NOM

accept − Accepter une connexion sur une socket.

SYNOPSIS

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

int accept(int *sock***, struct sockaddr ****adresse***, socklen_t ****longueur***);**

DESCRIPTION

accept() est utilisé généralement avec des processus serveurs orientés connexion. Cet appel système est employé avec les sockets utilisant un protocole en mode connecté (**SOCK_STREAM** et **SOCK_SEQPACKET**). Il extrait la première connexion de la file des connexions en attente, crée une nouvelle socket connectée, et renvoie un nouveau descripteur de fichier qui fait référence à cette socket. La nouvelle socket n'est plus en état d'écoute. La socket originale *sock* n'est pas modifiée par l'appel système.

L'argument *sock* est une socket qui a été créée avec la fonction **socket**(2). attachée à une adresse avec **bind**(2). et attend des connextions après un appel **listen**(2).

L'argument *adresse* est un pointeur sur une structure *sockaddr*. La structure sera remplie avec l'adresse du correspondant se connectant, telle qu'elle est connue par la couche de communication. Le format exact du paramètre *adresse* dépend du domaine dans lequel la communication s'établit. (Voir **socket**(2) et la page de manuel correspondant au protocole). L'argument *longueur* est un paramètre-résultat : il doit contenir initialement la taille de la structure pointée par *adresse*, et est renseigné au retour par la longueur réelle (en octet) de l'adresse remplie. Quand *adresse* vaut NULL, rien n'est rempli.

S'il n'y a pas de connexion en attente dans la file, et si la socket n'est pas marquée comme non-bloquante, **accept**() se met en attente d'une connexion. Si la socket est non-bloquante, et qu'aucune connexion n'est présente dans la file, **accept**() échoue avec l'erreur EAGAIN.

Pour être prévenu de l'arrivée d'une connexion sur une socket on peut utiliser **select**(2) ou **poll**(2). Un événement « lecture » sera délivré lorsqu'une tentative de connexion aura lieu, et on pourra alors appeler **accept**() pour la valider. Autrement, on peut configurer la socket pour qu'elle envoie un signal **SIGIO** lorsqu'une activité la concernant se produit, voir **socket**(7) pour plus de détails.

Pour certains protocoles nécessitant une confirmation explicite, comme DECNet, **accept**() peut être considéré comme extrayant simplement la connexion suivante de la file, sans demander de confirmation. On peut effectuer la confirmation par une simple lecture ou écriture sur le nouveau descripteur, et le rejet en fermant la nouvelle socket. Pour le moment, seul DECNet se comporte ainsi sous Linux.

NOTES

Il n'y a pas nécessairement de connexion en attente après la réception de **SIGIO** ou après que **select**(2) ou **poll**(2) indiquent quelque chose à lire. En effet la connexion peut avoir été annulée à cause d'une erreur réseau asynchrone ou par un autre thread avant que **accept**() ne se termine. Si cela se produit, l'appel bloquera en attendant une autre connexion. Pour s'assurer que **accept**() ne bloquera jamais, la socket *sock* transmise doit avoir l'attribut **O_NONBLOCK** (voir **socket**(7)).

VALEUR RENVOYÉE

S'il réussit, **accept**() renvoie un entier non-négatif, constituant un descripteur pour la nouvelle socket. S'il échoue, l'appel renvoie −1 et *errno* contient le code d'erreur.

GESTION DES ERREURS

Sous Linux, **accept**() renvoie les erreurs réseau déjà en attente sur la socket comme une erreur de l'appel système. Ce comportement diffère d'autres implémentations des sockets BSD. Pour un comportement fiable, une application doit détecter les erreurs réseau définies par le protocole après le **accept**() et les traiter comme des erreurs **EAGAIN**, en réitérant le mécanisme. Dans le cas de TCP/IP, ces erreurs sont **ENETDOWN**, **EPROTO**, **ENOPROTOOPT**, **EHOSTDOWN**, **ENONET**, **EHOSTUNREACH**, **EOPNOTSUPP**, et **ENETUNREACH**.

ERREURS

accept() doit échouer si :

EAGAIN ou **EWOULDBLOCK**
La socket est non-bloquante et aucune connexion n'est présente dans la file.

EBADF
Le descripteur est invalide.

ECONNABORTED
Une connexion a été abandonnée.

EINTR
L'appel système a été interrompu par l'arrivée d'un signal avant qu'une connexion valide ne survienne.

EINVAL
La socket n'est pas en attente de connexion, ou *addrlen* n'est pas valide (par exemple, est négative).

EMFILE
La limite des descripteurs ouverts pour le processus a été atteinte.

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

ENOTSOCK
Le descripteur n'est pas celui d'une socket.

EOPNOTSUPP
La socket de référence n'est pas de type **SOCK_STREAM**.

accept() peut échouer si :

EFAULT
adresse n'est pas dans l'espace d'adressage accessible en écriture.

ENOBUFS, ENOMEM
Par assez de mémoire disponible. En général, cette erreur due à la taille limitée du tampon des sockets, et pas à la mémoire système proprement dite.

EPROTO
Erreur de protocole.

La version Linux de **accept**() peut échouer si :

EPERM
Les règles du firewall interdisent la connexion.

De plus il peut se produire des erreurs réseau dépendant du protocole de la socket. Certains noyaux Linux peuvent renvoyer d'autres erreurs comme **ENOSR, ESOCKTNOSUPPORT**, **EPROTONOSUPPORT**, **ETIMEDOUT.** L'erreur **ERESTARTSYS** peut être rencontrée durant un suivi dans un débogueur.

CONFORMITÉ

SVr4, BSD 4.4 (La fonction **accept**() est apparue dans BSD 4.2).

Sous Linux, la nouvelle socket renvoyée par **accept**() *n'*hérite *pas* des drapeaux d'état de fichier de la socket à l'écoute comme **O_NONBLOCK** et **O_ASYNC**. Ce comportement est différent d'autres implémentations BSD. Les programmes portables ne doivent pas s'appuyer sur cette particularité, et doivent reconfigurer les attributs sur la socket renvoyée par **accept**().

NOTE

Le troisième argument de **accept**() était, à l'origine, déclaré comme un « int * » (ceci dans libc4 et libc5 ainsi que pour beaucoup d'autres systèmes comme BSD 4.*, SunOS 4, SGI). Une proposition de standard POSIX.1g l'a modifié en « size_t * » et c'est ce qu'utilise SunOS. Les dernières propositions POSIX en ont fait un « socklen_t * », ce que suivent les spécifications Single Unix, et la glibc2. Pour citer Linus

Torvalds :

« *Toute* bibliothèque sensée *doit* garder « socklen_t » équivalent à un int. Toute autre chose invaliderait tout le niveau des sockets BSD. POSIX l'avait d'abord remplacé par un size_t, et je m'en suis plaint violemment (ainsi que d'autres heureusement, mais bien entendu pas tant que ça). Le remplacement par un size_t est complètement inutile car size_t à exactement la même taille qu'un int sur les architectures 64 bits par exemple. Et il *a* la même taille qu'un « int » parce que c'était l'interface des sockets BSD. Quoiqu'il en soit, les gens de POSIX ont compris et ont créé un « socklen_t ». Ils n'auraient jamais dû y toucher, mais une fois commencé, ils ont décidé de créer un type spécifique, pour des raisons inavouées (probablement quelqu'un qui ne veut pas perdre la face en expliquant que le premier travail était stupide et ils ont simplement renommé leur bricolage). »

VOIR AUSSI

bind(2), **connect**(2), **listen**(2), **select**(2), **socket**(2)

TRADUCTION

Ce document est une traduction réalisée par Christophe Blaess <http://www.blaess.fr/christophe/> le 9 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 accept** ». N'hésitez pas à signaler à l'auteur ou au traducteur, selon le cas, toute erreur dans cette page de manuel.