Programmation réseau sous Linux (II)

Cédric Meuter, Eythan Levy

18 novembre 2004

1 Rappels

Vous trouverez ci-dessous les déclarations des différentes fonctions dont vous pourriez avoir besoin dans ce TP. Pour plus de détails, vous êtes renvoyés aux transparents de la séance précédente, et aux pages de manuel (man pages).

```
int socket(int domain, int type, int protocol);

int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen);

int listen(int s, int backlog);

int accept(int s, struct sockaddr *addr, socklen_t *addrlen);

int  connect(int  sockfd,  const  struct sockaddr *serv_addr, socklen_t
             addrlen);

ssize_t send(int s, const void *buf, size_t len, int flags);

ssize_t recv(int s, void *buf, size_t len, int flags);

ssize_t read(int fd, void *buf, size_t count);

ssize_t write(int fd, const void *buf, size_t count);

int gethostname(char *name, size_t len);

struct hostent *gethostbyname(const char *name);

int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
           struct timeval *timeout);
```

Les connexions de socket côté serveur doivent faire, dans l'ordre, un `socket()`, un `bind()`, un `listen()`, et un `accept()`. Le client devra quant à lui, effectuer un `socket()` puis un `connect()`. Une fois la connexion établie, les deux processus pourront communiquer à l'aide de `send()` et `recv()` ou de `read()` et `write()`.

2 Exercices

2.1 Exercice 1: nom et adresse IP

Ecrivez un programme qui affiche l'adresse IP et le nom de la machine locale.

2.2 Exercice 2: communication réseau de base

Ecrivez deux programmes communiquant par socket. Nous appellerons "client" le programme qui effectuera le `connect`, et "serveur" celui qui effectuera l'`accept`.

Une fois la connexion établie, le serveur affichera l'adresse IP du client, puis lui enverra un message "bonjour client!", que le client affichera.

Le serveur et le client termineront ensuite leur exécution.

Vous ferez d'abord tourner les deux programmes sur la machine locale, puis vous ferez une expérience distribuée sur deux machines avec l'un de vos voisins.

2.3 Exercice 3: client/serveur simple

Enrichissez le programme serveur de l'exercice précédent, de manière à ce qu'il corresponde au modèle de serveur permanent vu au cours. Votre serveur devra donc pouvoir accepter un nombre indéfini de connexions en cyclant infiniment, et forkera à chaque connexion en laissant à son fils le soin de répondre "bonjour client!" au client qui vient de se connecter. Nous demandons de plus qu'à chaque connexion le serveur affiche l'adresse IP du client avant de forker.

2.4 Exercice 4: client/serveur à deux services

Enrichissez le programme serveur de l'exercice précédent, de manière à ce qu'il puisse offrir deux services sur deux ports différents.

Le premier service correspondra à celui de l'exercice précédent (répondre "bonjour client!" aux clients qui se connectent). Le second service correspondra à lire deux entiers longs (32 bits) sur le socket, les additionner, puis envoyer le résultat à la machine distante.

Les services seront proposés en parallèle, grâce à la commande `select`.

A chaque connexion, le serveur affichera un message donnant l'adresse IP du client et le numéro du service demandé (1="bonjour", 2=calculatrice).

Note: la fonction `atol()` vous permet de convertir une chaîne de caractères représentant un entier en un `long int`.

2.5 Exercice 5: download de pages web

Le protocole http permet de communiquer avec un serveur web pour lui demander une page html, ou même un autre type de fichier. C'est ce protocole qui est utilisé par les naviguateurs web. Le protocole est très simple: afin d'obtenir un fichier dont le chemin d'accès est "a", sur un serveur web dont le nom est "b", il faut se connecter sur le port 80 du serveur, et lui envoyer la chaîne suivante:

```
GET a HTTP/1.1\r\nHost: b\r\nConnection: close\r\n\r\n
```

Pour la page du tp d'archi: `www.ulb.ac.be/di/ssd/cmeuter/archi/index.htm` le nom du serveur serait donc `www.ulb.ac.be` et le chemin d'accès serait `di/ssd/cmeuter/archi/index.htm`
Note: si votre URL consiste juste en un nom d'hôte, vous pouvez spécifer "/" pour le chemin d'accès.

Nous vous demandons d'écrire un petit programme qui prend en paramètre une URL, se connecte sur le serveur web correspondant, lui envoie la commande http adéquate, reçoit le fichier (html ou autre) demandé, puis le sauvegarde dans un fichier local.

S'il s'agit d'un fichier html, vous pouvez vérifier ensuite que le fichier a bien été reçu en l'ouvrant avec netscape (ou mozilla).