

TP 3 – Conditions POSIX

Exercice 1

On veut écrire un programme pour gérer un centre de calcul, dédié aux applications scientifiques parallèles :

- le centre de calcul contient m machines ;
- les n utilisateurs simultanés lancent chacun p « jobs » en séquence ;
- le job j de l'utilisateur u nécessite k_{uj} machines ($1 \leq k_{uj} \leq m$).

On décide de représenter les utilisateurs par des threads. L'algorithme du thread pour l'utilisateur u est :

```

pour  $j$  variant de 1 à  $p$ 
    attendre que  $k_{uj}$  machines soient disponibles
    exécuter le job  $j$ 
    remettre les  $k_{uj}$  machines à disposition
    
```

Écrivez le programme correspondant, qui doit admettre les trois arguments m , n et p . Les valeurs k_{uj} seront déterminées aléatoirement entre 1 et m . Chaque job ne fait rien d'autre qu'appeler `sleep(t)` avec t tiré aléatoirement entre 1 et 3. Vous utiliserez une variable globale partagée pour représenter le nombre de machines actuellement disponibles, et vous utiliserez une condition POSIX pour synchroniser les utilisateurs.


Exercice 2

On considère un système composé de 6 cuisiniers et d'un épicier, dont le but est de produire des gâteaux. La production d'un gâteau nécessite 4 ingrédients : du beurre, des œufs, de la farine et du sucre.

Les cuisiniers disposent chacun de deux ingrédients, en quantité illimitée, selon le tableau ci-contre. Notez que tous les cuisiniers ont des ingrédients différents.

Périodiquement, l'épicier livre sur une table deux ingrédients quelconques, choisis aléatoirement, en quantité suffisante pour produire un gâteau. Il attend ensuite un gâteau, avant de partir chercher à nouveau deux ingrédients. Notez que lorsque l'épicier apporte deux ingrédients, un seul des cuisiniers est à même de produire un gâteau (mais l'épicier ignore lequel).

Cuisinier :	1	2	3	4	5	6
Beurre			✓		✓	✓
Œufs		✓		✓		✓
Farine	✓			✓	✓	
Sucre	✓	✓	✓			

Le fichier ci-attaché contient un squelette de ce programme, que vous pouvez compléter : 

(Si vous ne parvenez pas à extraire le document attaché, consultez votre source habituelle.)

Exercice 3

On souhaite implémenter les sémaphores en utilisant les « conditions » offertes par l'API POSIX des threads.

On rappelle que les sémaphores sont constitués d'un compteur et de deux opérations P et V, dont l'implémentation peut être :

```

P
-----
compteur--
si compteur < 0
    alors attendre
fin si
    
```

```

V
-----
compteur++
si compteur ≤ 0
    alors réveiller un thread
fin si
    
```

Les algorithmes ci-dessus doivent être implémentés en section critique.

- définissez le type `monsem_t` nécessaire pour représenter un sémaphore;
- implémentez la fonction `int monsem_init (monsem_t *, int)`, où le deuxième paramètre est la valeur initiale du compteur;
- implémentez la fonction `int monsem_P (monsem_t *)`;
- implémentez la fonction `int monsem_V (monsem_t *)`.

Comme toutes les fonctions `sem_*`, ces fonctions doivent renvoyer 0 si tout s'est bien passé ou -1 en cas d'erreur (et modifier la variable `errno`).

À l'aide des sémaphores que vous avez implémentés, écrivez un programme composé de deux threads : le thread généré lit des octets sur l'entrée standard et les place dans un tableau de 10 éléments que le thread principal lit et affiche sur la sortie standard.

Votre programme doit bien évidemment :

- ne comporter aucune variable globale;
- s'arrêter lorsqu'une fin de fichier est rencontrée sur l'entrée standard;
- fonctionner sans problème avec des données binaires;
- produire une sortie identique à l'entrée; par exemple, la commande suivante doit afficher « ok ».

```
$ ./a.out < /bin/ls > toto && cmp /bin/ls toto && echo ok
```