# TD 3 : threads, processus,mémoire partagée

## Exercice 1. Lettres

Vous allez écrire un programme dont le but est de compter le nombre de « minuscules », « majuscules » et « autres signes » d'un texte. Les variables globales qui contiennent le texte (un tableau de char), et le nombre d'éléments de chaque type de lettres (des entiers) sont partagées entre les différents threads que vous allez mettre en œuvre.

Vous utiliserez comme base le squelette **lettres.c** donné sur Moodle.

Votre programme va lancer **trois threads** distincts qui auront comme rôle respectif de compter les minuscules, les majuscules et les autres signes. Une variable entière `position` , elle-aussi globale, permettra de parcourir les caractères de la chaîne. Chaque thread se comportera de manière similaire. Dans le cas des **minuscules**, on aura :

```
si la fin de la chaîne est atteinte alors
    fin du thread
sinon
    si le  caractére courant est une minuscule alors
        incrémenter le nombre de minuscules
        incrémenter position
```

Chaque thread n'incrémente donc `position` que si le caractère courant est de la classe de caractères qu'il traite (ici les minuscules).

**Remarques :**

- pour savoir si un caractère est une minuscule/majuscule pensez à utiliser `islower`/`isupper`.
- Les variables globales pourront/devront être protégées par des **mutex**.
- Options de compilation : **-pthread**

## Exercice 3. Aquarium

L'ascenseur de l'aquarium de La Rochelle ne peut accepter que **n** personnes à la fois. Quand l'ascenseur arrive, il ouvre ses portes et tous les visiteurs présents montent dans l'ascenseur. S'ils sont plus de n, seuls n entrent dans l'ascenseur, les autres attendent le retour de l'ascenseur.

Pour des raisons de sécurité, si de nouveaux visiteurs arrivent quand l'ascenseur a commencé à faire monter des visiteurs, ils devront attendre le prochain tour même s'il reste de la place dans l'ascenseur.

Quand les passagers sont montés dans l'ascenseur, il ferme les portes et démarre puis revient pour faire monter les visiteurs suivants.

S'il n'y a pas de visiteurs, l'ascenseur attend qu'il en arrive pendant une seconde (!) puis recommence son cycle (ouverture des portes…).

Vous allez écrire un programme **ascenseur.c** qui réalise un simulateur de ces comportements en utilisant un thread pour l'ascenseur et un thread pour chacun des visiteurs dont le nombre est variable.

Les actions de l'ascenseur et des visiteurs seront de simples affichages par printf.

Voici un exemple de simulation avec **n=5** et **10 visiteurs** :

```
Ascenseur : J'ouvre les portes
        Visiteur : Je monte dans 'lascenseur
Ascenseur : Je ferme les portes
Ascenseur : départ avec 1 personnes
Ascenseur : J'ouvre les portes
        Visiteur : Je monte dans l'ascenseur
        Visiteur : Je monte dans l'ascenseur
        Visiteur : Je monte dans l'ascenseur
Ascenseur : Je ferme les portes
Ascenseur : départ avec 3 personnes
Ascenseur : J'ouvre les portes
        Visiteur : Je monte dans l'ascenseur
        Visiteur : Je monte dans l'ascenseur
        Visiteur : Je monte dans l'ascenseur
        Visiteur : Je monte dans l'ascenseur
        Visiteur : Je monte dans l'ascenseur
Ascenseur : Je ferme les portes et je démarre
Ascenseur : départ avec 5 personnes
Ascenseur : J'ouvre les portes
        Visiteur : Je monte dans l'ascenseur
Ascenseur : Je ferme les portes
Ascenseur : départ avec 1 personnes
Ascenseur : J'ouvre les portes
Ascenseur : Personne, je ferme les portes
Ascenseur : j'attends
Ascenseur : J'ouvre les portes
Ascenseur : Personne,je ferme les portes
Ascenseur : j'attends
Ascenseur : J'ouvre les portes
Ascenseur : Personne,je ferme les portes
Ascenseur : j'attends
```

Vous utiliserez pour ce programme :

- une variable globale `en_attente` : qui contient le nombre de visiteurs présents prêts à monter. Elle sera incrémentée dans les threads visiteurs (arrivée d'un visiteur) et décrémentée dans le thread ascenseur (montée de visiteurs).

- un **mutex** qui servira dans le thread ascenseur pour garantir que quand la procédure d'embarquement des visiteurs commence, seuls les visiteurs présents vont embarquer. Dans les threads visiteurs, il permettra de protéger la variable enattente.
- un sémaphore **ascenseur initialisé à zéro**, qui dans le thread ascenseur, permet de signaler aux visiteurs qu'un d'entre eux peut monter. Dans les thread visiteur, ce sémaphore servira à attendre le signal pour monter dans l'ascenseur.
- un sémaphore **monte (monté !) initialisé à zéro**, qui dans le thread ascenseur, permet d'attendre qu'un visiteur soit monté. Dans les thread visiteur, ce sémaphore servira à signaler à l'ascenseur qu'un visiteur est monté.

Chaque montée de visiteur dans l'ascenseur utilisera donc ces deux sémaphores dans les deux types de thread.

**Remarque :** pour faire varier les scénarios, le temps d'attente entre deux visiteurs sera donné aléatoirement par `usleep(rand()%500);`