**UNIVERSITÉ DU QUÉBEC À MONTRÉAL**

**Département d'informatique**

**Cours : INF-3173-20-30 (Principes des systèmes d'exploitation)**

**Travail pratique de session # 2 (20%)**

**Date de remise : 20 décembre 2019 via Moodle**

**Ammar Hamad**

## 1. Objectifs pédagogiques

Durant le cours, nous avons parlé d'un certain nombre de problèmes de synchronisation et contrôle de concurrence. Nous avons vu que, souvent, nous utilisons des sémaphores pour la synchronisation (c'est la manière traditionnelle de présenter de telles solutions). Cependant, les implémentations réelles de ces solutions pourraient utiliser des verrous mutex à la place des sémaphores binaires. Ce travail va vous aider à mettre en pratique la théorie vue en classe et surtout de bien comprendre les fonctionnements de ces genres d'algorithmes. En plus, nous allons inclure dans ce projet l'utilisation de système de gestion fichier ainsi que certains appels systèmes de manipulation de fichier notamment ceux qui concernent le verrouillage de fichier.

## 2. Description d'un problème classique de synchronisation : Le problème des philosophes qui dînent

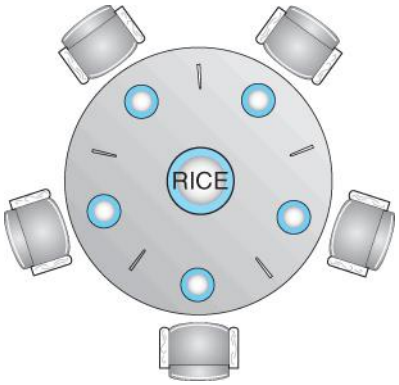


Figure 1.0

Considérez cinq philosophes qui passent leur vie à penser et à manger. Les philosophes partagent une table circulaire entourée de cinq chaises appartenant à un philosophe. Au centre de la table se trouve un bol de riz. La table est posée avec cinq baguettes simples (Figure 1.0). Quand un philosophe pense, il n'interagit pas avec ses collègues. De temps en temps, un philosophe a faim et tente de ramasser les deux baguettes qui lui sont les plus proches (les baguettes qui se trouvent entre lui et ses voisins de gauche et de droite). Un philosophe ne peut prendre qu'une baguette à la fois. De toute évidence, il ne peut pas ramasser une baguette qui est déjà dans la main d'un voisin. Lorsqu'un philosophe affamé a ses deux baguettes en même temps, il mange sans relâcher les baguettes. Quand il a fini de manger, il pose les deux baguettes et recommence à penser.

## 3. Description du projet : Le problème des philosophes qui dînent

Dans le point 2, ci-dessus, nous avons donné un aperçu d'un problème (Les philosophes qui dînent). Ce problème nécessitera la mise en œuvre d'une solution utilisant des verrous mutex et des variables de condition Pthreads.

Nous vous demandons de programmer en langage C cet algorithme. Le résultat devrait nous donner, pour chaque philosophe, 5 fois en train de manger et 5 fois entrain de penser. La première action de chaque philosophe est de manger et après il pense. Bien sûr, les 5 philosophes devraient alterner entre manger et penser.

Le résultat de cet algorithme doit être enregistré dans un fichier « résultat ». Ce fichier doit avoir les champs suivants :

- Un code unique qui identifie le philosophe de type entier de [0..4]
- Un nom de philosophe de type caractère [1..15]
- Une action de philosophes de type caractère [1..5]

Rappelez-vous que l'algorithme devrait nous donner 50 lignes : 25 lignes qui montrent les philosophes en train de manger et 25 lignes entrain de penser. L'ordre de ces deux actions (manger et penser) est aléatoire ainsi que l'ordre des philosophes.

Une fois le fichier « résultat » est prêt, votre programme doit verrouiller le fichier soit en entier, soit juste une ligne ou seulement un champ de la ligne. Avant de quitter le programme, nous affichons à l'utilisateur une liste de choix sous cette forme :

- **Consulter résultat** : Aucun verrouillage n'est nécessaire
- **Modifier le nom d'un philosophe** : Le verrouillage est fait sur le champ « nom »

- **Supprimer le nom d'un philosophe** : Le verrouillage est fait sur le fichier en entier
- **Modifier le nom et l'action d'un philosophe** : Le verrouillage est fait sur l'enregistrement (toute la ligne)
- **Quitter** : Votre programme nous affiche le résultat initial de votre algorithme et termine.

**Important** : Vous allez créer, en même temps, un deuxième programme qui va s'appeler « Valider » pour valider les verrouillages qui vont être faites par le programme principal sur le fichier « résultat ». Ce programme doit avoir les mêmes options ci-dessus (Consulter résultat, Modifier le nom d'un philosophe, Supprimer le nom d'un philosophe, Modifier le nom et l'action d'un philosophe et Quitter). Nous allons l'utiliser pour valider toutes les options de programme principal. La seule chose qui va partager ce programme avec le programme principal est le fichier « résultat ».

a. **Les philosophes**

Commencez par créer cinq philosophes, chacun est identifié par un nombre de 0 à 4. Chaque philosophe fonctionnera comme un thread séparé. Les philosophes alternent entre penser et manger. Pour simuler les deux activités, laissez le thread en veille pendant une période aléatoire comprise entre une et trois secondes. Quand un philosophe veut manger, il invoque la fonction.

pickup_forks (int philosopher_number)

Où philosopher_number identifie le numéro du philosophe souhaitant manger. Quand un philosophe a fini de manger, il invoque

return_forks (int philosopher_number)

### b. **Variables de condition Pthreads**

Les variables de condition dans Pthreads utilisent le type de données pthread_cond_t et sont initialisées à l'aide de la fonction pthread_cond_init (). Le code suivant crée et initialise une variable de condition ainsi que le verrou mutex associé.

```
pthread_mutex_t mutex;
pthread_cond_t cond_var;
pthread_mutex_init (&mutex, NULL);
pthread_cond_init (&cond_var, NULL);
```

### c. **Éléments importants à considérer**

- La function pthread_cond_wait () est utilisée pour attendre une variable de condition. Le code suivant illustre comment un thread peut attendre que la condition « a == b » devienne vraie à l'aide d'une variable de condition Pthread.

```
pthread_mutex_lock (&mutex);
while (a!= b)
        pthread_cond_wait(&mutex, &cond_var);
pthread_mutex_unlock (&mutex);
```

- Le verrou mutex associé à la variable de condition doit être verrouillé avant l'appel de la fonction pthread_cond_wait (), car il est utilisé pour protéger les données de la clause conditionnelle contre une éventuelle condition de concurrence. Une fois ce verrou acquis, le thread peut vérifier la situation. Si la condition n'est pas vraie, le thread appelle ensuite pthread_cond_wait () en transmettant le verrou mutex et la variable de condition en tant que paramètres.

- L’appel de pthread_cond_wait () libère le verrou mutex, permettant ainsi à un autre thread d’accéder aux données partagées et éventuellement de mettre à jour sa valeur afin que la clause de condition soit évaluée vraie. (Pour vous protéger contre les erreurs de programme, il est important de placer la clause conditionnelle dans une boucle afin que la condition soit revérifiée après avoir été signalée.) Un thread qui modifie les données partagées peut appeler la fonction pthread_cond_signal (), signalant ainsi un thread en attente de la variable de condition.

  Ceci est illustré ci-dessous :

```
pthread_mutex_lock (&mutex);
a = b ;
pthread_cond_signal (&cond_var);
pthread_mutex_unlock (&mutex);
```

- Il est important de noter que l’appel à pthread_cond_signal () ne libère pas le verrou mutex. C’est l’appel ultérieur à pthread_mutex_unlock () qui libère le mutex. Une fois que le verrou mutex est relâché, le processus signalé devient le propriétaire du verrou mutex et renvoie le contrôle à partir de l’appel de pthread_cond_wait ().

**Contraintes**

Une bonne compréhension du langage C, l’utilisation des verrous mutex et des Pthreads et les verrouillages de fichier sous Linux (exmple : lockf) pour réaliser ce travail.

**Réalisation**

- La réalisation de ce travail se fera par groupes d'au plus deux personnes;
- Chaque membre peut être questionné pour savoir s'il maîtrise le travail;
- Le programme doit être en C sous linux (Unix).

**Résultats attendus**

Prenons un exemple avec trois philosophes, lors de l'exécution de votre programme, l'algorithme produit le résultat ci-dessous – en prenant comme hypothèse qu'il n'y a pas d'erreur. Avant d'afficher quelque chose à l'utilisateur, comme nous l'avons mentionné auparavant, votre programme va enregistrer le résultat ci-dessous dans le fichier « résultat ».

| Code | Nom | Action |
|---|---|---|
| 1 | Philosophe 1 | mange |
| 2 | Philosophe 0 | mange |
| 3 | Philosophe 1 | pense |
| 4 | Philosophe 2 | mange |
| 5 | Philosophe 0 | pense |
| 6 | Philosophe 2 | pense |

Une fois l'enregistrement est fait, votre programme nous affiche les 5 options suivantes (Consulter résultat, Modifier le nom d'un philosophe, Supprimer le nom d'un philosophe, Modifier le nom et l'action d'un philosophe et Quitter) et il attendrait notre choix.

**Option 1** : Si je choisi, Consulter résultat, aucun verrouillage n'est fait et je vois sur mon écran ce qui suit :

| | | |
|---|---|---|
| 1 | Philosophe 1 | mange |
| 2 | Philosophe 0 | mange |
| 3 | Philosophe 1 | pense |
| 4 | Philosophe 2 | mange |
| 5 | Philosophe 0 | pense |
| 6 | Philosophe 2 | pense |

**Option 2** : Si je choisi, Modifier le nom d'un philosophe. Votre programme doit avant tout me demander le code de philosophe que je vais modifier son nom et il verrou le champ « nom » qui est associé à ce code.

**Exemple** : Je vais modifier « Philosophe 1 » associé au code 1 en le remplaçant par le nom « Philosophe B ». Une fois que je suis en train de faire la modification et j'ouvre une autre fenêtre (terminal), j'exécute le programme « valider » et j'essaye de faire la même opération (même option). Votre programme devrait me refuser l'accès au nom de code 1 en m'affichant « Le champ « nom » est verrouiller, veuillez essayer plus tard ».
Si je retourne à ma fenêtre initiale et je termine mon opération. Le verrouillage est supprimé.

**Option 3** : Si je choisi, Supprimer le nom d'un philosophe. Votre programme doit verrouiller le fichier en entier « résultat ».

**Exemple** : Je vais supprimer « Philosophe 1 » associé au code 1. Une fois que je suis en train de supprimer le nom, je me déplace vers l'autre terminal (fenêtre) et j'essaye de faire n'importe quelle opération (option). Votre programme (valider) devrait me refuser l'accès au fichier au « résultat » en m'affichant « Le fichier « résultat » est verrouillé, veuillez essayer plus tard ».

Si je retourne à ma fenêtre initiale et je termine mon opération. Votre programme libère le verrou. Et je peux retourner vers mon programme « valider » pour vérifier.

**Option 4** : Si je choisi, Modifier le nom et l'action d'un philosophe. Votre programme doit avant tout me demander le code de philosophe que je vais modifier son nom et son action. Il verrou l'enregistrement (la ligne) qui est associé à ce code.

**Exemple** : Je vais modifier « Philosophe 1 » associé au code 1 à « Philosophe B » et son action (au lieu de manger, je vais le changer à penser). Une fois que je suis en train de faire la modification, je me déplace vers l'autre terminal (fenêtre) et j'essaye de faire la même opération ou une autre opération sur la même ligne. Votre programme (valider) devrait me refuser l'accès à la ligne en m'affichant « L'enregistrement que vous voulez accéder est verrouillé, veuillez essayer plus tard ».

Si je retourne à ma fenêtre initiale et je termine mon opération. Le verrouillage est supprimé.

**Option 5** : Si je choisi, Quitter, votre programme se ferme en affichant le contenu de fichier « résultat ».

**À rendre le 20 décembre 2019**

- Votre programme;
- Le fichier « résultat »
- 1 fichier de manuel d'utilisation (assez simple : comment on compile votre programme? Comment on l'exécute? Et toutes autres informations que vous jugiez pertinentes pour que je puisse tester adéquatement votre travail).

**Évaluation:**

- Le respect des instructions demandées;
- La qualité de la solution technique proposée;
- La qualité du code fourni (commentaires clairs, utilisez juste les appels systèmes ou les fonctions dont vous aurez besoin, etc.);
- Si le programme ne compile pas, vous aurez automatiquement 0 comme note.