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

En C, tout est bas-niveau, c'est-à-dire que contrairement à Java, beaucoup de choses sont à faire « à la main ». Voici un rappel des principales fonctions C que nous utiliserons.
Les prototypes des fonctions supplémentaires nécessaires se trouvent dans les fichiers en-tête suivants :

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
```

Les fonctions principales sont :

```
int socket(int domaine, int type, int protocole);
int bind(int socket, const struct sockaddr *adresse, socklen_t longueur);
int listen(int socket, int attente);
int accept(int socket, struct sockaddr *adresse, socklen_t *longueur);
int connect(int socket, const struct sockaddr *adresse, socklen_t longueur);
int shutdown(int socket, int how);
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);
int getaddrinfo(const char *hostname, const char *servname,
      const struct addrinfo *hints, struct addrinfo **res);
```

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

L'objectif de cet exercice est d'écrire deux programmes en C déterminant si la machine qui les exécute est en *little-endian* ou en *big-endian*.

1. En utilisant des entiers non signés sur 32 bits (de type (`uint32_t`)) et la fonction `htonl`, déterminez si votre machine est *big-endian* ou non.
2. Vérifiez maintenant comment sont encodés les entiers en mémoire sur votre machine. Pour cela vous pouvez convertir un entier codé sur 32 bits en tableaux de `char`, donc en tableaux de 4 octets, et ensuite afficher la valeur de chacun des octets du tableau (pour rappel `printf` avec `%x` permet d'afficher la valeur d'un octet donné).
3. Conclure quant à la correction de la fonction `htonl`

Remarque : dans les 2 cas, utilisez comme témoin un entier dont les 4 octets sont différents. Cela est facile à écrire en notation hexadécimale, par exemple : `uint32_t witness = 0x01020304;`.

Exercice 2 : Un client TCP pour `daytime` en C

1. Déterminez l'adresse IPv4 de la machine `lampe` de l'ufr.
2. Sur `lampe`, le service `daytime` tourne. Écrivez un client qui se connecte à ce service, affiche la date et l'heure renvoyées par le service et se déconnecte. Vous utiliserez pour cette question l'adresse IPv4 de `lampe`.

Exercice 3 : Un client TCP pour `echo` en C

Dans cet exercice, faîtes 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).

Exercice 4 : Faire un client echo à l'envers

Faites un client qui, une fois connecté à un service, fait en boucle 4 fois les actions suivantes : attendre un message (d'au plus 99 caractères) du serveur et lui renvoyer son message à l'envers. Vous devrez tester ce client avec netcat (en mode serveur).