Travaux Dirigés de Réseau n°1

Cours de Réseau

—IG 2000 - IR 2—

Décembre 2003

▸ **Exercice 1.** *[Les Threads] Écrire un programme qui crée deux instances de la classe* `Affiche1` *et qui les démarre.*

1 *Écrire une classe* `Affiche1` *qui étend la classe* `Thread`*. Cette classe possède comme champ un entier identifiant chaque instance. Elle redéfinit la méthode* `run()` *de la classe* `Thread` *pour qu'elle affiche, dans une boucle de longueur* `n` *fixée, un message indiquant le numéro de l'instance courante et l'indice de la boucle.*

2 *Écrire une classe* `Affiche2` *fournissant les mêmes fonctionnalités que* `Affiche1` *mais sans hériter de* `Thread`*. Tester cette classe dans le main précédemment écrit. Que remarquez vous ?*

▸ **Exercice 2.** *[Problèmes d'exclusion mutuelle] On cherche à étudier le comportement d'un objet modifié par plusieurs threads.*

1 *Écrire une classe* `Point` *avec deux champs entiers* `x` *et* `y`*, une méthode* `void set(int x, int y)` *(qui déplace le point aux coordonnées passées en arguments) et une méthode* `toString()` *qui affiche ce point au format* `(x, y)`*.*

2 *Écrire une classe* `Imp` *ayant un constructeur prenant en paramètre un* `point` *et un entier* `i`*. La classe* `Imp` *possède une méthode* `run()` *qui démarre un processus léger qui appelle la méthode* `set(i, i)` *sur le point puis l'affiche, et ce dans une boucle infinie.*

Créer deux instances de la classe `Imp` *construites avec le même point* `p` *et des valeurs différentes (par exemple,* `Imp(p, 1)` *pour l'un et* `Imp(p, 2)` *pour l'autre). Démarrer les deux instances. Quels sont les différents affichages possibles pour le point ? Que faire pour y remédier ?*

▸ **Exercice 3.** *[Nombres premiers] Écrire une petite application interactive qui demande à l'utilisateur, dans une boucle, de saisir des nombres entiers. Quand un nombre entier est saisi, il est mis dans une file d'attente* `f1`*. Un processus léger va récupérer les nombres dans la file d'attente, et calculer leur primalité. Quand il a terminé le calcul*

de primalité du nombre `n`*, il insère le couple* `(n, p)`*, où* `p` *est un booléen indiquant la primalité de* `n`*, dans une autre file d'attente* `f2`*. Quand l'utilisateur indique, par le menu interactif, qu'il ne souhaite plus rentrer de nombres, le programme récupère tous les couples de* `f2` *et affiche l'ensemble de couples ainsi récupérés.*

▸ **Exercice 4.** *[Sommes partielles] Écrire un programme qui tire 10 000 nombres entiers au sort, les stockent dans un tableau, et calcule la somme des éléments du tableau de la manière suivante: on divise le tableau en dix parties et on confie le calcul de la somme partielle d'une partie à un processus léger. Quand les dix processus légers ont terminés leurs sommes partielles, on récupère les résultats et on en fait la somme pour obtenir le résultat final.*

▸ **Exercice 5.** *[Mini ramasse-miettes] Dans une classe* `Graphe` *représentant les graphes enracinés (c'est-à-dire avec un sommet de départ) le constructeur des noeuds du graphe stocke les noeuds dans un tableau au fur et à mesure de leur construction. On souhaite que de temps en temps, le ménage soit fait à l'intérieur du tableau, pour récupérer de la place: c'est-à-dire que les éléments qui sont inaccessibles à partir de la racine doivent disparaître. Écrire une classe* `RamasseMiettes` *interne à* `Graphe` *qui s'active de temps en temps pour faire le ménage à l'intérieur du tableau, sans pour autant que le reste de l'application s'arrête. Afin d'éviter la fragmentation du tableau, le ramasse-miettes réalise aussi une opération de compactage du tableau quand il l'a débarrassé de ses noeuds inutiles.*