PR6 – Programmation réseaux
TP n° 6 : Un client dict

On rappele des principales fonctions C que nous utiliserons. Vous pouvez utiliser « `man` » pour avoir plus d'informations sur chacune.

```
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) Utilisation de `getaddrinfo`

Exercice 1 : modifier un client IPv6

Au dernier exercice du TP2, 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, vous y connecter pour récupérer son adresse IPv6, puis écrire à la main cette adresse dans le code du client. On va maintenant automatiser cette étape.

1. 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.
   NB : si besoin, vous pouvez récupérer une version de ce code sur moodle : `client_echo_ipv6.c`.
2. En modifiant le code du client, faites en sorte que le client puisse se connecter à une machine dont le nom, et le port, sont passés en paramètres de la ligne de commande.
3. Testez avec `./exo2 lampe 7` depuis *lulu*.

Exercice 2 :

Ecrivez une fonction qui affiche les adresses d'une machine (IPv4 et IPv6) à partir de son nom.

II) Un client *polymorphe* 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 3 : se familiariser avec le protocole

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

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 la machine qui s'appele *lampe* : (`lampe.informatique.univ-paris-diderot.fr`). On rappele que cette machine n'est pas accessible depuis l'exterieur du réseau de l'UFR, donc pour y accéder, il faut passer par lulu.

3. À l'aide de « `telnet` » ou de « `nc` », connectez-vous au service 2628 sur *lampe* (ou *localhost* si *lampe* ne marche pas).
4. 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 : établir la connexion

Dans cet exercice vous allez écrire un client IPv6 polymorphe qui essaie de se connecter à `dict`. On ne sait pas à l'avance l'adresse de la machine, par conséquent, on ne sait pas si le serveur est en IPv4 ou en IPv6.

1. Étant donné que l'on connaît le nom de la machine à connecter, utilisez `getaddrinfo` pour récupérer son adresse. Faites attention : nous voulons que le client soit en IPv6, donc il faut choisir une bonne valeur pour `hints.ai_family`, et qu'il soit polymorphe, donc il faut choisir une bonne valeur pour `hints.ai_flags`.
2. Après la récupération de l'adresse, vous devez établir la connexion. Votre client doit afficher un message dès qu'il réussit à se connecter ; s'il n'arrive pas à se connecter, il affichera un message d'erreur.
3. 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 que votre client se comporte comme prévu en vérifiant différents scénarios de connexion.

Exercice 5 : demander la définition

1. Modifiez votre client pour que après la réception de l'invitation, il envoie une ligne de la forme

   `DEFINE *` *mot*

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

2. Le client doit afficher la réponse du serveur. La réponse peut être longue, donc il est possible qu'elle arrive en plusieurs morceaux. Vous devez donc lire jusqu'à la fin de la réponse. Faites attention au format des réponses.

   Rappel : on ne fait pas de lecture octet par octet !