IUT DE PARIS

2018—2019

Principe des Systèmes d'exploitation

Programmation Client Serveur

Arsène Lapostolet

Groupe 205

I/ Présentation du projet

Le projet consiste en un Quiz en réseau. Le Quiz est une série de questions stockées sur le serveur. Lorsqu'un client se connecte au serveur, on lui demande son pseudo, puis le Quiz commence. Les questions sont ensuite envoyées au client, qui les affiche, récupère les réponses sur l'entée standard et envoie ces dernières au serveur. Grâce à la parallélisation du traitement des clients du quiz, plusieurs clients peuvent se connecter et effectuer le quiz en même temps

Quand toutes les questions ont été posées, le serveur envoie au client un message de fin ainsi que son score, puis ferme la connexion.

II/ Réseau

Gestion des adresses

sockaddr_in est utilisé pour représenter une adresse IP v4 avec également sa famille et son port lié :

```
struct sockaddr_in {
    short           sin_family;   // Famille : AF_INET ou AF_INET6
    unsigned short  sin_port;     // Port : à calculer avec htons()
    struct in_addr  sin_addr;     // L'adresse elle même
    char            sin_zero[8];
};
```

sockaddr est le type utilisé pour stocker toutes les adresses IP. C'est en quelque sorte une « abstraction à la C » des adresses IP. Ainsi, on utilise *sockaddr* dans les fonctions d'interaction avec les sockets, en effet, leur structure est assez proche pour pouvoir transtyper des *sockaddr_in** en *sockaddr**.

```
struct sockaddr {
    unsigned short    sa_family;    // Famille : AF_INET ou AF_INET6
    char              sa_data[14];  // L'adresse elle même
};
```

On utilise ensuite la fonction *htons()* pour remplir le port. *htons()* permet de traduire un *short* de l'hôte en *short* du réseau. C'est pourquoi on l'utilise pour traduire le port de la *sock_addr_in* qui est un *short*.

```
unsigned short int htons (unsigned short int hostshort);
```

Pour remplir l'adresse à partir d'un *char**, on utilise la fonction *inet_aton()*. Elle va en effet traduire l'adresse IP dans la chaîne de caractère dans la structure *sin_addr* de la *sockaddr_in*

```
int inet_aton(const char *addr, socklen_t addrlen);
```

Connexion

Pour commencer, on utilise la fonction *socket()* qui permet de créer un point de communication réseau et d'en récupérer le descripteur en valeur de retour. Il faut spécifier le domaine (on choisit *AF_INET* pour la communication IP v4), le type (on choisit *SOCK_STREAM* pour le mode connecté TCP/IP) et le protocole (on choisit 0 pour le protocole IP).

```
int socket(int domain, int type, int protocol);
```

En ce qui concerne le côté serveur, on utilise les fonctions *bind()* et *listen()*. *bind ()* permet de lier un socket à une adresse locale. C'est « l'affectation d'un nom au socket ». Nous allons donc passer en paramètre le descripteur obtenu avec *socket()*, notre *sockaddr_in* transtypée en *sockaddr* et la taille de l'adresse.

```
int bind(int sockfd, const struct sockaddr *cp, struct sin_addr *inp);
```

Ensuite, listen() permet de marquer le socket dont le descripteur est passé en paramètre comme un socket passif : il sera utilisé pour accepter les demandes de connexion entrantes. Avec le paramètre *backlog*, on peut spécifier la longueur de la file d'attente de connexion. Nous passons donc le descripteur du socket serveur précédemment attaché et nous choisissons une file d'attente de 15 clients.

```
int listen(int sockfd, int backlog);
```

Enfin, la fonction *accept()* permet d'accepter la connexion en attente d'un client et de récupérer en valeur de retour le descripteur du socket de communication avec ce client. On lui passe en paramètre le descripteur du socket serveur mis en écoute passive, l'adresse locale et sa longueur.

```
int accept(int sock, struct sockaddr* adr, socklen_t longueur);
```

En ce qui concerne le côté client, on utilise la fonction *connect()* pour établir la connexion avec le serveur qui permet de mettre en place la connexion avec le serveur dont on fournit l'adresse sur le socket client passé en paramètre. On passe donc en paramètre le descripteur du socket client obtenu par la fonction *socket()*, notre *sockaddr_in* transtypée en *sockaddr*, ainsi que sa longueur.

```
int connect (int sock_client, struct sockaddr * servadr, socklen_t longueur);
```

Communication Client/Serveur

Pour la communication client/serveur, nous n'utilisons pas *sendto()* et *recvfrom ()* car ces fonctions sont dédiées à la communication selon le protocole UDP en mode non connecté (datagrammes).

```c
int recvfrom (int sock, char * buffer, int taille_buffer, int attributs, struct
sockaddr * source, socklen_t * taille);


int sendto (int sock, char * buffer, int taille_buffer, int attributs, struct
sockaddr * destination, socklen_t taille);
```

Les fonctions dédiées à la communications TCP/IP sont *send()* et *recv()*. Elles permettent d'écrire sur le réseau en précisant des attributs. *send()* (resp. *recv()*) permet d'écrire (resp. de lire) un flux en texte sur le réseau à partir d'une socket dont la liaison est ouverte.

```c
int recvfrom (int sock, char * buffer, int taille_buffer, int attributs);
int sendto (int sock, char * buffer, int taille_buffer, int attributs);
```

Nous n'avons pas besoin de spécifier de tels attributs, c'est pourquoi nous utilisons les fonctions *write()*, et *read()*, qui nous permettent d'écrire et de lire des informations sur le flux en texte sur le réseau de la même façon que *send()* et *recv()*

```c
int read (int fd, char * buffer, int taille_buffer);
int write (int fd, char * buffer, int taille_buffer);
```

III/ Entrée et sorties

Côté client

Pour les entrées (pseudo et réponses au quiz) on utilise la fonction *fgets()* qui permet de lire une ligne depuis un flux en texte. On l'utilise sur l'entrée standard *stdin*. Pour les sorties, on utilise simplement *printf()*.

```c
char *fgets(char *s, int size, FILE *stream);
```

Côté Serveur

Sur le serveur, pas d'entrée, seulement des sorties (les logs des opérations effectuées par le serveur) qui sont simplement affichées avec *printf()*.