UNIVERSITÉ DE TOULOUSE

Licence Informatique

Université Paul Sabatier
118 route de Narbonne
31062 Toulouse Cedex 09

SR2 – Partie « Systèmes » - Travaux pratiques n°4&5
Threads Posix - Sémaphores

RAPPELS. Dans les applications mettant en jeu des threads (et même des processus d'ailleurs), vous devez tester **plusieurs** fois leur exécution, avec des paramètres **différents**, avant d'en conclure que leur comportement est celui **attendu**.

Au besoin, vous pouvez utiliser *usleep()* ou *nanosleep()* pour **temporiser** un peu les exécutions des threads à des endroits stratégiques (dans les boucles, par exemple) afin de ralentir un peu leur vitesse de traitement pour vérifier que l'application continue à fonctionner comme souhaité quelles que soient les actions des threads.

Vous pouvez ajouter de **l'aléatoire** en utilisant *srand()* et *rand(),* la graine sera alors l'identificateur du thread appelant *srand().*

Toutes les primitives ne positionnent pas *errno* et vous pouvez utiliser *strerror()* pour gérer les erreurs.

A vous de consulter le manuel en ligne (*man*) pour plus d'informations sur ces fonctions.

Exercice 1 – Alternance d'affichage

On veut synchroniser l'affichage de NBT threads de manière à ce qu'ils affichent NBM messages (constitué de NBL lignes) à **tour de rôle** à l'écran (même principe que l'exercice vu en TD).

L'application sera **paramétrée** par les valeurs de NBT, NBM et NBL.

Attention : Chaque thread sera **paramétré** par le nombre de messages et le nombre de lignes qu'il devra afficher (ce ne seront pas des variables partagées).

Version 1. La synchronisation sera réalisée à l'aide de **verrous d'exclusion mutuelle** Posix (*pthread_mutex_t*).

Exemples d'exécution : ./exo1v1 2 3 2 // 2 threads, 3 messages de 2 lignes

```
Afficheur 0 (140370571626240), j'affiche ligne 0/2 du message 0/3
Afficheur 0 (140370571626240), j'affiche ligne 1/2 du message 0/3
Afficheur 1 (140370563172096), j'affiche ligne 0/2 du message 0/3
Afficheur 1 (140370563172096), j'affiche ligne 1/2 du message 0/3
Afficheur 0 (140370571626240), j'affiche ligne 0/2 du message 1/3
Afficheur 0 (140370571626240), j'affiche ligne 1/2 du message 1/3
Afficheur 1 (140370563172096), j'affiche ligne 0/2 du message 1/3
Afficheur 1 (140370563172096), j'affiche ligne 1/2 du message 1/3
Afficheur 0 (140370571626240), j'affiche ligne 0/2 du message 2/3
Afficheur 0 (140370571626240), j'affiche ligne 1/2 du message 2/3
Afficheur 0 (140370571626240), je me termine
Afficheur 1 (140370563172096), j'affiche ligne 0/2 du message 2/3
Afficheur 1 (140370563172096), j'affiche ligne 1/2 du message 2/3
Afficheur 1 (140370563172096), je me termine

Fin de l'execution du thread principal
```

Version 2. La synchronisation sera réalisée à l'aide de **sémaphores Posix** (*sem_t*).

Exemples d'exécution : ./exo1v2 2 3 2

La même chose que ci-dessus.

Exercice 2 – Compte bancaire

Ajoutez la **synchronisation** nécessaire (avec les sémaphores de votre choix) pour résoudre les problèmes constatés lors de l'exécution de l'exercice 2 du TP 3.

Exercice 3 – Modèle du producteurs-consommateurs

Si ce n'est fait, terminez l'exercice 3 du TP3. **Ajoutez la synchronisation** nécessaire (avec les sémaphores de votre choix) pour résoudre les problèmes de synchronisation et permettre aux threads de déposer et retirer leurs messages de manière cohérente.

Rappel du principe : Les dépôts se font de manière **circulaire** dans l'ordre des indices croissants et les retraits se font dans **l'ordre des dépôts**.

Exemples d'exécution : ./exo3 2 2 2 2 1

➔ 2 producteurs et 2 consommateurs déposant ou retirant chacun 2 messages, tableau de taille 1

```
        Prod 0 (140442678724352) : Message depose = Bonjour 1 de prod 0
                Conso 0 (140442661816064) : Message retire = Bonjour 1 de prod 0
        Prod 1 (140442670270208) : Message depose = Bonjour 1 de prod 1
                Conso 1 (140442653361920) : Message retire = Bonjour 1 de prod 1
        Prod 0 (140442678724352) : Message depose = Bonjour 2 de prod 0
                Conso 0 (140442661816064) : Message retire = Bonjour 2 de prod 0
        Prod 1 (140442670270208) : Message depose = Bonjour 2 de prod 1
                Conso 1 (140442653361920) : Message retire = Bonjour 2 de prod 1

Fin de l'execution du main
```

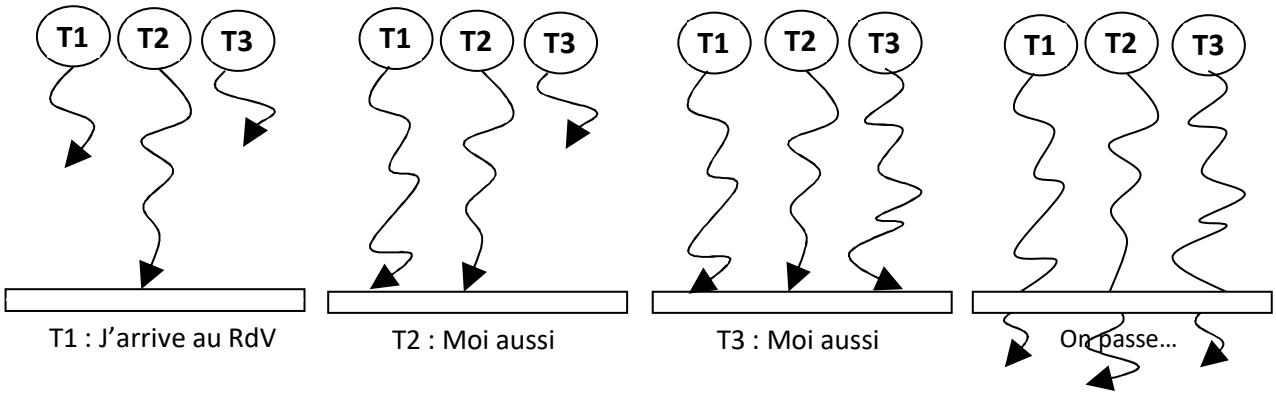
Exercice 4 – Rendez-vous entre threads

En vous basant sur ce qui a été évoqué éventuellement en CTD, programmez le **rendez-vous entre N** threads en assurant la **synchronisation** entre ces threads à l'aide des sémaphores de votre choix..

L'application sera **paramétrée** par le nombre N de threads.

Rappel du principe du rendez-vous : un thread arrivant au point de rendez-vous se met en attente s'il existe au moins un autre thread qui n'y est pas arrivé. Tous les threads bloqués sur cette « barrière » peuvent la franchir **lorsque le dernier y est arrivé**.

La figure ci-dessous illustre ce comportement pour un rendez-vous à 3 :



Un thread aura le comportement suivant :

```
Début
      Je fais un certain traitement ;
      J'arrive au point de rendez-vous
            et j'attends que tous les autres y soient aussi… ;
      …Avant de pouvoir continuer mon traitement ;
Fin
```

Exemples d'exécution : ./exo4 3

/ 3 threads ont un RdV */*

```
Thread 0 (140388848436992) : J'effectue un traitement en parallele avec les autres
Thread 0 (140388848436992) : J'arrive au RdV
Thread 0 (140388848436992) : Je ne suis pas le dernier au RdV
Thread 1 (140388839982848) : J'effectue un traitement en parallele avec les autres
Thread 1 (140388839982848) : J'arrive au RdV
Thread 1 (140388839982848) : Je ne suis pas le dernier au RdV
Thread 2 (140388831528704) : J'effectue un traitement en parallele avec les autres
Thread 2 (140388831528704) : J'arrive au RdV
        Thread 2 (140388831528704) : Je suis le dernier au RdV
        Thread 2 (140388831528704) : Je passe le point de RdV
Thread 2 (140388831528704) : Je continue un traitement en parallele avec les autres
Thread 2 (140388831528704) : Je termine mon traitement
        Thread 1 (140388839982848) : Je passe le point de RdV
Thread 1 (140388839982848) : Je continue un traitement en parallele avec les autres
Thread 1 (140388839982848) : Je termine mon traitement
        Thread 0 (140388848436992) : Je passe le point de RdV
Thread 0 (140388848436992) : Je continue un traitement en parallele avec les autres
Thread 0 (140388848436992) : Je termine mon traitement

Fin de l'execution du thread principal
```

Pour aller plus loin…

Vous pouvez programmer la question 3 des CC4 qui se trouvent dans les annales Moodle.