PERLINSKI Eric
VINCENT Kevin

Projet Réseaux

1) Protocole Applicatif

1.1) Description

L'application qui nous a été demandé de concevoir réalise le schéma suivant :

- Le serveur est attente d'un client
- Le client se connecte au serveur
- Il demande au serveur de participer à la conférence
- Une fois que le serveur a accepté, ce premier client à le jeton par défaut
- Le prochain client qui demande de participer est automatiquement en attente de celui qui a le jeton
- Une fois que le client qui a le jeton le libère il envoie un message au serveur qui alloue le jeton au client suivant
- Un client peut se déconnecter quand il veut soit par un message, soit par un CTRL-C mais dans le premier cas il doit attendre la réponse de la part du serveur

Pour réaliser cette application, nous avons implémenté la méthode multicast/multipoint et de ce fait le serveur ne parle à un client que pour lui distribuer le jeton afin de lui donner la parole. De ce fait, le client qui a le jeton parle directement sur l'adresse du groupe sans utiliser le serveur.

1.2) Type et format des messages

Voici ci-dessous la liste des messages qu'utilise le client pour parler au serveur :

- « p » le client demande au serveur de participer à la conférence
- « l » le client décide de laisser le jeton
- « d » le client décide de se déconnecter du serveur

D'autres messages circulent dans les sockets de dialogues entre le client et le serveur.
Lorsqu'un serveur donne le jeton à un client il envoie le mot « jeton ». De ce fait, le client qui reçoit ce mot sait qu'il a maintenant le droit de parler. Lorsqu'un client laisse le jeton il envoie un dernier message à tous les autres clients afin qu'il sorte de la phase écoute et ce message est «Token transmitted to a new client».

2) Choix d'Implémentation

Pour la partie « serveur », «nous avons décider d'implémenter un serveur multi-client TCP à l'aide de la fonction « select ». Au départ, nous voulions utiliser « fork » afin de donner à chaque client son processus mais cela commencer à devenir compliqué lorsqu'il fallait entretenir un tableau à jour contenant la socket de dialogue de chaque client.

Pour la partie « client », il nous a fallu créer trois sockets, une pour discuter avec le serveur, une autre pour émettre des messages multicast aux autres clients et enfin une dernière pour les clients qui recevait ces messages. De ce fait, lorsqu'un client est en attente de message avec la fonction

« select », il n'attend que le message « fin » de la part du client émetteur pour savoir si oui ou non c'est lui qui a le jeton au prochain tour de boucle. Si c'est effectivement lui qui le possède alors il change de statut et utilise la socket de l'émetteur pour commencer à émettre des message multicast, si non il continue d'écouter la socket du récepteur. Il n'utilise la socket du serveur que pour renseigner le serveur qu'il laisse le jeton. Le serveur à son tour cherche dans le tableau contenant tous les clients le prochain client qui aura le jeton puis envoie le message « jeton » à ce client.