TP1 : Architecture client/serveur

1 L'horloge parlante – UDP

Implémenter une application client/serveur, où dès que le serveur reçoit un datagramme UDP, il répond au client l'heure actuelle.

2 Vers un serveur de temps

Le protocole NTP (Network Time Protocol) est utilisé pour synchroniser les horloges de différentes machines. Il fait intervenir une machine (client) et un serveur de temps (relié à une horloge précise). Une version simplifiée de ce protocole suit les interactions suivantes :

- Le client envoie un datagramme UDP contenant son heure actuelle T_1.
- Quand le serveur reçoit un datagramme, il note l'heure T_1' de réception, et renvoie un paquet contenant T_1, T_1' et l'heure d'envoi T_2' de ce paquet.
- Quand le client reçoit ce datagramme, il note l'heure T_2 de réception.
- Le client estime alors le délai de transmission est $\delta = T_2 - T_1 - (T_2' - T_1')$ et l'écart entre les horloges $\theta = \frac{T_1'+T_2'}{2} - \frac{T_1+T_2}{2}$, et peut alors réajuster son horloge en lui ajoutant θ.

Implémenter une application client/serveur ayant le comportement décrit ci-dessus (le client affichera δ et θ sur sa sortie standard). Justifier l'utilisation d'UDP.

3 L'horloge parlante – TCP

Implémenter une application client/serveur, où dès qu'une socket TCP est ouverte, le serveur envoie l'heure actuelle au client, puis ferme la socket.

4 L'horloge parlante++ – TCP

On souhaite enrichir le protocole précédent pour faire en sorte que le client puisse demander la date, l'heure ou les deux (en envoyant `DATE`, `HOUR` ou `FULL`), et que la connexion reste active tant que le client n'en demande pas la fermeture (en envoyant le message `CLOSE`). Implémenter une application remplissant ces spécifications.

5 Capitalisation

Implémenter une application client/serveur, où le serveur met en capitales puis renvoie chaque ligne de texte écrite sur l'entrée standard du client. (Le client affichera sur sa sortie standard les réponses du serveur).

6 Puissance 4

Le but de cet exercice est d'implémenter une application client/serveur permettant de jouer au morpion en réseau. Le serveur doit gérer la connexion de deux clients, tirer au hasard qui commence, recevoir les coups des joueurs, tester s'ils sont valides, et informer les clients de l'évolution du jeu (quel coup a été joué par leur adversaire, qui a gagné, etc). Les clients, eux, doivent interpréter les messages du serveur afin de mettre à jour l'interface avec l'utilisateur.

1. Créer une classe Jeu, qui permet de retenir l'état du jeu et de tester si un coup est valide.
2. Décrire (sur papier) le protocole de communication qui sera utilisé entre le client et le serveur.
3. Créer une classe Joueur (représentant un Thread du serveur), qui gère la communication avec un client.
4. Finaliser la classe Serveur : son main va créer une instance de Jeu, et deux threads Joueur. *Sous linux et mac, vous pouvez tester le serveur en utilisant* `nc <adresse> <port>`.
5. Implémenter une classe Client permettant d'interfacer l'utilisateur et le serveur, afin que l'utilisateur puisse jouer sans connaître le protocole de communication avec le serveur. *Vous pouvez, si vous le souhaitez, programmer une jolie interface graphique, mais ce n'est pas le but de l'exercice. Jouer dans un terminal suffira largement.*