Tristan BILOT G205

Compte-rendu du TP n°4 de PSE

Les sockets en C

Côté client

Côté client, nous devons indiquer l'adresse IP de la machine (localhost) donc 127.0.0.1 :

```
inet_aton("127.0.0.1", &adresse.sin_addr);
```

Le principe est de créer une structure sockaddr_in et d'y stocker :

- L'adresse IP (ici 127.0.0.1)
- Le port (ici 5000)
- La famille de socket (ici AF_INET)

Il faut ensuite caster cette structure en une autre structure sockaddr qui sera ensuite utilisable.

```
connect(sock, (struct sockaddr *)&adresse, &longueurAdresse
```

Il est important à noter pour la connexion au serveur qu'il faut utiliser la longueur de l'adresse avec une référence, sinon la connexion échouera, il faut donc faire attention au manuel qui nous dit d'utiliser un socklen_t sans référence.

Le reste du code permet la lecture du message tapé par l'utilisateur (scanf), puis de l'envoi au serveur (send) et de la réception du message du serveur (recv) : la réception du message est une amélioration (expliquée à la fin de ce compte-rendu). Tout ce code est placé dans une boucle pour répéter l'opération.

Côté serveur

Côté serveur, nous devons comme pour le client créer une socket ainsi qu'une structure sockaddr_in (nommée adresse) que l'on va remplir de la façon suivante :

```
adresse.sin_family = AF_INET;
adresse.sin_port = htons(PORT);
adresse.sin_addr.s_addr = htonl(INADDR_ANY);
```

Nous utilisons htonl(INADDR_ANY) pour permettre la connexion de toutes les machines, dans le cas où nous voulions autoriser que la connexion à la machine locale nous aurions renseigné 127.0.0.1 mais le but est ici de permettre à plusieurs machines de se connecter au serveur.

Nous utilisons la commande bind pour réserver une adresse pour la socket, nous devons donc impérativement l'utiliser côté serveur.

```
int bind_success = bind(server_socket, (struct sockaddr *)&adresse, longueurAdresse);
```

L'étape numéro 5 nous a permis de créer la structure de base côté serveur mais ne permet pas d'accepter encore des clients, nous allons donc coder la partie qui autorise les connexions au serveur pour pouvoir ensuite communiquer avec le ou les clients.
Nous utilisons listen() avant la boucle while pour mettre la socket en paramètre en mode passive et donc en mode écoute.

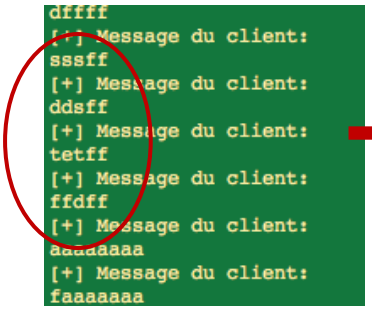
```
listen(server_socket, 1000);
```

Les paramètres de la fonction sont la socket à mettre en écoute et le nombre de connexions en attente. Nous mettons ici 1000 à titre d'exemple car nous savons qu'il n'y aura pas plus de 1000 machines dans la file d'attente.
On peut alors ensuite faire une boucle pour permettre plusieurs connexions sur le serveur. Il est impératif d'utiliser ensuite accept() dans la boucle pour accepter les connexions qui viennent des clients.

Seulement, notre serveur doit pouvoir accepter plusieurs clients à la fois, il va donc falloir utiliser la méthode fork() à chaque connexion d'un client pour qu'il soit géré dans un nouveau processus.

```
int ret = fork();
```

Après avoir récupéré la valeur de retour du fork ou va gérer avec un if (ret == 0), le processus fils et dans le else le processus père et tout ça en boucle pour permettre à plusieurs clients de se connecter.
Dans chaque processus, on place un while(allumé) pour faire une boucle infinie, l'intérêt de placer une variable à la place d'un 1 est que nous pouvons éventuellement éteindre le serveur en changeant la variable à 0.
Il ne reste plus qu'à envoyer et réceptionner les messages en utilisant les fonctions recv/send ou read/write.
C'est à ce moment que le problème majeur s'est posé, le buffer servant d'intermédiaire pour les transmissions et affichage du message ne se vidait pas complètement et laissait le message précédent encore dans la variable buffer.



SOLUTION

Lors de l'envoi de message et de sa réception, le tableau de caractères retourné ne possède pas de caractère nul donc il n'est pas possible de connaître la vraie longueur de la chaîne.
Il suffit donc de rajouter ce caractère (\0) juste après l'utilisation de la fonction recv pour ainsi définir la fin du tableau. C'est pour cela que nous récupèrons le nombre d'octets retournés par recv pour pouvoir l'exploiter après pour rajouter ce caractère.

```
(read_size = recv(client_sock , client_message , MAX_READ , 0))
```

On utilise ensuite read_size comme indice pour ajouter ce caractère :

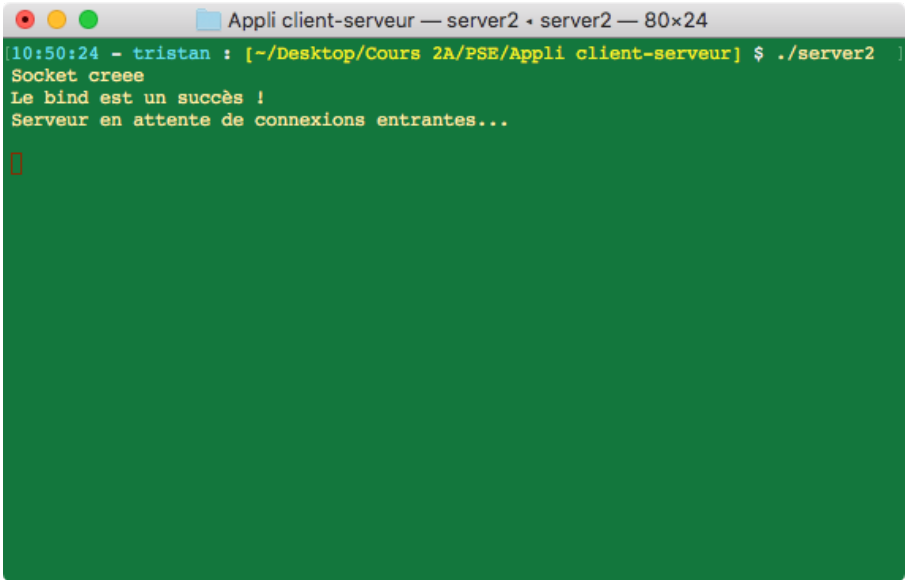
```
client_message[read_size] = '\0';
```

Il ne reste plus qu'à vider la chaîne en utilisant memset() :

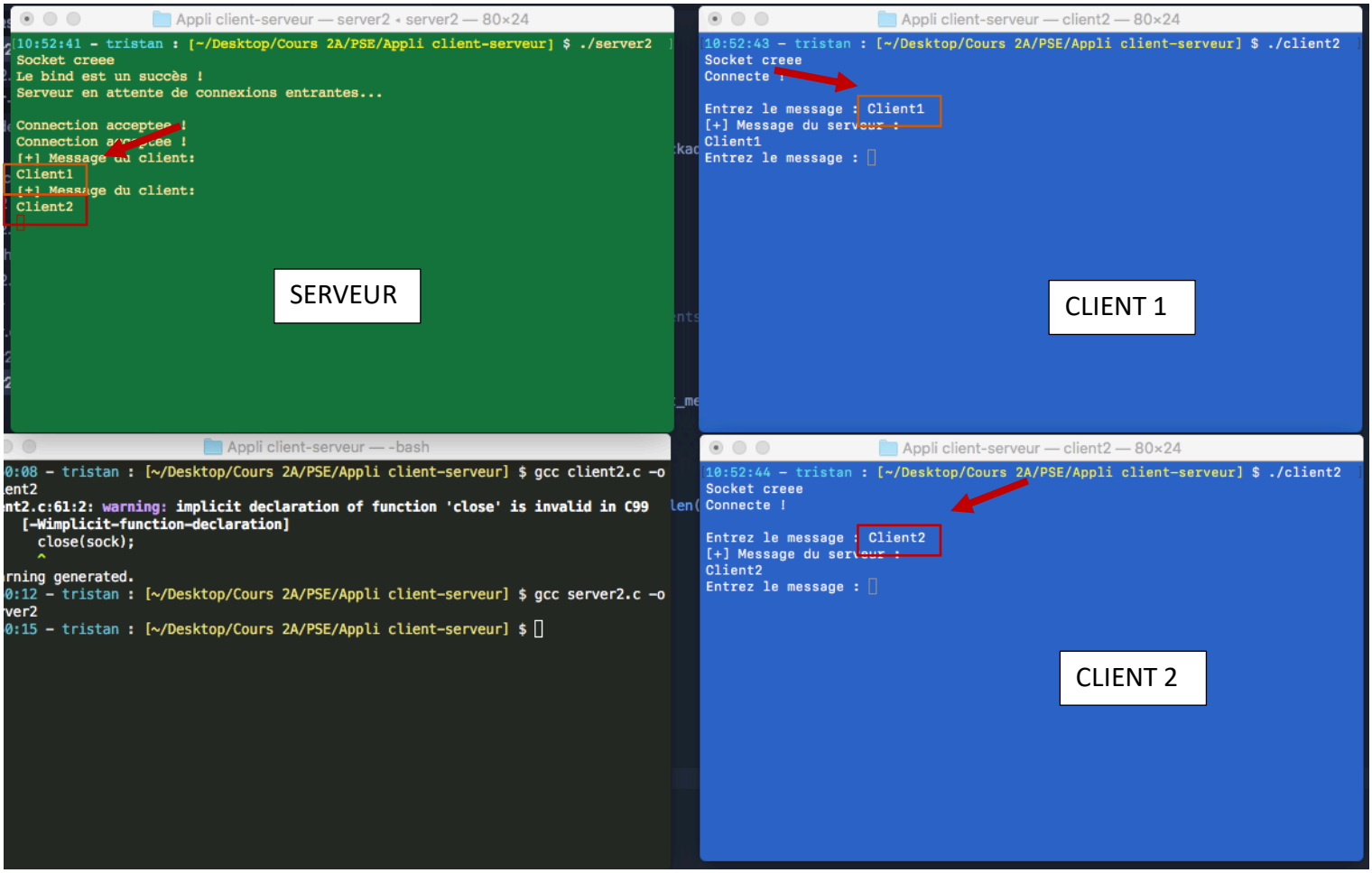
```
memset(client_message,'0', read_size);
```

Utilisation

Lancement du serveur



Connexion de 2 clients



Dans l'exemple précédent, j'ai ouvert 4 terminaux, 1 serveur, 2 clients et 1 permettant la compilation des programmes. La première étape est d'allumé le serveur (et non les clients en premier car ils se connecteraient à un serveur qui n'existe même pas), puis de lancer autant de clients que l'on désire.
Les messages envoyés par les clients d'affichent l'un après l'autre : le serveur gère donc plusieurs clients !

Améliorations

Une amélioration que j'avais envie d'ajouter à ce TP est que le serveur renvoie la chaîne de caractères envoyés par le client envoyeur. Dans la pratique, je vous l'accorde, cela n'a pas de grand intérêt mais ça m'a permis de me familiariser avec les échanges de données entre client et serveur en C. J'ai donc ajouté cette légère amélioration au programme.

Une autre amélioration que j'aurai aimé ajouter est que le serveur puisse envoyer un message qu'il intercepte à tous les clients qui y sont connectés. Pour cela, il faudrait pouvoir stocker les clients dans une structure client par exemple et ensuite créer une fonction qui envoie à tous les clients le message mis en paramètre. Malheureusement, les prochains partiels ne m'ont pas laissé le temps pour ajouter cette fonction au serveur…

Contenu de l'application

- Le fichier serveur.c (code du serveur)
- Le fichier client.c (code du client)
- Le fichier data.h (constantes de préprocesseur)
- Le fichier serveur (exécutable serveur)
- Le fichier client (exécutable client)

Compilation

```
gcc serveur.c –o serveur
gcc client.c –o client
```