PR6 – Programmation réseaux
TP n° 5 : Un client dict

Voici un rappel des principales fonctions C que nous utiliserons. Pour rappel, vous pouvez utiliser « `man` » pour avoir plus d'informations sur chacune.

```
// Reseaux
int close(int fd);
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
void freeaddrinfo(struct addrinfo *res);
int getaddrinfo(const char *node, const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);
int inet_pton(int af, const char *src, void *dst);
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
ssize_t send(int sockfd, const void *buf, size_t len, int flags);
int snprintf(char *str, size_t size, const char *format, ...);
int socket(int domain, int type, int protocol);
```

I) Une première utilisation de `getaddrinfo`

Exercice 1 : modifier un client IPv6

Lors du TP2, au dernier exercice, vous avez écrit un client IPv6 qui communiquait avec une autre machine sur le réseau local, sur laquelle on faisait tourner un pseudo-`echo`. Pour cela, vous deviez choisir une machine allumée, vous y connecter pour récupérer son adresse IPv6, puis écrire à la main cette adresse dans le code du client.
NB : si besoin, vous pouvez récupérer une version de ce code sur moodle : `client_echo_ipv6.c`.

- Modifiez le code, en utilisant `getaddrinfo` plutôt que `inet_pton`, de façon à trouver automatiquement les adresses IPv6 de la machine à partir de son nom.
- Vérifiez que votre code marche toujours.

II) Un client pour `dict`

Le démon `dictd` permet d'accéder à travers le réseau à un ou plusieurs dictionnaires en ligne. À la différence des outils du type *Google Translate*, qui dépendent de HTTP, il utilise son propre protocole *dict* (RFC 2229). Le but de ce TP est d'implémenter un client *dict*.

Exercice 2 : se familiariser avec le protocole

Quel est le numéro de port qu'utilise `dict` ? Rappel : voir `/etc/services`.
Téléchargez la RFC 2229. Utilisez-la comme référence. Vous pourrez en particulier regarder les sections sur les commandes `HELP`, `DEFINE` et `SHOW`.

Exercice 3 : une première rencontre avec le service

Le protocole *dict* est un protocole textuel où les lignes se terminent par un `CRLF` (la chaîne « `"\r\n"` »). C'est le serveur qui parle le premier : il commence par envoyer une ligne ayant la forme suivante :

```
220 lampe dictd <auth> <5@foo>
```

Ensuite, le client est incité à effectuer des requêtes (`HELP`, `DEFINE`, `SHOW` parmi d'autres).

Le serveur auquel on va se connecter se trouve sur l'hôte nommé *lampe* (`lampe.informatique.univ-paris-diderot.fr`), pas accessible depuis l'exterieur du réseau de l'UFR. Pour y accéder, il faut soit travailler directement sur lulu (comme on a vu au TP2), soit faire une redirection de port en passant par lucy : ouvrez un terminal et executez la commande

`ssh -L 2628:lampe:2628 lucy.informatique.univ-paris-diderot.fr -l` *loginUFR*

(gardez-le ouvert) ; ainsi toute demande de connexion sur `localhost:2628` sera automatiquement redirigée vers `lampe:2628`.

- À l'aide de « `telnet` » ou de « `nc` », connectez-vous au service 2628 sur *lampe* ou *localhost* (selon votre choix ci-dessus).
- Testez quelques requêtes, par exemple :
  - trouver la liste des dictionnaires,
  - trouver le site sur lequel est publié un des dictionnaires,
  - trouver la définition du mot “network” dans tous les dictionnaires (indication : `DEFINE *`),
  - trouver la traduction en français du mot “computer”.

  Pour rappel, `HELP` vous donnera des informations sur les différentes commandes.

Exercice 4 : un client poli mais inutile

Dans cet exercice vous allez écrire un client TCP primitif, qui essaie de se connecter à l'adresse socket `adresse_ip_serveur:2628`. Dès qu'il arrive à se connecter, votre client affichera un message amical puis terminera ; s'il n'arrive pas à se connecter, il affichera un message d'erreur.
Tout d'abord, il faut récupérer l'adresse ip du serveur (en format binaire).

1. Dans un premier temps vous allez la trouver manuellement et la traduire comme **`struct in_addr`** en utilisant `inet_pton`. Si vous avez choisi de travailler sur *lulu*, il faut découvrir manuellement l'adresse ip de *lampe* (« `nslookup` » ou « `host` »). Sinon, utilisez l'adresse de l'hôte local (`127.0.0.1`).
2. Une façon préférable vu qu'on connait le nom d'hôte du serveur est de découvrir ses adresses IP en utilisant `getaddrinfo`. Votre client doit essayer de se connecter au port TCP correspondant au service `dict` de chacune d'entre elles.

Exercice 5 : un client qui lit

Modifiez votre client pour qu'il lise le premier message du serveur, puis qu'il vérifie que la ligne commence par la chaîne « `220␣` » (notez l'espace). Si ce n'est pas le cas, votre client affichera un message d'erreur. Testez.

Exercice 6 : un client "curieux"

Modifiez votre client pour que, une fois reçue l'invitation, il envoie une ligne de la forme

```
DEFINE * machin
```

où la chaîne «machin» est remplacée par le paramètre de la ligne de commande `argv[1]`. (Indication : `snprintf` est votre ami.)

Affichez la première réponse du serveur avant de terminer. Il devrait s'agir du nombre de définitions trouvées par *dict*, ou "no match" si le mot n'est pas dans les bases de données.

En pratique, les définitions sont envoyées dans un ou plusieurs autres messages. On verra dans un prochain TP comment traiter des réponses sur plusieurs messages.