ANNEXE D7

Programmation Réseau

Syntaxe de quelques appels systèmes

Nota : Les informations qui suivent sont tirées de l'aide en ligne d'un système Linux. La conformité de ces fonctions à BSD 4.4 garantit une certaine portabilité de ces appels. Le système qui nous intéresse respecte intégralement ces appels...

IP(4) Manuel du programmeur Linux IP(4)

NAME

ip - Linux IPv4 protocol implementation

SYNOPSIS

```
#include <sys/socket.h>
#include <net/netinet.h>

tcp_socket = socket(PF_INET, SOCK_STREAM, 0);
raw_socket = socket(PF_INET, SOCK_RAW, protocol);
udp_socket = socket(PF_INET, SOCK_DGRAM, protocol);
```

DESCRIPTION

Linux implements the IPv4 protocol described in RFC791 and RFC1122. ip contains a level 2 multicasting implementation comforming to RFC1112. It also contains an IP router including a packet filter.

The protocol is implemented in the kernel on the basis of a BSD compatible socket interface. For more information on sockets, see socket(4).

An IP socket is created by calling the socket(2) function with a PF_INET socket family argument. Valid socket types are SOCK_STREAM to open a tcp(4) socket, SOCK_DGRAM to open a udp(4) socket, or SOCK_RAW to open a raw socket. protocol is the IP protocol in the IP header be received or sent. For TCP and UDP sockets, only 0, IPPROTO_TCP , or IPPROTO_UDP are valid. For SOCK_RAW you may specify a valid IANA IP protocol defined in RFC1700 assigned numbers.

Raw sockets may only be opened by a process with effective user id 0 or when the process has the CAP_NET_RAW capability.

When a process wants to receive new incoming packets or connections, it should be bound to a local interface address using bind(2). When INADDR_ANY is specified it will bind to any local interface. A bound TCP socket is unavailable for time after closing, unless the SO_REUSEADDR flag is set.

ADDRESS FORMAT

An IP socket address is defined as a combination of an IP interface address and a port number.

```
struct sockaddr_in
{
      sa_family_t       sin_family; /* address family: AF_INET */
      u_int16_t         sin_port;   /* port in network byte order */
      struct in_addr    sin_addr;   /* internet address */
};
```

```
/* Internet address. */
struct in_addr
{
      u_int32_t          s_addr;      /* IPv4 address in network byte order */
};
```

sin_family is always set to AF_INET. This is required; in Linux 2.2 most networking functions return EINVAL when this setting is missing. sin_port contains the port in network byte order. The port numbers below 1024 are called reserved ports. Only processes with the effective user id 0 or the CAP_NET_BIND_SERVICE attribute set may bind(2) to these sockets. Note that the raw IPv4 protocol as such has no concept of a port, they are only implemented by higher protocols like tcp(4) and udp(4).

sin_addr is the host address. The addr member of struct in_addr contains the host interface address in network order. in_addr should be only accessed using the inet_aton(3), inet_addr(3), inet_makeaddr(3) library functions or directly with the name resolver (see gethostbyname(3)). IPv4 addresses are divided into unicast, broadcast and multicast addresses. Unicast addresses specify a single interface of a host, broadcast addresses specify all host on a network and multicast addresses address all hosts in a multicast group. Datagrams to broadcast addresses are only passed to the user when the socket broadcast flag is set. To send datagrams to broadcast addresses it has to be set too. Connection oriented sockets are only allowed to use unicast addresses.

Note that the address and the port are always stored in network order, this particulary means that you need to call htons(3) on the number that is assigned to a port. All address/port manipulation in the standard library automatically convert to network order.

SEE ALSO

sendmsg(2), recvmsg(2), socket(4), netlink(4), tcp(4), udp(4), raw(4), ipfw(4)

RFC791, RFC1122, RFC1812

NOM

socket - Créer un point de communication.

SYNOPSIS

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

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

DESCRIPTION

Socket crée un point de communication, et renvoie un descripteur.

Le paramètre domain indique un domaine de communication, à l'intérieur duquel s'établira le dialogue. Ceci permet de sélectionner la famille de protocole à employer. Ces familles sont définies dans le fichier linux/socket.h.

Les formats actuellement proposés sont :

AF_UNSPEC	Famille non spécifiée, laisser le système la déterminer.
AF_UNIX	Protocoles locaux internes UNIX (pipe,...)
AF_INET	Protocoles Internet (UDP, TCP, etc...)
AF_AX25	Radio amateurs AX.25
AF_IPX	Protocoles Novell.
AF_APPLETALK	Protocoles Apple.
AF_NETROM	Radio amateurs NetRom.
AF_BRIDGE	Passerelle multi-protocoles.
AF_AAL5	Réservé pour l'ATM Werner
AF_X25	Réservé pour le projet CCITT X.25
AF_INET6	Réservé pour le projet IP version 6

Les sockets ont le type, indiqué, ce qui fixe la sémantique des communications. Les types définis actuellement sont :

SOCK_STREAM Support de dialogue garantissant l'intégrite,fournissant un flux de données binaires, et intégrant un mécanisme pour les transmissions de données hors-bande.

Les sockets de ce type sont des flux full-duplex, similaires à des tubes (pipes).

SOCK_DGRAM	Transmissions sans connexion, non garantie, de datagrammes de longueur fixe, généralement courte.
SOCK_RAW	Transmissions internes au système, le type SOCK_RAW, ne peut être utilisé que par le Super-User.
SOCK_RDM	Transmission garantie de datagrammes
SOCK_SEQPACKET	Dialogue garantissant l'intégrite, pour le transport de datagrammes de longueur fixe. Le lecteur peut avoir à lire le paquet de données complet à chaque appel système read

Le protocole à utiliser sur la socket est indiqué par l'argument protocol. Normalement il n'y a qu'un seul protocole par type de socket pour une famille donnée. Néanmoins

rien ne s'oppose à ce que plusieurs protocoles existent, auquel cas il est nécessaire de le spécifier.

Le numéro de protocole dépend du domaine de communication de la socket. Voir protocols(5).

Une socket de type stream doit être connectée avant que des données puisse y être lues ou écrites. Une connexion sur une autre socket est établie par l'appel système connect(2). Une fois connectée les données y sont transmises par read(2) et write(2) ou par des variantes de send(2) et recv(2).

Quand une session se termine,on referme la socket avec close(2).

Les données hors-bande sont envoyées ou reçues en utilisant send(2) et recv(2).

Le protocole de communication utilisé pour implémenter les sockets stream garantit qu'aucune donnée n'est perdue ou dupliquée. Si un bloc de données, pour lequel le correspondant a suffisamment de place dans son buffer, n'est pas transmis correctement dans un délai raisonnable, la connexion est considérée comme inutilisable, et les appels systèmes renverront une valeur -1 en indiquant une erreur ETIMEDOUT dans la variable globale errno.

Eventuellement les protocoles peuvent maintenir les sockets en service en forçant des transmissions directes toutes les minutes en l'absence de toute autre activité. Une erreur est indiquée si aucune réponse n'est reçue sur une socket inactive pendant une période prolongée (par exemple 5 minutes).

Un signal SIGPIPE est envoyé au processus tentant d'écrire sur une socket inutilisable, forçant les programmes ne gérant pas ce signal à se terminer.

Les sockets de type SOCK_SEQPACKET emploient les mêmes appels systèmes que celles de types SOCK_STRREAM, à la différence que la fonction read(2) ne renverra que le nombre d'octets requis, et toute autre donnée restante sera éliminée.

Les sockets de type SOCK_DGRAM ou SOCK_RAW permettent l'émission de datagrammes à des correspondants indiqués au moment de l'appel système send(2). Les datagrammes sont généralement lus par la fonction recvfrom(2), qui fournit également l'adresse du correspondant.

Un appel à fcntl(2) permet de préciser un groupe de processus qui recevront un signal SIGURG lors de l'arrivée de données hors-bande. Cette fonction permet également de valider des entrées/sorties non bloquantes, et une notification asynchrone des évènements par le signal SIGIO.

Les opérations sur les sockets sont représentées par des options du niveau socket. Ces options sont définies dans sys/socket.h. Les fonctions setsockopt(2) et getsockopt(2) sont utilisées respectivement pour fixer ou lire les options.

VALEUR RENVOYÉE

socket retourne un descripteur référençant la socket créée en cas de réussite. En cas d'échec -1 est renvoyé, et errno contient le code d'erreur.

ERREURS

EPROTONOSUPPORT	Le type de protocole, ou le protocole lui-même n'est pas disponible dans ce domaine de communication.
EMFILE	La table des descripteurs par processus est pleine.
ENFILE	La table des fichiers est pleine.
EACCES	La création d'une telle socket n'est pas autorisée.
ENOBUFS	Pas suffisamment d'espace pour allouer les buffers nécessaires.

NOM

bind - Fournir un nom une socket.

SYNOPSIS

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

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

DESCRIPTION

bind fournit à la socket sockfd, l'adresse locale my_addr. my_addr est longue de addrlen octets. Traditionnellement cette operation est appelée "assignation d'un nom à une socket" (Quand une socket est créée, par l'appel-système socket(2),nelle existe dans l'espace des noms mais n'a pas de nom assigné).

NOTES

Assigner un nom dans le domaine UNIX crée une socket dans le système de fichier, qui devra être détruite par le créateur une fois qu'il n'en a plus besoin, en utilisant unlink(2).

Les règles d'assignation de nom varient suivant le domaine de communication. Consultez le manuel du programmeur Linux section 4 pour de plus amples informations.

VALEUR RENVOYÉE

bind renvoie 0 s'il réussit, ou -1 s'il échoue, auquel cas errno contient le code d'erreur.

ERREURS

EBADFF	sockfd n'est pas un descripteur valide.
EACCESS	L'adresse est protégée et l'utilisateur n'est pas le Super-User.
ENOTSOCK	L'argument est un descripteur de fichier, pas une socket.
EADDRINUSE	La socket a déjà une adresse assignée.

Les erreurs suivantes sont spécifiques au domaine UNIX(AF_UNIX):

EINVAL	La longueur addr_len est fausse, ou la socket n'est pas de la famille AF_UNIX.
EROFS	L'i-noeud de la socket se trouverait dans un système de fichiers en lecture seule.
EFAULT	my_addr pointe en dehors de l'espace d'adresse accessible.
ENAMETOOLONG	my_addr est trop long
ENOENT	Le fichier n'existe pas.
ENOMEM	pas assez de mémoire pour le noyau.
ENOTDIR	Un composant du chemin d'accès n'est pas un répertoire.
EACCES	L'accès à un composant du chemin d'accès n'est pas autorisé.
ELOOP	my_addr contient des références circulaires (à travers un lien symbolique).

NOM

send, sendto, sendmsg - Envoyer un message sur une socket.

SYNOPSIS

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

int send(int s, const void *msg, int len , unsigned int flags);
int sendto(int s, const void *msg, int len , unsigned int flags,
           const struct sockaddr *to, int tolen);
int sendmsg(int s, const struct msghdr *msg , unsigned int flags);
```

DESCRIPTION

Send, sendto, et sendmsg permettent de transmettre un message à destination d'une autre socket. Send ne peut être utilisé qu'avec les sockets connectée alors que sendto et sendmsg peuvent être utilisés tout le temps.

L'adresse de la cible est donnée par to avec la longueur tolen. La longueur du message est indiquée dans len. Si le message est trop long pour être transmis intégralement au protocole sous-jacent, l'erreur EMSGSIZE sera déclenchée et rien ne sera émis.

Aucune indication d'échec de distribution n'est fournie par send. Seules les erreurs locales sont détectées, et indiquées par une valeur de retour -1.

Si la socket ne dispose pas de la place suffisante pour le message, alors send va bloquer, à moins que la socket ait été configurée en mode d'entrées/sorties non-bloquantes. On peut utiliser l'appel système select(2) pour vérifier s'il est possible d'émettre des données.

Le paramètre flags peut contenir une ou plusieurs des options suivantes

```
#define  MSG_OOB        0x1 /* Traiter les données hors-bande */
#define  MSG_DONTROUTE  0x4 /* Contourner le routage      */
```

L'option MSG_OOB est utilisée pour émettre des données hors-bande sur une socket qui l'autorise (par ex : SOCK_STREAM). Le protocole sous-jacent doit également autoriser l'émission de données hors-bande.

MSG_DONTROUTE est utilisée par les programmes de diagnostique ou de routage.
Voir recv(2) pour une description de la structure msghdr.

VALEUR RENVOYÉE

Ces appels systèmes renvoient le nombre de caractères émis, ou -1 s'ils échouent, auquel cas errno contient le code d'erreur.

ERREURS

EBADF	Descripteur de socket invalide.
ENOTSOCK	L'argument s n'est pas une socket.
EFAULT	Un paramètre pointe en dehors de l'espace d'adressage accessible.
EMSGSIZE	La socket nécessite une émission intégrale du message mais la taille de celui-ci ne le permet pas.
EWOULDBLOCK	La socket est non-bloquante et l'opération demandée bloquerait.
EPIPE	L'écriture est impossible (correspondant absent), et le signal SIGPIPE a été ignoré par le processus.
ENOMEM	Pas assez de mémoire pour le noyau.
ENOBUFS	La file d'émission de l'interface réseau est pleine. Ceci indique généralement une panne de l'interface réseau, mais peut également être dû à un engorgement passager.

NOM

recv, recvfrom, recvmsg - Recevoir un message sur une socket.

SYNOPSIS

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

int recv(int s, void *buf, int len, unsigned int flags);
int recvfrom(int s, void *buf, int len, unsigned int flags,
                struct sockaddr *from, int *fromlen);
int recvmsg(int s, struct msghdr *msg, unsigned int flags);
```

DESCRIPTION

recvfrom et recvmsg sont utilisés pour recevoir des messages depuis une socket s, et peuvent servir à la lecture de données que la socket soit orientée connexion ou non.

Si from est non nul, et si la socket n'est pas orientée connexion, l'adresse de la source du message y est insérée. Fromlen est un paramètre résultat, initialisé à la taille du buffer from, et modifié en retour pour indiquer la taille réelle de l'adresse enregistrée.

L'appel recv est normalement utilisé sur une socket connectée (voir connect(2)) et est équivalent à recvfrom avec un paramètre from nul. Comme ceci est redondant il est possible que recv ne soit plus maintenu dans le futur.

Ces trois routines renvoient la longueur du message si elles réussissent. Si un message est trop long pour tenir dans le buffer, les octets supplémentaires peuvent être abandonnés suivant le type de socket utilisé (voir socket(2)).

Si aucun message n'est disponible sur la socket, les fonctions de réception se mettent en attente, à moins que la socket soit non bloquante (voir fcntl(2)) auquel cas la valeur -1 est renvoyée, et errno est positionnée à EWOULDBLOCK.

Les fonctions de réception renvoient normalement les données disponibles dans la limite du paramètre len sans attendre d'avoir reçu le nombre exact réclamé. Ce comportement peut être modifié avec les options de socket SO_RCVLOWAT et SO_RCVTIMEO décrites dans getsockopt(2). L'appel select(2) peut permettre de déterminer si des données supplémentaires sont disponibles.

L'argument flags de l'appel recv est constitué par un OU binaire (|) entre les valeurs suivantes

```
MSG_OOB      traiter les données hors-bande
MSG_PEEK     lire sans enlever les données
MSG_WAITALL attendre le nombre exact ou une erreur
```

L'option MSG_OOB permet la lecture des données hors-bande qui ne seraient autrement pas placées dans le flux de données normales. Certains protocoles placent ces données hors-bande en tête de la file normale, et cette option n'a pas lieu d'être dans ce cas.

L'option MSG_PEEK permet de lire les données en attente dans la file sans les enlever de cette file. Ainsi une lecture ultérieure renverra à nouveau les mêmes données.

L'option MSG_WAITALL demande que l'opération de lecture soit bloquée jusqu'à ce que la requête complète soit satisfaite. Toutefois la lecture peut renvoyer quand même moins

de données que prévu si un signal est reçu, ou si une erreur ou une déconnexion se produisent.

L'appel recvmsg utilise une structure msghdr pour minimiser le nombre de paramètres à fournir directement. Cette structure à la forme suivante, définie dans <sys/socket.h>

```
struct msghdr {
   caddr_t    msg_name;         /* optional address */
   u_int      msg_namelen;      /* size of address */
   struct     iovec *msg_iov;   /* scatter/gather array */
   u_int      msg_iovlen;       /* # elements in msg_iov */
   caddr_t    msg_control;      /* ancillary data, see below */
   u_int      msg_controllen;   /* ancillary data buffer len */
   int        msg_flags;        /* flags on received message */
};
```

Ici msg_name et msg_namelen spécifient l'adresse destination si la socket n'est pas connectée, msg_name peut être un pointeur nul si le nom n'est pas nécessaire. Msg_iov et msg_iovlen décrivent les buffers de réception comme décrit dans readv(2). Msg_control, de longueur msg_controllen, pointe sur un buffer utilisé pour les autres messages relatifs au protocole, ou à d'autres données. Les messages ont la forme

```
struct cmsghdr {
   u_int      cmsg_len;   /* data byte count, including hdr */
   int        cmsg_level; /* originating protocol */
   int        cmsg_type;  /* protocol-specific type */
/* followed by
   u_char     cmsg_data[]; */
};
```

Par exemple, on peut utiliser ceci pour être informe des changements du flux de données avec XNS/SPP, ou pour obtenir des données d'identification en demandant un recvmsg sans buffer immédiatement après l'appel de accept avec ISO.

Les descripteurs de fichiers annexes sont désormais passés en tant que données annexes dans le domaine AF_UNIX avec cmsg_level valant SOL_SOCKET et cmsg_type valant SCM_RIGHTS.

Le champ msg_flags est rempli au retour avec les informations concernant le message. MSG_EOR indique une fin d'enregistrement, les données reçues terminent un enregistrement (utilisé généralement avec les sockets du type SOCK_SEQPACKET). MSG_TRUNC indique que la portion finale du datagramme a été abandonnée car le datagramme était trop long pour le buffer fourni. MSG_CTRUNC indique que des données de contrôle ont été abandonnées à cause d'un manque de place dans le buffer de données annexes. MSG_OOB indique que des données hors-bande ont été reçues.

VALEUR RENVOYÉE

Ces fonctions renvoient le nombre d'octets reçus si elles réussissent, ou -1 si elles échouent, auquel cas errno contient le code d'erreur.

ERREURS

EBADF	L'argument s n'est pas un descripteur valide.
ENOTCONN	La socket est associée à un protocole orienté connexion et n'a pas encore été connectée (voir connect(2) et accept(2)).
ENOTSOCK	L'argument s ne correspond pas à une socket.
EWOULDBLOCK	La socket est non-bloquante et aucune donnée n'est disponible, ou un délai de timeout a été indiqué, et il a expiré sans que l'on ait reçu quoi que ce soit.
EINTR	Un signal a interrompu la lecture avant que des données soient disponibles.
EFAULT	Un buffer pointe en dehors de l'espace d'adressage accessible.