TP : modèle producteur-consommateur

Dans ce TP on étudie un schéma classique de synchronisation appelé *producteur-consommateur*. Dans ce modèle, un fil d'exécution *producteur* produit des données qui doivent être traitées par un fil d'exécution *consommateur*.

Dans notre cas particulier, des données produites et en atttente de traitement seront sauvegardées dans un *buffer* de taille bornée. Ainsi, le *producteur* doit se mettre en attente s'il n'y a plus de place dans le buffer, tandis que le *consommateur* doit se mettre en attente lorsqu'il n'y a plus de données à consommer.

Ce schéma est utilisé dans les systèmes d'exploitation pour mettre en œuvre les *pipes* : c'est-à-dire les commandes de type `prog1 | prog2` où la sortie de `prog1` est envoyée sur l'entrée de `prog2`.

Pour simplifier, on considèrera que le *buffer* est un tableau de caractères de longueur
`#define N 5`

Le producteur sera appelé *A* et il écrira aléatoirement des *a* et des *b* dans le buffer. Le consommateur appelé *B* afficher un 0 lorsqu'il lit un *a* et un 1 lorsqu'il lit un *b*.

1 Le buffer circulaire

On commence par coder la structure qui stockera les données produites et en attente d'être consommées.

```
struct buffer_s {
    char data[N];
    int debut; // position du debut des donnees
    int taille; // nombre de données actuelles
};
typedef struct buffer_s buffer;
```

On définira également les codes d'erreur suivants :

```
const int ERR_BUFFER_PLEIN = 1;
const int ERR_BUFFER_VIDE = 2;
```

Question 1

Écrire une fonction `void buffer_init(buffer* b)` qui initialise un buffer vide.

Question 2

Écrire une fonction `void buffer_write(buffer* b, char c)` qui écrit un nouveau caractère dans le buffer. Si le buffer est plein le programme termine avec le code d'erreur `ERR_BUFFER_PLEIN`. Le buffer sera *circulaire* c'est-à-dire que lorsqu'on arrive à l'extrémité droite du tableau `data`, on continue à écrire depuis l'extrémité gauche.

Question 3

Écrire une fonction `char buffer_read(buffer* b)` qui lit le prochain caractère disponible dans le buffer. Si le buffer est vide, le programme termine avec le code d'erreur `ERR_BUFFER_VIDE`.

Question 4

Écrire une fonction `void buffer_print(buffer* b)` qui affiche le contenu du buffer à l'écran sur le flux `stderr`.

Question 5

Dans le contexte de la programmation concurrente, on souhaite sécuriser le buffer contre les accès simultanées.

a. Modifier le type buffer pour ajouter une adresse de *mutex*
b. Modifier `buffer_init` pour qu'il initialise le *mutex*
c. Modifier les fonctions d'accès en lecture et en écriture du buffer pour qu'elles utilisent le *mutex*.

Question 6

Tester le programme en écrivant, puis en lisant quelques caractères dans le buffer.

2 Le producteur et le consommateur

Question 7

Écrire une fonction `void attendre(int duree)` qui produit une attente active de `duree` secondes. On pourra utiliser l'appel de fonction[1] `time(NULL)` pour obtenir la date actuelle en secondes.

Question 8

Écrire une fonction `void rever(int duree_max)` qui produit une attente aléatoire entre 0 et `duree_max` secondes. On rappelle qu'une valeur aléatoire entière quelconque peut être obtenue avec la fonction `rand` déclarée dans `stdlib.h`.

La synchronisation de l'ensemble `producteur-consommateur` sera réalisée à l'aide deux sémaphores :

- `libre` : qui comptabilise le nombre de places libres dans le buffer
- `occupe` : qui comptabilise le nombre d'éléments en attente dans le buffer

Question 9

Écrire une fonction `producteur` destinée à être initiée dans un nouveau `thread` et dont le fonctionnement est la boucle suivante :

1. rêver au maximum 3 secondes
2. attendre qu'une place soit disponible dans le buffer
3. écrire aléatoirement un caractère `a` ou `b` dans le buffer
4. afficher le buffer à l'écran (et un message)

[1] déclarée dans `time.h`

Question 10

Écrire une fonction `consommateur` destinée à être initiée dans un nouveau `thread` et dont le fonctionnement est la boucle suivante :

1. rêver au maximum 3 secondes
2. attendre qu'un élément soit disponible dans le buffer
3. lire un caractère depuis le buffer puis : si c'est un `a` afficher `0` à l'écran, sinon afficher `1` à l'écran.

3 La fonction main

Question 11

Écrire la fonction `main` mettant en place tous les éléments du modèle producteur-consommateur. On jouera sur les durées de rêverie des deux processus pour vérifier la bonne synchronisation.

Question 12

Modifier l'ensemble du programme pour avoir un nouveau fil d'exécution dont la mission est d'afficher à l'écran à intervalles réguliers l'état du buffer. Dans ce cas on enlèvera l'affichage du buffer par le producteur.