

## PR6 – Programmation réseaux

### TP n° 3 : Clients et Serveurs TCP en C

On rappelle à nouveau les fichiers d'en-tête et les prototypes des principales fonctions à utiliser dans ce TP. Les fichiers d'en-tête à inclure sont les suivants :

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

Les principales fonctions à utiliser sont les suivantes :

```
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);
```

**Remarque :** Dans le document le signe □ représentera un simple caractère d'espacement (ASCII 32). De plus les messages circulant sont indiqués entre guillemets, **les guillemets ne faisant pas partie du message**.

#### Exercice 1 : Serveur echo IPv4

Écrire un serveur TCP implémentant le service **echo** (pour tout client se connectant en IPv4, le serveur répète au client tout ce que celui-ci lui envoie). On pourra écrire une première version où le serveur ne traitera qu'un seul client et s'arrêtera, et une deuxième où le serveur pourra traiter un nombre quelconque de clients *à la suite* (on verra plus tard comment traiter plusieurs clients *en parallèle* avec les *threads*). Attention à bien distinguer la socket associée au serveur des sockets associées aux clients. On vérifiera que l'on ferme bien les sockets des clients à la fin des interactions.

Tester le serveur en utilisant **nc** comme client (utilisez **localhost** comme nom d'hôte) :

- D'abord en passant l'option **-4** pour forcer une connexion IPv4.
- Puis en passant l'option **-6** pour forcer une connexion IPv6.

Que constatez-vous ?

#### Exercice 2 : Serveur echo IPv6

Adapter le code de l'exercice précédent pour accepter des connexions en IPv6.

Tester le serveur avec **nc** comme dans l'exercice précédent. Que constatez-vous ?

Remarque : nous verrons dans un cours ultérieur comment restreindre les connexions à IPv6 seulement.

### Exercice 3 : Jeu

On souhaite implémenter un serveur qui exécute le jeu décrit ci-après. Lorsqu'un client se connecte, le serveur choisit au hasard un nombre  $n$  entre 0 et 65535. Le client peut alors faire jusqu'à 20 tentatives pour deviner  $n$  en envoyant des requêtes de la forme " $k \backslash n$ " au serveur, où  $k$  est un entier compris entre 0 et 65535 représenté comme chaîne de caractères. À chaque requête, le serveur répond :

- "PLUS\_ $r \backslash n$ " si  $k$  inférieur à  $n$ , où  $r$  est le nombre de tentatives restantes ;
- "MOINS\_ $r \backslash n$ " si  $k$  est supérieur à  $n$ , où  $r$  est le nombre de tentatives restantes ;
- "GAGNE\_ $n$ " si  $k$  est égal à  $n$ . Le serveur clôt alors la communication ;
- "PERDU\_ $n$ " si  $k$  est différent de  $n$  et qu'il ne reste plus de tentative au client. Le serveur clôt alors la communication.

1. Implémenter un serveur qui exécute le protocole ci-dessus.
2. Tester le serveur en utilisant `nc`.
3. Écrire un client qui exécute automatiquement une partie avec le serveur. Peut-on écrire un client qui gagne à tous les coups avec les règles ci-dessus ?
4. On décide de changer le protocole afin que les messages entre le client et le serveur soient de taille fixe (de cette manière, nous n'avons pas besoin de chercher le caractère " $\backslash n$ " pour détecter la fin d'un message). Au lieu d'envoyer et de recevoir des nombres comme des séquences de caractères 0,...,9, changer le protocole et le code de façon à ce que :
  - le client envoie un entier au serveur sous la forme de deux octets correspondant à son codage en *big-endian*, et
  - le serveur réponde également sur deux octets : un entier sur un octet indiquant le nombre d'essais restant (0 si la partie est terminée), suivi de 0 pour MOINS/PERDU ou 1 pour PLUS/GAGNE.

Tester cette nouvelle version.

### Exercice 4 : Transfert de fichier

Dans cet exercice, on implémente un protocole simple décrit ci-après permettant de transférer un fichier d'un client à un serveur. Lorsqu'un client se connecte, le serveur envoie le message "DEBUT\_ $n$ " au client. Une fois que le client a reçu ce message, il envoie au serveur " $n \backslash n$ ", où  $n$  est l'écriture décimale d'un entier positif indiquant le nombre d'octets (ou nombre de caractères) du fichier à transmettre, suivi du contenu du fichier. Le serveur écrit alors le contenu reçu dans un fichier « `data.txt` » et répond "BIEN RECU\_ $n$ " au client et clôt la communication.

1. Écrire le code d'un serveur implémentant ce protocole. Votre serveur doit fonctionner même si la taille du fichier est supérieure à la taille de votre buffer de lecture.
2. Tester l'implémentation en utilisant `nc` comme client.
3. Écrire le code d'un client exécutant le protocole avec le serveur. On fera en sorte de pouvoir préciser le nom du fichier à transmettre en argument du programme lors de son lancement dans le terminal.