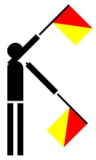
TP 6 : Sémaphores, Threads



Exercice 1. Threads

1. Ecrire un programme **n_threads.c** qui crée **N** threads. Chaque fonction de thread reçoit en argument un numéro : le premier thread reçoit le numéro 0, le deuxième 1…Chaque thread ajoute la valeur de son argument à une variable globale **s**, initialisée à 0. Le thread principal attend la terminaison de tous les threads fils et affiche la valeur de s.

Que constatez-vous ?

Includes	Fonctions
`#include <pthread.h>`	`int pthread_create(pthread_t *thread, const pthread_attr_t *attr,void *(*start_routine)(void *), void *arg);`

Ne pas oublier de compiler/lier avec **-pthread**.

2. Pour éviter le problème rencontré précedemment, utilisez un **mutex** : fonctions `pthread_mutex_lock` et `pthread_mutex_unlock`.

Includes	Fonctions
`#include <pthread.h>`	`int pthread_mutex_lock(pthread_mutex_t *mutex);`
`#include <pthread.h>`	`int pthread_mutex_unlock(pthread_mutex_t *mutex);`

Rappel : la somme devrait être N(N-1)/2.

3. Une autre façon d'éviter le problème est de récupérer le retour des fonctions de thread dans `pthread_join`, et cette fois il n'y a pas besoin de mutex. Ecrivez le code correspondant.

Includes	Fonctions
`#include <pthread.h>`	`int pthread_join(pthread_t thread, void **retval);`

Exercice 2. Reprise de l'exercice de l'exercice 4 du TP n°5

Ecrire un programme **threads_100.c** qui crée un thread fils. Le thread principal affichera les entiers pairs compris entre 1 et 100, le fils affichera les entiers impairs compris dans le même intervalle. Synchroniser les processus à l'aide de sémaphores anonymes pour que l'affichage soit 1 2 3 … 100.

Cette solution est-elle sure ?

Includes	Fonctions
`#include <semaphore.h>`	`int sem_init(sem_t *sem, int pshared, unsigned int value);`
`#include <semaphore.h>`	`int sem_wait(sem_t *sem);`
`#include <semaphore.h>`	`int sem_post(sem_t *sem);`

Ne pas oublier de compiler/lier avec **-pthread**.

Exercice 4. Pi

Sur le modèle de la dernière approche de l'exercice 1 (la 3)), calculez une approximation de π à partir de la formule suivante :

$$\frac{\pi^2}{6} \simeq \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + ...$$

Chaque terme sera calculé dans un thread séparé, le programme s'appellera **threads_pi.c**. Quel problème rencontrez-vous dans le passage de paramètre de la fonction threadée ?

Pour le résoudre, il vous faudra probablement utiliser **malloc** et **free**.

Includes	Fonctions

```c
#include <semaphore.h>
```

```c
void pthread_exit(void *retval);
```

Ne pas oublier de compiler/lier avec *-pthread* **-lm**.

Exercice 5. Dentiste

Vous allez écrire une série de programmes en C qui simulent le fonctionnement d'un cabinet de dentiste avec sa salle d'attente. Le dentiste est simulé par un **thread** et il soigne un patient à la fois. Chaque patient sera simulé par un thread (il y a donc autant de **threads** créés que de patients arrivant au cabinet dentaire).

Vous utiliserez comme base le squelette donné sur Moodle.

1. Programme dentiste0

Dans le main de ce programme, vous allez créer un thread pour le dentiste puis, toujours dans le main, dans une boucle « infinie » attendre l'appui sur une touche, chaque appui déclenche la création d'un patient arrivant au cabinet dentaire. C'est-à-dire que vous allez créer un thread simulant ce patient (voir **squelette_dentiste.c**)

Le dentiste ne traitant qu'un patient à la fois il sera associé à un sémaphore binaire « dentiste » : si la valeur est 1 le dentiste s'occupe d'un patient, 0 sinon.

Les patients sont associés à un sémaphore qui servira à la synchronisation avec le dentiste.

Le fonctionnement du *thread dentiste* est le suivant :

- Affichage d'un message "Ouverture du cabinet."
- Dans une boucle infinie :
  - attente d'un patient : opération P() sur le sémaphore patients
  - réception du patient : opération V() sur le sémaphore dentiste
  - affichage du message "Dentiste : je reçois un nouveau patient"
  - le dentiste soigne le patient. Ce sera simulé par un sleep de 2 secondes.
  - affichage du message de fin du soin : "Dentiste : Au suivant !"

Le fonctionnement d'*un thread patient* est le suivant :

- informe le dentiste qu'il arrive : opération V() sur le sémaphore patients

- attente du dentiste : opération P() sur le sémaphore dentiste
- Affichage d'un message d'arrivée : "Patient : Je rentre dans la piece et me fait soigner"

Réfléchissez aux valeurs initiales des sémaphores.

Exemple d'exécution avec 2 appuis sur la touche entrée :

```
$ ./dentiste0

Ouverture du cabinet.

> Arrivé d'un patient.

Dentiste : je reçois un nouveau patient
Patient  : Je rentre dans la piece et me fait soigner.

> Arrivé d'un patient.

Dentiste : Au suivant !
Dentiste : je reçois un nouveau patient
Patient  : Je rentre dans la piece et me fait soigner.
Dentiste : Au suivant !
```