# TD 5 : Sur les paradigmes de la concurrence

**Exercice 1 :**

Une application contient N tâches « ouvrière » lancées en concurrence par une tâche « reine ». La tâche « reine » fonctionne de la façon suivante : elle lit un caractère au clavier, le stocke dans une variable V globale au programme (et donc à toutes les tâches de celui-ci) et signale les N tâches « ouvrière ». Elle attend alors un signal de fin de travail de chacune des tâche « ouvrière » avant d'afficher un saut à la ligne et de recommencer (lecture d'un caractère, signale, attente, etc.).

Le comportement des tâches « ouvrière » consiste à attendre le signal de la tâche « reine », à afficher le caractère stocké dans la variable globale V et à envoyer un signal de fin d'action à la tâche « reine ».

Implanter cette application en Java en utilisant le paradigme du passage de témoin (vous prendrez N égal à 10) ; vous direz pourquoi ce paradigme est suffisant pour résoudre ce problème.

**Exercice 2 :**

N=10 taches (T1 à T10) affichent chacune un caractère de ('A' à 'J') ; on souhaite grâce à l'utilisation de 10 sémaphores faire en sorte que le résultat de l'exécution de ces 10 taches lancées en parallèle produise systématiquement ABCDEFGHIJABCDEFGHIJABC....
Le paradigme utilisé ici s'appelle le « passage de témoins » ou « passage de bâton » : le main passe le témoin à T1 qui passe le témoin à T2, qui passe le témoins à T3, ..., qui passe le témoins à T10 qui passe le témoin à T1, etc. Implémenter une solution de ce problème en C/ POSIX et en Java.

**Exercice 3 :**

On souhaite maintenant que N tâches fonctionnent en pipeline : la tâche 0 reçoit un caractère du main, puis le passe à la tâche 1 qui le passe à la tâche 2 et ainsi de suite ; la dernière tâche (N-1) affiche le caractère à l'écran.

Donner une solution à ce problème en C/POSIX en utilisant le paradigme du passage de témoins et des Mutex (signal) pour 3 tâches ; généraliser à N tâches.

**Exercice 4 :**

On souhaite implémenter le crible d'Ératosthène à l'aide de tâche Java ; on rappelle que cet algorithme permet de calculer tous les nombres premiers inférieurs à une constante N donnée.

Son implémentation à l'aide de tâches se fera de la façon suivante : chaque tâche *i* est créée avec un nombre entier $n_i$ et devra « filtrer » les nombres multiples de $n_i$ et « passer » à la tâche *i+1* les nombres qu'elle reçoit de la tâche *i-1* et qui ne sont pas multiples de $n_i$. La tâche *i* sera créée par la tâche *i-1* lorsque cette dernière reçoit le premier nombre non multiple de $n_{i-1}$. Lorsque la tâche i reçoit la valeur 0 elle imprime son nombre ni et termine. Une tâche « générateur » génère les nombres de 2 à N et les « passe » à la tâche 2 puis passe la valeur 0 pour terminer l'algorithme.

Implémenter cet algorithme en utilisant le paradigme « producteur consommateur » ; lorsque la tâche *i* crée la tâche *i+1* elle crée aussi le tampon d'échange entre *i* et *i+1* et passe cette instance en paramètre à la tâche créée (de même que le nombre dont elle a la charge).