TINAZZI Giovanni VANSON Nathan

RAPPORT TP5

Objectifs

L'objectif principal de ce TP etait de creer une architecture client/server qui joue au jeu "devine a quel numero je pense". Pour faire cette architecture, il etait necessaire de manipuler les appels systemes liées aux sockets (en particulier read and write). Notre programme devait etre capable de gerer plusieurs clients pour un seul server, ce qui impliquait qu'il fallait faire attention a ne pas se retrouver avec des processus zombies.

Pour se faire, la partie code est divise en 2 modules:

> - la partie serveur, qui va s'occuper de envoyer a travers les sockets les informations aux client, qui va gerer les processus parents, enfants et zombie et qui va generer un nombre aleatoire pour que le client puisse jouer.

> - La partie client, qui va recevoir les informations du server, et envoyer sa proposition, tout en gerant les erreurs possibles lors de la creation du client (ex: Si le client arrive pas a se connecter)

Protocole de communication

Tout d'abord, pour que le client et le serveur puissent communiquer entre eux, il faut definir les messages d'initialisation. On va donc definir, dans la partie du code du server et de client, les commmandes suivantes:

> - TOO_LOW : on va communiquer que la valeur est trop basse
> - TOO_HIGH : on va communiquer qu la valeur est trop eleve
> - WIN : on va communiquer qu'il a gagne
> - LOSE : on va communiquer qu'il a perdu
> - NB_ESSAIS : on va communiquer le nombres d'essais deja utilise

---

Partie Server

Tout d'abord, dans notre partie serveur on va mettre notre prototype de la fonction permettant de generer des nombres aleatoires:

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

Cette fonction a pour but de generer des nombres aleatoires compris entre 0 et 100 (ca aurait pu etre des nombres differents, il suffit de modifier les variables `min` et `max`). On va ici plutot utiliser `rand()`.

Dans notre fonction `main` on va tout d'abord mettre en place le serveur. On va utiliser `struct sockaddr_in` pour initialiser la structure qui va contenir toutes les informations necessaire pour notre adresse.

Pour le port que devra utiliser l'utilisateur, on veux que l'utilisateur choisisse un port, donc on va utiliser `arg[1]` pour prendre en entree l'input de l'utilisateur, et on va verifier que le port choisis soit bien compris entre 1024 et 65535.

On va maintenant pourvoir creer le socket avec:

```
 int serverSock = socket(AF_INET, SOCK_STREAM, 0);
```

et ensuite lier notre socket a notre adresse:

```
int lier = bind(serverSock, (struct sockaddr*) &address, sizeof(address));
```

Enfin, etant donner que l'on veux que plusieurs client puissent etre traites par un seul server, on va creer une queue(une liste d'attente) et notre serveur pourra "ecouter" les clients avec:

```
int ecoute = listen(serverSock, queue);
```

Gestion du client

Dans notre boucle `while(1)`, on va d'abord initialiser notre client. Une fois initialise, notre serveur va accepter une connection avec l'appel systeme suivant:

```
int clientSock = accept(serverSock, (struct sockaddr *) &clientAddress,
&clientLength);
```

Pour l'instant, notre serveur peut accepter un seul client. On va donc utiliser `fork()` pour creer un nouveau processus a chaque fois qu'un nouveau client essaye de se connecter:

```
pid_t pid = fork(); // pid_t se trouve dans l include sys/types.h
```

En faisant des fork, on va creer des processus enfant. Le probleme des processus enfants et que on peut se retrouver avec des processus zombies, c'est a dire des processus qui n'ont pas ete tues. On va donc devoir les gerer, en travaillant sur les PID(Processus Id). Si on se retrouve dans le processus parent, on va devoir "tuer" les enfants avec la fonction:

```
waitpid(pid, NULL, 0);
```

Une fois qu'on est sur qu'on se trouve dans le processus enfant, on va pouvoir jouer avec le client et on est sur qu'il n y a pas de processus zombie.

Une fois cree le nombre aleatoire (different pour chaque client), on peux jouer. Ici, ce qui est important a retenir c'est que on peux envoyer des informations au client avec:

```
write(clientSock, &cmd, 1);
```

et on peux lire le nombre que le client a ecrit avec:

```
read(clientSock, NULL, 1);
```

enfin, en utilisant les protocole de communication pour que le client puisse voir ce qu'il doit faire.

Partie Client

Dans la partie client, on initialise de la meme facon que dans la partie serveur, sauf que nos sockets cette fois sont pour le client et non le server. On va donner a notre client l'adresse IP: "127.0.0.1", Pour le reste on verifie qu'il n y a pas d'erreur lors de la creation des different socket.

Une fois avoir check le bon fonctionnement des socket, on va lire le min et le max envoyes par le serveur. Une fois qu'on connait ces deux variables, On va pouvoir faire jouer le client.

La boucle while marche un peu de la meme facon que le serveur:

- Le client ecrit son choix et on prend la valeur tapee par le client
- On envoie la proposition du client au serveur avec: `write(client, &input_mod, 1);`
- notre serveur va recevoir la reponse du client, lui communiquer s'il a juste ou pas
- avec `read(client, &random_number, 1);` le client recupere la reponse du serveur.
- A l'aide des protocoles de communication, on va dire si le client a gagne, perdu, si c'est trop petit ou trop grand

Comment utiliser le programme

Pour l'utilisation du programme, on va devoir ouvrir un terminal pour le serveur, et un terminal pour chaque client.

Dans le terminal du serveur on devra ecrire le port de la facon suivante:

```
./server numero_port
```

Attention a bien penser quel port chosir (compris entre 1024 et 65535)

Dans le terminal du client, on va ecrire dans le terminal:

```
./client adresse_IP numero_port
```

Le numero du port devra etre le meme que celui ecrit dans le server et l'adresse IP va etre "127.0.0.1" comme dit auparavant.