

HELMo - Haute Ecole Libre Mosane
Informatique de gestion

Systèmes d'exploitation et réseaux, partie II

Correction
Exercice d'intégration

par Louis SWINNEN

Ce document est disponible sous licence Creative Commons indiquant qu'il peut être reproduit, distribué et communiqué pour autant que le nom de l'auteur reste présent, qu'aucune utilisation commerciale ne soit faite à partir de celui-ci et que le document ne soit ni modifié, ni transformé, ni adapté.



<http://creativecommons.org/licenses/by-nc-nd/2.0/be/>

La Haute Ecole Libre Mosane (HELMo) attache une grande importance au respect des droits d'auteur. C'est la raison pour laquelle nous invitons les auteurs dont une oeuvre aurait été, malgré tous nos efforts, reproduite sans autorisation suffisante, à contacter immédiatement le service juridique de la Haute Ecole afin de pouvoir régulariser la situation au mieux.

Juin 2008

1. Introduction

Le présent corrigé est *une correction type* d'un exercice d'intégration. L'exercice résolu est celui présenté dans le cours (dia 336 à 338).

2. Enoncé

Soit le code suivant :

```
struct hostent* hostEntry;
struct sockaddr_in address;
int my_socket;
char buffer[15000];

hostEntry = gethostbyname("www.hemes.be");
address.sin_family = AF_INET;
address.sin_port = htons(80);
memcpy((char*) &(address.sin_addr), hostEntry->h_addr, hostEntry->h_length);
my_socket = socket(PF_INET, SOCK_STREAM, 0);
connect(my_socket, (struct sockaddr*) &address, sizeof(address));
recv(my_socket, buffer, sizeof(buffer)-1, 0x0);
close(my_socket);
```

Il faut ~~peut-être~~ savoir que :

- Le MTU est de 1500 octets
- `www.hemes.be` correspond à `81.188.2.20`
- Le serveur DNS à l'adresse `195.207.137.26`
- L'adresse IP de la machine exécutant le code est `195.207.137.28` et le masque associé est `255.255.255.248`
- Le routeur de sortie est `195.207.137.30`
- Les tables, fichier `host` et les mémoires caches (et autre information temporaire) sont vides.
- La taille de l'information reçue est de 5000 octets
- Il s'agit d'un réseau Ethernet, connecté en bus.

On vous demande :

- d'expliquer ce qui se passe à **chaque couche** lorsque les instructions **en gras** sont exécutées par la machine
- de considérer uniquement le cas de l'émetteur
- d'expliquer tous les concepts que vous utilisez
- de toujours mentionner la couche responsable de l'opération que vous décrivez
- d'être précis sur les échanges qui s'opèrent
- d'utiliser si nécessaire des schémas pour décrire les échanges
- de proposer des valeurs réalistes, si des données sont manquantes.

Il est clair qu'à l'examen, l'exercice ne sera pas si conséquent. Ici, nous avons plusieurs exercices regroupés dans cet énoncé ce qui me permet d'expliquer plusieurs éléments. Il y a, bien sûr, plusieurs façons de présenter la réponse.

3. Résolution

Avant de se lancer dans le raisonnement de la solution, il faut se poser la question suivante : *Quels sont les thèmes du cours qui sont abordés au travers de cet énoncé ?*

Il s'agit ici d'un programme C qui exécute le code. Les lignes de code à considérer concernent les instructions C *gethostbyname*, *connect* et *recv*. On voit donc les thèmes suivants se dégager :

1. la résolution de nom DNS par l'instruction *gethostbyname*
2. l'établissement d'une connexion TCP via le *connect*
3. la réception d'information via l'instruction *recv*.

Nous allons structurer la réponse en fonction de ces différents thèmes.

3.1 La résolution DNS

L'instruction *gethostbyname* permet de résoudre un nom et obtenir l'adresse IP en utilisant le service DNS. Le service DNS fonctionne comme suit : la machine contacte son serveur DNS (dans notre cas : 195.207.137.26) en utilisant le protocole UDP sur le port 53.

La requête envoyée par la machine contient la question suivante : quelle est l'adresse IP correspondant au nom `www.hemes.be`. Pour y répondre, le serveur DNS 195.207.137.26 va parcourir l'arbre DNS depuis la racine, tout d'abord pour obtenir l'adresse du serveur DNS gérant `.be` puis celle du serveur DNS gérant `.hemes.be` afin d'obtenir l'adresse de la machine `www`. Une fois cette adresse obtenue, le serveur DNS 195.207.137.26 répond au client.

Construction du TPDU UDP

Le protocole de transport UDP est un protocole minimaliste analogue au service proposé par la Poste. Les TPDU UDP sont envoyés sans aucune garantie que celui-ci arrivera à destination. C'est un protocole orienté sans connexion et non fiable. Parmi les informations importantes dans un TPDU UDP, on trouve également les numéros de port source et destination qui identifient les applications émettrice et destinataire de l'information.

- Port source : 1025 (par exemple)
- Port destination : 53
- Données : « quelle est l'adresse IP de `www.hemes.be` »

Construction du paquet IP

Le paquet IP ne contiendra aucune option particulière. Nous aurons les champs habituels : *version*, *tailles*, *TTL*, *checksum*. Le champ *Up-Protocol* désignera UDP (puisque les données contiennent un TPDU UDP).

- Adresse IP source (codée sur 32-bits) : 195.207.137.28
- Adresse IP destination (codée sur 32-bits) : 195.207.137.26
- Up-Protocol : UDP
- Données : TPDU UDP (contenant la requête DNS)

Construction de la trame Ethernet

La trame Ethernet contient différentes informations comme *l'adresse destination*, *l'adresse source*, *le type*, *les données* et *le CRC*.

L'adresse physique source est connue, il s'agit de celle de la machine exécutant le code C. L'adresse physique destination, par contre, n'est pas connue. Pour l'obtenir, il faut envoyer une requête ARP.

ARP (*Address Resolution Protocol*) est un mécanisme permettant d'obtenir l'adresse physique d'une machine au départ de son adresse IP. L'idée est la suivante, la machine envoie une requête ARP en broadcast (reçue par toutes les machines connectées au même sous-réseau) contenant la question suivante : *quelle est l'adresse physique de la machine dont l'adresse IP est 195.207.137.26 ?* La machine concernée répond en fournissant son adresse physique. La trame ARP est envoyée avec comme adresse physique source, l'adresse de la machine qui pose la question et comme adresse physique destination, `ff:ff:ff:ff:ff:ff` qui correspond à l'adresse broadcast. La machine concernée envoie une trame pour répondre à la requête et de ce fait, donne son adresse physique.

Une fois l'adresse physique destination obtenue, la trame peut être construite.

- Adresse destination : Obtenue par la requête ARP. Ex : 00:01:1b:3f:92:16
- Adresse source : celle de la machine. Ex. 00:01:1b:48:6e:9a
- Type : protocole IP
- Données : paquet IP (contenant le TPDU UDP + requête DNS)
- CRC : valeur de contrôle calculée

Le CRC est un système polynomial permettant de vérifier que l'information reçue n'a pas été altérée. Le principe est simple : rendre l'information exactement divisible par un polynôme générateur déterminé. La destination vérifie si l'information est toujours exactement divisible (le reste vaut 0) par le polynôme générateur. Si c'est le cas, l'information est réputée être arrivée sans modification. Le CRC permet de détecter toutes les erreurs simples, doubles et en rafale de longueur \leq au degré du polynôme générateur. Ce mécanisme est donc bien meilleur que la parité.

La trame est alors envoyée sur le réseau en utilisant CSMA/CD. Afin d'éviter toute collision, la machine va écouter le réseau afin de déterminer si une autre machine est en train de transmettre. Si c'est le cas, elle va attendre un temps aléatoire et puis réessayer. Dans le cas contraire, la trame est émise et est reçue par le serveur DNS. Celui-ci construit la réponse et la retourne à la machine émettrice.

Lors de la réception des informations, la couche « accès réseau » va analyser la trame reçue, vérifier que l'adresse physique correspond à celle de la machine et ensuite, elle va contrôler les informations (le CRC est-il correct ?) et ensuite, grâce au champ *type* (qui mentionne qui est destinataire des données dans la trame), elle va transmettre les données à la couche réseau IP.

La couche réseau IP va contrôler le paquet (via le checksum) et vérifier que l'adresse IP destination correspond bien à une des adresses IP configurées sur la machine. Ensuite, grâce au champ *Up-protocol*, la couche IP détermine quelle couche transport est destinataire des informations (dans notre cas, la couche transport UDP).

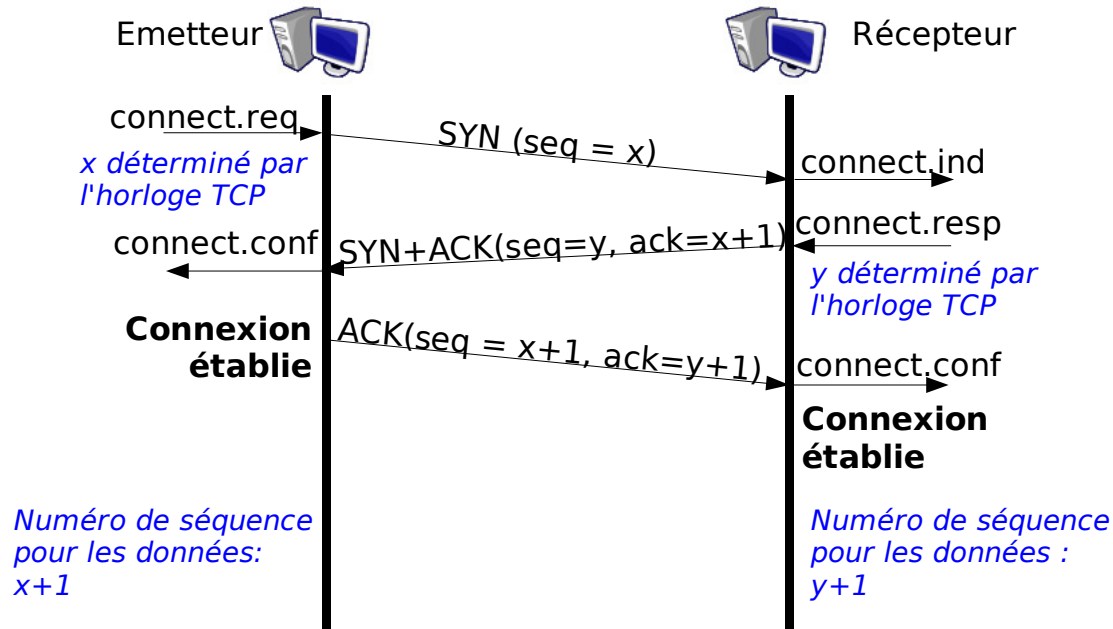
La couche transport UDP reçoit les informations, vérifie le checksum s'il est rempli, et, en fonction du port destination, transmet le contenu du TPDU UDP à l'application concernée (dans notre exemple, le programme C exécutant la requête).

Comme les concepts sont analogues à ceux présentés plus haut, ils n'ont pas fait l'objet d'une nouvelle description.

3.2 Etablissement de la connexion TCP

La connexion TCP permet l'envoi d'information entre un émetteur et un récepteur de manière fiable. Cette phase est obligatoire pour toute communication utilisant le protocole de transport TCP.

La connexion TCP s'établit en 3 phases en suivant le schéma du 3-way handshake :



Le TPDU TCP contient comme information principale le port source et destination, le numéro de séquence et d'acquit, les flags, le checksum et la taille de la fenêtre

Construction du TPDU TCP

- Port source : 1026 (tcp)
- Port destination : 80 (tcp, web)
- Numéro de séquence : x (déterminé par l'horloge transport émetteur)
- Flag : SYN (positionné à toute ouverture de connexion TCP)

Construction du paquet IP

Le paquet IP ne contiendra aucune option particulière. Nous aurons les champs habituels : *version, tailles, TTL, checksum*. Le champ *Up-Protocol* désignera TCP (puisque les données contiennent un TPDU TCP).

- Adresse IP source (codée sur 32-bits) : 195.207.137.28
- Adresse IP destination (codée sur 32-bits) : 81.188.2.20 (car résolution DNS faite)
- Up-Protocol : TCP
- Données : TPDU TCP

Construction de la trame Ethernet

La trame Ethernet contient différentes informations comme *l'adresse destination, l'adresse source, le type, les données et le CRC*.

L'adresse destination du paquet IP est hors du réseau local. Il faut, par conséquent, adresser ce paquet au routeur afin que ce dernier puisse transmettre le paquet à la destination. Pour savoir si la machine destination est *directement connectée* à la machine source, il faut simplement appliquer un ET logique entre l'adresse destination et le masque. Si l'adresse

obtenue est identique à l'adresse réseau (celle-ci est obtenue en effectuant un ET logique entre l'adresse IP d'une machine du réseau et le masque – par exemple 195.207.137.28 ET 255.255.255.248 => 195.207.137.24), alors c'est que la machine source et destination sont directement connectée sinon, il est nécessaire d'adresser le paquet à un routeur.

Pour adresser l'information au routeur, il faut simplement construire la trame ethernet pour que l'adresse physique destination corresponde à celle du routeur. Cela sera toujours possible car, pour interconnecter plusieurs réseaux, le routeur doit être accessible directement.

Pour obtenir l'adresse physique du routeur, le protocole ARP est utilisé. ARP a déjà été décrit précédemment.

Une fois l'adresse physique destination obtenue, la trame peut être construite.

- Adresse destination : Obtenue par la requête ARP. Ex : 00:01:1b:67:67:9f
- Adresse source : celle de la machine. Ex. 00:01:1b:48:6e:9a
- Type : protocole IP
- Données : paquet IP (contenant le TPDU TCP)
- CRC : valeur de contrôle calculée

Le CRC est un système polynomial permettant de vérifier que l'information reçue n'a pas été altérée. Celui-ci a déjà été décrit précédemment.

La trame est alors envoyées sur le réseau en utilisant CSMA/CD. Afin d'éviter toute collision, la machine va écouter le réseau afin de déterminer si une autre machine est en train de transmettre. Si c'est le cas, elle va attendre un temps aléatoire et puis réessayer. Si 2 machines transmettent en même temps, il se produit une collision et alors, toute transmission est arrêtée afin de régler la collision. Dans le cas contraire, la trame est émise et est reçue par le routeur. Celui-ci est chargé d'orienter le paquet vers la destination.

Lors de la réception des informations, la couche « accès réseau » va analyser la trame reçue, vérifier que l'adresse physique correspond à celle de la machine et ensuite, elle va contrôler les informations (le CRC est-il correct ?) et enfin, grâce au champ *type* (qui mentionne qui est destinataire des données dans la trame), elle va transmettre les données à la couche réseau IP.

La couche réseau IP va contrôler le paquet (via le checksum) et vérifier que l'adresse IP destination correspond bien à une des adresses IP configurées sur la machine. Ensuite, grâce au champ *Up-protocol*, la couche IP détermine quelle couche transport est destinataire des informations (dans notre cas, la couche transport TCP).

La couche transport TCP reçoit les informations, vérifie le checksum, et examine le TPDU. Le TPDU est, en fait, une réponse à la demande de la connexion TCP et les flags SYN et ACK sont positionnés.

La couche transport va répondre à cette demande via un TPDU TCP dont le flag *ACK* est positionné. Ces étapes ont déjà été décrites. La connexion est établie entre les deux machines distantes.

3.3 Réception des informations

En exécutant la fonction *recv*, la couche application, par un appel système, commande une entrée-sortie (puisqu'il faut attendre la réception d'information). La couche application est alors en attente d'information car l'appel système *recv* est bloquant (l'exécution du programme est suspendue en attendant que les informations soient arrivées).

La taille des informations reçues est de 5000 octets au total. Or le MTU est placé à 1500 octets. La conséquence est que l'information originale sera *fragmentée* en plusieurs morceaux. En effet, d'un bout à l'autre du réseau, plusieurs technologies réseaux peuvent cohabiter avec pour conséquence des limites sur la taille des informations qui peuvent être traitées (par exemple, une trame ethernet peut faire jusqu'à 1500 octets). Le MTU (*Maximum Transfer Unit*) renseigne cette longueur.

La fragmentation est un mécanisme propre à IPv4 pour « couper » les paquets trop grands (i.e. dont la taille est supérieure au MTU) en petits paquets. Le rassemblement de tous les morceaux est pris en charge par la couche réseau IPv4 destination qui transmet alors le paquet complet à la couche transport supérieure. Pour fonctionner, la fragmentation utilise le champ *fragmentation offset*, *identifier* et les *flags* présents dans le paquet IPv4. Les flags importants sont : DF (Don't Fragment) qui mentionne que le paquet ne peut être fragmenté et MF (More Fragment) qui mentionne que des fragments suivent ce fragment (et donc ce drapeau est positionné sur tous les fragments excepté le dernier). Le champ *fragmentation offset* permet de reconstituer le paquet original car il mentionne le déplacement de l'information par rapport au paquet original (positionnement du 1^{er} octet de donnée du fragment dans le paquet original). Le champ *identifier* permet de reconnaître tous les fragments d'un même paquet. Ce champ est positionné par la source et est présent dans chaque fragment.

Réception des trames Ethernet

Les trames Ethernet contenant les différents fragments contiennent principalement les adresses physiques (source et destination), le type (couche réseau destinataire de l'information), les données (contenant le paquet) et le CRC.

La couche « accès réseau » va vérifier que l'adresse physique correspond à celle de la machine et ensuite, elle va contrôler les informations (le CRC est-il correct ?) et ensuite, grâce au champ *type* (qui mentionne qui est destinataire des données dans la trame), elle va transmettre les données à la couche réseau IP.

- Adresse destination : 00:01:1b:48:6e:9a (celle de la machine)
- Adresse source : 00:01:1b:67:67:9f (celle du routeur)
- Type : protocole IP
- Données : paquet IP (contenant le TPDU TCP)
- CRC : valeur de contrôle calculée

Réception des paquets IP

Comme expliqué plus haut, nous allons recevoir plusieurs fragments IP. Comme l'information fait 5000 octets et qu'on a, dans chaque fragment IP 1480 octets de données (et 20 octets d'en-tête), il nous faudra 4 fragments répartis comme suit :

<u>Fragment 1</u> : 0 à 1479 – flag MF (1500 octets)	<u>Fragment 2</u> : 1480 à 2959 – flag MF (1500 octets)
<u>Fragment 3</u> : 2960 à 4439 – flag MF (1500 octets)	<u>Fragment 4</u> : 4440 à 5000 (580 octets)

Tous les fragments disposent d'une en-tête IP reprenant les informations suivantes :

- IP source : 81.188.2.20 (information du serveur web)
- IP destination : 195.207.137.28 (IP de la machine)
- Flag : MF sauf sur le dernier fragment
- Identifiant : valeur placée par la source, codée sur 16 bits
- Fragmentation offset : suivant le tableau ci-dessus
- Up-protocol : TCP
- Données : morceau du TPDU TCP

Une fois tous les fragments arrivés, le paquet IP original est reconstitué et vérifié. La couche réseau vérifie que l'adresse destination correspond à une des adresses de la machine. Elle vérifie également que le checksum est correct. Ensuite, le TPDU est extrait. Via le champ *Up-Protocol*, la couche réseau sait à quelle couche transport le TPDU doit être transmis (ici TCP).

Réception du TPDU TCP

Le TPDU TCP contient les données demandées par la couche application. Les informations utiles dans le TPDU TCP sont *les numéros de port (source et destination), le numéro de séquence, les flags*.

- Port source : 80 (tcp, web)
- Port destination : 1026 (tcp)
- Numéro de séquence : $y + 1$ (car réception : destinataire vers émetteur)
- Flag : Aucun

Suite à la réception de ce TPDU, la machine émettra un TPDU de contrôle pour acquitter la réception de ce TPDU.

Les informations sont transmises à la couche application.

4. Questions à se poser

Que se passerait-il si :

- Des informations étaient perdues ?
 - Go-Back-n ↔ Selective Repeat
 - Fast Retransmit ↔ Expiration des temporisateurs et contrôle de congestion
- Vous deviez gérer la déconnexion TCP (instruction `close`) ?
 - La déconnexion TCP est effectuée dans chaque sens !