TP 2 - Programmation Réseau et Système

3TC PRS - INSA Lyon

Mise en place d'un serveur TCP echo (suite)

Dans ce TP, vous allez poursuivre le développement du serveur et client TCP que vous avez commencé dans le TP1. Vous pouvez démarrer avec votre propre code ou, si vous n'avez pas fini le TP1, avec la solution proposée sur moodle.

Dans un premier temps, vous allez rajouter au serveur et au client la possibilité de prendre des arguments depuis la ligne de commande. A la fin de cette opération, vous devez pouvoir lancer vos programmes en utilisant les commandes :
./serveur <port_serveur>
./client <ip_serveur> <port_serveur>

Pour avoir accès aux paramètres de la lignes de commande dans un programme C, vous devez utiliser la déclaration complète de la fonction *main()* :
int main(int argc, char argv[])*

La variable *argc* récupère automatiquement le nombre d'arguments donnés sur la ligne de commande, tandis que le tableau *argv* permet l'accès aux valeurs saisies par l'utilisateur, dans un format chaîne de caractères. La manière classique d'utiliser ces variables est de vérifier d'abord si la valeur *argc* est celle attendue et d'exploiter ensuite les différentes variables *argv* (ou d'arrêter éventuellement le programme avec un message indiquant l'utilisation correcte).

Question 1 *Quelles sont les valeurs attendues pour la variable argc dans le cas du serveur et du client ? Quelle est la valeur de argv[0] ?*

Note : La fonction *atoi()*, définie dans l'en-tête *<stdlib.h>*, vous permet de convertir une chaîne de caractères dans l'entier correspondant, ce qui peut être très utile pour récupérer le numéro de port.

Lancez maintenant votre serveur sur le port 3456 et y connectez un client. Sans arrêter ce premier client, essayez de connecter un deuxième client au serveur.

Question 2 *Quel est le résultat ? Comment l'expliquez vous ?*

Pour résoudre ce problème et permettre la gestion concurrente de plusieurs clients, la solution la plus simple est de créer, du côté serveur, un nouveau processus pour chaque client connecté. La fonction *fork()* permet la création d'un processus fils, qui est au départ une copie identique du processus initial. En cas de réussite, *fork()* retourne des valeurs différentes dans le processus fils et le processus père, ce qui permet de distinguer entre les deux processus : une valeur de 0 est retournée dans le processus fils et une valeur supérieure à 0 dans le processus père.

Question 3 *Trouvez la valeur retournée par* fork() *dans le processus père. Que signifie cette valeur ?*

Question 4 *Dans le processus fils, affichez la valeur des deux sockets ouvertes (celle sur laquelle le serveur attend les connexions et celle créée par la fonction* accept()*). Comment expliquez-vous ces valeurs ?*

Une fois l'opération *fork()* réussie, dans le processus père vous devez fermer la socket de service créée par *accept()* et revenir en attente d'un nouveau client. Dans le processus

fils, vous devez fermer la socket de connexion et traiter le client sur la socket restant ouverte. Après avoir finalisé le traitement dans le processus fils, vous pouvez utiliser la fonction *exit()* pour arrêter ce processus.

> **Note :** La commande *fork()* n'est disponible que dans les systèmes d'exploitation compatible avec les standards POSIX. Par exemple, le système d'exploitation Windows ne respecte pas le standard POSIX, donc cette solution ne peut pas s'appliquer sur ce type de système.

Un serveur peut avoir besoin d'écouter sur des sockets multiples pour d'autres choses que recevoir et traiter des connexions. Par exemple, le serveur peut gérer une connexion TCP afin de se synchroniser avec d'autres serveurs et, en même temps, accepter des connexions TCP de la part des clients. Utiliser des processus parallèles à l'aide de *fork()* serait possible, mais avec un gros surcoût, surtout pour donner la priorité aux opérations de synchronisation face aux demandes des clients. Il faut trouver donc une autre solution pour que le serveur écoute en même temps sur plusieurs ports.

Question 5 *Est-ce qu'on peut associer deux ports différents à une même socket ? Quelles seraient les conséquences ?*

Question 6 *Que se passerait-il si on utilisait deux sockets différentes, en essayant d'accepter des connexions sur les deux ?*

L'API Sockets propose une manière de gérer des opérations sur plusieurs descripteurs de fichier (donc de socket) par le moyen de la fonction *select()* (définie dans les entêtes *<sys/select.h>* et *<sys/time.h>*). Cette fonction bloquante permet au processus d'attendre qu'un ou plusieurs événements se passent sur des descripteurs dans une liste donnée, avant de continuer avec le traitement de ces événements. Pour représenter un ensemble de descripteurs, *select()* utilise une structure spécifique : un set de descripteurs - **fd_set**. Ce set de descripteurs est représenté comme un tableau de valeurs binaires, où chaque élément du tableau correspond à un descripteur de fichiers. Les fonctions importantes liées à la structure **fd_set** sont :

- **void** *FD_ZERO(fd_set* fdset)* - remet à zéro la totalité du tableau ;
- **void** *FD_SET(int fd, fd_set* fdset)* - active le bit associé au descripteur *fd* ;
- **void** *FD_CLR(int fd, fd_set* fdset)* - désactive le bit associé au descripteur *fd* ;
- **int** *FD_ISSET(int fd, fd_set* fdset)* - vérifie si le bit associé au descripteur *fd* est activé.

La fonction *select()* prend 5 paramètres :

- **int** *maxfdp* - spécifie le nombre de descripteurs à tester. Faites attention, la fonction va tester tous les descripteurs entre 0 et *maxfdp*-1, même ceux qui ne vous intéressent.
- **fd_set*** *readset* - donne les descripteurs pour lesquels vous voulez tester une écriture.
- **fd_set*** *writeset* - donne les descripteurs pour lesquels vous voulez tester une lecture.
- **fd_set*** *exceptset* - donne les descripteurs pour lesquels vous voulez tester une exception, par exemple la réception de données hors-bande sur une socket.
- **struct timeval*** *timeout* - définit le temps maximal d'attente pour la fonction. Si la valeur de ce paramètre est une structure *timeval* initialisée à zéro, *select()* ne bloque pas et il s'agit d'une opération de *polling*. Au contraire, si vous ne voulez pas définir de timer et que vous ne voulez débloquer le processus que dans le cas d'une opération d'entrée-sortie, il suffit de donner comme un argument un pointeur *NULL*.

Le même principe s'applique aussi aux paramètres de type *fd_set* : si vous ne voulez pas tester des écritures, des lectures ou des exceptions, vous pouvez utilisez le pointeur *NULL*. Les trois paramètres *fd_set* sont des paramètres-résultat, modifiés par *select()*. Dans le *fd_set* passé comme argument à la fonction, vous devez activer tous les descripteurs dans lesquels vous êtes intéressés. Par contre, dans le résultat, seulement le(s) descripteur(s) sur le(s)quel(s) une activité a été détectée resteront activés.

Question 7 *Donnez une fonction équivalente à un appel* select() *avec les trois paramètres* fd_set *à* NULL *et le paramètre* timeval *initialisée à 5 secondes.*

Pour tester *select()*, vous devez définir une deuxième socket d'écoute dans votre serveur, sur un port différent de celui associé à la première socket et en utilisant cette fois-ci le protocole UDP (vous pouvez lire ce port sur la ligne de commande, comme le précédent). Déclarez aussi un set de descripteurs et initialisez le à zéro. Le pas suivant est d'activer dans la variable *fd_set* les bits correspondant aux deux sockets qui vous intéressent.

Question 8 *Est-ce que vous allez effectuer cette opération d'activation à l'intérieur ou à l'extérieur de la boucle infinie du serveur ?*

Vous allez ensuite modifier la structure du serveur. Au lieu de faire directement un *accept()* pour une demande de connexion arrivant sur la première socket, vous allez attendre les demandes en appelant la fonction *select()*.

Question 9 *Quelle sera la valeur du paramètre* timeval *pour être équivalents au fonctionnement préalable, qui utilisait* accept() *?*

Une fois que *select()* annonce une écriture sur une des sockets, vous allez vérifier quelle socket a été activé. S'il s'agit de la socket TCP, vous allez accepter la connexion et traiter le client dans un nouveau processus. Si la socket UDP est activée, vous allez simplement afficher le message envoyé par le client UDP (qu'il ne faut pas oublier de programmer).

Question 10 *Donnez un squelette du code utilisée dans la boucle infinie du serveur.*

Note : Même si l'utilisation de *select()* est souvent présentée comme une alternative à *fork()*, surtout pour des systèmes d'exploitation qui n'implémente pas ce dernier appel système, cet exemple vous montre que les deux fonctions peuvent être complémentaires.