TP 3- INFO 632
modèle PRODUCTEUR-CONSOMMATEUR

Dans ce TP nous allons utiliser des sémaphores pour implanter un modèle producteur/ consommateur similaire à celui que vous avez vu en TD.

Rappel du modèle producteur/consommateur
Le fonctionnement du modèle producteur/consommateur est le suivant :

- Un écrivain en cours de modification interdit tout autre accès à la ressource partagée, que ce soit en lecture ou en écriture,
- Un lecteur en consultation interdit les accès en écriture, mais autorise d'autres consultations simultanées de la ressource partagée.

Sémaphores Unix
Dans le cours, vous avez vu les semaphores. Ces structures de blocage se gèrent avec 3 opérations `P`, `V` et `Init` . Les fonction C proposées par le système UNIX pour effectuer les opération P, V, et Init sont décrites dans le tableau ci-dessous :

Opération	API POSIX C
`Init(sem, compteur)`	`sem_t * sem = sem_open(Nom, O_CREAT, 0644, compteur)` Signification des paramètres: ▪ `Nom` : chaine de caractères correspondant au nom du sémaphore, ce nom doit commencer par `/` ▪ `O_CREAT` : création d un nouveau sémaphore ▪ `0644` : droits d'accès (lecture et écriture)
`P(sem)`	`sem_wait(sem);` Notez ici que le paramètre sem est celui retourné par la fonction sem_open; il est du type sem_t*.
`V(sem)`	`sem_post(sem);` Notez ici que le paramètre sem est celui retourné par la fonction sem_open; il est du type sem_t*.

Dans la suite nous allons essayer de résoudre plusieurs problèmes simples avec les sémaphores.

Question 1- Signalisation

C'est l'utilisation la plus simple des sémaphores. Supposez que deux threads s'exécutent et que la tache **`a();`** dans le thread 1 doit obligatoirement s'executer avant la tache **`b();`** dans le thread 2. Pour ceci les deux threads ont besoin d'un moyen de synchronisation. Supposons que la variable `sync` joue ce rôle

```
init(&sync, 0);
```

Process One

```
a;
signal(&sync);
```

Process Two

```
wait(&sync);
b;
```

Comment faire cela avec une sémaphore ? complétez le code dans `ex1.c` pour faire cela

Question 2- Rendez vous

Generaliser le cas précédent pour que cela marche dans les deux sens. Le thread 1 contient deux opérations `a1()` et `a2()`, et le thread 1 deux opérations `b1()` et `b2()`. Nous voulons nous assurer que `a1()` à lieu avant `b2()` et que `b1()` a lieu avant `a2()`.
Pour ceci nous aurons besoin de deux sémaphores. Indiquer pour la solution suivante ne marchera pas.

Process One

```
a1;
wait(&sem2);
signal(&sem1);
a2;
```

Process Two

```
b1;
wait(&sem1);
signal(&sem2);
b2;
```

Proposez une solution qui marche.

Question 3- exclusion mutuelle

```
init(&mutex, 1);
```

Thread A

```
wait(&mutex);
/* CS */
count = count + 1;
signal(&mutex);
```

Thread B

```
wait(&mutex);
/* CS */
count = count + 1;
signal(&mutex);
```

Dans cette question nous utiliserons les sémaphores pour garantir que seul un thread peux accéder à une mémoire partagée. Par exemple nous supposons que deux threads souhaitent accéder à une variable partagée `count`.
Expliquer pourquoi le pseudo code plus haut garantie l'exclusion mutuelle?

Compléter le code dans ex2.c.

Question 4- Vous avez maintenant les éléments pour implanter tout les exercices du TD. Nous supposerons que `Produire()` est une fonction qui génère un message « Message : *k* » ou *k* est incrémenté à chaque appel de la fonction `Produire()`. La fonction `Consommer()` met le message reçu en majuscule. Dans un premier temps nous supposerons qu'il n'y a qu'un seul producteur et un seul consommateur. Dans ce cas en quoi consiste les fonctions `Deposer()` et `Retirer()` du TD. Ecrivez le code relatif à la premiere partie du TD.

Question 5- Maintenant nous allons faire le multi-producteur, multi-consommateur. Dans ce cas pourquoi avons nous besoin d'une structure de données entre les producteurs et les consommateurs ?
Dans la suite nous utiliserons une file d'attente (queue en anglais) pour jouer le role de structure de données. Voir la description d'une file d'attente dans https://www.geeksforgeeks.org/queue-set-1introduction-and-array-implementation/
Pourquoi une file d'attente est bien indiqué pour le besoin que nous avons ?
Voir le code dans le fichier `queue.c` et bien le comprendre.

Si nous utilisons une file d'attente pour connecter des producteurs et des consommateurs, que doivent faire concrètement les fonctions `Deposer()` et `Retirer()`. Utiliser le code dans queue.c pour implanter un multi-producteur, multi-consommateur avec les fonction Produire et Consommer précédente. Supposez qu'à la fin du traitement de Consommer un délai aléatoire est généré avant d'imprimer le message. Comment pouvez vous assurer que les messages en majuscule après consommation sont imprimé dans le bon ordre.