TP11

Page web du cours :

`http://www.lsv.ens-cachan.fr/~schwoon/enseignement/systemes/ws1516/`.

1 Élection d'un leader

Un problème important dans la programmation concurrente est de choisir un *leader*, p.ex. pour l'organisation et coordination d'un réseau. On part avec un nombre n des ordinateurs ou processus (on parlera des *nœuds*) qui sont à priori tous pareil, sauf un identifiant unique. On suivant un même protocol, tous les nœuds vont se mettre d'accord sur un seul nœud qu'ils accepteront comme leader.

Il existe une grande variété de protocoles pour cet objectif. On va en étudier (et réaliser) un protocole par Dolev, Klawe, and Rodeh qui se distingue par le faible nombre de messages nécéssaire pour l'élection qui sera de $\mathcal{O}(n \log n)$. Par comparaison, les approches simples ont tendance a nééssiter prendre $\mathcal{O}(n^2)$ messages. Le protocole va être presenté pendant le TP, les diapos sont aussi disponible sur la page web.

Le protocole part du principe que les nœuds sont organisé en anneau, chaque nœud envoie des messages vers son voisin à droite. La page web contient un code squelette qui va créer n processus (les nœuds) pour un n donné ; ces processus seront déjà équipées des pipes pour communiquer. Votre tâche est de compléter le programme en réalisant le protocole :

- Au départ, tous les nœuds sont *actifs* et possèdent un identifiant unique.
- Un nœud actif attend une petite période aléatoire (fonction `delay()`), puis envoie son identifiant à son voisin à droit. Ensuite, il attendra les identifiants des *deux* plus proches voisins actifs à gauche. Il décidera ensuite s'il reste actif, devient passif, ou se déclare leader. (Les conditions seront précisées dans la présentation.) S'il reste actif, il répète le comportement ci-dessus.
- Un nœud *passif* transmet simplement tous les messages reçus de gauche vers son voisin à droit. En plus, si le message déclare un leader, il affiche un message correspondant à l'écran.
- Il y a trois types de messages échangés par les nœuds, tous dans le format (*type*,*identifiant*).
  - "voisin" (v) : pour envoyer son identifiant vers le voisin à droit.
  - "prochain" (p) : un nœud qui a reçu l'identifiant de son voisin à gauche l'envoie vers son voisin à droit.
  - "gagnant" (g) : un nœud se déclare leader.

Essayez le protocole pour des différente valeurs de n. Observez si votre programme termine toujours et s'il y a toujours un seul leader.

Précisions : Vous devez modifier la fonction `node` et la compléter avec les réactions aux trois types de messages. La fonction `send_message` vous permet d'envoyer des messages.

2 Serveur tchat

Notre objectif sera de créer un simple serveur tchat. La page web contient un client et un serveur incomplet.

Le client (déjà complété) se connecte au serveur, puis il envoye tout ce que tape l'utilisateur vers le serveur, et il affiche tout ce qu'il reçoit de la part du serveur. Le client prend deux arguments facultatifs sur la ligne de commande pour spécifier la machine et le port (par défaut : `localhost` et 4004).

Le serveur est la partie à compléter. Vous pouvez utilisez les fonctions dans `network.h` pour établir le serveur sur son port. (Attention, au démarrage il se peut que le port 4004, utilisé par défaut, est toujours bloqué par un test préalable ; dans ce cas on peut changer le port à utiliser sur la ligne de commande.)

1. Créez un serveur simple que attend deux clients et les laisse échanger des messages.
2. Rajoutez une possibilité pour les client d'adopter un nom ; lorsque le client envoie `NICK <nom>`, ce nom sera antéposé à ses messages suivants envoyés aux clients.
3. Améliorez le serveur tel qu'il n'attend pas un nombre fixe de clients ; des clients peuvent se connecter à n'importe quel instant, et le serveur gère une liste de clients dynamiquement.