Université de Caen
U.F.R. de Sciences

Année 2016 - 2017
Devoir maison
A. Lechervy M. Spaniol

L3 Informatique

Devoir maison

Systèmes d'exploitation avancés et réseaux

à rendre le 5 décembre 2016 à 23h59 au plus tard

Serveur DNS

1 Présentation

Le but de ce devoir est de réaliser un serveur DNS (Domain Name System) permettant la résolution de nom de domaine. Vous aurez à réaliser un script ou un ensemble de scripts permettant cette opération sur les machines en libre-services du département informatique en implémentant le protocole DNS.

Le principe de base du travail à réaliser est le suivant : une fois lancé, le serveur permet, au travers d'un client classique (dig, host, navigateur web...), de demander une résolution de nom de domaine.

Ce projet est à rédiger **en C** en utilisant la commande **netcat pour faire la partie réseau du serveur**. Vous n'avez **pas le droit d'utiliser de bibliothèques extérieures** mis à part les bibliothèques standards du langage vu en cours.

2 Rappels généraux

2.1 Outils réseaux

Vous utiliserez **netcat** (ou **nc**) pour programmer un mini serveur. Il est nécessaire de développer des programmes appelés par **netcat** côté serveur. Pour la réalisation du projet, on rappelle l'existence d'une option **-e** de l'utilitaire **nc.traditional**. Ainsi qu'on l'a vu en T.P., il existe plusieurs versions de netcat. Prenez soin de choisir une version qui accepte le transfert du contrôle à un programme extérieur (option **-e**).

2.2 Documentations

Vous vous appuierez sur la 2ème version du protocole DNS. Le descriptif détaillée du protocole DNS est disponible dans les RFC 1034 et 1035. Référez vous à ces documents pour réaliser ce projet :

`https://tools.ietf.org/html/rfc1034`,
`https://tools.ietf.org/html/rfc1035`.

La notation prendra en compte le respect de votre code à la norme. Le sujet est donné à titre indicatif, en cas d'erreur dans le sujet ou de passage pas assez clair, vous devez implémenter ce qui est décrit dans les deux RFC cités plus haut.

Pour tester votre serveur vous pouvez vous aider des commande **dig** comme vu en TP. Par exemple, la commande suivant interroge le serveur DNS à l'adresse 127.0.0.1 sur le port 4567 et lui demande l'adresse IP associé au nom www.unicaen.fr :

dig @127.0.0.1 -p4567 www.unicaen.fr

Votre serveur devra pouvoir répondre correctement à ce type de commande.

2.3 Rappel de C

Le code que vous aurez à rendre est à écrire en C. Vous avez le droit d'utiliser uniquement les bibliothèques vu en cours, TD et TP et notamment stdio.h, stdlib.h, errno.h, unistd.h, signal.h, et string.h .

Vous aurez besoin de manipuler des chaines de caractères en C. Pour cela utiliser la libraires string.h et notamment les fonctions strlen, strcpy et strcmp. Vous pouvez vous référer au TP 5 pour des exemples d'utilisation de ces fonctions.

La librairie stdint.h introduit différent type d'entier avec une taille précise. Par exemple int16_t est un entier signé composé de 16 bits. Vous pouvez donc utiliser ce type pour stocker des objets de 2 octets. La page de man de stdint.h, vous donnera tout les types disponibles.

Vous pouvez facilement passer d'un tableau de 2 entiers sur un octet à un entier de 2 octets et réciproquement en utilisant des instructions similaires à celles qui suivent :

```
void convert_int16_to_2int8(int16_t in,int8_t *res){
    int8_t *c=(int8_t*)&in;
    res[0]=c[0];
    res[1]=c[1];
}

int16_t convert_2int8_to_int16(int8_t *in){
    return *(int16_t*)in;
}
...
int16_t i_16=12345;
int8_t i_2x8[2];
convert_int16_to_2int8(i_16,i_2x8);
int16_t j_16 = convert_2int8_to_int16(i_2x8);
```

Fig. 1 : Conversion d'un entier sur 16bits en un tableau de deux entiers sur 8bits et réciproquement.

Les fonctions write et read prennent en argument un descripteur de fichier. Si vous utiliser un descripteur de fichier de valeur 0 ou 1, vous pouvez lire et écrire des données binaires sur l'entrée et la sortie standard.

On rappel également l'existence d'opération bits à bits en C (à ne pas confondre avec les opérateurs logiques) : << (décalage binaire vers la gauche), >> (décalage binaire vers la droite), & (et binaire),^ (ou exclusif binaire), | (ou inclusif binaire) , ∼ (négation binaire). Vous pouvez ainsi testez si le 3ème bits (en commençant la numérotation à partir de 0 de la droite vers la gauche) d'un nombre est à 1 avec le code suivant :

```
int i =12345;
if(i & 1<<3)
        printf("Le␣3ème␣bit␣de␣%d␣est␣1\n",i);
```

FIG. 2 : Test si le 3ème bits de 12345 est un 1.

3 Travail à faire

3.1 Analyse de protocole

Commencez par **lire le RFC1035, vous en ferez une synthèse** dans votre rapport en présentant les points qu'il vous semble important. Vous lirez notamment le chapitre 4 de la page 25 à la page 33.

Sur le moodle, vous trouverez le fichier Wireshark ***exemples_dns.pcapng*, vous en ferez une analyse** en expliquant de manière détaillée chacun des paquets. Ces paquets sont les résultats type que votre serveur devra retourner.

3.2 Réception et envoi d'une requête DNS valide

Un serveur DNS doit être à l'écoute sur un port réservé à cet usage (53 pour le protocole DNS). Votre serveur devra pourvoir être lancé sur n'importe quel port. En particulier, nous prendrons le **port 4567** pour ce projet afin de pouvoir lancer le serveur sans droit root.

Le protocole DNS utilise généralement des communications de type UDP (cf. page 31 du RFC1035), pensez à mettre l'**option -u à netcat** .

Vous utiliserez la commande netcat pour lancer votre serveur. Dans ce cas les commandes venant du client seront envoyé (en binaire) sur l'entrée standard de votre application. Vous répondrez en écrivant directement sur la sortie standard. Si vous souhaitez faire des affichages de debug, vous utiliserez la sortie d'erreur. Par exemple, pour lire le premier octet envoyé par le client et l'afficher dans le terminal du serveur, vous pouvez écrire en C :

```
int8_t b;
read(0,&b,1);
fprintf(stderr,"%d\n",b);
```

FIG. 3 : Code d'exemple de lecture du flux d'entrée en C.

En lisant la page 25 du RFC 1035, on voit qu'un paquet DNS est composé de 5 parties :

- une entête (Header),

- une partie Question,
- une partie Réponse,
- une partie Autorité,
- une partie Information additionnelle.

En lisant la section *Header section format* commençant à la page 26 du RFC 1035, on a la composition de la partie entête. On y trouve notamment que l'entête est composé de 12 octets que l'on peut décomposer en 6 blocs de 2 octets apparaissant dans l'ordre suivant :

1. un identifiant, identique dans la question et la réponse,
2. des flags,
3. le nombre d'entrée dans la section Question,
4. le nombre d'entrée dans la section Réponse,
5. le nombre d'entrée dans la section Autorité,
6. le nombre d'entrée dans la section Information additionnelle.

La structure de la section Question est décrite page 28 du RFC 1035 et la structure des trois autres sections est donné en pages 29-30.

Les paquets que vous recevrez et que vous enverrez auront cette structure. On rappel aussi que la norme recommande d'utiliser UDP pour des paquets de moins de 512 octets (cf page 10 du RFC 1035). Vous pourrez donc supposer que les messages feront tous moins de 512 octets. Vous pouvez ainsi lire 512 octets d'un seul coup et les traités ensuite. La valeur renvoyée par la fonction read est le nombre d'octet lu. Cette valeur peut être inférieur aux nombres d'octet demandé si vous êtes arrivé en fin de fichier. Une fois la requête récupérée dans un tableau de 512 octets, écrivez une fonction permettant de récupérer le n-ième mot de 16 bits du tableau d'octet. Écrivez ensuite une fonction permettant de recopier un mot de 16 bits à une position donnée dans un tableau d'octet transmis en paramètre.

Vous pouvez à présent écrire des fonctions de lecture et d'écriture de la partie entête d'un paquet DNS (que vous aurez récupéré sous la forme d'un tableau de 512 octets). Vous réfléchirez à une structure de donnée pertinente pour représenter un paquet dans votre code. Vous pouvez éventuellement utiliser plusieurs variables. Vous expliquerez dans votre rapport votre choix et donnerez toutes les informations nécessaires à la compréhension de votre code.

Écrivez les fonctions de lecture et d'écriture de la section Question du paquet DNS dans un tableau de 512 octets.

Écrivez les fonctions de lecture et d'écriture des autres sections du paquet DNS dans un tableau de 512 octets.

Assemblez vos fonctions pour lire et écrire des paquets DNS valide. Vous pouvez dans un premier temps faire que renvoyer au client la requête qu'il vous a soumis.

Vous testerez votre code à l'aide d'une application client (par exemple la commande dig). Le client devra pouvoir lire la réponse que vous lui transmettez. On rappel l'existence de la commande *od -cd* permettant de visualiser un fichier binaire octet par octet.

Vous porterez une attention particulière à la section 4.1.4 du RFC 1035 qui décrit la façon de gérer les noms de domaines. Vous pouvez regarder en détaille le fichier Wireshark fournis pour y voir ce mécanisme.

Pour tester cette question, vous répondrez de la manière à toute les requêtes. Vous devez pour l'instant juste lire et écrire des requêtes valides sans regarder le contenu de la requête initiale.

3.3 Gestion des erreurs

La page 27 du RFC 1035 présente le champs RCODE de l'entête permettant de gérer les erreurs. Mettez en place un mécanisme permettant de gérer les différents types d'erreurs, notamment votre serveur devra pouvoir dire au client si le paquet envoyé est invalide ou si la requête demandée est non implémentée.

3.4 Lecture depuis un fichier

Vous avez à votre disposition un fichier de configuration de DNS sur le moodle. Ajoutez dans votre code le code nécessaire à la lecture de ce fichier. Vous analyserez les requêtes et vous répondrez en fonction des valeurs du fichier.

3.5 Relai vers un autre DNS

Vous allez mettre en place un relais vers le DNS de l'université (dns.unicaen.fr). Les requêtes valident n'ayant pas de réponse dans le fichier de votre DNS devront être transmise et la réponse communiquée ensuite au client.

Pour réaliser cette fonctionnalité, vous utiliserez un processus fils qui appellera la commande *nc -u dns.unicaen.fr 53*. Vous aurez créé **deux pipes** pour communiquer entre le père et le fils. Dans le fils, avant l'appel à la commande netcat, vous redirigerez l'entrée standard et la sortie standard vers les deux pipes à l'aide de dup ou dup2 comme vu en cours, TD et TP. Votre processus principal enverra la requête sur un pipe et récupérera la réponse sur l'autre.

Vérifiez le bon fonctionnement du relai en fessant une requête dont la réponse n'est pas connu par votre DNS.

4 Rendu

Le travail est à réaliser en binôme (ou seul). Vous devrez rendre votre travail en utilisant le système *Devoir* de moodle. Pour cela, vous devez déposer une archive targz dans moodle.

Rappel de la date limite de la remise : **5 décembre 2016 à 23h59**.

Dans le cas présent cette archive doit contenir au moins un fichier **noms.txt** contenant les noms des étudiants ou étudiantes composant le binôme. Chaque étudiant(e) doit déposer au minimum ce fichier, qu'il ou elle soit seul(e), ou en binôme. Les autres fichiers (liste ci-dessous) ne seront déposées qu'une seule fois par binôme.

- les sources de vos programmes et les scripts éventuelles ;
- un fichier **README.txt** indiquant clairement les instructions pour lancer le serveur ;
- un rapport expliquant la manière dont vous avez résolu les problèmes, et donnant le mode d'emploi de votre réalisation ;
- tout autre fichier dont la présence est nécessaire pour faire fonctionner l'ensemble de votre réalisation.

5 Évaluation

Le non-fonctionnement du serveur sera fortement pénalisé, vous devrez donc respecter scrupuleusement le protocole et vous assurer du bon fonctionnement de votre serveur. Il est donc inutile d'essayer d'introduire des fonctionnalités si ce qui a déjà été fait ne marche pas correctement.

La qualité et le contenu du rapport ainsi que le respect des consignes seront également pris en compte dans la notation.