NOM

socket − Créer un point de communication

SYNOPSIS

#include <sys/types.h> /* Consultez NOTES */
#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
AF_UNIX, **AF_LOCAL**	Communication locale	**unix**(7)
AF_INET	Protocoles Internet IPv4	**ip**(7)
AF_INET6	Protocoles Internet IPv6	**ipv6**(7)
AF_IPX	IPX − Protocoles Novell	
AF_NETLINK	Interface utilisateur noyau	**netlink**(7)
AF_X25	Protocole ITU−T X.25 / ISO−8208	**x25**(7)
AF_AX25	Protocole AX.25 radio amateur	
AF_ATMPVC	Accès direct ATM PVCs	
AF_APPLETALK	AppleTalk	**ddp**(7)
AF_PACKET	Interface paquet bas−niveau	**packet**(7)

La socket a le *type* indiqué, ce qui indique 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 maximale fixe.

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

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. Consultez **packet**(7).

Certains types de sockets peuvent ne pas être implémentés par toutes les familles de protocoles.

Depuis Linux 2.6.27, le paramètre *type* a un autre objectif : en plus d'indiquer le type de socket, il peut inclure les valeurs suivantes en les combinant par un OU binaire, pour modifier le comportement de **socket**() :

SOCK_NONBLOCK
Placer l'attribut d'état de fichier **O_NONBLOCK** sur le nouveau descripteur de fichier ouvert. Utiliser cet attribut économise des appels supplémentaires à **fcntl**(2) pour obtenir le même résultat.

SOCK_CLOEXEC
Placer l'attribut « close−on−exec » (**FD_CLOEXEC**) sur le nouveau descripteur de fichier. Consultez la description de l'attribut **O_CLOEXEC** dans **open**(2) pour savoir

pourquoi cela peut être utile.

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 ; consultez **protocols**(5). Consultez **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 système 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 dans le paquet 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).

Un appel à **fcntl**(2) avec l'argument **F_SETOWN** permet de préciser un processus ou un 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 définir le processus ou groupe de processus qui recevront une notification asynchrone des événements d'entrées−sorties par le signal **SIGIO**. L'utilisation de **F_SETOWN** est équivalent à un appel **ioctl**(2) avec l'argument **FIOSETOWN** ou **SIOCSPGRP**.

Lorsque le réseau indique une condition d'erreur au module du protocole (par exemple avec un message ICMP pour 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, consultez **IP_RECVERR** dans **ip**(7).

Les opérations sur les sockets sont contrôlées 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 définir ou lire les options.

VALEUR RENVOYÉE

socket() renvoie 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

EACCES

La création d'une socket avec le type et le protocole indiqués n'est pas autorisée.

EAFNOSUPPORT

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

EINVAL
Protocole inconnu, ou famille de protocole inexistante.

EINVAL
Attributs incorrects dans *type*.

EMFILE
La table des fichiers est pleine.

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

ENOBUFS ou **ENOMEM**
Pas suffisamment d'espace pour allouer les tampons nécessaires. La socket ne peut être créée tant que suffisamment de ressources ne sont pas libérées.

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.

Les attributs **SOCK_NONBLOCK** et **SOCK_CLOEXEC** sont spécifiques à Linux.

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).

NOTES

POSIX.1−2001 ne requiert pas l'inclusion de <*sys/types.h*>, et cet en-tête n'est pas nécessaire sous Linux. Cependant, il doit être inclus sous certaines implémentations historiques (BSD), et les applications portables devraient probablement l'utiliser.

Les constantes explicites utilisées sous BSD 4.x pour les familles de protocoles sont **PF_UNIX**, **PF_INET**, etc. alors que **AF_UNIX**, **AF_INET**, etc. 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'adresses », et les standards ultérieurs utilisent AF_* partout.

EXEMPLE

Un exemple d'utilisation de **socket**() se trouve dans la page de manuel de **getaddrinfo**(3).

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)

« An Introductory 4.3BSD Interprocess Communication Tutorial » et « BSB Interprocess Communication Tutorial », réimprimés dans *UNIX Programmer's Supplementary Documents Volume 1*.

COLOPHON

Cette page fait partie de la publication 3.70 du projet *man−pages* Linux. Une description du projet et des instructions pour signaler des anomalies peuvent être trouvées à l'adresse http://www.kernel.org/doc/man−pages/.

TRADUCTION

Depuis 2010, cette traduction est maintenue à l'aide de l'outil po4a <http://po4a.alioth.debian.org/> par l'équipe de traduction francophone au sein du projet perkamon <http://perkamon.alioth.debian.org/>.

Christophe Blaess <http://www.blaess.fr/christophe/> (1996-2003), Alain Portal <http://manpagesfr.free.fr/> (2003-2006). Julien Cristau et l'équipe francophone de traduction de Debian (2006-2009).

Veuillez signaler toute erreur de traduction en écrivant à <perkamon−fr@traduc.org>.

Vous pouvez toujours avoir accès à la version anglaise de ce document en utilisant la commande « **LC_ALL=C man** <*section*> <*page_de_man*> ».