser les activités du coiffeur et de ses clients avec des sémaphores.

a- Un processus principal créera une thread Coiffeur et aléatoirement au cours du temps des threads Clients. Les clients devront attendre que le coiffeur ait fini de les coiffer (synchronisation) avant de partir, le temps de coiffure étant bien sur aléatoire.

b- Un deuxième coiffeur vient d'être embauché. Ce nouveau coiffeur est spécialisé dans la coiffure féminine. Les coiffeurs décident de s'organiser de la manière suivante : le *coiffeur1* coiffera en priorité les femmes, le *coiffeur2* les hommes. Si, à un moment, la file d'attente n'est composée que de femmes ou d'hommes, les deux coiffeurs coifferont alors indifféremment les clients. Un processus principal créera deux threads Coiffeur1 et Coiffeur2 et aléatoirement au cours du temps des Threads Clients et Clientes.