Haute école d'ingénierie et de gestion du Canton de Vaud
Département TIC
Laboratoire de programmation répartie PRR

Temps à disposition : 8 périodes (travail débutant le mardi 20 septembre 2016)

Objectifs

- Écrire son premier programme Java réparti.
- Réaliser ses premières communications par UDP et par diffusion partielle.
- Se familiariser avec un environnement de programmation Java.

Énoncé

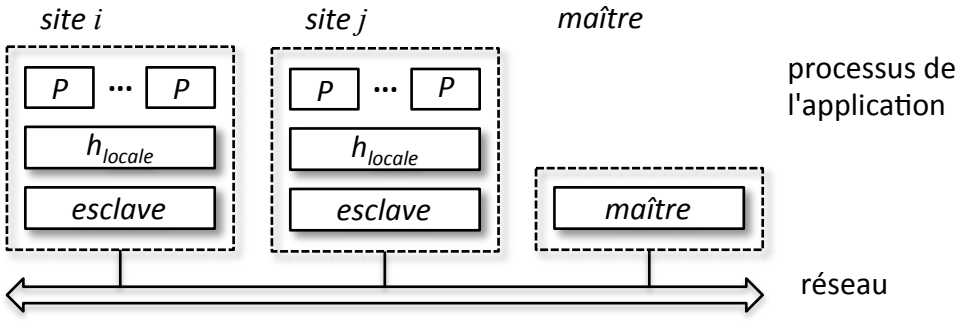
Nous souhaitons implémenter un algorithme simple permettant de synchroniser approximativement les horloges locales des processus s'exécutant sur un ensemble de sites. Comme nous le savons, chaque site d'un environnement réparti possède sa propre horloge système mais aussi, cette horloge a un décalage et une dérive qui lui est propre. Le but de notre algorithme est de rattraper cette dérive sans pour autant corriger l'horloge du système. Pour ce faire, nous distinguons 2 horloges. L'horloge système est l'heure retournée par un appel système; cette horloge désigne la minuterie mise à jour par le système d'exploitation d'un site. Sous un système protégé, il faut avoir des privilèges administrateur pour le modifier et, pour contourner ce problème, un processus usager peut interpréter le temps comme la valeur de l'horloge système additionnée à un décalage. Dans ce qui suit, le résultat de cette opération est appelé l'horloge locale. Ainsi,

$h_{locale} = h_i + décalage_i$ (c.-à-d. l'heure système sur le site i + son décalage)

La synchronisation des horloges revient alors à trouver $décalage_i$ pour chaque site i de l'environnement réparti de telle sorte que h_{locale} est identique pour chaque processus de l'application.

Algorithme

Les sites se divisent en 2 groupes : le maître qui est unique et les esclaves qui sont subordonnés au maître pour l'exécution de l'algorithme. Notons qu'un site représente une machine virtuelle Java (VM) et il peut y avoir plusieurs VM par site physique.



Périodiquement, le maître demande à ses esclaves de retourner l'écart séparant son horloge système (h_i) et avec celui qu'il a reçu ($h_{maître}$). Après avoir reçu toutes les réponses, le maître détermine la valeur du décalage maximale et le diffuse à ses esclaves ($h_{maître} + décalage_{max}$). Les esclaves peuvent ensuite réajuster leurs horloges locales : $décalage_i = (h_{maître} + décalage_{max}) - h_i$.

L'algorithme décrit est inspiré de celui pour synchroniser des postes de travail sous UNIX Berkeley.

Propriétés

- En réalisant des diffusions, la variabilité du temps de transmission entre des sites différents est réduite, si cette diffusion se fait sur un média de diffusion commun.
- L'horloge locale d'un site progresse toujours dans le même sens et ne reviendra jamais à une valeur du passé.

Travail à faire

Implémenter cet algorithme en Java avec les hypothèses suivantes :

1. Le maître et ses esclaves s'exécutent sur des plates-formes homogènes et ils sont reliés par un LAN à diffusion. Le nombre d'esclaves peut varier de 1 à *n* (une constante).
2. Les sites et le réseau qui les interconnecte sont entièrement fiables (pas de panne).
3. Les communications du maître vers ses esclaves sont des diffusions.
4. Les communications des esclaves au maître se font par datagrammes UDP point-à-point.
5. Les messages doivent être les plus courts possibles. (Les chaînes de caractères ne doivent pas se substituer à des nombres.)

Remarques

- Vous devez nous rendre un listage complet de vos sources et aussi nous les transmettre par courrier électronique.
- La description de l'implémentation, ses différentes étapes et toute autre information pertinente doivent figurer dans les programmes rendus. Aucun rapport n'est demandé.
- Inspirez-vous du barème de correction pour connaître là où il faut mettre votre effort.
- Vous pouvez travailler en équipe de deux personnes.

Barème de correction

Conception (structure, décomposition et simplicité)	15%
Portabilité sur un autre environnement réparti, réutilisation du code dans un projet plus conséquent, robustesse du code	15%
Exécution et fonctionnement	20%
Codage (choix des variables, opérations, lisibilité, localité de référence, etc.)	20%
Documentation des en-têtes (programme et méthodes)	20%
Commentaires au niveau du code (qualité et complétude)	10%