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

Temps à disposition : 10 périodes
Distribué le : mardi 18 septembre 2018 à 14h55
A rendre le : mardi 30 octobre 2018 à 14h55 **(avant la séance de labo)**

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

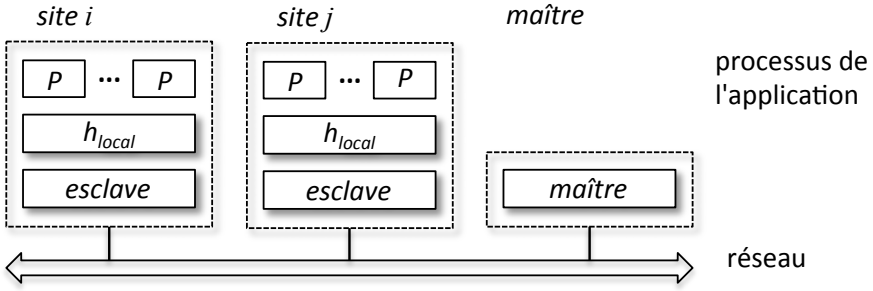
Nous souhaitons implémenter un algorithme simple permettant de synchroniser approximativement les horloges locales des tâches d'une application répartie. Comme nous le savons, chaque site d'un environnement réparti possède sa propre horloge système, mais aussi, cette horloge a un décalage et une dérive qui lui est propre. Le but de notre algorithme est de rattraper ce décalage sans pour autant corriger l'horloge du système. Pour ce faire, nous distinguons 2 horloges. L'horloge système h_{sys} 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 les privilèges administrateurs pour le modifier et, pour contourner ce problème, une tâche applicative peut interpréter le temps comme la valeur de l'horloge système sur le site où elle réside additionnée à un décalage. Dans ce qui suit, le résultat de cette opération est appelé l'horloge locale. Ainsi pour la tâche applicative i, nous avons

$$h_{locale}(i) = h_{sys}(\text{site de } i) + décalage(i)$$

La synchronisation des horloges revient alors à trouver $décalage(i)$ pour chaque tâche i de telle sorte que $h_{locale}(i)$ est identique pour toutes les tâches formant l'application répartie.

Protocole de synchronisation

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. Cette structure se représente ainsi :



Si *P* désigne les tâches de l'application, chaque tâche peut alors interroger son gestionnaire d'horloge (*esclave* sur la figure) résidant sur le même site qu'elle. Elle obtient alors une heure qui est approximativement synchrone avec les autres tâches sur d'autres sites de l'application. Ce sont alors uniquement les esclaves et le maître qui doivent se synchroniser.

La procédure de synchronisation est réalisée en 2 étapes. La première détermine l'écart temporel, $écart(i)$, entre le maître et les esclaves, alors que la seconde corrige le délai de transmission et les latences, $délai(i)$, entre le maître et ses esclaves. Leur somme donne $décalage(i) = écart(i) + délai(i)$, pour le site i.
Périodiquement, le maître diffuse 2 messages. Le premier message est de type SYNC et

contient un identifiant. Le second message, FOLLOW_UP, diffusé immédiatement après le premier, contient l'heure, $t_{maître}$, auquel le maître a émis le message SYNC.

boucle
  identifiant ← *identifiant* + 1
  msg ← <SYNC,*identifiant*>
  $t_{maître}$ ← h_{sys}(*maître*)
  diffusion de *msg*
  msg ← <FOLLOW_UP,$t_{maître}$,*identifiant*>
  diffusion de *msg*
  attente de k secondes
fin boucle

La correction de l'écart se réalise indépendamment par tous les esclaves lors de la réception du message FOLLOW_UP. Les esclaves obtiennent $t_i = h_{sys}(i)$ quand ils reçoivent le message SYNC, puis à la réception du message FOLLOW_UP, ces esclaves calculent

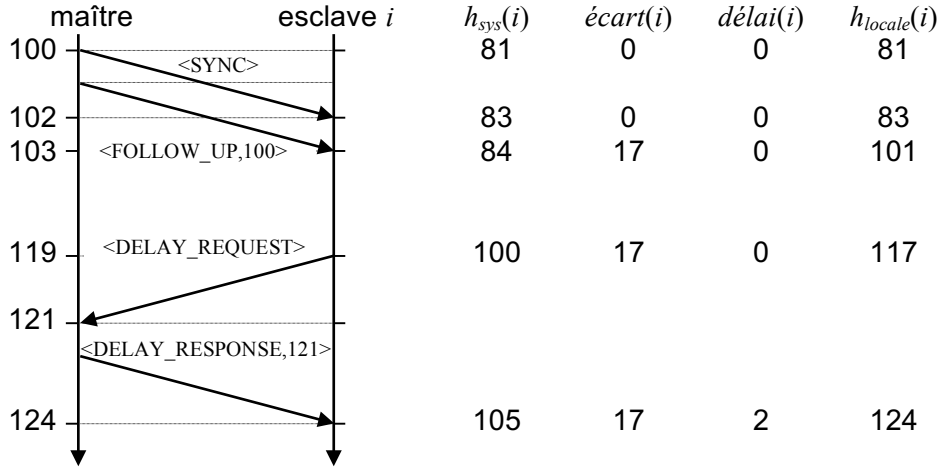
$écart(i) = t_{maître} - t_i$.

Sur un environnement idéal (temps de calcul instantané et délai de transmission nul), les horloges du maître et de ses esclaves seraient synchrones à l'issue de cette première étape.
La seconde étape est lancée par les esclaves pour mesurer approximativement la latence de transmission entre les tâches esclaves et le maître. Un message de type DELAY_REQUEST est émis au maître. À la réception de ce message, le maître renvoie un message de type DELAY_RESPONSE à l'esclave contenant l'heure qu'il a reçu le message DELAY_REQUEST. L'esclave peut ensuite raffiner son décalage par rapport au maître : si t_{es} est l'heure locale d'émission de DELAY_REQUEST et t_m celle de sa réception par le maître, alors

$délai(i) = (t_m - t_{es}) / 2$.

Notons que l'esclave insère aussi un identifiant dans le message DELAY_REQUEST. Cet identifiant est renvoyé par le maître et sert de contrôle.
Cette seconde étape est exécutée irrégulièrement et à des intervalles de temps supérieurs à k. L'esclave entame la seconde étape pour la première fois après la première étape et après un temps aléatoire tiré de l'intervalle $[4k,60k]$. Toutes les fois subséquentes, cette étape se fait après la précédente et après un temps aléatoire tiré depuis le même intervalle.

La figure ci-dessous illustre un échange typique.



$h_{sys}(i)$	$écart(i)$	$délai(i)$	$h_{locale}(i)$
81	0	0	81
83	0	0	83
84	17	0	101
100	17	0	117
105	17	2	124

Remarques sur le protocole

diffusions?

- 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.
- Le protocole suppose que les latences sont symétriques, c.-à-d. que les délais des communications entre le maître et ses esclaves ne dépendent pas du sens de la communication.
- Le protocole décrit correspond à la norme IEEE 1588 admise en 2002 utilisé pour

synchroniser des horloges temps réel sur un bus de terrain et réaliser la cohérence temporelle des acquisitions. Ce protocole est aussi connu sous le nom de « Precision Time Protocol » ou PTP.

Travail à faire

Implémenter cet algorithme en Java avec les hypothèses suivantes :

1. Le maître et ses esclaves s'exécutent sur des plates-formes différentes (JVM) et ils sont reliés par un LAN à diffusion. Le nombre d'esclaves peut varier de 1 à n.
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 partielles.
4. Les messages DELAY_REQUEST et DELAY_RESPONSE se font par datagrammes point-à-point UDP.
5. Les messages doivent être les plus courts possibles. (Les chaînes de caractères ne doivent pas se substituer à des nombres!) Il vous faudra travailler avec des octets.

Remarques sur le travail à effectuer

- Vous devez nous rendre un listage (papier) complet de vos sources (fichiers sources Java).
- 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.
- Une démo de votre programme pourrait être demandée.

Barème de correction

Conception (structure et décomposition)	10%
Portabilité sur un autre environnement réparti, réutilisation du code dans un projet plus conséquent, robustesse	15%
Exécution et fonctionnement (démo sur 2 PC différents)	20%
Codage (choix des variables, opérations, lisibilité, localité de référence, etc.)	20%
Documentation des en-têtes (programme et méthodes)	25%
Commentaires au niveau du code (qualité et complétude)	10%