**TP: Les THREADS**

**Notes du cours :**

Creation d’un thread

*int pthread\_create(pthread\_t \* thread, pthread\_attr\_t \* attr,*

*void \*(\*start\_routine) (void \*), void \*arg);*

* La fonction renvoie une valeur de type int : 0 si la création a été réussie ou une autre valeur si il y a eu une erreur.
* Le premier argument est un pointeur vers l'identifiant du thread (valeur de type pthread\_t).
* Le second argument désigne les attributs du thread. Vous pouvez choisir de mettre le thread en état joignable (par défaut) ou détaché, et choisir sa politique d'ordonnancement (usuelle, temps-réel...). Dans nos exemples, on mettra généralement NULL.
* Le troisième argument est un pointeur vers la fonction à exécuter dans le thread. Cette dernière devra être de la forme void \*fonction(void\* arg) et contiendra le code à exécuter par le thread.
* Enfin, le quatrième et dernier argument est l'argument à passer au thread.

Suppression d’un thread

*void pthread\_exit(void \*ret);*

Elle prend en argument la valeur qui doit être retournée par le thread, et doit être placée en dernière position dans la fonction concernée.

*int pthread\_join(pthread\_t th, void \*\*thread\_return);*

Elle prend en paramètre l'identifiant du thread et son second paramètre, un pointeur, permet de récupérer la valeur retournée par la fonction dans laquelle s'exécute le thread (c'est-à-dire l'argument de pthread\_exit).

Initialisation d’un mutex

Conventionnellement, on initialise un mutex avec la valeur de la constante PTHREAD\_MUTEX\_INITIALIZER.

Verrouillage d’un mutex

*int pthread\_mutex\_lock(pthread\_mutex\_t \*mut);*

Déverrouillage un mutex

*int pthread\_mutex\_unlock(pthread\_mutex\_t \*mut);*

**Exercice 1 :**

Créer une structure ***TypeTableau*** qui contient :

* + Un tableau d’entiers ;
  + Le nombre d’éléments du tableau ;
  + Un entier x.

Écrire un programme qui crée un thread qui initialise un ***TypeTableau*** avec des valeurs aléatoires entre 0 et 99. Le nombre d’éléments du tableau est passé en paramètre.

Dans le même temps, le thread principal lit un entiers x au clavier. Lorsque le tableau est fini de générer, le programme crée un thread qui renvoie **1** si l’élément x est dans le tableau, et **0** sinon.

**Exercice 2 :**

Écrire un programme, avec un compteur global ***compt***, qui crée deux threads :

* Le premier thread itère l’opération suivante : on incrémente le compteur et on attend un temps aléatoire entre 1 et 5 secondes.
* Le deuxième thread affiche la valeur du compteur toutes les deux secondes. Les accès au compteur seront bien sûr protégés par un ***mutex***.
* Les deux threads se terminent lorsque le compteur atteint une valeur limite passée en argument (en ligne de commande) au programme.

**Exercice 3 :**

Ecrire un programme qui crée un thread appelé « ***Producteur*** » qui remplit un tableau de 10 entiers. Chaque case contient un nombre aléatoire que le producteur génère chaque 100 ms et l’affiche sur la console. Lorsqu’il arrive à la fin du tableau, il recommence à partir de la première case.

Le programme crée aussi 3 autres threads appelés « ***Consommateur*** ». Ils ont pour mission d’afficher le contenu de chaque case du même tableau chaque 200 ms. Il faut prévoir les mécanismes de synchronisation nécessaires entre le producteur et les consommateurs.

**Exercice 4 :**

Des processus producteurs produisent des objets et les insère un par un dans un ***tampon*** de **n** places. Bien entendu des processus consommateurs retirent, de temps en temps les objets (un par un).

Résolvez le problème pour qu’aucun objet ne soit ni perdu ni consommé plusieurs fois.

Écrire un programme avec ***N*** threads producteurs et ***M*** threads consommateurs, les nombres N et M étant saisi au clavier. Les producteurs et les consommateurs attendent un temps aléatoire entre 1 et 3 secondes entre deux produits. Les produits sont des octets que l’on stocke dans un tableau de 10 octets avec gestion LIFO. S’il n’y a plus de place, les producteurs restent bloqués en attendant que des places se libèrent.