Université de Strasbourg
UFR Mathématique et Informatique

Licence 3 Informatique
Année 2011/2012

Réseaux et Protocoles

TP01 : Introduction aux sockets

Les deux premiers exercices du TP nécessitent de récupérer l'archive suivante :

- clarinet.u-strasbg.fr/~iova/files/tp1_src.tar.gz

Pour ouvrir cette archive, consultez le manuel de la commande *tar*.

1 Transferts UDP

Dans cet exercice nous allons écrire deux petits clients UDP qui s'échangent une chaîne de caractères. L'archive que vous avez récupéré précédemment contient le répertoire *udp* qui comporte les fichiers *sender-udp.c* et *receiver-udp.c*. Le fichier *sender-udp.c* contient un squelette en C d'un petit client UDP qui envoie une chaîne de caractères à l'adresse IPv4 et au numéro de port spécifiés sur la ligne de commande au lancement du programmme. De même, le fichier *receiver-udp.c* contient un squelette en C d'un petit client UDP qui réceptionne une chaîne de caractères et l'affiche à l'écran. Le numéro de port utilisé par le recepteur doit être passé en paramètre au lancement du programme.

Exercices :

- A l'aide du polycopié sur les sockets distribué en TD, analysez le code C des fichiers *sender-udp.c* et *receiver-udp.c* : structure générale du code, fonctions utilisées, etc.
- Ajoutez le code manquant dans les fichiers *sender-udp.c* et *receiver-udp.c* pour qu'ils soient fonctionnels (includes, paramètres des fonctions, etc.).
- Après avoir compilé les deux programmes à l'aide du Makefile, lancez le récepteur (*receiver-udp*) puis l'expéditeur (*sender-udp*) avec les bons paramètres et vérifiez que la chaîne envoyée par l'expéditeur est correctement affichée du côté du récepteur.

2 Transferts TCP

Dans cet exercice nous allons écrire un client et un serveur TCP qui s'échangent une chaîne de caractères. L'archive que vous avez récupéré précédemment contient également le répertoire *tcp* qui comporte les fichiers *client-tcp.c* et *server-tcp.c*. Le fichier *client-tcp.c* contient un squelette en C d'un petit client TCP qui se connecte à un serveur TCP et lui envoie une chaîne de caractères. L'adresse IPv4 et le numéro de port du serveur TCP doivent être passés en paramètre au lancement du programme. De même, le fichier *server-tcp.c* contient un squelette en C d'un petit serveur TCP qui réceptionne une chaîne de caractères (après connexion d'un client) et l'affiche à l'écran. Le numéro de port utilisé par le serveur doit être passé en paramètre au lancement du programme.

Exercices :

- A l'aide du polycopié sur les sockets distribué en TD, analysez le code C des fichiers *client-tcp.c* et *server-tcp.c* : structure générale du code, fonctions utilisées, etc.
- Ajoutez le code manquant dans les fichiers *client-tcp.c* et *server-tcp.c* pour qu'ils soient fonctionnels (includes, paramètres des fonctions, etc.).

– Après avoir compilé les deux programmes à l'aide du Makefile, lancez le serveur (*server-tcp*) puis le client (*client-tcp*) avec les bons paramètres et vérifiez que la chaîne envoyée par le client est correctement affichée du côté du serveur.

3 IPv6

Modifier les exercices 1 et 2 pour qu'ils fonctionnent avec des adresses IPv6.

4 Dialogue avec un serveur existant

On souhaite réaliser un petit navigateur web (très simplifié) en mode texte. Pour ce faire, vous allez réaliser en langage C un client TCP. Ce client devra pouvoir se connecter en TCP sur le serveur web dont le nom (e.g. www.google.fr) sera passé sur la ligne de commande au lancement du programme. Une fois connecté, votre programme devra récupérer et afficher dans le terminal la page web qui aura été spécifiée sur la ligne de commande (e.g. index.html). Pour effectuer la résolution de nom, vous devrez utiliser la fonction *getaddrinfo()* (la fonction *gethostbyname()* est obsolète).

On rappelle que les échanges de page web se font à l'aide du protocole HTTP (pour *Hyper Text Transfer Protocol*). Le port de communication par défaut d'un serveur web est le port numéro 80. La spécification du protocole HTTP se trouve dans le RFC (Request For Comments) 2068. Ce RFC est disponible à l'adresse suivante :
http://ietf.org/rfc.html.

Votre client ne devra supporter que la commande *GET* du protocole HTTP.

La commande *telnet m p* établie une connexion TCP sur le port *p* de la machine *m* permettant alors de tester le protocole HTTP.

5 Talk multi-utilisateurs

On souhaite réaliser un système client/serveur permettant de communiquer à plusieurs simultanément sur le réseau. Le programme client devra au préalable se connecter au serveur, puis récupérer les chaînes de caractères entrées par l'utilisateur sur l'entrée standard et les envoyer au serveur. A la réception d'une chaîne de caractères, le serveur doit prendre en charge l'envoi de cette chaîne à tous les clients connectés au serveur, sauf au client dont la chaîne est originaire. L'implémentation se fera à l'aide du protocole TCP en IPv6.

Exercices :

– Reprennez les fichiers de l'exercice 2, ils vont vous servir de base pour la création d'un client / serveur TCP.
– Le serveur va devoir écouter sur plusieurs sockets à la fois (une socket par client). Modifiez le programme *server-tcp* pour gérer la connexions de multiples clients. Il faut notamment définir une structure pour stocker les informations relatives à un client : paramètres réseaux et socket.
– Maintenant que le serveur accepte de multiples clients, il faut également qu'il puisse écouter simultanément sur les différentes sockets associées aux clients. Pour cela nous allons utiliser la fonction *select*. En vous aidant du polycopié distribué en TD et à l'aide du manuel utilisateur (*man select*), ajoutez dans le programme *server-tcp* la fonction *select* de manière à ce que le serveur écoute simultanément sur les différentes sockets associées aux clients.
– Pour finaliser le serveur, ajoutez le traitement que ce dernier doit réaliser lorsqu'il reçoit un message d'un des clients, c-à-d lorsque la fonction *select* se débloque, identifiez quel client a envoyé un message, récupérez ce message, et envoyez-le aux autres clients.

– Nous allons maintenant nous intéresser au client. Un client doit gérer deux événements : la réception d'un message du serveur et son affichage sur l'écran, et la réception d'un message tapé par l'utilisateur sur la ligne de commande et son envoi au serveur. Ces deux événements doivent être gérés simultanément, nous allons donc utiliser la fonction *select*. Modifiez le programme *client-tcp.c* en ajoutant la fonction *select* de manière à ce que le client écoute simultanément sur sa socket de communication avec le serveur et sur l'entrée standard.
– Pour finaliser le client, ajoutez le traitement que ce dernier doit réaliser lorsqu'il reçoit un événement, c-à-d lorsque la fonction *select* se débloque, identifiez si le client a reçu un message du serveur et dans ce cas l'afficher, ou si l'utilisateur a entré une chaîne de caractères et dans ce cas l'envoyer au serveur.
– Lancez un serveur et plusieurs clients et vérifiez qu'un message envoyé par un client est bien reçu par les autres clients connectés au serveur.

Fonctions utiles pour la conversion d'adresse

– `int inet_pton(int family, const char *src, void *dst);`

  La fonction *inet_pton()* permet de convertir une adresse dans sa représentation classique (*a.b.c.d* pour IPv4 ou *: :* pour IPv6) vers sa représentation binaire. Cette fonction peut être utilisée pour convertir des adresses IPv4 et IPv6. Le premier paramètre permet justement de préciser le type d'adresse. Le deuxième paramètre est un pointeur vers une adresse mémoire qui contient la chaîne de caractères qui représente l'adresse IP dans sa forme classique. Enfin le dernier paramètre est un pointeur sur la structure d'adresse réseau qui contiendra l'adresse IP convertie. L'espace mémoire pointé par le dernier paramètre doit déjà être alloué avant l'appel de la fonction *inet_pton()*. La fonction *inet_pton()* renvoie 1 en cas de succès, 0 ou -1 dans le cas d'une erreur (cf. voir le manuel utilisateur pour plus de détails).

– `const char *inet_ntop(int family, const void *src, char *dst, size_t cnt);`

  La fonction *inet_ntop()* permet d'effectuer la conversion inverse, c'est-à-dire la conversion d'une adresse IP de sa forme binaire vers sa forme classique. Le premier paramètre permet de préciser le type d'adresse. Le deuxième paramètre est un pointeur vers l'adresse IP dans sa forme binaire. Le troisième paramètre est un pointeur vers de l'espace mémoire qui contiendra l'adresse IP convertie dans sa forme classique. Enfin, le dernier paramètre permet de préciser la taille de l'espace mémoire pointé par le troisième paramètre. On pourra utiliser la maccro **INET_ADDRSTRLEN** pour une adresse IPv4 et la maccro **INET6_ADDRSTRLEN** pour une adresse IPv6. La fonction *inet_ntop()* renvoie un pointeur non nul en cas de succès et un pointeur nul en cas d'erreur.