IG2I
Institut de Génie Informatique et Industriel

LA1 - Année 2016-2017

TP n°3 - Sémaphores

Ce TP aborde la notion de sémaphore pour la résolution des problèmes liés à la concurrence. Après avoir expérimenté les différentes façons de mettre en œuvre les sémaphores, l'exercice de synthèse permettra d'implémenter la solution du problème classique du « déjeuner des philosophes ».

Ce TP fera l'objet d'un compte-rendu contenant :

- les réponses aux diverses questions présentes dans cet énoncé,
- les codes sources commentés des programmes réalisés,
- les jeux d'essais obtenus,
- une conclusion sur le travail réalisé, les difficultés rencontrées, les connaissances acquises ou restant à acquérir, …

Il sera tenu compte de la qualité de vos pratiques de production de code :

- Lisibilité du code : emploi de commentaires, indentation, constantes symboliques
- Automatisation : Fichiers makefile avec macros …
- Robustesse : Vérification des valeurs de retours des appels systèmes

1 Sémaphores (n.m.)

Le système Linux propose un mécanisme permettant de résoudre le problème d'accès à une ressource partagée, basé sur l'utilisation de primitives *atomiques* (i.e. non interruptibles) de manipulation de variables entières : les **sémaphores**.

Il existe deux types de sémaphore, selon la manière dont on souhaite les partager.

1.1 Utilisation de sémaphores non nommés

1. Indiquer quelles sont les manières de partager un sémaphore **non nommé**.
2. Avec quelle librairie faut-il lier une application utilisant des sémaphores ?
3. Utiliser le mécanisme de sémaphore non nommé pour protéger la ressource « écran » du dernier exercice du TP précédent.
4. Modifier cette application de façon à utiliser des processus à la place de threads. Le comportement est-il similaire à celui observé précédemment ?

1.2 Utilisation de threads mutex

La bibliothèque des threads **POSIX**, inclut un mécanisme proche de celui du sémaphore anonyme : le `pthread_mutex_t`**.**

1. Citer au moins deux cas de figure où il n'est pas possible de substituer un sémaphore non nommé par un `pthread_mutex_t`.
2. L'application à base de threads, traitée à la question 1.1.3 se prête-t-elle au remplacement du sémaphore non nommé par un `pthread_mutex_t` ? Si c'est le cas, réaliser et tester cette nouvelle version.

1.3 Utilisation de sémaphores nommés

1. Présenter le concept de **sémaphores nommés**. Dans quel contexte faut-il les utiliser ?
2. Indiquer le répertoire système qui accueille les fichiers implémentant les sémaphores. Est-ce que les permissions de votre sémaphore permettent son utilisation par un processus appartenant à un autre utilisateur ?

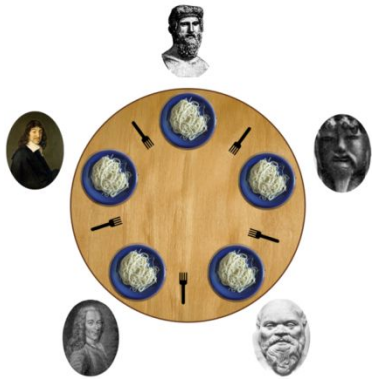
NB : La mise en œuvre de sémaphores nommés n'est pas demandée dans cette section. Elle sera réalisée à l'occasion de la section suivante, pour le cas multi-processing.

2 Le problème du déjeuner des philosophes

2.1 Version multi-threading (statique puis dynamique)

Nous souhaitons implémenter le problème du déjeuner des philosophes vu en cours (cf. diapositives 111-115 du poly de cours 2012 SEE_TR) :

> Un nombre n (>=3) de philosophes sont assis autour d'une table ronde, sur laquelle sont installées n assiettes de spaghettis et n fourchettes.
>
> Chaque philosophe dispose d'une fourchette de chaque côté de son assiette.
>
> Les philosophes commencent tous à penser. Lorsque l'un d'eux se décide à faire une pause pour manger, il cherche à prendre les deux fourchettes qui se trouvent de chaque côté de son assiette. Si un de ses voisins mange, il devra attendre…



Ce problème classique illustre la notion de partage de ressources et les phénomènes de blocage, qui peuvent se produire par exemple si chaque philosophe qui se réveille prend tout de suite l'une des fourchettes disponibles et ne la rend pas avant d'avoir récupéré la deuxième.

1. Modéliser sous forme de réseau de Petri, l'algorithme qui évite que des blocages se produisent.

2. Développer la version statique de ce programme (le nombre total de philosophes `NBPHILOS` est fixé dans le programme) à l'aide de threads et de sémaphores non nommés. Le programme prend fin dès que `NBCYCLES` cycles penser-manger ont été réalisés par chaque philosophe.

3. Modifier le programme précédent de façon à produire une version dynamique, pour laquelle le nombre de philosophes peut être défini au début de l'exécution du programme, par une lecture au clavier ou par un passage de paramètre en ligne de commande.

Indications

> Créer les macro-fonctions `DROITE(i)` et `GAUCHE(i)` chargées de fournir les indices des voisins de gauche et de droite du philosophe d'indice `i`.

Créer un tableau d'états pour les `NBPHILOS` philosophes en variable globale. Créer pour cela un type énuméré nommé `T_Etat` pour définir les états des philosophes : Penser, Attendre et Manger.

Utiliser un tableau de sémaphores privés, un pour chaque philosophe en attente de ses fourchettes. Remarquer que les tests permettant de modifier les sémaphores privés sont uniquement fonction du numéro du philosophe et en faire une fonction générique.

Créer une fonction qui affiche l'état des philosophes sur une SEULE ligne pour comparaison et test de bon fonctionnement. On se contentera d'afficher l'initiale du nom de leur état (P, A ou M), et on ne fera appel à cette fonction que s'il y a changement d'état. Ainsi, on peut vérifier les propriétés suivantes :

- Gestion correcte de la mutex : d'une ligne d'affichage à la suivante, un seul philosophe change d'état ;
- Gestion correcte des sémaphores privés : deux philosophes voisins ne mangent pas en même temps.

Le passage de la version statique du programme à la version dynamique est réalisé rapidement en notant que :

- chaque tableau dont la taille dépendait du nombre de philosophes se transforme en pointeur de …,
- il sera nécessaire, dès que le nombre sera connu, d'allouer les espaces de mémoire nécessaires, par `malloc()` ou `calloc()`.
- Il est souhaitable de récupérer ces espaces de mémoire dès qu'ils deviennent inutilisés, par `free()`.

2.2 Version multi-processing, sémaphores nommés

Il s'agit de développer une version en multi-processing de la version précédente. Pour cela, vous serez amené à remplacer les threads par des processus fils. **POSIX** permet le partage des sémaphores non nommés entre processus « frères » (fils d'un même père) alors que Linux ne respecte cette règle et ne permet ce partage que pour certaines versions récentes du noyau. Par conséquent, vous serez amené à utiliser les sémaphores nommés.

Les threads se partagent naturellement des données en mémoire, mais pas les processus. Vous serez donc amené à utiliser un fichier pour les états de philosophes.

Deux fonctions `lire()` et `ecrire()` seront mises en œuvre. On lit et on écrit le tableau en une seule fois grâce à `fread()` et `fwrite()`.