Programmation sockets
Prototypes de fonctions et structures de données

Le domaine AF_INET

On ne détaillera pas ici le contenu des fichiers headers à inclure dans le code. En règle générale, il faut simplement inclure :

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>              // pour la résolution de noms
```

Structures de données: sockaddr et sockaddr_in

```
struct sockaddr {
        unsigned short      sa_family;    // Famille d'adresse, AF_xxx
        char                sa_data[14];  // 14 octets
 };
```

sa_data contient le port et l'adresse de la socket. En pratique on utilise la structure sockaddr_in qui facilite l'accès aux différents champs. Les pointeurs vers ces deux types de structure sont "interchangeables".

```
struct sockaddr_in {
        short int           sin_family;   // AF_INET
        unsigned short int  sin_port;     // Numéro de port
        struct in_addr      sin_addr;     // Adresse IP
        unsigned char       sin_zero[8];  // 8 caractères nuls
};

struct in_addr {
        unsigned long       s_addr;       // Adresse IP de 4 octets
};
```

Remarque:

Les champs *sin_port* et *sin_addr* doivent être en format "Network Byte Order". On utilisera les fonctions **htons, htonl, ntohs, et ntohl** pour faire les conversions nécessaires.

Quelques fonctions bien utiles:

in_addr_t **inet_addr**(const char *cp);

Transforme l'adresse IP donnée au format "a.b.c.d" (cp) en un entier long non−signé (valeur de retour de la fonction).

int **inet_aton**(const char *cp, struct in_addr in);

Transforme l'adresse IP donnée au fomat "a.b.c.d" (cp) en une structure *in_addr* (in).

char * **inet_ntoa**(struct in_addr in);

Transforme une adresse IP du format *struct in_addr* au format "a.b.c.d" (valeur de retour).

Exemple:

```
struct sockaddr_in ma_socket;

ma_socket.sin_family = AF_INET;
ma_socket.sin_port = htons(4785);                    // conversion host −> network
ma_socket.sin_addr.s_addr = inet_addr("130.79.44.193");   // ou inet_aton( "130.79.44.193", &(ma_socket.sin_addr) );
memset( &(ma_socket.sin_zero), '\0', 8);    // on positionne les 8 octets à zéro

printf("%s", inet_ntoa(ma_socket.sin_addr) );          // affiche l'adresse IP au format a.b.c.d
```

La résolution de noms

Les mécanismes de résolution de noms de machines permettent d'associer un nom de machine (www.yahoo.fr) à une adresse IP (217.12.3.11).

```
struct hostent {
        char    *h_name;        // nom officiel
        char    **h_aliases;    // noms alternatifs (aliases)
        int     h_addrtype;     // type d'adresse (habituellement AF_INET)
        int     h_length;       // longueur de l'adresse en octets
        char    **h_addr_list;  // adresses alternatives en Network Byte Order
};
#define h_addr h_addr_list[0]          // la première adresse de h_addr_list
```

struct hostent * **gethostbyname**(char *name);

Cette fonction permet d'obtenir via les mécanismes de résolution de noms (DNS) l'adresse IP d'une machine à partir de son nom.

int **getpeername**(int sockfd, struct sockaddr *addr, int *addrlen);

Permet d'obtenir les informations relatives à la socket *sockfd*. Ces informations sont stockées à l'emplacement de la structure *addr*.

Exemple:

```
struct hostent *info_yahoo;
int i = 0;

info_yahoo = gethostbyname( "www.yahoo.fr" );

printf("L'adresse IP de %s est %s", info_yahoo−>h_name, inet_ntoa( * ((struct in_addr *)info_yahoo−>h_addr )));

/* affichage des aliases */

while( *(pc_ip−>h_aliases + i) != NULL )
{
        printf("\n\tAlias: %s", *(pc_ip−>h_aliases + i) );
        i++;
}
```

Les fonctions de programmation des sockets

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

Déclaration d'une socket: domain = AF_INET, type = SOCK_STREAM ou SOCK_DGRAM, protocol = 0.

Dans tous les prototypes de fonctions ci−dessous, *sockfd* est un descripteur de socket obtenu lors de l'appel de la fonction socket().

int **bind**(int sockfd, struct sockaddr *sock_info, int addrlen);

Avant de faire un listen(), associe un port à la socket *sockfd*. Si sock_info.sin_port = htons(0), un port libre sera choisi aléatoirement. Si sock_info.sin_addr.s_addr = htonl(INADDR_ANY), l'adresse IP est automatiquement positionnée avec l'adresse de la machine (*addrlen* = sizeof(struct sockaddr)).

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

La socket *sockfd* est en attente de connexions entrantes. L'entier *max_number* spécifie le nombre maximal de connexions pouvant être mise en file d'attente avant d'être traitées par accept().

int **accept**(int sockfd, struct sockaddr *sock_info, int addrlen);

Une connexion en attente de traitement après un listen() doit être accept()ée. Sockfd est la socket passée à la fonction listen(). La structure *sock_info* contiendra les informations de la connexion entrante et *addrlen* = sizeof(struct sockaddr). ATTENTION: accept() retourne un nouveau descripteur de socket qui devra être utilisé pour traiter la connexion. La socket "de départ" *sockfd* continuera à traiter les connexions entrantes.

int **connect**(int sockfd, struct sockaddr *sock_info, int addrlen);

Permet de connecter la socket *sockfd* à la destination décrite par *sock_info* (*addrlen* = sizeof(struct sockaddr)). Il n'est pas obligatoire de faire un bind() avant un connect() (dans ce cas accept() s'occupe du bind()).

Exemple:

```
int sockfd, new_fd;  // listen() sur sock_fd, nouvelle connexion sur new_fd
struct sockaddr_in my_addr;    // ma machine
struct sockaddr_in their_addr; // le destinataire
int sin_size;

sockfd = socket(AF_INET, SOCK_STREAM, 0);

my_addr.sin_family = AF_INET;                    // host byte order
my_addr.sin_port = htons(3490);                  // short, network byte order
my_addr.sin_addr.s_addr = INADDR_ANY;            // mon adresse IP
memset(&(my_addr.sin_zero), '\0', 8);            // on mets à zéro le reste de la structure

bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));

listen(sockfd, 5);              // on attends des connexions avec une longueur max de file d'attente = 5

sin_size = sizeof(struct sockaddr_in);
new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);   // on traitera la connexion avec new_fd
.
.
```

int **send**(int sockfd, const void *msg, int length, int flags);

Envoie *length* octets de données stockées à l'emplacement mémoire *msg* (voir plus loin pour la description du champs *flags*) vers la socket *sockfd* de type SOCK_STREAM.

int **sendto**(int sockfd, const void *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen);

Idem que send() mais avec une socket de type SOCK_DGRAM. La structure pointée par *to* contient l'adresse IP et le port destination (et *tolen* = sizeof(struct sockaddr)).

int **recv**(int sockfd, void *buffer, int length, unsigned int flags);

Attends la réception d'au maximum *length* octets de données lues depuis la socket *sockfd* et copiées à l'emplacement mémoire pointé par *buffer (sockfd* doit être de type SOCK_STREAM).

int **recvfrom**(int sockfd, void *buf, int len, unsigned int flags, struct sockaddr *from, int *fromlen);

Idem que recv() mais avec une socket de type SOCK_DGRAM. La structure sockaddr pointée par from contiendra l'adresse IP et le port source de la machine émettrice (et *fromlen* = sizeof(struct sockaddr)).

Quelques précisions au sujet des valeurs possibles de l'entier *flags*:

L'entier flags peut prendre les valeurs suivantes (**ou 0 pour le comportement par défaut**):

Pour **send** et **sendto**:

```
#define MSG_OOB          0x1   /* process out−of−band data */
#define MSG_PEEK         0x2   /* peek at incoming message */
#define MSG_DONTROUTE    0x4   /* bypass routing, use direct interface */
#define MSG_EOR          0x8   /* data completes record */
#define MSG_EOF          0x100 /* data completes transaction */
```

Pour **recv** et **recvfrom**:

```
MSG_OOB          /* process out−of−band data     */
MSG_PEEK         /* peek at incoming message     */
MSG_WAITALL      /* wait for full request or error  */
```

int **close**(int sockfd); Permet de fermer la socket *sockfd*

int **shutdown**(int sockfd, int how);

Permet de restreindre l'utilisation de la socket *sockfd* suivant la valeur de l'entier *how:* 0 = la réception n'est plus permise, 1 = l'émission n'est plus permise, 2 = réception et émission sont interdites. ATTENTION: il faudra quand même fermer la socket avec close().

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

Remarque: il faut rajouter les inclusions suivantes: <sys/time.h> et <unistd.h>

La fonction select() permet de "surveiller" des ensembles de descripteurs. Après exécution, les ensembles de descripteurs sont modifiés afin de n'y inclure que les descripteurs ayant des données à traiter. Les macros ci−dessous permettent de manipuler les ensembles de descripteurs.

FD_ZERO(fd_set *set)	Initialise l'ensemble *set*
FD_SET(int fd, fd_set *set)	Ajoute le descripteur *fd* à l'ensemble *set*
FD_CLR(int fd, fd_set *set)	Supprime le descripteur *fd* de l'ensemble *set*
FD_ISSET(int fd, fd_set *set)	Test l'appartenance du descripteur *fd* à l'ensemble *set*

Les descriptions détaillées des fonctions sont bien entendu disponibles dans les **man pages**.
Source du document: Beej's guide to network programming (http://www.ecst.csuchico.edu/~beej/guide/net/)