

## Système - TP3

### Exercice 1

Implémentez une file d'attente de type LIFO où FIFO de taille fixe. Vous devez écrire trois fonctions, une d'initialisation, une pour ajouter des éléments et une pour en enlever.

Lorsqu'un thread tente d'ajouter un élément à la file alors qu'elle est pleine, ou d'enlever un élément alors qu'elle est vide, la fonction concernée doit attendre une seconde et réessayer l'opération jusqu'à ce que celle-ci réussisse.

N'oubliez pas de créer deux threads simples pour tester vos fonctions.

### Les conditions

Les variables conditionnelles sont une autre forme de synchronisation qui permet de ce placer en attente d'une ressource. Lorsqu'un thread obtiens le verrouillage d'un mutex, la ressource obtenue n'est pas forcément dans le bon état, les conditions permettent de libérer le mutex et d'attendre que la ressource soit prête.

Par exemple, une variable peut-être protégée par un mutex ; elle va être affectée par un thread qui lira sa valeur dans un fichier et sera utilisée par un autre thread. Puisque les deux threads vont y accéder, il est nécessaire de la protéger par un mutex. Mais lorsque le deuxième thread obtiens le mutex, le premier peut ne pas avoir encore placé la valeur lue dans la variable, le deuxième thread doit alors libérer le mutex et attendre avant de recommencer.

Les conditions permettent de faire cela efficacement. Au lieu de libérer le mutex directement, le thread va ce mettre en attente de la condition, cela à pour effet de libérer le mutex et de placer le thread en sommeil. Lorsque le deuxième thread va modifier la variable, il va d'abord obtenir le mutex, puis avant de libérer le mutex, il va envoyer un signal pour réveiller le premier.

```
// Initialisation :
pthread_mutex_t mutex;
pthread_cond_t cond;

int init = 0;
int var = 0;

pthread_mutex_init(&mutex, NULL);
pthread_cond_init(&cond, NULL);
```

```

// Code du thread lecteur :
int tmp = lireValeur();
pthread_mutex_lock(&mutex);
init = 1;
var = tmp;
pthread_cond_signal(&cond);
pthread_mutex_unlock(&mutex);

// Code du thread consommateur :
pthread_mutex_lock(&mutex);
while (init == 0)
    pthread_cond_wait(&cond, &mutex);
    // Utilisation de la variable
pthread_mutex_unlock(&mutex);

```

Dans le thread consommateur, la fonction `pthread_cond_wait` va déverrouiller le mutex et attendre le signal `cond`. Lorsque celui-ci arrive, le mutex est reverouillé et l'exécution continue.

## Exercice 2

À l'aide de mutex et de conditions, modifiez votre file d'attente afin de ne plus faire d'attente semi-active (la boucle d'attente avec pause de une seconde).

Lorsqu'une opération impossible est demandée, le thread doit se mettre en sommeil jusqu'à ce qu'elle devienne possible.

## Exercice bonus

En utilisant votre file d'attente, modifier la fonction `quickSort` du TP 3 de manière à ce qu'il y ait un ensemble de threads qui prennent les tableaux à trier depuis la file d'attente, et ajoute dedans les sous-tableaux en attente de traitement.

Quels problèmes de synchronisation peuvent apparaître dans cette approche ?