TP 3 : threads et sémaphores

⏱️ 4h
📚 Algorithmie en langage C
📅 *Mis à jour le 23/10/2025*

🎯 Objectifs

- Découvrir la programmation concurrente avec les threads
-

🔧 Matériel nécessaire

- Ordinateur avec compilateur gcc
- CodeBlocks/VS Code

Partie 1 : les sémaphores

Rappels

Un sémaphore est une variable entière qui permet de contrôler l'accès à une ressource partagée par plusieurs threads. Il peut prendre des valeurs positives ou nulles.

Si le sémaphore est supérieur à 0, cela signifie que la ressource est disponible et qu'un thread peut y accéder. Si le sémaphore est égal à 0, cela signifie que la ressource est occupée et qu'un thread doit attendre avant de pouvoir y accéder.

Un sémaphore est généralement manipulé à l'aide de deux opérations atomiques :

- `wait()` (ou `P()`), qui décrémente la valeur du sémaphore. Si la valeur devient négative, le thread est bloqué jusqu'à ce que le sémaphore soit disponible.
- `signal()` (ou `V()`), qui incrémente la valeur du sémaphore. Si la valeur était négative, un thread bloqué est réveillé.

Exercice

Voici deux processus partageant deux ressources (A et B) :

```
// Processus 1
P(A);
V(A);
P(B);
V(B);
```

```
// Processus 2
P(A);
V(A);
P(B);
V(B);
```

Les conditions initiales sont les suivantes :

```
// Initialisation des sémaphores
a = 1;
b = 0;
```

Q1. Montrez que les processus entrent systématiquement en interblocage.

Q2. Proposez une modification des conditions initiales pour éviter l'interblocage.

Q3. Montrez graphiquement que votre solution fonctionne.

Partie 2 : l'étoile mouvante

Nous allons, dans cette partie, programmer un simple jeu d'étoile qui se déplace dans un plateau. Lorsqu'elle rencontre un obstacle, elle rebondit dans le sens opposé. Il est aussi possible de diriger l'étoile avec les touches ZQSD du clavier.

Q1. Récupérez l'archive `jeu.zip` sur Arche et importez les fichiers dans un nouveau projet CodeBlocks.

L'archive contient les fichiers suivants :

- `jeu.c` : le fichier principal du programme
- `jeu.h` : le fichier d'en-tête
- `utilitaires.c` : des fonctions utilitaires

Le fichier `jeu.c` contient la fonction `main()` qui effectue les opérations suivantes :

- lecture du fichier `plateau.txt` qui contient la configuration du plateau de jeu
- placement de l'étoile à une position aléatoire
- lancement de deux threads :
  - un thread `anim` qui gère l'animation de l'étoile
  - un thread `clavier` qui gère les entrées clavier

On stocke le plateau grâce à un tableau d'entiers qui doit respecter ce format :

- 0 : case vide
- 1 : obstacle
- 2 : étoile

> ⚠ **Warning**
>
> Les codes sont utilisables sous linux par défaut mais il suffira de décommenter les lignes spécifiques à Windows si vous êtes sous cet OS. En particulier, la fonction `sleep()` sous linux est remplacée par

> `Sleep()` sous Windows via la bibliothèque `windows.h`.

Q2. Dans le fichier `utilitaires.c`, complétez la fonction `afficher_plateau()` afin d'afficher le plateau de jeu dans la console. Il faudra penser à effacer la console avant chaque affichage.

Q3. Écrivez la fonction `placer_star()` qui place l'étoile à une position aléatoire dans le plateau.

> **Note**
>
> Vous pourrez utiliser vos codes sur la génération de nombres aléatoires vus en TP1.

> **Warning**
>
> Il faudra penser à vérifier que la position choisie n'est pas un obstacle.

Q4. Écrivez la fonction `calculer_direction()` qui met à jour la direction que doit prendre l'étoile (`star->direction`). Pour rappel, l'étoile change de direction lorsqu'elle rencontre un obstacle.

Q5. Écrivez la fonction `deplacer_star()` qui est appelée par le thread `anim`. Après avoir fait appel à la fonction `calculer_direction()`, cette fonction met à jour le plateau puis l'affiche avant de s'endormir pendant une seconde.

Gestion de la vitesse

Q6. Ajoutez un menu au lancement du programme permettant à l'utilisateur de choisir la vitesse de l'étoile.

Vitesse	**Délai**
1 - Lent	1000 ms
2 - Normal	500 ms
3 - Rapide	200 ms

Vous pourrez utiliser la fonction suivante :

```c
// Millisecond sleep
void msleep(int milliseconds) {
    struct timespec ts;

    ts.tv_sec = milliseconds / 1000;
    ts.tv_nsec = (milliseconds % 1000) * 1000000;

    nanosleep(&ts, NULL);
}
```

Thomas Mongaillard
thomas.mongaillard@univ-lorraine.fr