Licence L3 Informatique
HLIN503 -Réseaux

2015-2016

1 Un client/serveur UDP

Cette partie permet de mettre en place un clien /t serveur en UDP. La partie serveur vous sera fourni. La partie client sera à programmer, l'exercice qui suit permet de se poser les bonnes questions avant de coder.

Exercice 1

Un serveur jouant au « perroquet » tourne sur l'hôte `io.info-ufr.univ-montp2.fr`. En cas de conflit son code exécutable se trouve aussi dans le répertoire commun habituel, `/commun/info/L3/Reseau` et on peut le lancer sur tout hôte. Il ne fait qu'attendre des messages dans la boîte réseau numéro 31469, sous le protocole UDP, et renvoie le texte reçu à l'expéditeur.

On veut écrire un client qui expédie un message et attend la réponse (l'écho en quelque sorte).

1. Quelles sont les informations nécessaires à cette réalisation en plus des informations données ci-dessus ?
2. Est-on sûr de recevoir la réponse ?
3. Lors d'une réception, est-on sûr de recevoir tout le message ? (préciser si c'est en une seule réception).
4. Noter que le client ne doit pas se contenter d'expédier un seul message, mais doit envoyer en boucle chaque ligne entrée par l'utilisateur, jusqu'à une demande d'arrêt.

2 Communications UDP entre deux applications

Exercice 2

L'objectif de cet exercice est de mettre en évidence toutes les caractéristiques d'une communication en mode *non connecté* : synchronisations, tampons, spécificités des entrées-sorties (blocage), etc.

Soient `monclient` et `monserveur` deux applications (deux processus) sur deux hôtes distincts quelconques. Ces hôtes ne sont pas figés dans les applications (nom passé en paramètre), car on prévoit de lancer ces applications en plusieurs exemplaires sur plusieurs hôtes.

Première étape

On commencera par la mise en place d'une communication échangeant un seul message texte (chaque application envoie un message à l'autre).

On peut supposer le schéma algorithmique de base suivant pour `monclient` :

```
entier descLocal= allouerBRlocale (nomHôte, numLocal, proto) ;
initialiser BRdistante (nomDistant, numDistant, proto);
initialiser longueurBRdist;
initialiser tamponExp, longueurExp ;
initialiser tamponRec, longueurRec;
expédier (descLocal, tamponExp, longueurExp, 0, BRdistante, longueurBRdist);
recevoir (descLocal, tamponRec, longueurRec, 0, NULL, NULL);
```

On peut supposer un schéma algorithmique semblable pour `monserveur`, en inversant l'ordre d'expédition et réception.

1. Que se passe-t-il si on lance une seule des deux applications ? Jusqu'où se déroule-t-elle sans encombre ?

2. Que se passe-t-il si une application fait un envoi sans que l'autre soit en réception ? Pour illuster la réponse à cette question, il faut compléter et modifier le schéma algorithmique précédent. En effet, il faut s'assurer que l'application censée recevoir soit lancée, tout en étant sûr qu'elle n'est pas en réception. Proposer une solution répondant à ce besoin.
3. Réciproquement, est-ce qu'une application peut être en attente de réception sans qu'un expéditeur n'ait créé la boîte réseau assurant le service d'expédition [1] ?
4. Que se passe-t-il si on se trompe dans l'affectation des numéros des *boîtes réseaux* ?
5. Que se passe-t-il si on lance deux applications identiques, pour le même service sur un même hôte ? sur deux hôtes différents ?
6. Est-il possible d'envisager une communication entre une application qui vous appartient et une autre qui ne vous appartient pas ? On envisagera ici les deux cas :
   — l'autre personne lance l'application écrite par vous,
   — l'autre personne lance sa propre application.
   Quels sont les éléments sur lesquels il faut se mettre d'accord afin que deux applications écrites par deux personnes différentes puissent communiquer [2] ?

Deuxième étape

On veut généraliser ces exemples pour mettre en évidence les pertes de paquets alors que les deux applications sont lancées, c'est-à-dire que si on perd des paquets, ce n'est pas parce que le destinataire ou la boîte de destination n'existe pas.

1. Quels exemples peut-on proposer ? On peut proposer ici des exemples qui pourront facilement être mis en œuvre en TP, et d'autres cas possibles, qu'on ne pourra pas mettre en évidence.
2. Quelles sont les modifications à apporter aux programmes afin de mettre en évidence une telle perte ?
3. Est-il possible de créer un exemple d'interblocage des deux applications (au besoin en modifiant les programmes) ?
4. Est-il possible alors de les débloquer par des moyens autres que l'arrêt pur et simple ?

Troisième étape

On veut qu'une seule application ait une adresse de boîte réseau connue (le numéro de service est connu) par l'autre. Par exemple `monclient` est lancée avec un numéro connu par `monserveur`, mais cette dernière se fait attribuer un numéro quelconque par le système d'exploitation local. Que proposez-vous pour la mise en place d'un dialogue sans heurts ?

3 Circulation d'un paquet UDP

Exercice 3

Un paquet UDP part de la boîte réseau numéro 1025 (on peut dire aussi du port source UDP numéro 1025, de l'hôte 197.198.199.200 à destination de la boîte réseau numéro 1026 de l'hôte 205.206.207.208.

Les réseaux auxquels appartiennent ces deux hôtes sont séparés par deux routeurs. Ce sont deux réseaux de classe C, sans sous-adressage ni sur-adressage.

1. Choisir des adresses réseau (adresses IP) plausibles pour les deux routeurs. De combien d'adresses doit disposer au minimum chaque routeur ?
2. Esquisser avec des schémas simples la circulation de ce paquet vue des couches transport et réseau, en supposant qu'aucun découpage n'est effectué sur le paquet.
3. Quelles informations manquent pour illustrer la circulation de ce paquet dans la(les) couche(s) liaison de données ?

1. c'est un peu une question piège.
2. est-ce une question piège ?

4 Client/Serveur en TCP

Exercice 4

On veut créer un exemple de deux programmes permettant d'engendrer des processus communiquant par le protocole TCP, selon le modèle *Client - Serveur*. On va créer ces deux programmes étape par étape. La base de travail sera les classes `C++` distribuées précedemment et à votre disposition.

Objectif : Le Client `clientTCP` expédie un message contenant le numéro d'utilisateur ayant lancé ce processus client (voir l'appel système `getuid()`). Le serveur `serverTCP` lui répondra *Tiens, bonjour* suivi du nom d'utilisateur correspondant au numéro transmis (voir l'appel système vu en cours de *systèmes d'exploitation*, `getpwuid()`).

Le serveur traite les requêtes une à une, c'est-à-dire qu'on répond complètement à un client, on ferme la connexion puis on passe au client suivant.

4.1 Serveur - `serverTCP`

1. Que doit faire le serveur pour mettre en place ce que nous avons appelé une `Boîte Réseau` publique ? En quelque sorte :
   — Que faut-il décider pour l'allocation de la boîte publique ?
   — quel constructeur va être utilisé pour la mise en place ?
2. Doit-on prévoir dans le programme sur quel hôte on va faire tourner ce serveur ?
3. Pourra-t-on lancer plusieurs serveurs sur un même hôte ?
4. Pourra-t-on lancer plusieurs serveurs sur des hôtes différents ?
5. Si on lance le serveur, jusqu'à quelle étape doit-il se dérouler sans être bloqué ?

4.2 Client - `clientTCP`

1. Quelles informations sont nécessaires au client pour joindre le serveur ?
2. Si l'utilisateur se trompe dans le nom de l'hôte où le serveur est prévu, que se passe-t-il ?
3. En supposant que le serveur a été lancé, jusqu'à quelle étape est-ce que le client se déroule sans être bloqué ?
4. Peut-on lancer plusieurs clients sur un même hôte (en quelque sorte, si on lance plusieurs processus clients, que se passe-t-il pour chacun d'eux ?)
5. Peut-on lancer plusieurs processus clients sur plusieurs hôtes ?

4.3 La communication

1. Que se passe-t-il du côté du client, si le serveur ferme le circuit virtuel sans envoyer de message au client ?
2. Même question du côté du serveur si le client ferme le circuit sans envoyer de message au serveur ?
3. Décrire comment le client peut dépasser l'expédition de la requête, et rester bloqué ensuite (par la volonté du serveur).
4. Est-ce que le serveur peut annoncer de quel hôte et quelle boîte est venue la requête du client *avant* d'accepter la demande de connexion ?
5. En fin de compte, quels sont les moyens qui sont à la disposition d'un processus serveur pour refuser des clients ?

Exercice 5

Nous avons vu que le protocole TCP était orienté *flot de caractères*. Il n'y a pas de limite de message qui soit assurée par le protocole. Du coup, il n'y a pas de correspondance entre le nombre d'envois (écritures) et le nombre de réceptions (lectures) pour un *message*.

Est-ce que les processus que vous avez prévus tiennent compte de cette caractéristique ?

Enfin, cet exemple de transmission de l'`uid` nous permet de mettre en œuvre une cummunication d'informations autres que des chaînes de caractères. Mais quelles sont les limites du(des) réseau(x) pour un fonctionnement cohérent d'une telle application ?

5 Appels système pratiques

Exercice 6

— L'appel `getsockname()`, déjà vu dans un TD précédent, permet de récupérer l'adresse de la socket (boîte réseau) dont le descripteur est `s`.

```
int  getsockname(int   s, struct sockaddr * name, int * namelen )
```

— L'appel `getpeername()` permet de récupérer l'adresse de la socket connectée à la socket `s`.

```
 int  getpeername(int  s, struct sockaddr *name, int *namelen);
```

1. Un client en mode connecté peut laisser l'appel `connect()` compléter la demande d'allocation de sa boîte réseau. Écrire un programme permettant d'afficher le numéro de port réellement obtenu après avoir fait la demande `connect()`. On pourra afficher le numéro avant cette demande, pour constater que l'appel système a fait son travail.
2. Afficher sur un serveur en mode connecté les caractéristiques du client demandant une connexion.

6 Un serveur concurrent et un problème de performances

Exercice 7

On veut mettre en place un serveur de fichiers, à la manière de `ftp`, afin de faire des mesures et comparer ses performances aux outils classiques de transfert de fichiers (`ftp` ou ce même protocole sous un navigateur par exemple).

On va écrire un serveur et un client pour cette application. Le client ne peut que demander un fichier et le serveur le lui expédie lorsqu'il existe, ou lui annonce son inexistence : on ne prévoit pas d'autres services que le transfert de fichiers. Bien sûr plusieurs clients sont possibles et peuvent travailler simultanément. Chaque client peut demander plusieurs fichiers.

1. Rappler pourquoi on opte pour un serveur concurrent.
2. Proposer un protocole d'application entre serveur et client.
3. Écrire l'algorithme et préparer le programme pour les TP.
4. Est-ce que le client peut savoir s'il a reçu un fichier intégralement (est-ce prévu dans votre protocole) ?
5. Si on veut limiter le nombre de clients simultanés, quelle solution peut-on proposer ?

7 Problème d'héritage (adaptation de questions d'examens)

Exercice 8

Un fonctionnement classique d'un serveur concurrent consiste à créer un clône lors de chaque demande de service. On suppose ici que la communication se fait en mode connecté. Comme pour les fichiers, les descripteurs de *sockets*, c'est-à-dire pour nous *descripteurs des boîtes réseau*, sont dupliqués. Le fonctionnement des réceptions et expéditions reste aussi identique à celui des entrées-sorties classiques sur les fichiers (rappel : les entrées-sorties réalisées à partir du processus parent ou descendant s'influencent mutuellement pour tous les fichiers ouverts par le parent avant clônage).

Or, l'obtention d'une boîte réseau permet à un processus de s'assurer l'exclusivité d'accès à cette boîte : nous avons vu en TP qu'une erreur du type *adresse déjà utilisée* est signalée lorsqu'on demande l'allocation d'une boîte déjà allouée. Il n'empêche qu'après le clônage, on se retrouve avec deux processus ayant chacun un descripteur pointant sur la même boîte.

1. Si aucun des deux processus ne ferme son descripteur, peut-on déduire que c'est une méthode pour partager une boîte commune ? Est-ce que ce raisonnement reste valable que la boîte soit publique ou privée ?
2. Quelle que soit votre réponse à la question ci-dessus, supposons que la possibilité de partager une boîte aux lettres entre plusieurs processus existe. Donner un exemple dans lequel un tel partage peut être utile et un contre-exemple (partage néfaste).

3. Si un des deux processus décide de fermer son descripteur, supposons qu'un message arrive après la création du clône et avant la fermeture du descripteur. Y-a-t-il un risque de perte pour ce message ?

8 Blocage des entrées-sorties

Exercice 9

On veut faire le point sur le blocage des entrées-sorties dans les réseaux.

1. Faire la liste de tous les appels système utilisés jusque là pour les communications dans les réseaux que ce soit en mode connecté ou non. Voir les documents de cours et ne pas confondre ces appels avec les méthodes offertes dans les classes (*sock* et *sockdist*) distribuées en cours.
2. Pour chaque appel **bloquant**, préciser le type d'événement qui le réveille.
3. Proposer une solution permettant dans vos programmes de mettre en évidence cette caractéristique pour chacun des appels bloquants. En particulier, que proposez-vous pour l'appel `connect()` ?

9 Multiplexage

Exercice 10

On considère ici une application de type *vente aux enchères électronique* immédiate, c'est-à-dire sans délai *long* :
— Une application *serveur* propose des objets et prend en compte les enchères faites par des utilisateurs connectés ;
— tant que les enchères montent, cette application annonce à tous les utilisateurs l'annonce la plus haute en cours ;
— un délai maximal de 3 secondes permet aux utilisateurs de surenchérir ; sinon, l'objet est affecté à l'utilisateur ayant proposé la plus haute enchère ;
— des utilisateurs peuvent se connecter ou se déconnecter à tout instant.

1. Constater que des entrées-sorties non bloquantes ou multiplexées sont nécessaires au niveau du *serveur*.
2. Proposer une solution permettant de gérer correctement ces entrées-sorties.
3. Est-il nécessaire de multiplexer les entrées-sorties des clients ?
4. Dans tous les cas, vaut-il mieux privilégier le multiplexage ou les entrée-sorties non-bloquantes ?
5. Lorsqu'un utilisateur se déconnecte, envisager les deux situations suivantes :
   — il n'a pas fait l'enchère la plus haute ;
   — il a fait l'enchère la plus haute, mais la déconnexion entraîne l'annulation de son enchère.
   Étudier ces deux cas, et proposer une solution en particulier dans le deuxième cas ; attention, il est utile ici d'analyser l'ensemble de ce que le serveur peut recevoir simultanément.
6. Écrire les schémas algorithmiques des deux applications.

10 Notion de masque : adressage, sous-adressage et sur-adressage

Un petit rappel de Cours :

Un des problèmes clefs posés lors de l'acheminement d'un datagramme à partir d'un hôte H donné est : *est-ce que le destinataire du datagramme est connecté au même réseau physique (ou à un des réseaux physiques) au(x)quel(s) je suis moi-même (H) connecté ?*, ou en termes de la table de routage *est-ce que le contact avec le dsestinataire est direct, sans routeur intermédiaire ?*

La difficulté vient du fait qu'il s'agit d'une communication physique et qu'on se pose la question en ne disposant que de l'adresse réseau. En effet, le routage doit être résolu au niveau de la couche réseau.

Il faut donc organiser l'affectation des adresses du niveau *réseau* (adresses IP ici) en liaison avec l'organisation physique dudit réseau. Cette affectation est d'autant plus importante que l'étendue du réseau dont on dispose (donc

le volume d'adressage) est grande : on peut admettre qu'un réseau de type « classe A ou B » ne sera pas organisé en un seul grand réseau (bien que ce soit logiquement possible), mais en plusieurs sous-réseaux interconnectés.

Ceci se fait en définissant des masques de réseaux. Un masque est une donnée de la taille d'une adresse, permettant de calculer l'adresse globale du *réseau* à partir d'une adresse quelconque d'hôte (on peut dire aussi qu'on extrait la souche réseaux de l'adresse de l'hôte). L'opération effectuée est :

adresse réseau = (adresse hôte) **et** (masque)

Exercice 11

hôte	adresse	masque
départ H_1	194.195.196.197	255.255.255.0
destination H_2	194.195.196.206	255.255.255.0

Résultat : H_1 et H_2 sont sur le même réseau (vu de H_1).

hôte	adresse	masque
départ H_3	130.160.21.22	255.255.255.0
destination H_4	130.160.140.22	255.255.255.0

Résultat : H_3 et H_4 ne sont pas sur le même réseau(vu de H_3).

1. À quelle classe de réseau appartiennet les adresses des exemples ci-dessus ? Quelles sont les bornes des adresses allouées aux hôtes avec ces adresses ?
2. Vérifier en traduisant en binaire que les exemples sont corrects. Indiquer dans chaque cas quelles sont les bornes des adresses pour lesquelles on obtiendra une réponse négative au test « sommes nous connectés au même réseau ? ».
3. Plus difficile : Que se passe-t-il dans le premier exemplesi le masque est 255.255.255.192 ? Et si le masque est 255.255.254.0 ?
4. Pour spécifier dans la toute première phrase l'appartenance **à un ou à des réseaux physiques**, citer un exemple dans chaque cas.

Exercice 12

Pour les questions suivantes, il est conseillé de travailler en binaire et de passer à la représentation décimale à la fin.

1. On pourra se servir du second exemple pour répondre à cette partie. Dans un réseau de classe B on veut créer des sous-réseaux permettant de voir le réseau global comme un ensemble de réseaux de classe C. Combien de sous-réseaux peut on déterminer ? Quel est le nombre maximal d'hôtes possible dans chaque sous-réseau ? Comment sont représentées les adresses « réseau » et « tous » ? Quel masque faut-il utiliser (dans chaque sous-réseau) ?
2. On veut diviser un réseau de classe C en huit sous-réseaux. Proposer une solution ; donner un exemple en précisant la capacité d'adressage de chaque sous-réseau, les masques ainsi que les adresses réservées (« ce réseau » et « tous »).
3. Peut-on faire une division en six sous-réseaux, quatre de 30 hôtes chacun et deux autres de 62 chacun ?
4. On veut maintenant avoir une division en trois sous-réseaux. Quelle**s** solution**s** peut-on proposer ?

Exercice 13

On veut créer un réseau physique unique composé de plus de 254 et moins de 508 machines. Deux adresses de type « classe C » sont affectées.

1. Donner un exemple d'adresses consécutives permettant de réaliser ce réseau ; préciser comme ci-dessus la capacité, les masques et adresses spécifiques
2. Profiter pour corriger légèrement cet énoncé.
3. Donner un exemple de deux adresses de classe C consécutives, non compatibles pour former un seul réseau.
4. Est-il nécessaire de limiter les masques à une suite consécutive de bits à 1 et une suite consécutive de bits à 0 ?

11 Transformations subies par un message

Exercice 14

On s'intéresse à l'évolution d'un message partant d'une application sur un hôte M_1 jusqu'à son arrivée sur l'hôte destinataire M_2. La notion de message s'applique à l'entité échangée vue par les applications, par exemple un fichier lorsqu'il s'agit d'applications de transfert de fichiers, une page lorsqu'il s'agit de W3, etc.

On a déjà vu les divers emballages et déballages subis par un paquet traversant plusieurs réseaux physiques. Cette fois-ci on veut étudier les interfaces entre couches ainsi que les découpages et réassemblages. Le protocole utilisé est `TCP` et les ports (les numéros de boîtes à lettres) attribués par les systèmes respectifs sont 2048 sur M_1 et 4096 sur M_2.

Supposons que la taille du *message* soit supérieure à la taille maximale d'un paquet `IP` : un **gros** fichier, une image, ...Vu de l'application, il n'y a qu'un et un seul message à transférer, et de fait, une écriture suffit.

1. Décrire dans chacun des cas ci-dessous les transformations subies par le message au fur et à mesure de son évolution, jusquà son arrivée à destination.
2. Préciser les interfaces entre couches, il n'est pas nécessaire de détailler les encapsulations des diverses couches.

Exercice 15

1. M_1 et M_2 sont sur le même réseau local et ont une seule connexion au réseau chacune. Leurs numéros IP respectifs sont `193.2.4.8` (ip_1) et `193.2.4.16` (ip_2). Le réseau local est un réseau *ethernet* et les adresses physiques sont `8:4:CF:20:36:AB` (eth_1) et `8:20:FE:10:20:48` (eth_2).
2. On suppose maintenant que M_1 et M_2 sont sur deux réseaux distincts. Prendre pour adresse IP de M_2 `195.16.32.64` et ignorer la référence précédente.

   Décrire l'évolution du paquet lorsqu'un seul routeur relie les deux réseaux.
3. Si plus d'un routeur intervient, expliquer ce qui se passe dans chaque routeur.

Exercice 16

On veut mettre en place une communication entre deux hôtes A et B situés sur deux réseaux locaux, interconnectés par un routeur R. On rappelle qu'un routeur est une machine capable de prendre des décisions de routage à partir de l'adresse réseau de destination (adresse couche 3). Cette communication fait que A est amené à expédier beaucoup d'informations vers B. Toujours sur B, des gourmands insatiables lancent d'autre applications, faisant que d'autres hôtes, du même côté que A vont à leur tour envoyer des informations vers B. Vu de R, on a donc beaucoup de paquets venant du côté A et peu de celui de B.

1. Expliquer comment une communication en mode connecté TCP entre A et B peut être mise en place, le dialogue entre les applications se dérouler puis échouer en plein milieu, sans jamais se terminer.
2. Préciser comment se termine alors chacune des deux applications.

12 Routage dans les réseaux

Exercice 17

Donner un exemple de boucle ou circuit de routage comportant au moins trois routeurs, montrant comment un paquet (vu de la couche réseau) pourrait tourner indéfiniment entre plusieurs réseaux.

Une méthode pour empêcher un tel paquet de vivre trop longtemps dans les réseaux est d'utiliser la limite du nombre de routeurs que ce message peut *traverser*. Ce nombre peut être fixé au départ du message et permet en particulier d'éviter les boucles de routage.

1. Rappeler l'algorithme utilisé pour éviter que ces boucles soient infinies ;
2. Est-ce que cet algorithme est différent selon que les applications concernées par cet échange communiquent en utilisant un mode de transport connecté ou sans connexion ?

3. Montrer comment deux paquets avec les mêmes adresses IP *source* et *destination* peuvent l'un arriver, l'autre être supprimé.

Soit R_s le nom du routeur qui supprime le paquet dans votre exemple.

1. Que doit faire R_s en plus de la suppression du paquet ?
2. Décrire ce qui se passe à la suite d'une telle suppression lorsque le paquet supprimé fait partie d'une communication en TCP ? Préciser quelle entité doit être avertie de la suppression.
3. Même question avec UDP.