Travaux Dirigés de Réseau n°1

Cours de Réseau

—IG 2000 - IR 2—

Janvier 2004

▸ **Exercice 1.** *[Producteur et consommateur]*
On désire développer une petite application simulant les comportements concurrents d'un ensemble de producteurs et de consommateurs de messages. D'un côté, les producteurs produisent des messages qu'ils stockent dans une file d'attente commune ; de l'autre, les consommateurs récupèrent dans ce même buffer les messages (dans l'ordre où ils y ont été placés).

On doit respecter un certain nombre de contraintes et pouvoir paramétrer l'application:

1. *un producteur ne peut produire un message que si le buffer n'est pas plein et, bien sûr, un consommateur ne peut lire un message que si le buffer n'est pas vide;*
2. *le débit de chaque producteur et de chaque consommateur est paramétrable à sa construction (il représente le temps d'attente entre deux productions ou deux consommations).*

Écrire les classes `Producteur` *et* `Consommateur` *permettant de mettre en oeuvre ces spécifications. Chaque producteur et consommateur sera exécuté par un processus léger (*`Thread`*) différent, en concurrence avec les autres. Vous pouvez utiliser d'autres threads.*

▸ **Exercice 2.** *[Pool (simple) de processus légers]*
On souhaite écrire un programme qui teste la primalité de tous les entiers compris entre 2 et `N`*, par la méthode la plus stupide qui soit: pour chacun de ces entiers, on essaie tous les diviseurs possibles.*

Le test de primalité d'un entier `i` *sera confié à un processus léger, qui essaiera tous les diviseurs potentiels, et mettra un message indiquant la primalité de l'entier dans un fichier de résultats (le fichier est le même pour tous les résultats). Les créations/destructions successives de contrôleurs de processus légers, ainsi que leur ordonnancement, sont des opérations couteuses en temps et en mémoire. Il convient de les limiter au maximum.*

Pour cela, votre programme devra gérer un ensemble (pool) de processus légers de taille limitée par une constante `MAX_THREADS`*. L'idée est que quand un processus léger a terminé son travail, il n'est pas détruit: il est gardé en réserve. Quand l'application*

a de nouveau besoin d'un processus léger pour exécuter un autre travail, elle regarde dans la réserve s'il existe un processus léger disponible. Si oui, elle l'emploi. Si non, elle attend qu'un processus léger se termine.