PR6 – Programmation réseaux
TP n° 2 : Clients TCP en C

Voici un rappel des principales fonctions C que nous utiliserons.

```
int inet_aton(const char *cp, struct in_addr *pin);
int inet_pton(int af, const char * restrict src, void * restrict dst);
int socket(int domaine, int type, int protocole);
int connect(int socket, const struct sockaddr *adresse, socklen_t longueur);
int close(int socket);
ssize_t send(int socket,const void *tampon, size_t longueur,int options);
ssize_t recv(int socket,void *tampon, size_t longueur,int options);
```

Exercice 1 : Déterminer le « boutisme » (*endianness*) d'une machine

L'objectif de cet exercice est de déterminer de deux façons différentes si une machine est *little-endian* ou *big-endian*.

1. En regardant si la fonction `htonl` est l'identité (ce qui veut dire que *host order* et *network order* sont identiques) ou non. Tester un seul entier suffit.
2. En regardant comment sont encodés les entiers en mémoire sur votre machine. Pour cela on peut convertir un entier codé sur 32 bits, par exemple `uint32_t witness = 0x01020304;` en un tableau de 4 `unsigned char`, et les afficher dans l'ordre.

Exercice 2 : Un client TCP pour `daytime`

Écrire un client qui se connecte au service `daytime` de la machine `lampe` et affiche l'heure renvoyée. On peut utiliser "en dur" l'adresse IPv4 de `lampe`. Ne pas oublier de fermer la connexion.

Exercice 3 : Un client TCP pour `time`

Même question pour le service `time` de la machine `monjetas-etu`. Au lieu de renvoyer une chaîne comme `daytime`, un serveur `time` renvoie un entier de 32 bits non-signé, représentant le nombre de secondes écoulées depuis le 1er janvier 1900. C'est différent d'un `time_t` Unix qui, lui, compte les secondes depuis 1970 [1]. Plusieurs conversions sont à faire :

- depuis le *network order* au *host order*,
- puis en `time_t` en retirant le nombre de secondes entre les premiers janvier 1900 et 1970, soit `2208988800L`
- enfin en chaîne pour affichage, avec `ctime`.

Exercice 4 : Un client TCP pour `echo`

Faire un client qui se connecte au service `echo` de la machine `lampe` de l'ufr et qui en boucle lui envoie un message et affiche la réponse du service (par exemple le client enverra `Hello1`, `Hello2`, `Hello3`,...,`Hello10` au service et attendra la réponse du service entre chaque envoi).

1. Comme conséquence, le service `time` est victime du bug de l'an 2036, tandis que le temps Unix a le bug de l'an 2038 ou 2106 selon que `time_t` est implémenté en `signed long` ou, selon la norme POSIX, `unsigned long`.

Exercice 5 : Un client IPv6 pour `echo`

On veut adapter le code de l'exercice précédent pour que le client se connecte en IPv6, sur le port `4242`. Pour cela on va d'abord créer avec `netcat` un serveur du service `echo` sur le port 4242. Rien de plus simple, juste taper

```
mkfifo ncecho && cat ncecho | netcat -6 -l -k 4242 > ncecho; rm ncecho
```

Le serveur tourne sur l'adresse IPv6 locale soit `::1`, c'est elle qu'il faut utiliser "en dur" pour votre client IPv6. Il ne reste plus qu'à adapter le client de l'exercice précédent !

Si besoin, effacer manuellement le tube *ncecho* à la fin du TP.