Programmation socket - notes de cours ++

Prérequis

Afin que deux applications communiquent, il faut que les machines qui les exécutent soient accessibles :

- même réseau
- réseaux joignables
- penser à pinguer les machines entre elles (commande ping).

Fonctions utiles

Socket : créer un point de communication

Une **socket** est un point de communication, on pourra relier cette **socket** à un couple (adresse IP,port) de la machine sur laquelle s'exécute le futur logiciel communiquant que l'on écrit.
Socket retourne un **entier**, un **descripteur, -1 si erreur**.

#include <sys/types.h>
#include <sys/socket.h>
int socket(int *domain*, **int** *type*, **int** *protocol*);

Exemple (vu ci-après) :

```
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1)
{
    perror("socket");
    exit(1);
}
```

Source : http://manpagesfr.free.fr/man/man2/socket.2.html

Bind : fournir un nom à une socket

#include <sys/types.h>
#include <sys/socket.h>
int bind(int *sockfd*, **const struct sockaddr** **addr*, **socklen_t** *addrlen*);

Quand une socket est créée avec l'appel système **socket**(2), elle existe dans l'espace des noms mais n'a pas de nom assigné). **bind**() affecte l'adresse spécifiée dans *addr* à la socket référencée par le descripteur de fichier *sockfd*. *addrlen* indique la taille, en octets, de la structure d'adresse pointée par *addr*. Traditionnellement cette opération est appelée « affectation d'un nom à une socket ».

Il est normalement nécessaire d'affecter une adresse locale avec **bind**() avant qu'une socket **SOCK_STREAM** puisse recevoir des connexions ***(accept)***. **Bind** retourne 0 ok, -1 erreur.

Source : http://manpagesfr.free.fr/man/man2/bind.2.html. Exemple (vu ci-après) :

```
if (bind(sock, (struct sockaddr *)&local, sizeof(struct sockaddr)) == -1)
{
    perror("bind");
    exit(1);
}
```

Listen : attendre des connexions sur une socket

#include <sys/types.h>
#include <sys/socket.h>

int listen(int *sockfd*, **int** *backlog*);

listen() marque la socket référencée par *sockfd* comme une socket passive, c'est-à-dire comme une socket qui sera utilisée pour accepter les demandes de connexions entrantes en utilisant **accept**(2).

L'argument ***sockfd*** est un descripteur de fichier qui fait référence à une socket de type **SOCK_STREAM** ou **SOCK_SEQPACKET.**

L'argument ***backlog*** définit une longueur maximale jusqu'à laquelle la file des connexions en attente pour *sockfd* peut croître. Si une nouvelle connexion arrive alors que la file est pleine, le client reçoit une erreur indiquant **ECONNREFUSED**, ou, si le protocole sous-jacent supporte les retransmissions, la requête peut être ignorée afin qu'un nouvel essai réussisse. listen retourne -1 quand une erreur survient.

Exemple (vu ci-après) :

```
if (listen(sock, 5) == -1)
{
    perror("listen");
    exit(1);
}
```

Accept : accepter une connexion sur une socket

#include <sys/types.h>
#include <sys/socket.h>

int accept(int *sockfd*, **struct sockaddr** **adresse*, **socklen_t** **longueur*);

L'appel système **accept**() est employé avec les sockets utilisant un protocole en mode connecté (**SOCK_STREAM** et **SOCK_SEQPACKET**). Il extrait la première connexion de la file des connexions en attente de la socket *sockfd* à l'écoute, ***crée une nouvelle socket connectée, et renvoie un nouveau descripteur de fichier qui fait référence à cette socket***. La nouvelle socket n'est pas en état d'écoute. La socket originale *sockfd* n'est pas modifiée par l'appel système.

L'argument *sockfd* est une socket qui a été créée avec la fonction **socket**(2), attachée à une adresse avec **bind**(2), et attend des connexions après un appel **listen**(2). accept retourne -1 quand une erreur survient.

Exemple (vu ci-après) :

```
if ((socket2 = accept(sock, (struct sockaddr *)&distant, &lg)) == -1)
{
    perror("accept");
    exit(1);
}
```

Close : fermer un descripteur de fichier

#include <unistd.h>

int close(int *fd*);

Permet de fermer une socket (très important) !

Read : lire depuis un descripteur de fichier

#include <unistd.h>

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

read() lit jusqu'à *count* octets depuis le descripteur de fichier *fd* dans le tampon pointé par *buf.*

Si *count* vaut zéro, **read**() renvoie zéro et n'a pas d'autres effets. Si *count* est supérieur à **SSIZE_MAX**, le résultat est indéfini.

read() renvoie -1 s'il échoue, auquel cas *errno* contient le code d'erreur, et la position de la tête de lecture est indéfinie. Sinon, **read**() renvoie le nombre d'octets lus (0 en fin de fichier), et avance la tête de lecture de ce nombre. Le fait que le nombre renvoyé soit plus petit que le nombre demandé n'est pas une erreur. Ceci se produit à la fin du fichier, ou si on lit depuis un tube ou un terminal, ou encore si **read**() a été interrompu par un signal.

Exemple (vu ci-après) :

```
read(socket2, mess, 80);
```

Write : écrire dans un descripteur de fichier

#include <unistd.h>

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

write() écrit jusqu'à *count* octets dans le fichier associé au descripteur *fd* depuis le tampon pointé par *buf*.

Le nombre d'octets écrits peut être inférieur à *count* si, par exemple, la place disponible sur le support physique concerné est insuffisante, ou si la limite de ressource **RLIMIT_FSIZE** a été atteinte (voir **setrlimit**(2)), ou si l'appel a été interrompu par un gestionnaire de signal avant que les *count* octets aient été écrits. (Voir également **pipe**(7).)

Pour un fichier positionnable (c'est-à-dire un fichier sur lequel on peut appliquer **lseek**(2), par exemple, un fichier ordinaire) l'écriture s'effectue à la position courante de la tête de lecture, et la tête de lecture est déplacée du nombre d'octets effectivement écrits. Si le fichier a été ouvert avec l'attribut **O_APPEND** de **open**(2), la tête de lecture est d'abord positionnée à la fin du fichier avant que l'écriture ne commence. Le déplacement de la tête de lecture et l'opération d'écriture sont effectués en une étape atomique.

POSIX réclame qu'une lecture avec **read**(2) effectuée après le retour d'une écriture avec **write**(), renvoie les nouvelles données. Notez que tous les systèmes de fichiers ne sont pas compatibles avec POSIX.

write() renvoie le nombre d'octets écrits (0 signifiant aucune écriture), ou -1 s'il échoue, auquel cas *errno*

contient le code d'erreur. Si *count* vaut zéro, et si *fd* est associé à un fichier normal, **write**() peut renvoyer un code d'erreur si l'une des erreurs ci-dessous est détectée. Si aucune erreur n'est détectée, 0 sera renvoyé sans effets de bord. Si *count* vaut zéro, et si *fd* est associé à autre chose qu'un fichier ordinaire, les résultats sont indéfinis.

Exemple (vu ci-après) :

```
    write(socket2, "message recu !", 80);
```

Structures utilisées

sockaddr_in et in_addr

```
#include <netinet/in.h>

struct sockaddr_in {
   short          sin_family;   // e.g. AF_INET
   unsigned short   sin_port;     // e.g. htons(3490)
   struct in_addr   sin_addr;     // see struct in_addr, below
   char           sin_zero[8];  // zero this if you want to
};

struct in_addr {
   unsigned long s_addr;  // load with inet_aton()
};

=> Comment utiliser les structures ?
struct sockaddr_in local;
=> permettra d'avoir accès à
local.sin_family ⇒ =AF_INET           (constante)
local.sin_port    ⇒ =htons(PORT)      hexa to non signed short
local.sin_addr    ⇒ =INADDR_ANY    (constante)
local.sin_zero    ⇒ bzero(&(local.sin_zero),8)   bzero nécessite l'adresse de la
                              variable et donc on utilise le &, on met les 8 char à 0
```

Exemples

Socket bindé fermée.

On prépare le terrain : une socket sock, on la bind à l'adresse locale puis on ferme la socket.

```
   #include <stdio.h>
   #include <stdlib.h>
   #include <errno.h>
   #include <string.h>
   #include <netinet/in.h>
   #include <sys/socket.h>
   #include <sys/types.h>
   #include <strings.h>
   #include <unistd.h>

   #define PORT 12345
   int sock;
   bzero(&local, sizeof(local));       // mise a zero de la zone adresse
   local.sin_family = AF_INET;         // famille d adresse internet
   local.sin_port = htons(PORT);       // numero de port
   local.sin_addr.s_addr = INADDR_ANY; // types d adresses prises en charge
   bzero(&(local.sin_zero), 8);        // fin de remplissage

   lg = sizeof(struct sockaddr_in);
   // creation socket du serveur mode TCP/IP la fonction socket retourne un entier
   // que l'on place dans la variable sock
   if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1)
   {
       perror("socket");
       exit(1);
   }

   // nommage de la socket - on affecte le numero de port 12345
   if (bind(sock, (struct sockaddr *)&local, sizeof(struct sockaddr)) == -1)
   {
       close(sock);
       perror("bind");
       exit(1);
   }
}
int main(){
creer_socket();
close(sock);
return 0;
}
```

Serveur ⇒ Socket bindé à l'écoute accepte messages jusqu'à "fin" et stop quand ctrl-c, fermée

```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <strings.h>
#include <unistd.h>

#define PORT 12345
int sock, socket2, lg;
char mess[80];
struct sockaddr_in local;   // champs d entete local
struct sockaddr_in distant; // champs d entete distant

void creer_socket()
{                                       // preparation des champs d entete de la socket serveur
   bzero(&local, sizeof(local));       // mise a zero de la zone adresse
   local.sin_family = AF_INET;         // famille d adresse internet
   local.sin_port = htons(PORT);       // numero de port
   local.sin_addr.s_addr = INADDR_ANY; // types d adresses prises en charge
   bzero(&(local.sin_zero), 8);        // fin de remplissage

   lg = sizeof(struct sockaddr_in);
   // creation socket du serveur mode TCP/IP
   if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1)
   {
       perror("socket");
       exit(1);
   }

   // nommage de la socket
   if (bind(sock, (struct sockaddr *)&local, sizeof(struct sockaddr)) == -1)
   {
       close(sock);
       perror("bind");
       exit(1);
   }
}

void detourneCtrlC()
{
```

```
   close(sock);
   close(socket2);
   exit(0);
}

int main()
{
   // creation socket
   creer_socket();

   signal(SIGINT, &detourneCtrlC);

   // mise a l ecoute de la socket
   if (listen(sock, 5) == -1)
   {
       close(sock);
       perror("listen");
       exit(1);
   }

   // boucle sans fin pour la gestion des connexions
   while (1)
   { // attente connexion client
       printf("En  attente d un client, accept est bloquant\n");
       if ((socket2 = accept(sock, (struct sockaddr *)&distant, (socklen_t *)&lg)) == -1)
       {
           perror("accept");
           exit(1);
       }
       printf("client connecte \n");
       strcpy(mess, "");
       while (strncmp(mess, "fin", 3) != 0)
       {
           read(socket2, mess, 80);
           printf("le client me dit %s \n", mess);
           write(socket2, "message recu !", 80);
       }
       close(socket2); // on lui ferme la socket
   }
   exit(0);
}
```

Client ⇒ le client initialise une socket, se connecte au serveur, dialogue jusqu'à la saisie de "fin".

```c
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>
#include <fcntl.h>
#include <string.h>     // chaines de caracteres
#include <sys/socket.h> // interface socket
#include <netinet/in.h> // gestion adresses ip
#include <sys/types.h>
#include <unistd.h>

#define SERV "127.0.0.1" // adresse IP = boucle locale
#define PORT 12345       // port d'ecoute serveur
int port, sock;          // n°port et socket
char mess[80];           // chaine de caracteres

struct sockaddr_in serv_addr; // zone adresse
struct hostent *server;       // nom serveur

void creer_socket()
{
   port = PORT;
   server = gethostbyname(SERV); // verification existance adresse
   if (!server)
   {
       fprintf(stderr, "Problème serveur \"%s\"\n", SERV);
       exit(1);
   }
   // creation socket locale
   sock = socket(AF_INET, SOCK_STREAM, 0); // AF_INET=famille adresse internet
                                           // SOCK_STREAM= mode connecte-TCP
   bzero(&serv_addr, sizeof(serv_addr));   // preparation champs entete
   serv_addr.sin_family = AF_INET;         // Type d'adresses
   bcopy(server->h_addr, &serv_addr.sin_addr.s_addr, server->h_length);
   serv_addr.sin_port = htons(port); // port de connexion du serveur
}

int main()
{ // creation socket
   creer_socket();
   // connexion au serveur
   if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
```

```
    {
        perror("Connexion impossible:");
        close(sock);
        exit(1);
    }
    printf("connexion avec serveur ok\n");
    // dialogue avec le serveur
    strcpy(mess, "");
    while (strncmp(mess, "fin", 3) != 0)
    {
        printf("Question : ");
        gets(mess); //depuis entrée standard
        write(sock, mess, 80); //envoi du message au serveur
        read(sock, mess, 80);  //attente de la réponse
        printf("Reponse : %s\n", mess); //affichage message recu !
    }

    close(sock);
}
```

Todo : tester sur deux machines différentes

⇒ lancer le serveur
⇒ lancer le client, saisir coucou puis fin
⇒ killer le serveur avec un ctrl-c
⇒ relancer le serveur, y a-t-il le message ***bind: Address already in use*** ?