CHRISTOFOROU ANTHONY

RAPPORT

Server

Variables/Commands

Dans ce TP on cherche a faire une architecture serveur-client qui permettra a l'utilisateur de jouer a un jeu gerer par le serveur. Le plus important ici, sera le fait que l'on puisse connecter plusieurs client en meme temps sur un seul serveur.

La premiere chose a faire ici est de definir les commandes qui vont nous servir pour chaque communications:

> - `TOO_LOW` : Sera communique quand la valeur devinee par le client sera trop basse
> - `TOO_HIGH` : Sera communique quand la valeur devinee par le client sera trop haute
> - `WIN` : Sera communique quand la valeur devinee par le client est la bonne
> - `lOSE` : Sera communique quand la valeur devinee par le client est la mauvaise est que le nombre d'essais est 0
> - `MAX_TRIES` : Nombre d'essais restants

On a ensuite une fonction qui genere des nombres aleatoires, en effet,

```
unsigned char random_number(unsigned char min, unsigned char max);
```

Va prendre en argument 2 bits non signées, et va en retourner un au hasard borné par ces deux la en lisant dans le fichier `/dev/urandom`. Attention! Cela veut dire que l'on a donc une fonction qui ne marchera que dans un system UNIX.

Server Initialization

La premiere chose a faire a l'interieur de la fonction `main` est de mettre en place tout le necessaire pour lancer un serveur. On initialise donc `struct sockaddr_in address` qui va contenir tout le necessaire une adresse (cf. Structure Adressage Internet). On choisir alors un port (entre 1024 et 65535 puisque les addresses plus basses sont reservees) que l'ont va transformer d'abord avec `strtol`:

> Cette fonction permet de convertir un `string` en un `long`

puis ensuite avec `htons`:

> Fonction qui permet d'obtenir un numero de port valide et dans le bon byte-order

On va aussi initialiser une addresse de type `AF_INET` (IPv4) et ensuite la rendre valide avec `htonl` (comme `htons`)

Enfin on arrive a la partie ou l'on va utiliser tout ce qu'on a defini precedemment, tout d'abord viens la fonction `int socket(int domain, int type, int protocol)` qui va creer le socket que l'on va tout de suite utiliser avec la fonction `int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);` qui va permettre de lier le socket a notre addresse. Derniere etape, on met les client qui essaient de se connecter sur une liste d'attente grace a la fonction `int listen(int sockfd, int backlog);`, on choisit un maximum de 4 dans ce code.

On va bien evidemment tester le retout de chacune de ces fonctions et dans le cas ou l'on retourne `-1` on va afficher l'erreur grace a `errno`.

Server Connection

Ici tout va ce passer dans une boucle `for(;;)`, en effet on va devoir accepter des connections a chaque fois, mais avant ca on va initialiser chaque adresse de client on rappel alors `sockaddr_in` et on utilise la fonction `int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);` qui va retourner un descripteur de fichier qui va servir de socket client.

A partir de la on va creer un nouveau processus en utilisant `fork()`, mais on va `fork` une deuxieme fois pour pouvoir eviter les zombies. En effet de cette maniere on va tuer le premier enfant rendant le deuxieme enfant orphelin et lui permettant ainsi de ce relier a `init`, ou le premier processus parent qui lui aura `waitpid` le premier enfant. De cette maniere on evite les zombies.

A partir de la on va avoir la communication entre le client et le serveur ainsi que les conditions pour jouer. La toute premiere communication va etre la borne min et max, en utilisant `write` pour ecrire sur le descripteur de fichier.

> Le protocole choisit pour ce code sera de 3 bit pour chaque communications

On choisit alors un nombre aleatoire grace a la fonction `random_number` et dans une nouvelles boucle infinie on va recuperer la valeur mise par le client grace a un `read` et ainsi de suite. Enfin, on va tester la valeur a chaque fois et selon si elle est trop haute ou trop basse on va envoyer la commande correspondante comme vu au tout debut. On utilise alors `exit(int SIGNAL)` pour tuer le processus courant et continuer sur le `root`.

Client

Variables/Commands

De la meme facon que pour le serveur on va definir `TOO_LOW`, `TOO_HIGH`, ... ainsi que une structure `sockaddr_in` pour l'addresse client. Ici on prendra en entree l'ip et le port, et comme avec le serveur on va initialiser tout cela d'abord en utilisant `strtol` et `htons` pour le port. Pour l'addresse par contre on va utiliser `int inet_pton(int af, const char *src, void *dst);`

> On va traduire notre adresse ip vers une adresse valide tout en initialisant une structure `in_addr`

On continue alors avec `socket` mais au lieu de bind comme dans le serveur ici on va utiliser `int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);` qui permettra de connecter notre socket client a un serveur existant.

Tout comme avec le serveur on va tester chaque retour de fonction et afficher des erreurs dans le cas ou elles existent.

Communication

On recois alors notre premiere communication, les fameux `min` et `max` grace a `read`. On rentre alors dans une loop `for(;;)` et le jeu commence:

> Le client va demander a l'utilisateur d'entrer une valeur entre le `min` et le `max` et ensuite la transmettre au serveur qui lui va la tester. Le client recois alors une commande `cmd` et va donc effectuer une action selon la commande recue. Le client affichera toute les informations recu pour que l'utilisateur puisse en faire ce qu'il veut. Enfin dans le cas de perte ou reussite, le client se ferme.

Fonctionnement

L'utilisation du code est assez simple, en effet il suffit de lancer le programme suivie de certains arguments. Par exemple pour le serveur:

```
./server port_number
```

> Sachant que le port est compris entre 1024 et 65535 car les port plus bas sont reserve pour d'autres utilisations

Pour le client on aura une forme tres similaire:

```
./client ip_number port_number
```

> On va utiliser le meme port que celui choisit pour le serveur sinon la connectino ne sera jamais possible. Pour l'ip par contre, on va utliser `127.0.0.1` puisque on se connecte en `localhost`.