Projet NSY103

1ère étape :

Vous allez développer une application bi-thread **projet.c** :

- Le processus de l'application va créer un second thread (il deviendra alors lui-même le thread principal)
- Le thread principal va afficher : **<num thread> : Veuillez saisir un message**
- Le thread principal va saisir le message et le transmettre au second thread en utilisant une variable commune **char message[100].**
- Le second thread va afficher le message reçu : **<num thread> : <message>**

- Pour la saisie du message vous n'allez pas utiliser **scanf("%s",message)** car le message sera tronqué au premier blanc. Vous allez utiliser **fgets (message,100,stdin)** :

2ème étape :

- Le thread principal va reboucler sur la saisie du message et le second thread va reboucler sur l'affichage du message.

- Pour la synchronisation des 2 threads, c'est-à-dire pour éviter que le second thread ne lise un message incomplet ou que le thread principal ne remette un message avant que le précédent n'ait été lu nous utiliserons une variable commune **int flag** :
  1) Avant d'afficher "Veuillez saisir …" le thread principal attend que le **flag** soit à **0** et après retransmission il le positionne à **1**.
  2) Avant de lire et d'afficher le message reçu, le second thread attend que le **flag** soit à **1** et après affichage il le positionne à **0**.

3ème étape :

La synchronisation triviale des 2 threads proposée à l'étape précédente fonctionne bien, cependant à titre d'exercice de synchronisation, vous allez abandonner la variable commune **flag** et raisonner comme dans le cas du problème "producteur-consommateur" (cf. support de cours "Communication et Synchronisation"), vous allez donc développer une seconde application **projet2.c** et réécrire la partie "Concurrence d'accès à la variable **message**" :

1) Créer et initialiser 2 mutex : **droit_ecriture** et **droit_lecture.**

```
pthread_mutex_t droit_ecriture;
pthread_mutex_t droit_lecture;
pthread_mutex_init (&droit_ecriture, NULL);
pthread_mutex_init (&droit_lecture, NULL);
```

2) Libérer le **droit_ecriture** cela signifiera que le thread principal peut écrire.

```
pthread_mutex_unlock (&droit_ecriture);
```

3) Prendre le **droit_lecture,** cela signifiera que le second thread doit attendre le droit de lire:

```
pthread_mutex_lock (&droit_lecture);
```

4) Pour écrire un message le thread principal va :

- Prendre **droit_ecriture** (sinon en attente) :

```
pthread_mutex_lock (&droit_ecriture)
```

- Saisir le message dans **`message`**
- Libérer **`droit_lecture`** (permettre au second thread de lire) :
  `pthread_mutex_unlock (&droit_lecture)`

5) Pour lire un message le second thread va :
- Prendre **`droit_lecture`** (sinon en attente) :
  `pthread_mutex_lock (&droit_lecture)`
- Lire le message dans **`message`**
- Libérer **`droit_ecriture`** (permettre au thread principal d'écrire) :
  `pthread_mutex_unlock (&droit_ecriture)`

➢ Si vous avez des problèmes de compréhension de l'énoncé du projet n'hésitez pas à poser vos questions sur le Forum.