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 le domaine de communication pour le dialogue ; ceci sélectionne la famille de protocole à employer. Elles sont définies dans le fichier <*sys/socket.h*>. Les formats actuellement proposés sont :

Nom	Utilisation	Page
PF_UNIX,PF_LOCAL	Communication locale	**unix**(7)
PF_INET	IPv4 Protocoles Internet	**ip**(7)
PF_INET6	IPv6 Protocoles Internet	
PF_IPX	IPX - Protocoles Novell	
PF_NETLINK	Interface utilisateur noyau	**netlink**(7)
PF_X25	Protocole ITU-T X.25 / ISO-8208	**x25**(7)
PF_AX25	Protocole AX.25 radio amateur	
PF_ATMPVC	Accès direct ATM PVCs	
PF_APPLETALK	Appletalk	**ddp**(7)
PF_PACKET	Interface paquet bas-niveau	**packet**(7)

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'intégrité, fournissant un flux de données binaires, et intégrant un mécanisme pour les transmissions de données hors-bande.

SOCK_DGRAM
Transmissions sans connexion, non garantie, de datagrammes de longueur fixe, généralement courte.

SOCK_SEQPACKET
Dialogue garantissant l'intégrité, pour le transport de datagrammes de longueur fixe. Le lecteur peut avoir à lire le paquet de données complet à chaque appel système read.

SOCK_RAW
Accès direct aux données réseau.

SOCK_RDM
Transmission fiable de datagrammes, sans garantie de l'ordre de délivrance.

SOCK_PACKET
Obsolète, à ne pas utiliser dans les programmes actuels. Voir **packet**(7).

Certains types de sockets peuvent ne pas être implémentés par toutes les familles de protocoles. Par exemple, **SOCK_SEQPACKET** n'est pas implémenté pour **AF_INET**.

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, auquel cas l'argument *protocol* peut être nul. 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). Voir **getprotoent**(3) pour savoir comment associer un nom de protocole à un numéro.

Une socket de type **SOCK_STREAM** est un flux d'octets full-duplex, similaire aux tubes (pipes). Elle ne préserve pas les limites d'enregistrements. Une socket SOCK_STREAM doit être dans un état *connecté*

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).

Les protocoles de communication qui implémentent les sockets **SOCK_STREAM** garantissent qu'aucune donnée n'est perdue ou dupliquée. Si un bloc de données, pour lequel le correspondant a suffisamment de place dans son tampon, n'est pas transmis correctement dans un délai raisonnable, la connexion est considérée comme inutilisable. Si l'option **SO_KEEPALIVE** est activée sur la socket, le protocole vérifie, d'une manière qui lui est spécifique, si le correspondant est toujours actif. 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_STREAM,** à la différence que la fonction **read**(2) ne renverra que le nombre d'octets requis, et toute autre donnée restante sera éliminée. De plus, les frontières des messages seront préservées.

Les sockets de type **SOCK_DGRAM** ou **SOCK_RAW** permettent l'envoi de datagrammes aux correspondants indiqués dans l'appel système **sendto**(2). Les datagrammes sont généralement lus par la fonction **recvfrom**(2), qui fournit également l'adresse du correspondant.

Les sockets **SOCK_PACKET** sont obsolètes. Elles servent à recevoir les paquets bruts directement depuis le gestionnaire de périphérique. Utilisez plutôt **packet**(7).

L'opération **F_SETOWN** de **fcntl**(2) permet de préciser un processus ou groupe de processus qui recevront un signal **SIGURG** lors de l'arrivée de données hors-bande, ou le signal **SIGPIPE** lorsqu'une connexion sur une socket **SOCK_STREAM** se termine inopinément. Cette fonction permet également de valider des entrées-sorties non bloquantes, et une notification asynchrone des événements par le signal **SIGIO**. L'utilisation de **F_SETOWN** est équivalent à un appel **ioctl**(2) avec l'argument **FSIOSETOWN** ou **SIOCSPRGR**.

Lorsque le réseau indique une condition d'erreur au module du protocole (par exemple l'utilisation d'un message ICMP au lieu d'IP), un drapeau signale une erreur en attente sur la socket. L'opération suivante sur cette socket renverra ce code d'erreur. Pour certains protocoles, il est possible d'activer une file d'attente d'erreurs par socket. Pour plus de détails, voir **IP_RECVERR** dans **ip**(7).

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

S'il réussit, **socket**() retourne un descripteur référençant la socket créée. S'il échoue, il renvoie −1 et *errno* contient le code d'erreur.

ERREURS

EACCES
La création d'une telle socket n'est pas autorisée.

EAFNOSUPPORT
L'implémentation ne supporte pas la famille d'adresses indiquée.

EINVAL
Protocole inconnu, ou famille de protocole inexistante.

EMFILE
La table des fichiers est pleine.

ENFILE
La limite du nombre total de fichiers ouverts sur le système est atteinte.

ENOBUFS ou **ENOMEM**
Pas suffisamment d'espace pour allouer les tampons nécessaires.

EPROTONOSUPPORT
Le type de protocole, ou le protocole lui-même n'est pas disponible dans ce domaine de communication.

D'autres erreurs peuvent être dues aux modules de protocoles sous-jacents.

CONFORMITÉ

BSD 4.4, POSIX.1-2001. La fonction **socket**() est apparue dans BSD 4.2. Elle est généralement portable de/vers les systèmes non-BSD supportant des clones des sockets BSD (y compris les variantes de System V).

NOTE

Les constantes explicites utilisées sous BSD 4.* pour les familles de protocoles sont PF_UNIX, PF_INET... et AF_UNIX... sont utilisées pour les familles d'adresses. Toutefois, même la page de manuel de BSD indiquait « La famille de protocoles est généralement la même que la famille d'adresse », et les standards ultérieurs utilisent AF_* partout.

BOGUES

SOCK_UUCP n'est pas encore implémentée.

VOIR AUSSI

accept(2), **bind**(2), **connect**(2), **fcntl**(2), **getpeername**(2), **getsockname**(2), **getsockopt**(2), **ioctl**(2), **listen**(2), **read**(2), **recv**(2), **select**(2), **send**(2), **shutdown**(2), **socketpair**(2), **write**(2), **getprotoent**(3), **ip**(7), **socket**(7), **tcp**(7), **udp**(7), **unix**(7)

TRADUCTION

Ce document est une traduction réalisée par Christophe Blaess <http://www.blaess.fr/christophe/> le 13 octobre 1996 et révisée le 14 août 2006.

L'équipe de traduction a fait le maximum pour réaliser une adaptation française de qualité. La version anglaise la plus à jour de ce document est toujours consultable via la commande : « **LANG=C man 2 socket** ». N'hésitez pas à signaler à l'auteur ou au traducteur, selon le cas, toute erreur dans cette page de manuel.