ESIREM - 3ème année
ITC7-2 (1 séance)
Étude d'une communication entre deux hôtes du réseau

Michael Choisnard, Arnaud Da Costa, Benoît Darties

Mars 2010

L'objectif de ce TP est de développer et mettre en place une application type client / serveur écrite en C, et de visualiser les différents éléments constituant la communication entre ces derniers.

1 Développement et étude d'une application client / serveur en mode non connecté (UDP)

1.1 Rappels sur la communication non connectée (ex UDP)

La communication entre deux processus distants s'effectue selon un mode de communication client / serveur ; ce sont les clients qui initient des dialogues à destination du serveur. Le mode non connecté est un mode de communication simplifié reprenant les éléments suivants :

- chaque processus client ou serveur possède une socket dite de communication ;
- un serveur se contentera de lire sur sa socket les informations qu'il reçoit ;
- le client écrira des informations sur sa socket et ces dernières sont acheminées vers la socket du serveur.

Le client doit connaître l'adresse du serveur, plus précisément l'adresse de la socket du serveur. Cette adresse est l'association d'une adresse ip et d'un port de communication, généralement appelé "port d'écoute". Notons qu'un serveur doit, lorsqu'il reçoit un message, être capable d'identifier le processus client qui lui a envoyé ce dernier. Ce dernier point sera abordé plus tard. Pour arriver à une telle configuration chaque entité procède comme suit :

Au niveau du serveur :

1. Création d'une socket de communication ;
2. Attachement de la socket à une adresse (ip + port). Cette adresse identifie la socket auprès des clients ;
3. Lecture en boucle sur la socket.

Au niveau du client :

1. Création d'une socket de communication ;
2. Envoi d'un message sur la socket vers l'adresse de la socket du serveur. [1]

1. Nous verrons par la suite que certains éléments sont passés sous silence, notamment sur l'attribution d'une adresse à la socket du client

1.2 Utilisation d'un serveur et d'un client UDP

Nous allons développer un client et serveur UDP fonctionnant comme suit :

- Le rôle du serveur est d'écouter en continu sur une socket de communication, et d'afficher tout message reçu. Son exécution nécessite un paramètre : le port de communication sur lequel écouter les messages entrants.
- Le rôle du client est d'envoyer un message vers le serveur. Son exécution nécessite exactement trois paramètres entrés dans cet ordre :
  1. l'adresse ip du serveur,
  2. le numéro de port,
  3. le message à envoyer, encadré par des guillemets "...".

1. Téléchargez puis décompressez l'archive contenant le code du serveur et du client UDP disponible sur la page InfoTronique 3A/ ITC38 de `http://informatique.esirem.fr`
2. Parcourez le code de chaque programme et répondez aux questions suivantes :
   (a) La fonction **socket()** crée un descripteur de socket. Quels sont ses paramètres ?
   (b) A quoi servent les structures de type **sockaddr_in** ?
   (c) Quelles sont les fonctions C utilisées respectivement pour l'envoi et la réception de messages en UDP ?
3. Compilez le serveur sous le nom **UDP_serveur** et lancez-le sur une machine du réseau avec comme paramètre de port 4000
4. À l'aide de la commande **ifconfig**, déterminer l'adresse ip du serveur.
5. Compilez le client sous le nom **UDP_client** et lancez-le sur une autre machine du réseau [2] avec comme paramètres l'adresse ip du serveur, le port 4000 et le message de votre choix.
6. Vérifiez que le serveur UDP reçoit bien le message envoyé, que l'adresse de l'expéditeur affichée correspond bien à l'adresse ip du client.

1.3 Ports de communication associés au client / serveur

1. Avec le client, envoyez successivement le même message au serveur, et observez l'affichage sur le serveur. Quelle donnée change constamment ?
   explication : la socket client possède aussi une adresse, composée d'une ip et d'un port. L'adresse ip est celle de la machine exécutant le client. Mais le port n'a pas été défini, car a aucun moment dans le code du client nous n'avons associé d'adresse à la socket de communication (ce qui aurait pu être fait avec la fonction **bind()**). De ce fait, le système d'exploitation associe à chaque exécution du client un numéro aléatoire de port à la socket. Ce numéro de port est communiqué dans les en-têtes du message qui est envoyé au serveur. Si jamais le client se mettait en attente de réception une fois son message envoyé, le serveur pourrait répondre au client en utilisant le port mentionné dans le message entrant.
2. Sur la machine serveur, lancez dans un autre terminal un second serveur avec le port 3000, puis vérifiez avec le client que vous parvenez bien à communiquer tantôt avec l'un des serveurs, tantôt avec l'autre en faisant varier le numéro de port d'envoi.
3. Essayez alors de lancer un 3eme serveur sur le port **4000**. Que se passe-t'il et pourquoi ?
4. La commande netstat vous permet de vérifier quels sont les ports actuellement utilisés par les différentes applications. Exécutez la commande suivante :
   netstat -a -u -n
   pour afficher toutes les sockets de type **udp** dans un format numérique, et vérifiez que les adresses locales *** :4000** et *** :3000** sont présentes dans la table résultat. Arrêtez le serveur situé sur le port **3000** (avec la combinaison de touche ctrl + c dans le terminal exécutant le programme) et vérifiez que l'entrée *** :3000** a bien disparu de la table.

2. Pour ceux qui douteraient de l'intérêt de lancer le client sur une autre machine, sachez que la communication client serveur lorsque ces derniers sont sur la même machine utilise la boucle locale et non l'interface ethernet. Ces deux interfaces possèdent un MTU (Maximum Transfert Unit) différent. Utiliser la même machine ne vous donnera pas les bons résultats

1.4 Visualisation du contenu d'un packet sous Wireshark

Dans cette sous-section, nous allons effectuer une application directe du TD4 sur l'analyse de trames au moyen de l'outil **Wireshark**. **Wireshark** est un utilitaire d'analyse de protocole (sniffer) open-source permettant de voir l'ensemble des messages transitant sur le réseau.

1.4.1 Premières captures

1. Lancez l'utilitaire **Wireshark** sur le poste client.
2. Afin de ne visualiser que le trafic entre le serveur et le client, appliquez sur **Wireshark** le filtre suivant : **ip.addr** == ***ip_client*** **&&** **ip.addr** == ***ip_serveur***
3. Lancer une capture sur **Wireshark**, puis envoyez quelques messages du client vers le serveur et observez le résultat sur **Wireshark**
4. Sélectionnez un datagramme **UDP** et répondez aux questions suivantes :
   (a) Sur quelle couche du modèle TCP/IP se situe l'adresse IP du destinataire ?
   (b) Quelle est la taille d'un en-tête UDP ?
   (c) Dans le protocole IP, quelle valeur hexadécimale désigne le protocole UDP pour le datagramme encapsulé ?

1.4.2 Taille maximale d'un datagramme, fragmentation IP

Une manière de déterminer la taille maximum des données qui peuvent être contenu dans un paquet (sans fragmentation) consiste à incrémenter à chaque itération la taille du message à envoyer. Pour cela nous utiliserons la ligne de commandes suivante : **ITER=*valeur* ; for ((i=0 ; i < $ITER ; i++)) do echo -n 'A' ; done ;**

1. exécutez plusieurs fois la ligne de commande précédente en faisant varier la variable *valeur* affectée à ITER. Que fait cette ligne de commande ?
2. pour faire varier la taille du message envoyé vers le serveur UDP, on utilisera la ligne de commande suivante :
   ./UDP_client *ip_serveur* 4000 "`ITER=*valeur* ; for ((i=0 ; i < $ITER ; i++)) do echo -n 'A' ; done ;`". (attention, les apostrophes accolées aux guillemets sont des anti-apostrophes). Testez plusieurs envois en faisant varier la valeur affectée à **ITER**.
3. Lorsque l'on souhaite envoyer 1 octet de données, combien d'octets sont réellement envoyés ? même question avec 2 octets et 1000 octets ?
4. Quelle est la taille maximal des données que l'on peut envoyer en un seul datagramme ? Que se passe-t'il si l'on essaye d'envoyer plus de données depuis le client ? Quelle est selon vous la taille maximum d'un paquet IP, et combien de données peut-il contenir ? En exécutant la commande **ifconfig**, essayez de retrouver la donnée correspondant à taille maximum de ce paquet. Comment s'appelle cette information ? La taille du segment IP est-elle limitée par le protocole IP en lui-même ou par le protocole Ethernet qui encapsule le paquet ?
5. Envoyez un datagramme UDP de taille bien supérieure au MTU, observez la fragmentation IP ainsi que les en-têtes de ces paquets. Listez les éléments permettant au serveur de déterminer que les paquets IPs contiennent chacun une partie d'un même message. Quel champ permet notamment de remettre ces paquets dans l'ordre ?

2 Identification du protocole implémentant le "ping"

Un outil d'analyse réseau tel que **Wireshark** permet de visualiser à quelle couche de l'architecture TCP/IP se situent de nombreux services et comment ces derniers opèrent. Nous prendrons l'exemple de la commande **ping**. Cette dernière permet de savoir si un hôte distant est actif ou non, a partir de son adresse ip.

1. Sans changer les filtres appliqués sur **Wireshark**, exécutez la commande suivante :
   ping *ip_ serveur* (en remplaçant ***ip_ serveur*** par l'adresse ip du serveur en notation pointée)
   depuis le client et observez la communication.
2. Quel protocole est à l'origine de l'implémentation du ping, et comment ce dernier est encapsulé ?
3. Quelle est la taille des en-têtes de ce protocole ?
4. En cherchant dans votre cours, déterminez la liste des différentes valeurs que peut prendre le champ "Type" de ce protocole.

3 Développement et étude d'une application client / serveur en mode connecté (type TCP/IP)

La section suivante reprend les éléments présentés dans la section 1 en mode connecté

3.1 Rappels sur la communication connectée (ex TCP)

Le mode connecté est un mode de communication dans lequel on garantit l'acheminement des messages, ce qui suppose une certaine synchronisation. L'implémentation de ce protocole est plus compliquée que pour une communication non connectée. Les sockets de communication sont ici encore identifiées par une adresse ip et un port de communication. Le fonctionnement du serveur et du client est résumé comme suit :

Au niveau du serveur :

1. Création d'une socket principale ;
2. Attachement de la socket à une adresse (ip + port). Cette adresse identifie la socket auprès des clients ;
3. Ecoute de la socket principale.
4. A chaque nouvelle connexion :
   - Création d'une socket de travail dédiée à la connexion
   - Lecture et affichage du message reçu sur cette socket
   - Fermeture de la socket de travail

Au niveau du client :

1. Création d'une socket de communication ;
2. Envoi d'un message sur la socket vers l'adresse de la socket du serveur.

3.2 Utilisation d'un serveur et d'un client TCP

Nous utilisons ici un client serveur dont l'utilisation est identique à celle présentée dans la section 1, mais la communication est désormais en mode connecté.

1. Téléchargez puis décompressez l'archive contenant le code du serveur et du client TCPdisponible sur la page InfoTronique 3A/ ITC38 de `http://informatique.esirem.fr`
2. Parcourez le code de chaque programme et répondez aux questions suivantes :
   (a) Quels paramètres de la fonction **socket()** diffèrent entre les versions TCP et UDP ?
   (b) Quelles sont les fonctions C utilisées respectivement pour l'envoi et la réception de messages en TCP ?

3. Compilez le serveur sous le nom **TCP_serveur** et lancez-le sur une machine du réseau avec comme paramètre de port 4000
4. Compilez-le client sous le nom **TCP_client** et lancez-le sur une autre machine du réseau avec comme paramètres l'adresse ip du serveur, le port 4000 et le message de votre choix.
5. Vérifiez que le serveur TCP reçoit bien le message envoyé, que l'adresse de l'expéditeur affichée correspond bien à l'adresse ip du client.
6. A l'aide de **Wireshark**, identifiez l'ensemble des paquets échangés lors d'une session TCP, la taille des paquets, des en-têtes. Illustrez enfin sur un schéma l'ensemble des paquets échangés pour l'envoi d'un simple message du client vers le serveur, en faisant figurer sur chaque message le type de paquet (flag), le numéro de séquence et le numéro d'acquittement.
7. À quoi correspond le champ "window size" ?