# Sujet du TP n° 7

## Préambule

Les ressources du cours (supports, exemples, et autres documents) et des TP (les sujets, les fichiers de travail ainsi que les corrigés) se trouvent à l'URL : http://www.emse.fr/~lalevee/ismin/pse, dénommé **[site]** dans les documents fournis.

Ce TP suppose que vous avez installé l'archive **PSE.tar**. Si ce n'est pas le cas, consultez le sujet du premier TP.

Placez-vous dans le répertoire **PSE/TP7**.

## les mutex (transparents 127 à 130)

**Exercice 1**

- le répertoire **TP7** contient un fichier **exercice1.c**
- compilez-le en tapant **make**
- testez-le, en donnant des valeurs de plus en plus grandes (boucles et/ou threads)
- que se passe-t-il ?
- expliquez.

**Exercice 2**

- modifiez le programme précédent afin que le résultat soit correct

**Exercice 3**

- écrivez un programme qui crée 2 threads, puis qui attend leur fin
- le premier thread *produit* une information, que le second *consomme* ensuite
- pour simuler le fonctionnement, la *production* consistera à attendre 200 ms, puis à générer un nombre aléatoire entre 1 et 6
- la *consommation* consistera à afficher ce nombre et à attendre 100 ms.

Notes :

- pour attendre n millisecondes, utilisez la fonction **usleep()** ou **nanosleep()**
- pour la génération de nombres aléatoires, utilisez **random_r()**
  profitez du manuel de cette fonction, pour réviser ce qu'est une fonction réentrante, et pour comprendre pourquoi **random()** n'est pas utilisable dans cette application !
- pour initialiser le générateur, utilisez **initstate_r()**
- pour donner une valeur au paramètre **seed**, vous pouvez utiliser **time(NULL)** en incluant **<time.h>**

**Exercice 4**

- modifiez le programme précédent, pour qu'il y ait 3 threads consommateurs (le producteur tire le nombre aléatoire dans des intervalles successifs)
- qu'en est-il de la synchronisation ?
- que pensez-vous qu'il se passerait si vous ajoutiez un second producteur ?

# Application multithread (transparents 120 et 121)

Dans le TP précédent, pour la mise en œuvre d'une cohorte, nous avons utilisé des attentes actives, et des mécanismes de communication simples. De plus, la gestion des *workers* libres a été réalisée avec de la scrutation. Nous allons tenter de pallier ces problèmes.

**Projet**

- copiez les fichiers **serveur.c** et **client.c** du répertoire **TP6** dans **TP7**
- modifiez le serveur en utilisant des **semaphore**s pour *réveiller* un *worker* en attente de travail et pour gérer les **workers** libres
- vous devrez modifier le fichier **dataspec.h** dans le répertoire **include** pour y placer les nouvelles données spécifiques
- n'oubliez pas ensuite de régénérer la bibliothèque **libpse.a** en tapant **make** dans le répertoire **modules**.