Sockets réseau POSIX

Table des matières

Les fonctions de base

La description des fonctions est limitée à leur usage sur le domaine Internet en IP v4 et pour l'usage des protocoles TCP et UDP.

***Note** : pour des usages plus spécialisés se reporter à la documentation disponible en ligne sur le net.*

Création d'une socket : « socket() »

Include	`#include <sys/types.h>` `#include <sys/socket.h>`
Prototype	`int socket(int domain, int type, int protocol);`
Description	Crée une socket et retourne : • « -1 » si erreur • un descripteur sur la socket créée si OK
Arguments	• **`domain`** : indique le domaine d'utilisation. Pour Internet, utiliser la constante « AF_INET ». • **`type`** : indique le type de socket. Pour une socket en mode « connecté » utiliser « SOCK_STREAM » ; pour une socket en mode « non connecté », utiliser « SOCK_DGRAM ». • **`protocol`** : positionné à « 0 » ; dans ce cas le choix du protocole est déterminé par l'argument « type ». Si celui-ci vaut « SOCK_STREAM » le protocole utilisé sera « TCP » ; si « type » vaut « SOCK_DGRAM », le protocole utilisé sera « UDP » .

Exemple

On souhaite créer une socket en mode « non connecté » sur le domaine Internet :

```
int sock ;

sock = socket(AF_INET, SOCK_DGRAM, 0) ;
if (sock == -1) {
  printf(« Echec creation socket\n ») ;
  exit(1) ;
}
```

Fermeture et destruction d'une socket : « close() »

Include	`#include <unistd.h>`
Prototype	`int close(int sock);`
Description	Ferme et détruit la socket dont le descripteur est indiqué par « sock ». Retourne « -1 » si erreur.

Attachement d'une socket : « bind() »

Include	`#include <sys/types.h>` `#include <sys/socket.h>`
Prototype	`int bind(int sock, const struct sockaddr *address, socklen_t addr_len);`

Description	Nomme et attache la socket dont le descripteur est fourni par « sock » à l'adresse réseau indiquée par le pointeur « address ». La taille de l'adresse réseau est fournie par l'argument « addr_len ». Retourne « -1 » si erreur.
Arguments	• **`address`** : Pour Internet, le type de cet argument est en réalité « struct sockaddr_in* » (cf. page 10). • **`addr_len`** : indique la taille de l'adresse réseau à l'aide de la fonction « sizeof() ».

Exemple

On souhaite que le serveur attende sur n'importe quelle interface réseau (constante « INADDR_ANY ») de la machine serveur et sur le port 5896 :

```
...
struct sockaddr_in addr_srv ;
...

// on initialise la structure d'adresse en la remplissant de '0'
bzero(&addr_srv, sizeof(addr_srv));

// on initialise la structure d'adresse avec les valeurs souhaitées
addr_srv.sin_family = AF_INET ;
addr_srv.sin_port = htons(5896) ;
addr_srv.sin_addr.s_addr = htonl(INADDR_ANY) ;

// on appelle la fonction (on suppose que « sock » est initialisée)
if (bind (sock, (struct sockaddr*) &addr_srv, sizeof(addr_srv)) == -1) {
  printf(« Echec attachement\n ») ;
  exit(1) ;
}
```

ATTENTION *: Dans l'appel de « bind() » le type du 2ème argument doit être forcé comme indiqué puisque la fonction attend l'adresse d'une variable de type « struct sockaddr*» et pas de type « struct sockaddr_in*».*

ATTENTION *: On peut remarquer l'utilisation des fonctions « htons() » et « htonl() » servant à coder des entiers courts ou longs au format réseau (cf. page 8).*

Envoi en mode « non connecté » : « sendto() »

Include	`#include <sys/types.h>` `#include <sys/socket.h>`
Prototype	`ssize_t sendto(int sock, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addr_len);`
Description	Envoie les données stockées à l'emplacement pointé par « buf » sur la socket de descripteur « sock » et à l'adresse réseau indiquée par « dest_addr ». Retourne « -1 » si erreur.

Arguments	• **`buf`** : adresse d'une zone de données (tableau, structure, etc.). • **`len`** : taille en nombre d'octets des données pointées par « buf ». • **`flags`** : positionné la plupart du temps à « 0 » ; voir la doc en ligne sur le net pour plus d'information. • **`dest_addr`** : Adresse réseau du destinataire ; pour Internet, le type de cet argument sera en réalité « struct sockaddr_in* » (cf. page 10). • **`addr_len`** : indique la taille de l'adresse réseau pointée par « dest_addr » à l'aide de la fonction « sizeof() ».

Exemple

On souhaite envoyer une chaîne de caractères sur une machine à l'adresse IP « 192.168.1.3 » et sur le port 5896 :

```
...
struct sockaddr_in dest_addr ;
...

// on définit le message à envoyer
char* msg = "coucou";

// on initialise la structure d'adresse
bzero(&dest_addr, sizeof(dest_addr));

// on initialise la structure d'adresse avec les valeurs souhaitées
dest_addr.sin_family = AF_INET ;
dest_addr.sin_port = htons(5896) ;
inet_aton("192.168.1.3", &dest_addr .sin_addr) ;

// on appelle la fonction (on suppose que « sock » est initialisée)
if (sendto (sock, msg, strlen(msg), 0,
      (struct sockaddr*) &dest_addr, sizeof(dest_addr)) == -1) {
      printf("Echec envoi\n") ;
      exit(1) ;
}
```

Note *: le champ « sin_addr » est ici initialisé par la fonction « inet_aton() » (cf page 8) qui permet de convertir une adresse exprimée sous forme de chaîne de caractères en un entier sur 32 bits codé au format réseau.*

Réception en mode « non connecté » : « recvfrom() »

Include	`#include <sys/types.h>` `#include <sys/socket.h>`
Prototype	`ssize_t recvfrom(int sock, void *buf, size_t len, int flags, struct sockaddr *src_addr, socklen_t *addr_len);`
Description	Reçoit les données sur la socket de descripteur « sock » et les dépose à l'emplacement pointé par « buf ». Retourne : • « -1 » si erreur • le nombre d'octets reçus sinon (0 à n)
Arguments	• **`buf`** : adresse d'une zone de données (tableau, structure, etc.). • **`len`** : taille en nombre d'octets de la zone pointée par « buf ». • **`flags`** : positionné la plupart du temps à « 0 » ; voir la doc en ligne sur le net pour plus d'information. • **`src_addr`** : Adresse réseau de l'émetteur ; pour Internet, le type de cet argument sera en réalité « struct sockaddr_in* » (cf. page 10). • **`addr_len`** : adresse d'une variable entière qui sera initialisée par la fonction avec la taille de l'adresse réseau pointée par « src_addr ».

Gestion de l'argument « src_addr »

Cet argument peut être positionné :

- avec la valeur « NULL » : ce sera généralement le cas d'un client qui reçoit la réponse du serveur à qui il vient d'envoyer une requête (le client n'a pas besoin de récupérer l'adresse du serveur puisqu'il était obligé de la connaître au prélable pour lui envoyer sa requête).
  Dans ce cas l'argument « addr_len » doit lui aussi être positionné à « NULL ».

- avec l'adresse d'une variable de type « struct sockaddr_in » valide : ce sera généralement le cas d'un serveur qui reçoit une requête et qui doit connaître l'adresse du client pour pouvoir lui répondre.
  Dans ce cas l'argument « addr_len » doit être l'adresse d'une variable de type « socklen_t ».
  A l'appel de la fonction cette variable doit stocker la taille de la variable pointée par « src_addr » ; au retour de la fonction elle contient la taille effective de l'adresse reçue (si ces 2 tailles diffèrent, la variable pointée par « addr_len » a été tronquée et on ne peut donc pas lui faire confiance!)

***Note** : Parfois un client a besoin de récupérer l'adresse du serveur qui lui répond, il lui suffit de alors fournir les arguments valides correspondants pour « src_addr » et « addr_len ».*
Ce cas peut être rencontré dans certains services UDP comme « TFTP » dans lesquels le serveur possède (comme avec TCP) une socket d'écoute et traite les requêtes par des sockets de service.

Exemple 1 : réception d'un serveur

```
...
struct sockaddr_in src_addr ;
socklen_t addr_len ;
...

// on définit un tableau de réception
char buf[32] ;
```

```
// on appelle la fonction (on suppose que « sock » est initialisée)
addr_len = sizeof(src_addr) ;
if (recvfrom (sock, buf, sizeof(buf), 0,
      (struct sockaddr*) &src_addr, &addr_len) == -1) {
      printf(« Echec réception\n ») ;
      exit(1) ;
}
```

Exemple 2 : réception d'un client

```
...

// on définit un tableau de réception
char buf[32] ;

// on appelle la fonction (on suppose que « sock » est initialisée)
if (recvfrom (sock, buf, sizeof(buf), 0, NULL, NULL) == -1) {
      printf(« Echec réception\n ») ;
      exit(1) ;
}
```

Marquage en socket « d'écoute » : « listen() »

Cette fonction est utilisée par un serveur en mode « connecté ».

Include	`#include <sys/types.h>` `#include <sys/socket.h>`
Prototype	`int listen(int sock, int nb);`
Description	Marque la socket de descripteur « sock » comme une socket « d'écoute » et indique par « nb » le nombre maximum de demandes de connexions acceptées en attente. Retourne « -1 » si erreur.

Attente des connexions : « accept() »

Cette fonction est utilisée par un serveur en mode « connecté ».

Include	`#include <sys/types.h>` `#include <sys/socket.h>`
Prototype	`int accept (int sock, struct sockaddr *addr, socklen_t *addrlen)`
Description	Attend une connexion sur la socket de descripteur « sock ». Si une connexion se produit sans erreur, la fonction dépose l'adresse réseau du client dans la variable pointée par « addr » ainsi que la longueur de cette adresse dans la variable pointée par « addrlen ». Retourne : • « -1 »si erreur • le descripteur de la socket de service sinon.

***REMARQUE** : les arguments « addr » et « addrlen » peuvent être positionnés à « NULL » si la connaissance de l'adresse du client n'est pas nécessaire.*

Demande de connexion : « connect() »

Cette fonction est utilisée par un client en mode « connecté ».

Include	`#include <sys/types.h>` `#include <sys/socket.h>`
Prototype	`int connect(int sock, const struct sockaddr *addr, socklen_t addrlen);`
Description	Etablit une connexion sur la socket de descripteur « sock » avec le serveur dont l'adresse réseau est fournie par « addr » ; la longueur de cette adresse est indiquée par « addrlen ». Retourne « -1 » si erreur.

Émission en mode « connecté » : « send() »

Include	`#include <sys/types.h>` `#include <sys/socket.h>`
Prototype	`ssize_t send(int sock, const void *buf, size_t len, int flags);`
Description	Envoie les données contenues dans le tableau « buf » sur la socket de descripteur « sock » Retourne : • « -1 » si erreur. • le nombre d'octets émis sinon
Arguments	• `buf` : adresse d'une zone de données (tableau, structure, etc.). • `len` : nombre d'octets de données à envoyer. • `flags` : positionné la plupart du temps à « 0 » ; voir la doc en ligne sur le net pour plus d'information.

Réception en mode « connecté » : « recv() »

Include	`#include <sys/types.h>` `#include <sys/socket.h>`
Prototype	`ssize_t recv(int sock, void *buf, size_t len, int flags);`
Description	Reçoit des données sur la socket de descripteur « sock » et les dépose dans le tableau « buf » Retourne : • « -1 » si erreur. • le nombre d'octets reçus sinon
Arguments	• `buf` : adresse d'une zone de données (tableau, structure, etc.). • `len` : taille en nombre d'octets de la zone pointée par « buf ». • `flags` : positionné la plupart du temps à « 0 » ; voir la doc en ligne sur le net pour plus d'information.

Les fonctions de service

Conversion d'entiers

Include	`#include <arpa/inet.h>`

Prototypes	`uint32_t htonl(uint32_t hostlong);`
	`uint16_t htons(uint16_t hostshort);`
	`uint32_t ntohl(uint32_t netlong);`
	`uint16_t ntohs(uint16_t netshort);`

Description	• **`htonl()`** : retourne un entier 32 bits codé au format réseau à partir d'un entier 32 bits codé au format « hôte ». • **`htons()`** : retourne un entier 16 bits codé au format réseau à partir d'un entier 16 bits codé au format « hôte ». • **`ntohl()`** : retourne un entier 32 bits codé au format « hôte » à partir d'un entier 32 bits codé au format réseau. • **`ntohs()`** : retourne un entier 16 bits codé au format « hôte » à partir d'un entier 16 bits codé au format réseau.

Conversion d'adresse IP

Include	`#include <sys/socket.h>`
	`#include <netinet/in.h>`
	`#include <arpa/inet.h>`

Prototypes	`int inet_aton(const char *cp, struct in_addr *inp);` `char *inet_ntoa(struct in_addr in);`
Description	• **`inet_aton()`** : dépose à l'adresse indiquée par l'argument « inp » (au format réseau) l'adresse IP v4 exprimée par l'argument « cp ». Retourne « 0 » si erreur. • **`inet_ntoa()`** : Retourne une adresse IP v4 sous forme de chaîne de caractères à partir de l'argument « in » (« in » codé au format réseau). Retourne (-1) si erreur.

Exemple

Pour initialiser une structure de type « sockaddr_in » avec une adresse IP exprimée sous forme de chaîne de caractères « pointée » on utilise « inet_aton() » de la manière suivante :

```
struct sockaddr_in addrServeur ;
...
inet_aton ("10.0.200.224", &addrServeur.sin_addr);
```

Pour obtenir cette fois une adresse IP sous la forme d'une chaine de caractères « pointée » on utilise « inet_ntoa() » de la manière suivante :

```
struct sockaddr_in addrServeur ;
char adIP[32] ;
…
strcpy(adIP, addrServeur.sin_addr);
```

Obtenir l'adresse d'une machine

Include	`#include <netdb.h>` `#include <sys/socket.h>`
Prototypes	`struct hostent *gethostbyname(const char *name);`
Description	Retourne l'adresse d'une structure de type « struct hostent » (cf. page 11) correspondant au nom Internet fourni en argument (par exemple « www.free.fr »).

Exemple

```
/*
 * gethostbyname.c - Example of using gethostbyname(3)
 * Martin Vidner <mvidner@suse.cz>
 */

#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>

struct hostent *he;
struct in_addr a;

int
main (int argc, char **argv)
{
    if (argc != 2)
    {
        fprintf(stderr, "usage: %s hostname\n", argv[0]);
        return 1;
    }
    he = gethostbyname (argv[1]);
    if (he)
    {
        printf("name: %s\n", he->h_name);
        while (*he->h_aliases)
            printf("alias: %s\n", *he->h_aliases++);
        while (*he->h_addr_list)
        {
            bcopy(*he->h_addr_list++, (char *) &a, sizeof(a));
            printf("address: %s\n", inet_ntoa(a));
        }
    }
    else
        herror(argv[0]);
    return 0;
}
```

Obtenir un n° de port par un nom de service et la réciproque

Include	`#include <netdb.h>`
Prototypes	`struct servent *getservbyname(const char *name);` `struct servent *getservbyport(int port, const char *proto);`
Description	• **`getservbyname()`** : Retourne l'adresse d'une structure de type « struct servent » (cf. page 11) correspondant au service Internet fourni en argument (par exemple « ftp ») sur la machine locale. • **`getservbyport()`** : Retourne l'adresse d'une structure de type « struct servent » correspondant au service Internet dont le numéro de port est fourni par « port » et le protocole par « proto ».

Les structures de l'A.P.I.

Adresse de type Internet : « struct sockaddr_in »

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

Prototype	`struct sockaddr_in {`
	`  short sin_family ; // domaine de communication`
	`  u_short sin_port ; // le n° de port`
	`  struct in_addr sin_addr ; // l'adresse IP`
	`}`
	
	`struct in_addr {`
	`  u_long s_addr ; // l'adresse IP effective sur 4 octets`
	`}`

__Note__ : pour remplir le champ « s_addr » d'une structure « struct in_addr » on pourra utiliser les fonctions de service « htonl() » (cf. page 8) ou « inet_aton() » (cf. page 8).

Exemple

On peut initialiser une structure de type « sockaddr_in » pour le domaine Internet avec une adresse IP sous la forme d'une chaîne de caractères et un port valide de la manière suivante :

```
struct sockaddr_in addr;
...
addr.sin_family = AF_INET;
addr.sin_port = htons(2369);
inet_aton ("192.168.26.23", &addr.sin_addr);
```

Information sur une machine : « struct hostent »

Include	`#include <netdb.h>` `#include <sys/socket.h>`

Prototype	`struct hostent {`
	`  char  *h_name;  /* official name of host */`
	`  char **h_aliases;  /* alias list */`
	`  int  h_addrtype;  /* host address type */`
	`  int  h_length;  /* length of address */`
	`  char **h_addr_list;  /* list of addresses */`
	`}`
	`#define h_addr h_addr_list[0] /* for backward compatibility */`

Champs	**h_name** : le nom Internet – par exemple « www.free.fr » **h_aliases** : un tableau de noms Internet « alias » (chaque nom est exprimé sous forme d'une chaîne de caractère) **h_addrtype** : le type de l'adresse réseau(« AF_INET » ou « AF_INET6 ») **h_length** : la taille des adresses réseau contenues dans le tableau « h_addr_list » (4 octets pour IP v4, 16 octets pour IP v6) **h_addr_list** : tableau d'adresses réseau. Chaque adresse réseau est stockée au format réseau dans un tableau de « h_addrtype » octets.

Information sur un service : « struct servent »

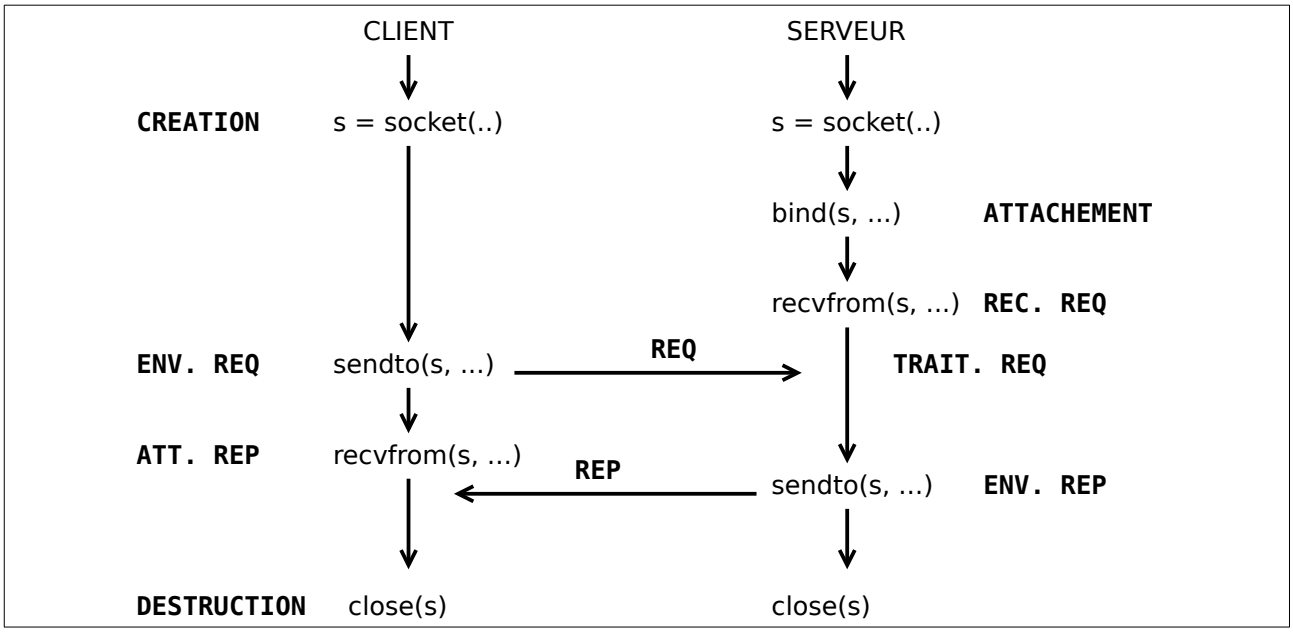
Include	`#include <netdb.h>` `#include <sys/socket.h>`

Prototype	`struct servent {`
	`  char  *s_name;  /* official service name */`
	`  char **s_aliases;  /* alias list */`
	`  int  s_port;  /* port number */`
	`  char  *s_proto;  /* protocol to use */`
	`}`

Champs	**s_name** : le nom du service – par exemple « ftp » **s_aliases** : un tableau de noms de service « alias » (chaque nom est exprimé sous forme d'une chaîne de caractère) **s_port** : le numéro du port correspondant au service codé au format réseau. **s_proto** : le nom du protocole associé au service

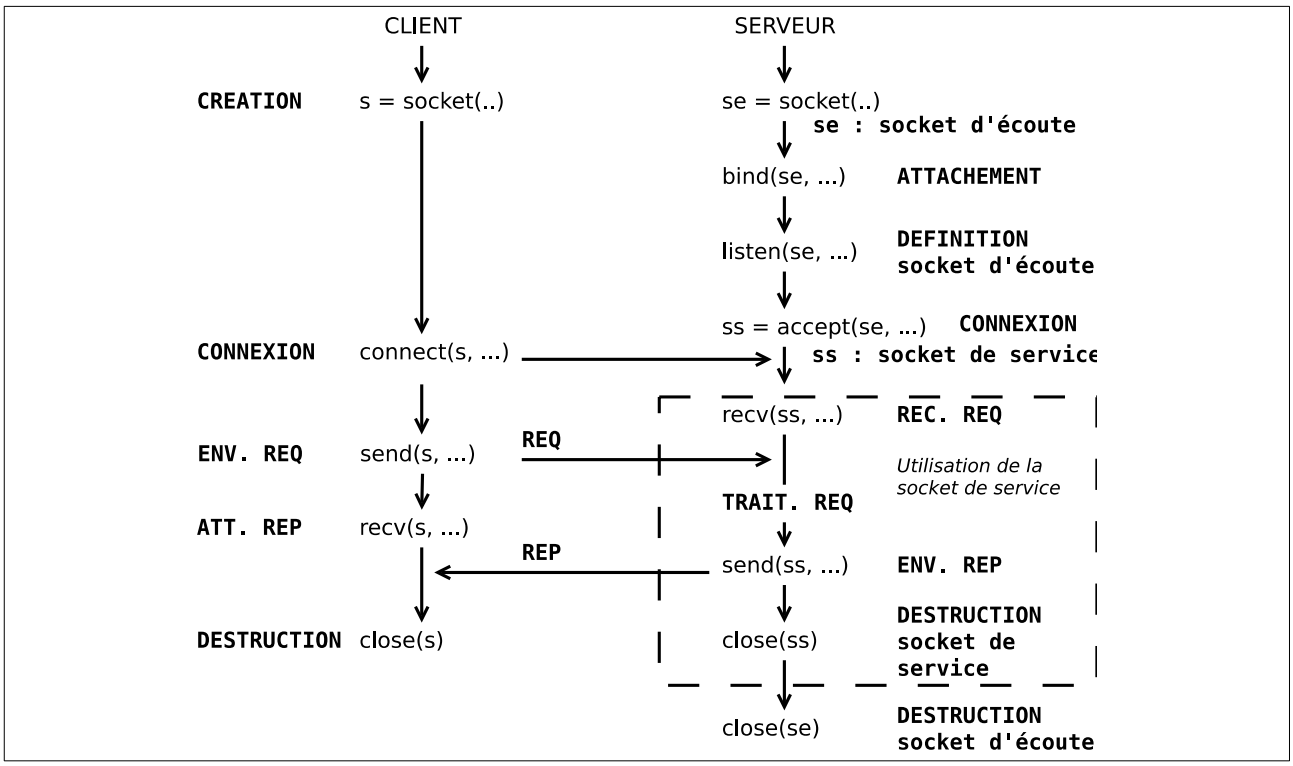
Exemples d'échanges

Mode « non connecté »



Exemple d'échange en mode non connecté (sans bouclage côté serveur)

Mode « connecté »



Exemple d'échange en mode connecté (sans bouclage côté serveur)