UNIVERSITÉ DE TOULOUSE

Licence Informatique

Université Paul Sabatier
118 route de Narbonne
31062 Toulouse Cedex 09

SR2 – Partie « Systèmes » - Travaux pratiques n°3
Processus Unix – Threads Posix

A LIRE. 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 – Comparaison Processus / Threads

Écrire une application dans laquelle l'activité principale crée NA activités parallèles.

Chaque activité parallèle affiche un message l'identifiant NF fois avant de se terminer en retournant à son créateur son rang de création.

L'activité principale devra afficher les informations retournées **au fur et à mesure** que les activités parallèles se terminent.

L'application sera paramétrée par les valeurs de NA et de NF.

Entre deux envois, une activité parallèle utilisera usleep() pour attendre un certain délai compris entre 0 et 100 ms.

Version 1. Les activités parallèles sont des **processus**.

Exemples d'exécution : ./exo1v1 2 4

/ 2 processus affichant chacun 4 fois */*

```
Activite rang 0 : identifiant = 1280 (delai = 2)
Activite rang 1 : identifiant = 1281 (delai = 62)
Activite rang 0 : identifiant = 1280 (delai = 27)
Activite rang 0 : identifiant = 1280 (delai = 17)
Activite rang 1 : identifiant = 1281 (delai = 77)
Activite rang 0 : identifiant = 1280 (delai = 6)
Activite rang 1 : identifiant = 1281 (delai = 72)
Activite rang 1 : identifiant = 1281 (delai = 1)
Valeur retournee par le fils 1280 = 0
Valeur retournee par le fils 1281 = 1
```

Version 2. Les activités parallèles sont des **threads** Posix.

Attention : pour apprendre à le faire, le nombre d'affichages sera fourni en **paramètre** de chaque thread (ce ne sera **pas** une variable partagée).

Exemples d'exécution : ./exo1v2 2 5

/ 2 threads affichant chacun 5 fois */*

```
Thread 0 : mon identificateur est 139980856313536 (delai = 54)
Thread 1 : mon identificateur est 139980847920832 (delai = 76)
Thread 0 : mon identificateur est 139980856313536 (delai = 54)
Thread 0 : mon identificateur est 139980856313536 (delai = 74)
Thread 1 : mon identificateur est 139980847920832 (delai = 56)
Thread 0 : mon identificateur est 139980856313536 (delai = 33)
Thread 1 : mon identificateur est 139980847920832 (delai = 7)
Thread 1 : mon identificateur est 139980847920832 (delai = 9)
Valeur retournee par le thread 139980856313536 = 0
Valeur retournee par le thread 139980847920832 = 1
```

Exercice 2 – Threads – Partage d'une variable

Écrire une application dans laquelle NT threads s'exécutent en parallèle et créditent ou débitent (d'un montant aléatoire) le solde d'un compte bancaire **commun**. Lors d'une opération de débit, si le solde est insuffisant, cela sera **signalé** par un message mais l'opération de débit sera quand même réalisée.

L'application sera paramétrée par la valeur de NT, le solde initial du compte et le nombre de débits/crédits.

À l'exécution, faites **varier** le nombre de threads et de débits/crédits, voire temporisez un peu les opérations, que **constatez-vous** ?

Ce sera un simple constat, si vous trouvez des problèmes, vous devrez les résoudre à la prochaine séance de TP.

Exemple d'exécution : ./exo2 2 0 3

/ 2 threads [1 créditeur/1 débiteur], solde initial à 0, 3 opérations chacun */*

```
Credit 140647206946560 : credit (2) => solde = 2
Credit 140647206946560 : credit (1) => solde = 3
Credit 140647206946560 : credit (9) => solde = 12
        Debit 140647198492416 : Si debit de 4 le solde (2) sera debiteur
Debit 140647198492416 : debit (4) => solde = 8
Debit 140647198492416 : debit (1) => solde = 7
Debit 140647198492416 : debit (1) => solde = 6
Solde = 6
```

Remarque : La fonction *int usleep(useconds_t ms)* permet de suspendre l'exécution d'un thread pendant au moins *ms* microsecondes.

Exercice 3 – Threads – Partage d'un tableau géré circulairement (version SANS synchronisation)

[Cet exercice servira de base à la séance 4 sur la synchronisation par sémaphores]

Écrire une application dans laquelle NP + NC threads s'exécutent en parallèle et **partagent** un tableau contenant des messages composés d'au plus 20 caractères et d'une valeur entière.

Les NP threads sont des « producteurs » de messages déposés dans le tableau partagé.

Les NC threads sont des « consommateurs » des messages qui se trouvent dans le tableau partagé.

Les dépôts se font dans l'ordre croissant des indices, de manière circulaire.

Les consommations se font dans l'ordre des dépôts.

L'application sera **paramétrée** par le nombre NP de « producteurs, » celui NC de « consommateurs », le nombre de « dépôts » et de « consommations » et la taille du tableau partagé.

Mettez au point l'application en considérant, dans un premier temps, des messages qui sont des entiers i.e. le tableau partagé ne contiendra que des entiers. Puis, quand votre application fonctionnera, complexifiez-la pour avoir des messages tels que décrits ci-dessus.

À l'exécution, faites **varier** les paramètres, voire temporisez un peu les threads, que **constatez-vous** ?

Exemple d'exécution : ./exo3 2 2 3 3 1

➔ 2 producteurs et 2 consommateurs déposant ou retirant chacun 3 messages, tableau de taille 1

```
        Prod 1 (139860106544896) : Message depose = Bonjour 1 de prod 1
        Prod 1 (139860106544896) : Message depose = Bonjour 2 de prod 1
        Prod 1 (139860106544896) : Message depose = Bonjour 3 de prod 1
                Conso 1 (139860089636608) : Message retire = xxxxx
                Conso 1 (139860089636608) : Message retire = Bonjour 3 de prod 1
                Conso 1 (139860089636608) : Message retire = xxxxx
        Prod 0 (139860114999040) : Message depose = Bonjour 1 de prod 0
        Prod 0 (139860114999040) : Message depose = Bonjour 2 de prod 0
        Prod 0 (139860114999040) : Message depose = Bonjour 3 de prod 0
                Conso 0 (139860098090752) : Message retire = Bonjour 1 de prod 1
                Conso 0 (139860098090752) : Message retire = Bonjour 3 de prod 0
                Conso 0 (139860098090752) : Message retire = xxxxx

Fin de l'execution du main
```