Intergiciels

1h45, documents autorisés

12 avril 2018

Les exercices sont indépendants.

1 Socket (7 pt)

On considère un serveur Web qui est couplé avec une application. Chaque client (navigateur) établit classiquement une connexion socket avec le serveur et envoie une requête HTTP. Le serveur répondra avec un document HTML. Pour ce faire, le serveur peut soit répondre directement avec le contenu d'un fichier (réponse statique), soit s'adresser à l'application qui calcule une réponse et renvoyer celle-ci (réponse dynamique). Le serveur et l'application sont reliés par un lien socket unique.

FIGURE 1 – Serveur web avec serveur applicatif

1. Dans quel sens peut ou doit se faire l'établissement du socket connecté entre le serveur web et le serveur applicatif ?
2. Donner les *algorithmes* d'initialisation du serveur web et du serveur applicatif pour établir la connexion entre eux (séquence des appels systèmes tels que `connect`, `accept`, `read`..., en indiquant informellement les valeurs des paramètres essentiels ; on ne demande pas du code C).
3. Donner l'*algorithme* du serveur web pour initialiser l'attente d'un client.
4. Donner l'*algorithme* du serveur web pour traiter une requête d'un client en explicitant les deux cas, réponse statique et réponse dynamique. Pour simplifier, on suppose qu'une requête ou une réponse HTTP est lue ou écrite en un seul appel de `read/write` avec un buffer de 1024 bytes. Ne pas se préoccuper pas du décodage des requêtes : supposer qu'il existe des fonctions `getURL(buffer)` pour obtenir l'URL contenue dans une requête client, `isStatic(URL)` pour savoir si la réponse doit être statique ou dynamique.
5. Si le serveur web est multi-threadé, quel problème cela pose-t-il ? Proposer une solution simple pour l'éviter.

2 RMI (9 pt)

On souhaite réaliser un service d'alarme auquel un processus peut s'abonner pour être prévenu quand une certaine date ou un certain événement survient. Nous avons donc un serveur d'alarme et un ou plusieurs clients. Le principe est le suivant :

— Le client appelle le serveur pour lui dire quand (ou de quoi) il souhaite être informé.

— Quand un événement (p.e. une date) survient qui concerne un (ou plusieurs) client(s), le serveur informe ce(s) client(s).

1. Quel est le schéma d'interaction de ce système ?
2. Proposer les interfaces RMI nécessaires pour le réaliser, et donner la classe qui contient une requête client pour être prévenu à une date donnée.
3. Donner un exemple de code client l'utilisant.
4. Donner la structure du serveur (sans rentrer dans les détails algorithmiques permettant de décider quel client est concerné).

Ce système n'est pas tolérant à l'arrêt du serveur. On propose alors d'utiliser un schéma avec primaire et backup. Les clients s'adressent au serveur primaire uniquement. Quand le serveur primaire reçoit une requête d'un client, il la transmet au backup s'il est présent. Le backup est passif tant que le primaire est présent. Quand le backup découvre que le primaire est arrêté, il prend en charge les déclenchements à venir, assurant ainsi que les clients seront informés à la date qu'ils avaient demandée.

5. Comment le primaire peut-il savoir si le backup est présent pour lui transmettre les requêtes des clients ?
6. Comment le backup peut-il détecter la panne du primaire ?
7. Proposer une interface RMI pour le backup.
8. Si nécessaire, modifier l'interface RMI du serveur primaire.
9. Donner la structure algorithmique du backup.

3 Intergiciel à messages (4 pt)

On souhaite réaliser le noyau Linda (TSpaces) vu en TP en s'appuyant sur un intergiciel à messages, et *sans* serveur centralisé. On ne s'intéresse qu'aux opérations write (dépôt d'un tuple), take (retrait d'un tuple), read (consultation d'un tuple). Dans l'architecture proposée, chaque application cliente contient un noyau qui conserve des tuples en local et qui communique avec les autres noyaux via un intergiciel à message. Chaque noyau local a son propre ensemble de tuples, indépendant des autres. L'espace global des tuples peut alors être vu comme l'union des espaces locaux (mais on ne construira jamais explicitement cette union).

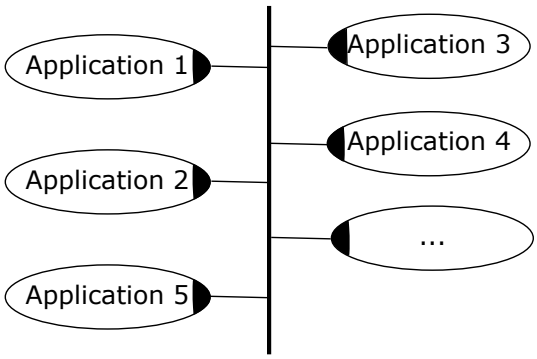

FIGURE 2 – MoM : Schéma général

1. Quand une application dépose un tuple (write), son noyau local l'envoie sur le MoM. La règle est qu'un seul noyau doit obtenir le tuple. Quelle est le type de la Destination (Topic ou Queue) ?
2. Quand une application demande un tuple (read ou take), le noyau local regarde d'abord dans son ensemble, puis s'il ne le trouve pas, il envoie un message demandant ce motif. Ce message doit parvenir à tous. Quel est le type de la Destination ? Peut-on utiliser la même Destination que pour les write ?
3. Quand un noyau local reçoit un message de requête pour un motif et s'il possède un tel tuple, il le retire de son ensemble local et il l'envoie sur la même destination qu'un write. Comment traiter le fait que plusieurs noyaux pourraient répondre à la requête (un tuple demandé mais plusieurs réponses) ?
4. Quand une application termine, il peut rester des tuples dans son espace local et il ne faut pas qu'ils disparaissent avec la fin de l'application. Proposer une solution pour cela.