Manuales Programacion Linux

TABLA DE CONTENIDO:

Ficheros

OPEN

```
OPEN(2)          Manual del Programador de Linux          OPEN(2)


NOMBRE
       open, creat - abren y posiblemente crean un fichero o dis-
       positivo

SINOPSIS
       #include <sys/types.h>
       #include <sys/stat.h>
       #include <fcntl.h>

       int open(const char *camino, int flags);
       int open(const char *camino, int flags, mode_t modo);
       int creat(const char *camino, mode_t modo);

DESCRIPCIÓN
       open intenta abrir un fichero y devolver un descriptor  de
       fichero  (un  entero pequeño, no negativo, de uso en read,
       write, etc.).

       flags es uno de O_RDONLY, O_WRONLY u O_RDWR que, respecti-
       vamente,  piden  que la apertura del fichero sea solamente
       para lectura, solamente para escritura, o para  lectura  y
       escritura.

       flags puede también ser la combinación, con el operador de
       bits OR (|), de una o más de las siguientes macros:

       O_CREAT Si el fichero no existe, será creado.

       O_EXCL  Cuando se combina con O_CREAT, es un error que  el
               fichero ya exista, y open fallará.  O_EXCL no fun-
               ciona en sistemas de ficheros de red NFS,  progra-
               mas que confíen en él para efectuar tareas de blo-
               queo contendrán  una  condición  de  concurso.  La
               solución  para  efectuar  un  bloqueo  atómico  de
               fichero mediante un fichero de bloqueo es crear un
               fichero único en el mismo sistema de ficheros (por
               ejemplo, incorporando el nombre del ordenador y el
               PID),  utilizar  link(2)  para  hacer un enlace al
               fichero de bloqueo y emplear stat(2) en el fichero
               único para comprobar si su número de enlaces se ha
               incrementado  a  2.  No  hay  que  usar  el  valor
               devuelto por la llamada link().

       O_NOCTTY
               Si  camino se refiere a una terminal -- vea tty(4)
               -- no se convertirá en  la  terminal  controladora
               del  proceso  ni  en  el caso de que el proceso no
               tenga ninguna.

       O_TRUNC Si el fichero ya existe, será truncado.

       O_APPEND
               El fichero se abrirá en modo de sólo-añadir.  Ini-
               cialmente, y antes de cada escritura, el apuntador
               del fichero es posicionado al final  del  fichero,
               como  se haría con lseek.  O_APPEND puede conducir
               a ficheros corruptos en sistemas de  ficheros  NFS
```

si más de un proceso añade datos a un fichero a la vez. Esto es así porque NFS no admite añadir a un fichero, así que el núcleo del cliente ha de simularlo, lo que no puede hacerse sin una condición de concurso.

O_NONBLOCK u O_NDELAY
El fichero se abre en modo no bloqueante. Ni el open ni ninguna operación subsecuente sobre el descriptor de fichero que es devuelto harán que el proceso que hace la llamada espere.

O_SYNC El fichero se abre para E/S síncrona. Cualesquiera operaciones de escritura write sobre el descriptor de fichero resultante bloquearán el proceso que ha efectuado la llamada hasta que los datos hayan sido físicamente escritos en el dispositivo subyacente. Vea no obstante más abajo la sección RESTRICCIONES.

Algunos de estos modificadores opcionales pueden alterarse mediante el empleo de fcntl tras que el fichero haya sido abierto.

El argumento modo especifica los permisos a emplear si se crea un nuevo fichero. Es modificado por la máscara umask del proceso de la forma habitual: los permisos del fichero creado son (modo & ~umask).

Se proporcionan las siguientes constantes simbólicas para modo:

S_IRWXU
00700 el usuario (el propietario del fichero) tiene permisos de lectura, escritura y ejecución

S_IRUSR (S_IREAD)
00400 el usuario tiene permiso de lectura

S_IWUSR (S_IWRITE)
00200 el usuario tiene permiso de escritura

S_IXUSR (S_IEXEC)
00100 el usuario tiene permiso de ejecución

S_IRWXG
00070 el grupo tiene permiso de lectura, escritura y ejecución

S_IRGRP
00040 el grupo tiene permiso de lectura

S_IWGRP
00020 el grupo tiene permiso de escritura

S_IXGRP
00010 el grupo tiene permiso de ejecución

S_IRWXO
00007 los otros tienen permiso de lectura, escritura y ejecución

S_IROTH
00004 los otros tienen permiso de lectura

S_IWOTH
00002 los otros tienen permiso de escritura

S_IXOTH
00001 los otros tienen permiso de ejecución

El argumento modo siempre debe especificarse cuando O_CREAT está en flags, y si no está, no es tenido en cuenta.

creat equivale a open con flags igual a O_CREAT|O_WRONLY|O_TRUNC.

VALOR DEVUELTO

open y creat devuelven el nuevo descriptor de fichero, o -1 si ha ocurrido un error (en cuyo caso, se da un valor apropiado a errno). Observe que open pueden abrir ficheros especiales de dispositivo, pero creat no puede crearlos; emplee mknod(2) en su lugar.

En sistemas de ficheros NFS con asociación de UIDs habilitada, open puede devolver un descriptor de fichero pero p. ej. llamadas a read(2) pueden denegarse con el error EACCES. Esto es así porque el cliente hace el open comprobando los permisos, pero la asociación de UID la hace el servidor sobre las peticiones de lectura y escritura.

ERRORES

EEXIST camino ya existe y se usaron O_CREAT y O_EXCL.

EISDIR camino se refiere a un directorio y el acceso pedido implicaba escribir.

ETXTBSY camino se refiere a una imagen ejecutable que se está ejecutando ahora mismo y se pidió un acceso de escritura.

EFAULT camino apunta afuera de su espacio de direcciones accesible.

EACCES El acceso pedido al fichero no es permitido, o uno de los directorios en camino no tenía permiso de búsqueda o paso (ejecución).

ENAMETOOLONG
camino era demasiado largo.

ENOENT Un componente directorio en camino no existe o es un enlace simbólico colgante (que apunta a un fichero inexistente).

ENOTDIR Un componente de camino empleado como un directorio no es, de hecho, un directorio.

EMFILE El proceso ya tiene abiertos todos los ficheros que puede.

ENFILE Se ha alcanzado el límite del número total de ficheros abiertos en el sistema.

ENOMEM No hay disponible bastante memoria del núcleo.

EROFS camino se refiere a un fichero de un sistema de ficheros de lectura exclusiva y se ha pedido un acceso de escritura.

ELOOP Se han encontrado demasiados enláces simbólicos al resolver camino.

ENOSPC camino iba a ser creado pero el dispositivo que lo habría de contener no tiene espacio para el nuevo

fichero.

CONFORME A
SVr4, SVID, POSIX, X/OPEN, BSD 4.3

RESTRICCIONES
Hay muchas impropiedades en el protocolo subyacente a NFS, afectando entre otros a O_SYNC y a O_NDELAY.

VÉASE TAMBIÉN
read(2), write(2), fcntl(2), close(2), unlink(2), mknod(2), stat(2), umask(2), mount(2), socket(2), fopen(3), link(2).

CLOSE

CLOSE(2) Manual del Programador de Linux CLOSE(2)

NOMBRE
close - cierra un descriptor de fichero

SINOPSIS
```
#include <unistd.h>

int close(int fd);
```

DESCRIPCIÓN
close cierra un descriptor de fichero de forma que ya no se refiera a fichero alguno y pueda ser reutilizado. Cualesquiera bloqueos mantenidos sobre el fichero con el que estaba asociado, y propiedad del proceso, son eliminados (sin importar qué descriptor de fichero fue utilizado para obtener el bloqueo).

Si fd es la última copia de cierto descriptor de fichero, los recursos asociados con dicho descriptor son liberados; si el descriptor fuera la última referencia a un fichero que haya sido eliminada mediante unlink entonces el fichero es borrado.

VALOR DEVUELTO
close devuelve 0 en caso de éxito y -1 si ocurre algún error.

ERRORES
EBADF fd no es un descriptor de fichero abierto válido.

CONFORME A
SVID, AT&T, POSIX, X/OPEN, BSD 4.3. SVr4 documenta una condición de error ENOLINK adicional.

NOTA
El no comprobar el valor devuelto por close es un error de programación común y no obstante serio. Aquellas implementaciones de sistemas de ficheros que usan técnicas tales como la conocida por ``write-behind'' (``escribe por detrás'') a fin de incrementar el rendimiento pueden resultar en que write(2) tenga éxito aunque aún no se hayan escrito los datos. El estado de error puede ser informado durante una posterior operación de escritura, pero está garantizado que será informado al cerrar el fichero. No comprobar el valor devuelto cuando se cierra un fichero puede dar lugar a una pérdida silenciosa de

```
       datos.   Esto  se  observa  especialmente en NFS y con las
       cuotas de discos.

VÉASE TAMBIÉN
       open(2), fcntl(2), shutdown(2), unlink(2), fclose(3).
```

READ

```
READ(2)          Manual del Programador de Linux          READ(2)


NOMBRE
       read - lee de un descriptor de fichero

SINOPSIS
       #include <unistd.h>

       ssize_t read(int fd, void *buf, size_t nbytes);

DESCRIPCIÓN
       read()  intenta  leer  hasta nbytes bytes del fichero cuyo
       descriptor de fichero es fd y guardarlos  en  la  zona  de
       memoria que empieza en buf.

       Si  nbytes  es  cero, read() devuelve cero y no tiene otro
       efecto.  Si nbytes es mayor que SSIZE_MAX, el resultado es
       indefinido.


VALOR DEVUELTO
       En  caso  de  éxito, se devuelve el número de bytes leídos
       (cero indica fin de fichero), y el indicador  de  posición
       del  fichero  avanza este número de bytes.  No es un error
       si este número es menor que el número  de  bytes  pedidos;
       esto  puede  suceder  por  ejemplo porque ahora mismo haya
       disponible un número menor de bytes (quizás porque estamos
       cerca  del fin-de-fichero, o porque estamos leyendo de una
       interconexión, o de una terminal), o porque read() ha sido
       interrumpido por una señal.  En caso de error, se devuelve
       -1, y se pone un valor apropiado en errno. En este caso se
       deja indeterminado si el indicador de posición del fichero
       (si lo hay) cambia o no.

ERRORES
       EINTR   La llamada ha  sido  interrumpida  por  una  señal
               antes de que se haya leído ningún dato.

       EAGAIN  Se  ha  seleccionado  E/S  no bloqueante empleando
               O_NONBLOCK y no había ningún  dato  inmediatamente
               disponible para la lectura.

       EIO     Error  de  E/S.  Esto  puede  ocurrir  por ejemplo
               cuando el proceso está en un grupo de procesos  en
               segundo  plano,  intenta  leer  de  su tty contro-
               ladora, y o  está  bloqueando  o  no  teniendo  en
               cuenta  a  SIGTTIN  o  su  grupo  de procesos está
               huérfano. También  puede  ocurrir  cuando  hay  un
               error  de  E/S de bajo nivel mientras se lee de un
               disco o cinta.

       EISDIR  fd se refiere a un directorio.

       EBADF   fd no es un descriptor de fichero válido o no está
               abierto para lectura.
```

EINVAL fd está asociado a un objeto que no es apropiado para su lectura.

EFAULT buf está fuera del espacio de direcciones accesible del usuario.

Pueden ocurrir otros errores, dependiendo del objeto conectado a fd. POSIX permite que un read que se interrumpa tras leer algunos datos devuelva -1 (con el valor EINTR en errno) o que devuelva el número de bytes ya leídos.

CONFORME A

SVr4, SVID, AT&T, POSIX, X/OPEN, BSD 4.3

RESTRICCIONES

En sistemas de ficheros NFS, leer cantidades pequeñas de datos sólo actualizará la fecha de acceso al fichero la primera vez, las demás llamadas pueden no hacerlo más. Esto está producido por el mecanismo de caché en la parte cliente, porque la mayoría si no todos los clientes NFS le dejan las actualizaciones de la fecha/hora de acceso al servidor y las lecturas en la parte del cliente satisfechas por el caché del cliente no provocarán actualizaciones del atime (fecha/hora de acceso) en el servidor puesto que no hay lecturas en la parte del servidor. La semántica de UNIX puede obtenerse deshabilitando el atributo de caché en la parte cliente, pero en la mayoría de las situaciones esto aumentará sustancialmente la carga del servidor y disminuirá el rendimiento.

VÉASE TAMBIÉN

readdir(2), write(2), write(2), fcntl(2), close(2), lseek(2), select(2), readlink(2), ioctl(2), fread(3).

WRITE

WRITE(2) Manual del Programador de Linux WRITE(2)

NOMBRE

write - escribe a un descriptor de fichero

SINOPSIS

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t num);
```

DESCRIPCIÓN

write escribe hasta num bytes en el fichero referenciado por el descriptor de fichero fd desde el búfer que comienza en buf. POSIX requiere que un read() que pueda demostrarse que ocurra después que un write() haya regresado, devuelva los nuevos datos. Observe que no todos los sistemas de ficheros son conformes con POSIX.

VALOR DEVUELTO

En caso de éxito, se devuelve el número de bytes escritos (cero indica pues que no se ha escrito nada). En caso de error, se devuelve -1 y se pone un valor apropiado en errno. Si num es cero y el descriptor de fichero se refiere a un fichero regular, se devolverá 0 sin que se cause ningún otro efecto. Para un fichero especial, los

resultados no son transportables.

ERRORES

EBADF fd no es un descriptor válido de fichero o no está abierto para escritura.

EINVAL fd está asociado a un objeto que no es adecuado para la escritura.

EFAULT buf está afuera del espacio de direcciones accesible.

EPIPE fd está conectado a una tubería o zócalo cuyo extremo de lectura está cerrado. Cuando esto ocurre el proceso de escritura recibirá una señal SIGPIPE; si la captura, bloquea o no tiene en cuenta, se devuelve el error EPIPE.

EAGAIN Se ha seleccionado E/S no bloqueante empleando O_NONBLOCK y no había sitio en la tubería o zócalo conectado a fd para escribir los datos inmediatamente.

EINTR La llamada ha sido interrumpida por una señal antes de que se haya escrito ningún dato.

ENOSPC El dispositivo que contiene al fichero referenciado por fd no tiene sitio para los datos.

EIO Ha ocurrido un error de E/S de bajo nivel mientras se estaba modificando el nodo-í.

Pueden ocurrir otros errores, dependiendo del objeto conectado a fd.

CONFORME A

SVr4, SVID, POSIX, X/OPEN, 4.3BSD. SVr4 documenta condiciones de error adicionales EDEADLK, EFBIG, ENOLCK, ENOLNK, ENOSR, ENXIO, EPIPE, o ERANGE. En SVr4 una escritura puede ser interrumpida y devolver EINTR en cualquier momento, no sólo justo antes de que se escriba algún dato.

VÉASE TAMBIÉN

open(2), read(2), fcntl(2), close(2), lseek(2), select(2), ioctl(2), fsync(2), fwrite(3).

LSEEK

LSEEK(2) Manual del Programador de Linux LSEEK(2)

NOMBRE

lseek - reposiciona el puntero de lectura/escritura de un fichero

SINOPSIS

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

off_t lseek(int fildes, off_t offset, int whence);
```

DESCRIPCIÓN

La función lseek reposiciona el puntero del descriptor de

fichero fildes con el argumento offset de acuerdo con la directiva whence as follows:

SEEK_SET
El puntero se coloca a offset bytes.

SEEK_CUR
El número de bytes indicado en offset se suma a la dirección actual y el puntero se coloca en la dirección resultante.

SEEK_END
El puntero se coloca al final del fichero más offset bytes.

La función lseek permite colocar el puntero de fichero después del final de fichero. Si después se escriben datos en este punto, las lecturas siguientes de datos dentro del hueco que se forma devuelven ceros (hasta que realmente se escriban datos dentro de ese hueco).

VALOR DEVUELTO

En el caso de una ejecución correcta, lseek devuelve la posición del puntero resultante medida en bytes desde el principio del fichero. Si se produce un error, se devuelve el valor (off_t)-1 y en errno se coloca el tipo de error.

ERRORES

EBADF fildes no es un descriptor de fichero abierto.

ESPIPE fildes está asociado a una tubería, socket, o FIFO.

EINVAL whence no es un valor adecuado.

CONFORME A

SVr4, POSIX, BSD 4.3

RESTRICCIONES

Algunos dispositivos son incapaces de buscar y POSIX no especifica qué dispositivos deben soportar la búsqueda.

Restricciones específicas de Linux: el uso de lseek sobre un dispositivo tty (terminal) devuelve ESPIPE. Otros sistemas devuelven el número de caracteres escritos, usando SEEK_SET para establecer el contador. Algunos dispositivos, como por ejemplo /dev/null, no provocan el error ESPIPE, pero devuelven un puntero cuyo valor es indefinido.

NOTAS

La utilización de whence en este documento es incorrecta en inglés, pero se usa por motivos históricos. Cuando convierta código antiguo, sustituya los valores para whence con las siguientes macros:

antiguo	nuevo
0	SEEK_SET
1	SEEK_CUR
2	SEEK_END
L_SET	SEEK_SET
L_INCR	SEEK_CUR
L_XTND	SEEK_END

SVR1-3 devuelve long en lugar de off_t, BSD devuelve int.

VÉASE TAMBIÉN

dup(2), open(2), fseek(3)

FCNTL

```
FCNTL(2)         Manual del Programador de Linux         FCNTL(2)


NOMBRE
       fcntl - manipula el descriptor de fichero

SINOPSIS
       #include <unistd.h>
       #include <fcntl.h>

       int fcntl(int fd, int cmd);
       int fcntl(int fd, int cmd, long arg);

DESCRIPCIÓN
       fcntl  realiza  una de las diversas y variadas operaciones
       sobre fd.  La operación en cuestión se determina  mediante
       cmd:

       F_DUPFD  Hace  que arg sea una copia de fd, cerrando fd si
                es necesario.

                El mismo resultado se  puede  obtener  fácilmente
                usando dup2.

                Los  descriptores  antiguo  y nuevo pueden usarse
                indistintamente.   Ambos    comparten    candados
                (locks),  indicadores  de  posición de ficheros y
                banderas (flags); por ejemplo, si la posición del
                fichero  se  modifica  usando lseek en uno de los
                descriptores, la posición del otro resulta  modi-
                ficada simultáneamente.

                Sin embargo, los dos descriptores no comparten la
                bandera close-on-exec  "cerrar-en-ejecución".  La
                bandera  close-on-exec  de la copia está desacti-
                vada, singificando que se cerrará en ejecución.

                En caso de éxito, se devuelve el  nuevo  descrip-
                tor.

       F_GETFD  Lee  la bandera close-on-exec. Si el bit de bajo-
                orden  es  0,  el  fichero  permanecerá  abierto
                durante  exec,  en  caso  contrario se cerrará el
                fichero.

       F_SETFD  Asigna el valor de la  bandera  close-on-exec  al
                valor  especificado  por arg (solamente se usa el
                bit menos significativo).


       F_GETFL  Lee las banderas del descriptor (todas  las  ban-
                deras,  según  hayan  sido asignadas por open(2),
                serán devueltas).

       F_SETFL  Asigna las banderas del descriptor al valor asig-
                nado por arg.  Sólo O_APPEND y  O_NONBLOCK pueden
                asignarse.

                Las banderas se comparten  entre  copias  (hechas
                con dup etc.) del mismo descriptor de fichero.
```

Las banderas y su semántica están descritas en open(2).

F_GETLK, F_SETLK y F_SETLKW
Gestionan candados de ficheros discrecionales (discretionary file locks). El tercer argumento arg es un puntero a una struct flock (que puede ser sobrescrita por esta llamada).

F_GETLK Devuelve la estructura flock que nos impide obtener el candado, o establece el campo l_type del candado a F_UNLCK si no hay obstrucción.

F_SETLK El candado está cerrado (cuando l_type es F_RDLCK o F_WRLCK) o abierto (cuando es F_UNLCK). Si el candado está cogido por alguien más, esta llamada devuelve -1 y pone en errno el código de error EACCES o EAGAIN.

F_SETLKW Como F_SETLK, pero en vez de devolver un error esperamos que el candado se abra. Si se recibe una señal a capturar mientras fcntl() está esperando, se interrumpe y regresa inmediatamente (devolviendo -1 y asignado a errno el valor EINTR).

F_GETOWN Obtiene el ID de proceso o el grupo de procesos que actualmente recibe las señales SIGIO y SIGURG para los eventos sobre el descriptor de fichero fd.

Los grupos de procesos se devuelven como valores negativos.

F_SETOWN Establece el ID de proceso o el grupo de procesos que recibirá las señales SIGIO y SIGURG para los eventos sobre el descriptor de fichero fd.

Los grupos de procesos se especifican mediante valores negativos.

Si activa la bandera de estado O_ASYNC sobre un descriptor de fichero (tanto si proporciona esta bandera con la llamada open como si usa la orden F_SETFL de fcntl), se enviará una señal SIGIO cuando sea posible la entrada o la salida sobre ese descriptor de fichero. El proceso o el grupo de procesos que recibirá la señal se puede seleccionar usando la orden F_SETOWN de la función fcntl. Si el descriptor de fichero es un enchufe (socket), esto también seleccionará al recipiente de las señales SIGURG que se entregan cuando llegan datos fuera de orden (out-of-band, OOB) sobre el enchufe. (SIGURG se envía en cualquier situación en la que select informaría que el enchufe tiene una "condición excepcional"). Si el descriptor de fichero corresponde a un dispositivo de terminal, entonces las señales SIGIO se envían al grupo de procesos en primer plano de la terminal.

El uso de O_ASYNC, F_GETOWN y F_SETOWN es específico de BSD. POSIX tiene E/S asíncrona y la estructura aio_sigevent para conseguir cosas similares.

VALOR DEVUELTO

Para una llamada con éxito, el valor devuelto depende de la operación:

F_DUPFD El nuevo descriptor.

F_GETFD Valor de la bandera.

F_GETFL Valor de las banderas.

F_GETOWN Valor del propietario del descriptor.

F_SETFD, F_SETFL, F_GETLK, F_SETLK, F_SETLKW Algún valor distinto de -1.

En caso de error el valor devuelto es -1, y se pone un valor apropiado en errno.

ERRORES

EACCES La operación está prohibida por candados mantenidos por otros procesos.

EAGAIN La operación está prohibida porque el fichero ha sido asociado a memoria por otro proceso.

EDEADLK Se ha detectado que el comando F_SETLKW especificado provocaría un interbloqueo.

EBADF fd no es un descriptor de fichero abierto.

EINTR El comando F_SETLKW ha sido interrumpido por una señal.

EINVAL Para F_DUPFD, arg es negativo o mayor que el valor máximo permitido.

EMFILE Para F_DUPFD, el proceso ya ha llegado al número máximo de descriptores de ficheros abiertos.

ENOLCK Demasiados candados de segmento abiertos, la tabla de candados está llena.

NOTAS

Los errores devueltos por dup2 son distintos de aquéllos dados por F_DUPFD.

CONFORME A

SVID, AT&T, POSIX, X/OPEN, BSD 4.3. Sólo las operaciones F_DUPFD, F_GETFD, F_SETFD, F_GETFL, F_SETFL, F_GETLK, F_SETLK y F_SETLKW se especifican en POSIX.1; F_GETOWN y F_SETOWN son BSD-ismos no aceptados en SVr4. Las banderas legales para F_GETFL/F_SETFL son aquéllas que acepta open(2) y varían entre estos sistemas; O_APPEND, O_NONBLOCK, O_RDONLY y O_RDWR son las que se mencionan en POSIX.1. SVr4 admite algunas otras opciones y banderas no documentadas aquí.

POSIX.1 documenta una condición de error adicional EINTR. SVr4 documenta las condiciones de error adicionales EFAULT, EINTR, EIO, ENOLINK y EOVERFLOW.

VÉASE TAMBIÉN

open(2), socket(2), dup2(2), flock(2).

DUP

```
DUP(2)           Manual del Programador de Linux           DUP(2)


NOMBRE
       dup, dup2 - duplica un descriptor de fichero

SINOPSIS
       #include <unistd.h>

       int dup(int oldfd);
       int dup2(int oldfd, int newfd);

DESCRIPCIÓN
       dup  y  dup2  crean  una  copia  del descriptor de fichero
       oldfd.

       Los descriptores antiguo y nuevo  pueden  usarse  indifer-
       entemente.  Comparten  candados  (locks),  indicadores  de
       posición de fichero y banderas (flags); por ejemplo, si la
       posición  del  fichero  se modifica usando lseek en uno de
       los descriptores, la posición en el otro también cambia.

       Sin embargo  los  descriptores  no  comparten  la  bandera
       close-on-exec, (cerrar-al-ejecutar).

       dup  usa  el descriptor libre con menor numeración posible
       como nuevo descriptor.

       dup2 hace que  newfd  sea  la  copia  de  oldfd,  cerrando
       primero newfd si es necesario.

VALOR DEVUELTO
       dup  y  dup2 devuelven el valor del nuevo descriptor, ó -1
       si ocurre algún error, en cuyo caso errno  toma  un  valor
       apropiado.

ERRORES
       EBADF  oldfd  no  es  un  descriptor de fichero abierto, o
              newfd está fuera del rango permitido para  descrip-
              tores de ficheros.

       EMFILE El  proceso  ya  tiene el máximo número de descrip-
              tores de fichero abiertos y se ha  intentado  abrir
              uno nuevo.

ADVERTENCIA
       El  error  devuelto por dup2 es diferente del devuelto por
       fcntl(...,F_DUPFD,...)  cuando newfd está fuera de  rango.
       En  algunos  sistemas  dup2  a  veces devuelve EINVAL como
       F_DUPFD.

CONFORME A
       SVID, AT&T, POSIX, X/OPEN, BSD  4.3.  SVr4  documenta  las
       condiciones  de error adicionales EINTR y ENOLINK. POSIX.1
       añade EINTR.

VÉASE TAMBIÉN
        fcntl (2), open (2), close (2).
```

FSTAT

```
STAT(2)                Llamadas al Sistema                STAT(2)


NOMBRE
```

stat, fstat, lstat - obtiene el estado de un fichero

SINOPSIS

```
#include <sys/stat.h>
#include <unistd.h>

int stat(const char *file_name, struct stat *buf);
int fstat(int filedes, struct stat *buf);
int lstat(const char *file_name, struct stat *buf);
```

DESCRIPCIÓN

Estas funciones devuelven información del fichero especificado. No se necesitan derechos de acceso al fichero para conseguir la información pero sí se necesitan derechos de búsqueda para todos los directorios del camino al fichero.

stat examina el fichero al que apunta file_name y llena buf.

lstat es idéntico a stat, solo que examina únicamente el enlace, no el fichero que se obtiene al seguir los enlaces.

fstat es idéntico a stat, pero sólo el fichero abierto apuntado por filedes (tal y como lo devuelve open(2)) es examinado en lugar de file_name.

Todos devuelven una estructura stat, que contien los siguientes campos:

```
struct stat
{
    dev_t         st_dev;      /* dispositivo */
    ino_t         st_ino;      /* inodo */
    mode_t        st_mode;     /* protección */
    nlink_t       st_nlink;    /* número de enlaces
físicos */
    uid_t         st_uid;      /* ID del usuario
propietario */
    gid_t         st_gid;      /* ID del grupo propietario
*/
    dev_t         st_rdev;     /* tipo dispositivo (si es
                                   dispositivo inodo) */
    off_t         st_size;     /* tamaño total, en bytes
*/
    unsigned long st_blksize;  /* tamaño de bloque para el
                                   sistema de ficheros de
E/S */
    unsigned long st_blocks;   /* número de bloques
asignados */
    time_t        st_atime;    /* hora último acceso */
    time_t        st_mtime;    /* hora última modificación
*/
    time_t        st_ctime;    /* hora último cambio */
};
```

Nótese que st_blocks podría no siempre estar en términos de bloques de tamaño st_blksize, y que st_blksize podría entonces dar una noción del tamaño de bloque "preferido"

para una E/S eficiente del sistema de ficheros.

No todos los sistemas de ficheros en Linux implementan todos los campos de hora. Por lo general, st_atime es cambiado por mknod(2), utime(2), read(2), write(2) y truncate(2).

Por lo general, st_mtime es cambiado por mknod(2), utime(2) y write(2). st_mtime no se cambia por modificaciones en el propietario, grupo, cuenta de enlaces físicos o modo.

Por lo general, st_ctime es cambiado al escribir o al poner información del inodo (p.ej., propietario, grupo, cuenta de enlaces, modo, etc.).

Se definen las siguientes macros POSIX para comprobar el tipo de fichero:

S_ISLNK(m) es un enlace simbólico?

S_ISREG(m) un fichero regular?

S_ISDIR(m) un directorio?

S_ISCHR(m) un dispositivo de caracteres?

S_ISBLK(m) un dispositivo de bloques?

S_ISFIFO(m) una tubería nombrada (fifo)?

S_ISSOCK(m) un enchufe (socket)?

Se definen las siguientes banderas para el campo st_mode:

S_IFMT máscara de bits 00170000 para los campos de bit del tipo de fichero (no POSIX)

S_IFSOCK 0140000 enchufe (no POSIX)

S_IFLNK 0120000 enlace simbólico (no POSIX)

S_IFREG 0100000 fichero regular (no POSIX)

S_IFBLK 0060000 dispositivo de bloques (no POSIX)

S_IFDIR 0040000 directorio (no POSIX)

S_IFCHR 0020000 dispositivo de caracteres (no POSIX)

S_IFIFO 0010000 fifo o tubería nombrada (no POSIX)

S_ISUID 0004000 poner bit UID

S_ISGID 0002000 poner bit GID

S_ISVTX 0001000 sticky bit (no POSIX)

S_IRWXU 00700 usuario (propietario del fichero) tiene permisos de lectura, escritura y ejecución

S_IRUSR 00400 usuario tiene premiso de lectura (igual que S_IREAD, que no es POSIX)

S_IWUSR 00200 usuario tiene premiso de escritura (igual que S_IWRITE, que no es POSIX)

S_IXUSR 00100 usuario tiene premiso de ejecución (igual que S_IEXEC, que no es POSIX)

S_IRWXG 00070 grupo tiene permisos de lectura, escritura y ejecución

```
                S_IRGRP  00040 grupo tiene permiso de lectura

                S_IWGRP  00020 grupo tiene permiso de escritura

                S_IXGRP  00010 grupo tiene permiso de ejecución

                S_IRWXO  00007 otros tienen  permisos  de  lectura,
                         escritura y ejecución

                S_IROTH  00004 otros tienen permiso de lectura

                S_IWOTH  00002 otros tienen permiso de escritura

                S_IXOTH  00001 otros tienen permiso de ejecución

VALOR DEVUELTO
       Se devuelve cero si hubo éxito. Si hubo error, se devuelve
       -1, y errno es actualizado apropiadamente.

ERRORES
       EBADF  filedes incorrecto.

       ENOENT No existe un componente del camino file_name  o  el
              camino es una cadena vacía.

       ENOTDIR
              Un componente del camino no es un directorio.

       ELOOP  Se  han encontrado demasiados enlaces simbólicos al
              recorrer el camino.

       EFAULT Dirección errónea.

       EACCES Permiso denegado.

       ENOMEM Fuera de memoria (es decir, memoria del núcleo).

       ENAMETOOLONG
              Nombre de fichero demasiado largo.

CONFORME A
       Las llamadas stat y fstat conforman con SVr4, SVID, POSIX,
       X/OPEN  y  BSD 4.3. La llamada lstat conforma con 4.3BSD y
       SVr4.  SVr4 documenta condiciones de error adicionales  de
       fstat:  EINTR, ENOLINK y EOVERFLOW.  SVr4 documenta condi-
       ciones de error  adicionales  de  stat  y  lstat:  EACCES,
       EINTR, EMULTIHOP, ENOLINK y EOVERFLOW.

VÉASE TAMBIÉN
       chmod(2), chown(2), readlink(2), utime(2)
```

Strings - Funciones de tratamiento

Listado de funciones

```
STRING(3)        Manual del Programador de Linux        STRING(3)


NOMBRE
       strcasecmp, strcat, strchr, strcmp, strcoll, strcpy, strc-
       spn, strdup, strfry, strlen,  strncat,  strncmp,  strncpy,
       strncasecmp,  strpbrk,  strrchr,  strsep,  strspn, strstr,
       strtok, strxfrm, index, rindex - operaciones  con  cadenas
       de caracteres
```

SINOPSIS

```
#include <string.h>

int strcasecmp(const char *s1, const char *s2);

char *strcat(char *dest, const char *orig);

char *strchr(const char *s, int c);

int strcmp(const char *s1, const char *s2);

int strcoll(const char *s1, const char *s2);

char *strcpy(char *dest, const char *orig);

size_t strcspn(const char *s, const char *reject);

char *strdup(const char *s);

char *strfry(char *string);

size_t strlen(const char *s);

char *strncat(char *dest, const char *orig, size_t n);

int strncmp(const char *s1, const char *s2, size_t n);

char *strncpy(char *dest, const char *orig, size_t n);

int strncasecmp(const char *s1, const char *s2, size_t n);

char *strpbrk(const char *s, const char *accept);

char *strrchr(const char *s, int c);

char *strsep(char **stringp, const char *delim);

size_t strspn(const char *s, const char *accept);

char *strstr(const char *haystack, const char *needle);

char *strtok(char *s, const char *delim);

size_t strxfrm(char *dest, const char *orig, size_t n);

char *index(const char *s, int c);

char *rindex(const char *s, int c);
```

DESCRIPCIÓN

Las funciones de manejo de cadenas de caracteres realizan operaciones en cadenas de caracteres acabadas en el carácer de código cero. Consulte las páginas individuales del Manual para ver las descripciones de cada función.

CONSULTE TAMBÉN

index(3), rindex(3), strcasecmp(3), strcat(3), strchr(3), strcmp(3), strcoll(3), strcpy(3), strcspn(3), strdup(3), strfry(3), strlen(3), strncat(3), strncmp(3), strncpy(3), strncasecmp(3), strpbrk(3), strrchr(3), strsep(3), str-spn(3), strstr(3), strtok(3), strxfrm(3)

STRCMP

```
STRCMP(3)        Manual del Programador de Linux        STRCMP(3)


NOMBRE
       strcmp, strncmp - comparar dos cadenas de caracteres

SINOPSIS
       #include <string.h>

       int strcmp(const char *s1, const char *s2);

       int strncmp(const char *s1, const char *s2, size_t n);

DESCRIPCIÓN
       La  función strcmp() compara las dos cadenas de caracteres
       s1 y s2.  Devuelve un entero menor, igual o mayor que cero
       si  se  encuentra  que  s1 es, respectivamente, menor que,
       igual a (concordante), o mayor que s2.

       La función strncmp() es similar, salvo que solamente  com-
       para los primeros n caracteres de s1.

VALOR DEVUELTO
       Las  funciones  strcmp()  y  strncmp() devuelven un entero
       menor que, igual a, o mayor que cero si s1 (o los primeros
       n bytes en el segundo caso) se encuentra que es, respecti-
       vamente, menor que, igual a, o mayor que s2.

CONFORME A
       SVID 3, POSIX, BSD 4.3, ISO 9899

VÉASE TAMBIÉN
       bcmp(3), memcmp(3),  strcasecmp(3),  strncasecmp(3),  str-
       coll(3)
```

SPRINTF

```
PRINTF(3)        Manual del Programador de Linux        PRINTF(3)


NOMBRE
       printf,  fprintf,  sprintf,  vprintf, vfprintf, vsprintf -
       conversión de salida formateada

SINOPSIS
       #include <stdio.h>

       int printf( const char *format, ...);
       int fprintf( FILE *stream, const char *format, ...);
       int sprintf( char *str, const char *format, ...);

       #include <stdarg.h>

       int vprintf( const char *format, va_list ap);
       int vfprintf( FILE *stream, const  char  *format,  va_list
       ap);
       int vsprintf( char *str, char *format, va_list ap);

DESCRIPCIÓN
       La  familia  de  funciones  printf  produce  una salida de
       acuerdo a format como se describe abajo.  Printf y vprintf
       escriben  su salida a stdout, el flujo de salida estándar;
       fprintf y vfprintf escriben su salida al stream de  salida
       dado;  sprintf, y vsprintf escriben a una cadena de carac-
```

teres str.

Todas estas funciones escriben la salida bajo el control de una cadena format que especifica cómo los argumentos posteriores (o los argumentos accedidos mediante las facilidades de argumentos de longitud variables proporionadss por stdarg(3)) son convertidos para su salida.

Estas funciones devuelven el número de caracteres impresos (no incluyendo el carácter `\0' usado para terminar la salida de cadenas). snprintf y vsnprintf no escriben más de size bytes (incluyendo el carácter terminador '\0'), y devuelven -1 si la salida se ha truncado debido a esta limitación.

La cadena format está compuesta de cero o más directivas: caracteres ordinarios (no %)- que se copian sin cambios al flujo de salida, e indicaciones de conversión, cada una de las cuales produce la búsqueda de cero o más argumentos posteriores. Cada especificación de conversión se introduce mediante el carácter %. Los argumentos deben corresponder adecuadamente (tras la promoción de tipos) con el indicador de conversión. Después de %, los siguientes caracteres pueden aparecer en secuencia:

· Cero o más de las siguientes banderas:

indica que el valor debe ser convertido a un ``formato alternativo''. Para las conversiones c, d, i, n, p, s, y u, esta opción no tiene efecto. Para la conversión o, se incrementa la precisión del número para hacer que el primer carácterer de la cadena de salida sea cero (excepto si se imprime el valor cero con una precisión explícita de cero). Para las conversiones x y X, la cadena `0x' (o `0X' para conversiones X) precede a los resultados que son distintos de 0. Para las conversiones e, E, f, g, y G, el resultado contendrá un punto decimal, aún si ningún dígito lo sigue (normalmente, sólo aparece un punto decimal en el resultado de aquellas conversiones que son seguidas de algún dígito). Para las conversiones g y G, en el resultado no se eliminan los ceros del final, como ocurriría en otro caso.

0 indica el relleno con ceros. Para todas las converiones excepto para n, el valor convertido es rellenado a la izquierda con ceros en vez de blancos. Si en una conversión numérica (d, i, o, u, x, y X), se indica una precisión, la bandera 0 se ignora.

- (una bandera de ancho de campo negativo) indica que el valor convertido es justificado a la izquierda sobre el límite del campo. Excepto para conversiones n, el valor convertido es rellenado a la derecha con blancos, en vez de a la izquierda con blancos o ceros. Un - sobreescribe un 0 si se indican ambos.

(un espacio) indica que se debe dejar un espacio en blanco delante de un número positivo producido por una conversión de signo (d, e, E, f, g, G, o i).

+ indica que siempre se colocará el signo delante de un número producido por una conversión con signo. Un + sobreescribe un espacio si se usan ambos.

' indica que en un argumento numérico la salida va a ser agrupada si la información de localización así lo indica. Dése cuenta que muchas versiones de gcc no pueden analizar esta opción y producirán un "warning".

· Una cadena de dígitos decimales opcionales indicando un ancho de campo mínimo. Si el valor convertido tiene menos caracteres que el ancho del campo, se rellenará con espacios a la izquierda (o a la derecha si se ha indicado la bandera de justificación a la izquierda) para abarcar el ancho del campo.

· Una precisión opcional, indicada por un punto (`.') seguido por una cadena de dígitos también opcional. Si se omite la cadena de dígitos, la precisión se toma como cero. Esto da el número mínimo de dígitos que deben aparecer en las conversiones d, i, o, u, x, y X, el número de dígitos que deben aparacer tras el punto decimal en las conversiones e, E, y f, el máximo número de dígitos significativos para las conversiones g y G, o el máximo número de caracteres a imprimir de una cadena en las conversiones s

· El carácter opcional h, que indica que la siguiente conversión d, i, o, u, x, o X corresponden a un argumento short int o unsigned short int, o que la siguiente conversión n corresponde a un puntero a un argumento short int

· El caracter opcional l (ele) indica que la siguiente conversión d, i, o, u, x, o X se aplica a un puntero a un argumento long int o unsigned long int, que la siguiente conversión n corresponde a un puntero a un argumento long int. Linux proporciona un uso no conforme a ANSI de dos banderas l como un sinónimo de q o L. Así ll se puede usar junto a las conversiones de coma flotante. Sin embargo este uso es fuertemente desaconsejado.

· El carácter L especifica que la siguiente conversión e, E, f, g, o G corresponde a un argumento long double, o que la siguiente conversión d, i, o, u, x, o X corresponden a un argumento long long Dése cuenta que long long no está especificado en el ANSI C y, por consiguiente, no es trasladable a todas las arquitecturas.

· El carácter opcional q. Es equivalente a L. Vea las secciones ESTÁNDARES Y FALLOS para comentarios acerca del uso de ll, L, y q,

· Un carácter Z especificando que la siguiente conversión de enteros (d, i, o, u, x, o X), corresponde a un argumento size_t.

· Un carácter que especifica el tipo de conversión a ser aplicado.

Se puede indicar un ancho de campo o una precisión, o

ambos, mediante un asterisco `*' en lugar de una cadena de dígitos. En este caso, un argumento int suministra el ancho de campo o la precisión. Un ancho de campo negativo se trata como una bandera de justificado a la izquierda seguida por un ancho de campo positivo; una precisión negativa se trata como si no se hubiese indicado.

Los indicadores de conversión y sus significados son:

diouxX El argumento int (o la variante apropiada) es convertida a un decimal con signo (d y i), a octal sin signo (o, a decimal sin signo (u, a a notación hexadecimal sin signo (x y X). Las letras abcdef son usadas para conversiones x; las letras ABCDEF son usadas para conversiones X. La precisión, si se ha indicado alguna, da el mínimo número de dígitos que deben aparecer; si el valor convertido requiere menos dígitos, éste es rellenado a la izquierda con ceros.

eE El argumento double es redondeado y convertido al formato [-]d.dddedd donde hay un dígito delante del carácter del punto decimal y el número de dígitos después de éste es igual a la precisión; si no se indica precisión, ésta es tomada como 6; si la precisión es cero, no aparece el carácter de punto decimal. Una conversión E usa la letra E (en vez de e) para introducir el exponente. El exponente siempre contiene al menos dos dígitos; si el valor es cero, el exponente es 00.

f El argumento double es redondeado y convertido a una notación decimal del estilo [-]ddd.ddd, donde el número de dígitos después del carácter del punto decimal es igual a la especificación de la precisión. Si no se indica precisión, ésta es tomada como 6; si la precisión es explícitamente cero, no aparece el carácter del punto decimal. Si aparece un punto decimal, al menos aparece un dígito delante de él.

g El argumento double es convertido al estilo de f o e (o E para conversiones G). La precisión especifica el número de dígitos significativos. Si no se indica precisión, se dan 6 dígitos; si la precisión es cero, ésta es tratada como 1. Se utiliza el formato de e si el exponente de su conversión es menor que -4 o más grande o igual a la precisión. Los ceros finales se eliminan de la parte fraccional del resultado; un punto decimal sólo aparece si es seguido de al menos un dígito.

c El argumento int es convertido a un unsigned char, y se escribe el carácter resultante.

s Se espera que el argumento ``char *'' sea un puntero a un arreglo (array) de tipo carácter (puntero a una cadena de caracteres). Se escriben caracteres del array hasta (pero no incluyendo) un carácter terminador NUL ; si se especifica una precisión, no se escriben más caracteres del número especificado. Si se da una precisión, no es necesario que aparezca ningún carácter nulo; si no especificada precisión, o es mayor que el tamaño de la cadena, la cadena debe contener un carácter de terminación NUL.

p El argumento de tipo puntero ``void *'' se imprime

en hexadecimal (como si se hubiera indicado %#x o %#lx).

n El número de caracteres escritos hasta ahora se guarda en el entero indicado por el argumento de tipo puntero ``int *'' (o una variante suya). No se convierte ningún argumento.

% Se escribe un `%'. No se convierte ningún argumento. La especificación completa de conversión es `%%'.

En ningún caso un ancho de campo pequeño o inexistente produce el truncamiento del campo; si el resultado de una conversión es más ancho que el campo, el campo se expande para contener el resultado convertido.

EJEMPLOS

Para imprimir una fecha y una hora de la forma `Sunday, July 3, 10:02', donde weekday y month son punteros a cadenas:

```
#include <stdio.h>
fprintf(stdout, "%s, %s %d, %.2d:%.2d\n",
     weekday, month, day, hour, min);
```

Para imprimir con cinco lugares decimales:

```
#include <math.h>
#include <stdio.h>
fprintf(stdout, "pi = %.5f\n", 4 * atan(1.0));
```

Para reservar una cadena de 128 bytes e imprimir dentro de ella:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
char *newfmt(const char *fmt, ...)
{
          char *p;
          va_list ap;
          if ((p = malloc(128)) == NULL)
               return (NULL);
          va_start(ap, fmt);
          (void) vsnprintf(p, 128, fmt, ap);
          va_end(ap);
          return (p);
}
```

VÉASE TAMBIÉN

printf(1), scanf(3)

ESTÁNDARES

Las funciones fprintf, printf, sprintf, vprintf, vfprintf, y vsprintf están conforme a ANSI C3.159-1989 (``ANSI C''). La bandera q es la notación de BSD 4.4 para long long, mientras que ll o el uso de L en las conversiones de enteros es la notación GNU.

La versión de Linux de estas funciones está basada en la biblioteca libio de GNU. Eche un vistazo a la documentación info de la libc (glibc-1.08) de GNU para una descripción más concisa.

FALLOS

Algunas conversiones de coma flotante bajo Linux producen pérdidas de memoria.

Todas las funciones cumplen completamente el estándar ANSI C3.159-1989, aunque adicionalmente proporcionan las banderas q, Z y ' así como el comportamiento adicional de las banderas L y l. Esto último se puede considerar como un fallo, ya que cambia el comportamiento de las banderas definidas en ANSI C3.159-1989.

El efecto de relleno del formato %p con ceros (bien por la bandera 0, bien por especificar una precisión), y el efecto benigno (es decir, ninguno) de la bandera # en las las conversiones %n y %p, así como las combinaciones sin sentido de las mismas, no son estándares; dichas combinaciones debe ser evitadas.

Algunas combinaciones de banderas definidas por ANSI C no tienen sentido (por ejemplo, %Ld). Aunque pueden tener un comportamiento bien definido en Linux, esto no tiene por qué ser así en otras arquitecturas. Por tanto, normalmente es mejor no usar en absoluto banderas que no estén definidas en ANSI C, o sea, usar q en lugar de L en combinación con las conversiones diouxX o ll.

El uso de q no es el mismo que en BSD 4.4, ya que puede ser utilizado en conversiones de coma flotante de forma equivalente a L.

Ya que sprintf y vsprintf asumen una cadena de longitud infinita, los invocadores deben tener cuidado de no sobrepasar el espacio actual; a menudo, esto es imposible de garantizar.

SCANF

SCANF(3) Manual del Programador de Linux SCANF(3)

NOMBRE

scanf, fscanf, sscanf, vscanf, vsscanf, vfscanf - conversión de la entrada con formato

SINOPSIS

```
#include <stdio.h>
int scanf( const char *formato, ...);
int fscanf( FILE *flujo, const char *formato, ...);
int sscanf( const char *str, const char *formato, ...);

#include <stdarg.h>
int vscanf( const char *formato, va_list ap);
int vsscanf( const char *str, const char *formato, va_list ap);
int vfscanf( FILE *flujo, const char *formato, va_list ap);
```

DESCRIPCIÓN

La familia scanf de funciones escudriña la entrada según un formato como se describe más adelante. Este formato puede contener especificadores de conversión; los resultados de tales conversiones, si las hay, se guardan donde apunten los argumentos punteros. La función scanf lee la entrada del flujo de entrada estándar stdin, fscanf lee su entrada del puntero a FILE flujo, y sscanf lee su entrada de la cadena de caracteres a la que apunte str.

La función vfscanf es análoga a vfprintf(3) y lee la entrada del puntero a FILE flujo utilizando una lista variable de argumentos de punteros (vea stdarg(3)). La función vscanf rastrea una lista variable de argumentos de

la entrada estándar y la función vsscanf la analiza de una cadena de caracteres; estas funciones son análogas a vprintf y vsprintf respectivamente.

Cada argumento puntero sucesivo debe corresponder correctamente con cada especificador de conversión sucesivo (pero vea más adelante lo referente a `supresión'). Todas las conversiones empiezan con el carácter % (signo de porcentaje). La cadena de caracteres formato puede también contener otros caracteres. El espacio en blanco (como espacios, tabuladores, o saltos de línea) en la cadena de formato concuerda con cualquier cantidad de espacio en blanco, incluyendo ninguna, en la entrada. Cualquier otra cosa concuerda solamente consigo misma. El análisis acaba cuando un carácter de la entrada no concuerda con un carácter del formato. También cuando una conversión no puede realizarse (vea más adelante).

CONVERSIONES

Después del carácter % que marca el comienzo de una conversión puede haber una serie de caracteres de opción, como sigue:

* Suprime la asignación. La conversión que sigue ocurre como si nada, pero no se usa ningún puntero; el resultado de la conversión simplemente se descarta.

a Indica que la conversión será s, el espacio de memoria necesario para la cadena se obtendrá mediante malloc() y el puntero a ella se asignará a la variable puntero char, que no tiene que haber sido inicializada anteriormente. Esta opción no existe en C ANSI.

h Indica que la conversión será una de dioux o n y que el puntero siguiente lo es a un short int (en vez de a un int).

l Indica bien que la conversión será una de dioux o n y el puntero siguiente lo es a un long int (en vez de a un int), o bien que la conversión será una de efg y el puntero siguiente lo es a un double (en vez de a un float). Especificar dos opciones l equivale a la opción L.

L Indica que la conversión será o bien efg y el siguiente puntero lo es a un long double o bien que la conversión será dioux y el siguiente puntero lo será a un long long. (Observe que long long no es un tipo de C ANSI. Un programa que utilice esto no será transportable a todas las arquitecturas).

q equivalente a L. Esta opción no existe en C ANSI.

Además de estas opciones, puede haber una anchura máxima de campo opcional, expresado como un entero en base diez, entre el signo % y la conversión. Si no se da la anchura, se supone `infinita' (con una excepción, vea más abajo); si se da, como mucho se miran los caracteres especificados en ella cuando haya que procesar la conversión. Antes de que ésta comience, la mayoría descartan el espacio en blanco; este espacio no cuenta para la anchura de campo.

Están disponibles las siguientes conversiones:

% Concuerda con un '%' literal. Esto es, `%%' en el formato concuerda con un solo carácter '%' en la

entrada. No se hace ninguna conversión, y no hay ninguna asignación.

d Concuerda con un entero en base diez con signo opcional; el siguiente puntero debe serlo a int.

D Equivalente a ld; esto existe solamente por compatibilidad con una forma antigua. (Nota: esto ocurre sólo en libc4. En libc5 y glibc %D se ignora silenciosamente, provocando el fallo mistorioso de programas antiguos.)

i Concuerda con un entero con signo opcional; el siguiente puntero debe serlo a int. El entero se lee en base 16 si empieza por `0x' ó `0X'; en base 8 si empieza por `0', y en base 10 si empieza por otro dígito. Sólo se usan caracteres que correspondan a la base.

o Concuerda con un entero octal sin signo; el siguiente puntero debe serlo a unsigned int.

u Concuerda con un entero en base diez sin signo; el siguiente puntero debe serlo a unsigned int.

x Concuerda con un entero hexadecimal sin signo; el siguiente puntero debe serlo a unsigned int.

X Equivalente a x

f Concuerda con un número en coma flotante con signo opcional; el siguiente puntero debe serlo a float.

e Equivalente a f.

g Equivalente a f.

E Equivalente a f

s Concuerda con una secuencia de caracteres distintos de blancos; el siguiente puntero debe serlo a char, y el vector debe ser lo suficientemente grande como para aceptar toda la secuencia y el carácter 0 ó NUL final. El análisis de la cadena de entrada acaba en el siguiente espacio blanco o cuando se llega a la anchura de campo máxima, lo que ocurra antes.

c Concuerda con una secuencia de anchura caracteres (1 por omisión); el siguiente puntero debe serlo a char, y debe haber suficiente espacio para todos los caracteres (no se añade el carácter NUL terminador). Se suprime el descarte de los blancos iniciales. Para saltar un espacio en blanco inicial, emplee un espacio explícito en el formato.

[Concuerda con una secuencia no vacía de caracteres del conjunto especificado de caracteres aceptados; el siguiente puntero debe serlo a char, y debe haber bastante sitio para todos los caracteres en la cadena, más un NUL terminal. Se suprime el descarte usual de los espacios en blanco iniciales. La cadena se forma con caracteres de (o no de) un conjunto particular; el conjunto se define por los caracteres entre el corchete abierto [y un carácter de corchete de cierre]. El conjunto excluye esos caracteres si el primero después del corchete abierto es un acento circunflejo ^. Para incluir un corchete de cierre en el conjunto,

póngalo el primero tras el corchete abierto o el circunflejo; en cualquiera otra posición indicará que termina el conjunto. El carácter guión - es también especial; cuando se pone entre dos otros caracteres, añade todos los de enmedio al conjunto. Para incluir un guión, póngalo como el último carácter antes del corchete de cierre final. Por ejemplo, `[^]0-9-]' significa el conjunto de `todos los caracteres excepto el corchete de cierre, los dígitos del cero al nueve, y el guión'. La cadena acaba con la aparición de un carácter que no es (o, con un circunflejo, que sí es) del conjunto, o cuando se llega a la anchura opcional especificada.

p Concuerda con un valor de tipo puntero (como se imprima por `%p' en printf(3)); el siguiente puntero debe serlo a void.

n No se espera concordar con nada; en su lugar, se guarda el número de caracteres consumidos o leídos hasta ahora de la entrada en donde apunte el siguiente puntero, que debe serlo a int. Esto no es una conversión, aunque pueda suprimirse con la opción *. El estándar de C dice: `La ejecución de una directriz %n no incrementa el número de asignaciones devuelto al final de la ejecución', pero el Añadido de Correcciones parece contradecir esto. Probablemente es lo mejor no hacer ninguna suposición sobre el efecto de las conversiones %n en el valor de retorno de la función.

VALOR DEVUELTO

Estas funciones devuelven el número de elementos de la entrada asignados, que pueden ser menores que los formatos suministrados para conversión, o incluso cero, en el caso de un fallo de concordancia. Cero indica que, mientras había caracteres disponibles en la entrada, no ocurrió ninguna asignación; normalmente esto es debido a un carácter de entrada inválido, como un carácter alfabético para una conversión `%d'. Se devuelve el valor EOF si ha habido un fallo de entrada antes de ninguna conversión, como cuando se llega al final de la entrada. Si ocurre un error de lectura o se llega al final de la entrada después de que se haya hecho alguna conversión al menos, se devuelve el número de conversiones completadas hasta ese punto con éxito.

VÉASE TAMBIÉN

strtol(3), strtoul(3), strtod(3), getc(3), printf(3)

CONFORME A

Las funciones fscanf, scanf, y sscanf son conformes al estándar ANSI C3.159-1989 (``C ANSI'').

La opción q es la notación de BSD 4.4 para el tipo long long, mientras que ll o el empleo de L en conversiones de enteros, es la notación de GNU.

La versión de Linux de estas funciones se basa en la biblioteca libio de GNU. Eche un vistazo a la documentación en formato info de GNU libc (glibc-1.08) para una descripción más concisa.

FALLOS

Todas las funciones son conformes completamente con el estándar ANSI C3.159-1989, pero proporcionan las opciones

```
adicionales q y a así como un comportamiento adicional  de
las  opciones L y l.  Lo último puede ser considerado como
un fallo, puesto  que  cambia  el  comportamiento  de  las
opciones definidas en el estándar ANSI C3.159-1989.

Algunas  combinaciones de opciones definidas por C ANSI no
tienen sentido en C  ANSI  (p.ej.,  %Ld).   Aunque  pueden
tener  un  comportamiento  bien definido en Linux, esto no
tiene por qué ser así en otras arquitecturas. Por lo tanto
es  normalmente  mejor  usar opciones que no son definidas
por C ANSI en absoluto, i.e., usar q en vez de L en combi-
nación con conversiones diouxX o ll.

El  empleo  de q no es el mismo que en BSD 4.4, puesto que
puede utilizarse en conversiones de coma flotante de forma
equivalente a L.
```

Manipulacion de Memoria

MEMMOVE

```
MEMMOVE(3)       Manual del Programador de Linux       MEMMOVE(3)


NOMBRE
       memmove - copia un área de memoria.

SINOPSIS
       #include <string.h>

       void *memmove(void *dest, const void *src, size_t n);

DESCRIPCIÓN
       La función memmove() copia n bytes del área de memoria src
       al área de memoria dest.   Las  áreas  de  memoria  pueden
       solaparse.

VALOR DEVUELTO
       La función memmove() devuelve un puntero a dest.

CONFORME A
       SVID 3, BSD 4.3, ISO 9899

VÉASE TAMBIÉN
       bcopy(3), memccpy(3), memcpy(3), strcpy(3), strncpy(3)
```

MEMCPY

```
MEMCPY(3)        Manual del Programador de Linux        MEMCPY(3)


NOMBRE
       memcpy - copiar area de memoria

SYNOPSIS
       #include <string.h>

       void *memcpy(void *dest, const void *src, size_t n);
```

```
DESCRIPCIÓN
       La funcion memcpy() copia n bytes desde el area de memoria
       src al area dest.  Dichas areas de memoria no deben  tener
       ningun punto de interseccion; en tal caso utilizar la fun-
       cion  memmove(3) en lugar de memcpy().

VALOR DEVUELTO
       La funcion memcpy() devuelve un puntero a dest.

CONFORME A
       SVID 3, BSD 4.3, ISO 9899

VEA TAMBIÉN
       bcopy(3), memccpy(3), memmove(3), strcpy(3), strncpy(3)
```

MEMCCPY

```
MEMCCPY(3)       Manual del Programador de Linux       MEMCCPY(3)


NOMBRE
       memccpy - copia un área de memoria

SINOPSIS
       #include <string.h>

       void *memccpy(void *dest, const void *orig, int c, size_t n);

DESCRIPCIÓN
       La  función  memccpy()  copia  como mucho n bytes desde el
       área de memoria orig al  área  de  memoria  dest,  parando
       cuando se encuentra el carácter c.

VALOR DEVUELTO
       La  función  memccpy()  devuelve  un  puntero al siguiente
       carácter de dest tras c, o NULL si c no estaba  en  los  n
       primeros caracteres de orig.

CONFORME A
       SVID 3, BSD 4.3

VÉASE TAMBIÉN
       bcopy(3), memcpy(3), memmove(3), strcpy(3), strncpy(3)
```

MEMSET – Rellena con bytes repetidos

```
MEMSET(3)        Manual del Programador de Linux        MEMSET(3)


NOMBRE
       memset - rellena una zona de memoria con bytes repetidos

SINOPSIS
       #include <string.h>

       void *memset(void *s, int c, size_t n);

DESCRIPCIÓN
       La  función memset() rellena los primeros n bytes del área
       de memoria apuntada por s con el byte constante c.
```

```
VALOR DEVUELTO
       La función memset() devuelve un puntero al área de memoria
       s.

CONFORME A
       SVID 3, BSD 4.3, ISO 9899

VÉASE TAMBIÉN
       bzero(3), swab(3)
```

Varios

SLEEP

```
SLEEP(3)         Manual del Programador de Linux         SLEEP(3)


NOMBRE
       sleep - Duerme durante el número de segundos especificado

SINOPSIS
       #include <unistd.h>

       unsigned int sleep(unsigned int segundos);

DESCRIPCIÓN
       sleep()  hace  que el proceso en curso se duerma hasta que
       hayan transcurrido segundos segundos o  hasta  que  llegue
       una señal que sea tenida en cuenta.

VALOR DEVUELTO
       Cero  si el tiempo pedido ha pasado, o el número de segun-
       dos que quedan de sueño.

CONFORME A
       POSIX.1

FALLOS
       sleep() puede estar implementada con SIGALRM; mezclar lla-
       madas a alarm() y a sleep() es una mala idea.

       Utilizar  longjmp()  desde un manejador de señales o modi-
       ficar el manejo de SIGALRM  mientras  se  está  durmiento,
       producirá resultados no definidos.

VÉASE TAMBIÉN
       signal(2), alarm(2)
```

Procesos - Comandos de ejecucion

FORK

```
FORK(2)          Manual del Programador de Linux          FORK(2)


NOMBRE
       fork, vfork - crean un proceso hijo
```

```
SINOPSIS
       #include <unistd.h>

       pid_t fork(void);
       pid_t vfork(void);

DESCRIPCIÓN
       fork  crea un proceso hijo que difiere de su proceso padre
       sólo en su PID y PPID, y en el hecho  de  que  el  uso  de
       recursos esté asignado a 0.  Los candados de fichero (file
       locks) y las señales pendientes no se heredan.

       En linux, fork está implementado usando páginas de  copia-
       en-escritura  (copy-on-write),  así  que  la  única penal-
       ización en que incurre fork es  en  el  tiempo  y  memoria
       requeridos  para  duplicar  las  tablas de las páginas del
       padre, y para crear una única estructura  de  tarea  (task
       structure) para el hijo.

VALOR DEVUELTO
       En  caso  de éxito, se devuelve el PID del proceso hijo en
       el hilo de ejecución de su padre, y se devuelve un  0   en
       el  hilo  de  ejecución  del  hijo.  En  caso de fallo, se
       devolverá un -1 en el contexto del  padre,  no  se  creará
       ningún proceso hijo, y se pondrá en errno un valor apropi-
       ado.

ERRORES
       EAGAIN fork no puede  reservar  sufficiente  memoria  para
              copiar las tablas de páginas del padre y alojar una
              estructura de tarea para el hijo.

       ENOMEM fork no pudo obtener las necesarias estructuras del
              núcleo porque la cantidad de memoria era escasa.

FALLOS
       En Linux, vfork es simplemente un alias para fork.

CONFORME A
       La  llamada  al  sistema  fork es conforme con SVr4, SVID,
       POSIX, X/OPEN y BSD 4.3.

VÉASE TAMBIÉN
       clone(2), execve(2), wait(2)
```

EXEC

```
EXEC(3)          Manual del Programador de Linux          EXEC(3)


NOMBRE
       execl, execlp, execle, execv, execvp - ejecutan un fichero

SINOPSIS
       #include <unistd.h>

       extern char **environ;

       int execl( const char *camino, const char *arg, ...);
       int execlp( const char *fichero, const char *arg, ...);
       int execle( const char *camino, const  char  *arg  ,  ...,
       char * const envp[]);
       int execv( const char *camino, char *const argv[]);
```

```
int execvp( const char *fichero, char *const argv[]);
```

DESCRIPCIÓN

La familia de funciones exec reemplaza la imagen del proceso en curso con una nueva. Las funciones descritas en esta página del Manual son interfaces para la primitiva **execve**(2). (Consulte la página del Manual de execve para información detallada acerca del reemplazo del proceso en curso.)

El primer argumento de estas funciones es el camino de un fichero que va a ser ejecutado.

El const char *arg y puntos suspensivos siguientes en las funciones execl, execlp, y execle pueden ser contemplados como arg0, arg1, ..., argn. Todos juntos, describen una lista de uno o más punteros a cadenas de caracteres terminadas en cero, que representan la lista de argumentos disponible para el programa ejecutado. El primer argumento, por convenio, debe apuntar al nombre de fichero asociado con el fichero que se esté ejecutando. La lista de argumentos debe ser terminada por un puntero NULL.

Las funciones execv y execvp proporcionan un vector de punteros a cadenas de caracteres terminadas en cero, que representan la lista de argumentos disponible para el nuevo programa. El primer argumento, por convenio, debe apuntar al nombre de fichero asociado con el fichero que se esté ejecutando. El vector de punteros debe ser terminado por un puntero NULL.

La función execle también especifica el entorno del proceso ejecutado mediante un parámetro adicional que va detrás del puntero NULL que termina la lista de argumentos de la lista de parámetros o el puntero al vector argv. Este parámetro adicional es un vector de punteros a cadenas de caracteres acabadas en cero y debe ser terminada por un puntero NULL. Las otras funciones obtienen el entorno para la nueva imagen de proceso de la variable externa environ en el proceso en curso.

Algunas de estas funciones tienen una semántica especial.

Las funciones execlp y execvp duplicarán las acciones del shell al buscar un fichero ejecutable si el nombre de fichero especificado no contiene un carácter de barra inclinada (/). El camino de búsqueda es el especificado en el entorno por la variable PATH. Si esta variable no es especificada, se emplea el camino predeterminado ``:/bin:/usr/bin''. Además, ciertos errores se tratan de forma especial.

Si a un fichero se le deniega el permiso (la función intentada execve devuelve EACCES), estas funciones continuarán buscando en el resto del camino de búsqueda. Si no se encuentra otro fichero, empero, retornarán dando el valor EACCES a la variable global errno .

Si no se reconoce la cabecera de un fichero (la función intentada execve devuelve ENOEXEC), estas funciones ejecutarán el shell con el camino del fichero como su primer argumento. (Si este intento falla, no se busca más.)

VALOR DEVUELTO

Si cualquiera de las funciones exec regresa, es que ha ocurrido un error. El valor de retorno es -1, y en la variable global errno se pondrá el código de error adecuado.

FICHEROS
/bin/sh

ERRORES
Todas estas funciones pueden fallar y dar un valor a errno para cualquiera de los errores especificados para la función execve(2).

VÉASE TAMBIÉN
sh(1), execve(2), fork(2), environ(5), ptrace(2)

COMPATIBILIDAD
En algunos otros sistemas, el PATH predeterminado tiene el directorio de trabajo listado detrás de /bin y /usr/bin, como una medida anti-caballo de Troya. En libc 5.4.7, Linux aún utiliza el valor de PATH predeterminado con el tradicional "el directorio de trabajo, el primero".

El comportamiento de execlp y execvp cuando hay errores al intentar ejecutar el fichero es una práctica de antiguo, pero tradicionalmente no ha sido documentada y el estándar POSIX no lo especifica. BSD (y quizás otros sistemas) hacen un sleep() automático y un reintento a continuación, si se encuentra el error ETXTBSY. Linux trata esto como un error importante y el regreso de la función es inmediato. Tradicionalmente, las funciones execlp y execvp hacían caso omiso de todos los errores salvo los descritos arriba y ENOMEM y E2BIG, que provocaban el regreso. Ahora también regresan si tiene lugar algún error distinto de los descritos anteriormente.

CONFORME A
Execl, execv, execle, execlp y execvp son conformes con IEEE Std1003.1-88 (``POSIX.1'').

EXECVE Primitiva Exec

EXECVE(2) Manual del Programador de Linux EXECVE(2)

NOMBRE
execve - ejecuta un programa

SINOPSIS
#include <unistd.h>

int execve (const char *filename, const char *argv [], const char *envp[]);

DESCRIPCIÓN
execve() ejecuta el programa indicado por filename. filename debe ser bien un binario ejecutable, bien un guión shell (shell script) comenzando con una línea de la forma "#! intérprete [arg]". En el segundo caso, el intérprete debe ser un nombre de camino válido para un ejecutable que no sea él mismo un guión y que será ejecutado como intérprete [arg] filename.

execve() no regresa en caso de éxito, y el código, datos, bss y la pila del proceso invocador se reescriben con los correspondientes del programa cargado. El programa invocado hereda el PID del proceso invocador y cualquier

descriptor de fichero abierto que no se halla configurado para "cerrar en ejecución" (close on exec). Las señales pendientes del proceso padre se limpian. Cualquier señal capturada por el proceso invocador es devuelta a su comportamiento por defecto.

Si el programa actual está bajo inspección de ptrace, se le enviará una señal SIGTRAP trás la ejecución exitosa de execve().

Si el ejecutable es un ejecutable binario a.out enlazado dinámicamente que contiene "stubs" de bibliotecas compartidas, se llama al enlazador dinámico de Linux, ld.so(8), al comienzo de la ejecución para traer al núcleo las bibliotecas compartidas necesarias y enlazar el ejecutable con ellas.

Si el ejecutable es un ejecutable ELF enlazado dinámicamente, se usa el intérprete especificado en el segmento PT_INTERP para cargar las bibliotecas compartidas necesarias. Este intérprete es usualmente /lib/ld-linux.so.1 para los binarios enlazados con la versión 5 de la libc de Linux o /lib/ld-linux.so.2 para los binarios enlazados con la versión 2 de la libc de GNU.

VALOR DEVUELTO

En caso de éxito execve() no regresa mientras que en caso de error el valor devuelto es -1, y a la variable errno se le asigna un valor apropiado.

ERRORES

EACCES El fichero o el intérprete de guiones no es un fichero regular.

EACCES Se ha denegado el permiso de ejecución para el fichero o el intérprete de guiones.

EACCES El sistema de ficheros está montado con la opción noexec.

EPERM El sistema de ficheros está montado con la opción nosuid, el usuario no es el superusuario y el fichero tiene activo el bit SUID o SGID.

EPERM El proceso está siendo ejecutado paso a paso, el usuario no es el superusuario y el fichero tiene activo el bit SUID o SGID.

E2BIG La lista de argumentos es demasiado grande.

ENOEXEC El ejecutable no se encuentra en un formato reconocible, es para una arquitectura incorrecta o tiene algún otro error de formato que impide su ejecución.

EFAULT filename apunta fuera de su espacio de direcciones accesible.

ENAMETOOLONG
filename es demasiado largo.

ENOENT El fichero filename no existe o no existe un intérprete de guiones o no existe un intérprete ELF.

ENOMEM No hay suficiente memoria disponible en el núcleo.

```
        ENOTDIR Un componente del camino filename o del camino del
                intérprete de guiones o del intérprete ELF  no  es
                un directorio.

        EACCES  Se  ha  denegado  el permiso de búsqueda en uno de
                los componentes del camino filename o  del  camino
                del intérprete de guiones.

        ELOOP   Se  han econtrado demasiados enlaces simbólicos al
                resolver filename, el  nombre  del  intérprete  de
                guiones o el nombre del intérprete ELF.

        ETXTBUSY
                Uno  o más procesos han abierto el ejecutable para
                escritura.

        EIO     Se ha producido un error de E/S.

        ENFILE  Se ha alcanzado el  límite  del  número  total  de
                ficheros abiertos en el sistema.

        EMFILE  El  proceso  ya tiene abiertos el número máximo de
                fichero.

        EINVAL  El  ejecutable  ELF  tiene  más  de  un   segmento
                PT_INTERP  (es decir, ha intentado especificar más
                de un intérprete).

        EISDIR  El intérprete ELF es un directorio.

        ELIBBAD El intérprete ELF no está en un formato  reconoci-
                ble.

CONFORME A
       SVr4,  SVID, X/OPEN y BSD 4.3.  POSIX no documenta el sig-
       nificado de #! pero, en  cualquier  caso,  es  compatible.
       SVr4   documenta  las  condiciones  de  error  adicionales
       EAGAIN, EINTR, ELIBACC, ENOLINK y EMULTIHOP; POSIX no doc-
       umenta  las  condiciones  de error ETXTBSY, EPERM, EFAULT,
       ELOOP, EIO, ENFILE, EMFILE, EINVAL, EISDIR ni ELIBBAD.


NOTAS
       Procesos SUID y SGID  no  pueden  ser  inspeccionados  con
       ptrace().

       La  longitud  máxima de línea en un ejecutable del tipo #!
       es de 127 caracteres en la primera línea del fichero.

       Linux ignora los bits SUID y SGID en los guiones shell.

VÉASE TAMBIÉN
       ld.so(8), execl(3), fork(2)
```

FIFOS

FIFO Introduccion

```
FIFO(4)             Linux Programmer's Manual             FIFO(4)


NAME
       fifo - first-in first-out special file, named pipe
```

DESCRIPTION

A FIFO special file (a named pipe) is similar to a pipe, except that it is accessed as part of the file system. It can be opened by multiple processes for reading or writing. When processes are exchanging data via the FIFO, the kernel passes all data internally without writing it to the file system. Thus, the FIFO special file has no contents on the file system, the file system entry merely serves as a reference point so that processes can access the pipe using a name in the file system.

The kernel maintains exactly one pipe object for each FIFO special file that is opened by at least one process. The FIFO must be opened on both ends (reading and writing) before data can be passed. Normally, opening the FIFO blocks until the other end is opened also.

A process can open a FIFO in non-blocking mode. In this case, opening for read only will succeed even if noone has opened on the write side yet; opening for write only will fail with ENXIO (no such device or address) unless the other end has already been opened.

Under Linux, opening a FIFO for read and write will succeed both in blocking and non-blocking mode. POSIX leaves this behaviour undefined. This can be used to open a FIFO for writing while there are no readers available. A process that uses both ends of the connection in order to communicate with itself should be very careful to avoid deadlocks.

NOTES

When a process tries to write to a FIFO that is not opened for read on the other side, the process is sent a SIGPIPE signal.

FIFO special files can be created by mkfifo(3), and are specially indicated in ls -l.

SEE ALSO

mkfifo(3), mkfifo(1), pipe(2), socketpair(2), open(2), signal(2), sigaction(2)

MKFIFO

MKFIFO(3) Manual del Programador de Linux MKFIFO(3)

NOMBRE

mkfifo - construye un fichero especial FIFO (una tubería con nombre)

SINOPSIS

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

int mkfifo ( const char *camino, modo_t modo );
```

DESCRIPCIÓN

mkfifo construye un fichero especial FIFO con el nombre camino. modo especifica los permisos del FIFO. Son modificados por la máscara umask del proceso de la forma

habitual: los permisos del fichero recién creado son (modo & ~umask).

Un fichero especial FIFO es similar a una interconexión o tubería, excepto en que se crea de una forma distinta. En vez de ser un canal de comunicaciones anónimo, un fichero especial FIFO se mete en el sistema de ficheros mediante una llamada a mkfifo.

Una vez que Ud. ha creado un fichero especial FIFO de esta forma, cualquier proceso puede abrirlo para lectura o escritura, de la misma manera que con un fichero normal. Sin embargo, tiene que ser abierto en los dos extremos simultáneamente antes de que se pueda proceder a cualquier operación de entrada o salida. Abrir un FIFO para lectura normalmente produce un bloqueo hasta que algún otro proceso abre el mismo FIFO para escritura, y viceversa.

VALOR DEVUELTO

El valor de retorno normal, si todo va bien, de mkfifo, es 0. En caso de error, se devuelve -1 (en este caso, errno toma un valor apropiado).

ERRORES

EACCES Uno de los directorios en camino no tiene permiso de paso (ejecución).

EEXIST camino ya existe.

ENAMETOOLONG
O la longitud total de camino es mayor que PATH_MAX, o un componente nombre de fichero individual tiene una longitud superior a NAME_MAX. En el sistema GNU, no hay un límite impuesto a la longitud total del nombre de un fichero, pero algunos sistemas de ficheros pueden poner límites en la longitud de un componente.

ENOENT Un componente directorio en camino no existe o es un enlace simbólico colgante.

ENOSPC El directorio o sistema de ficheros no tiene sitio para el nuevo fichero.

ENOTDIR
Un componente usado como directorio en camino no es, de hecho, un directorio.

EROFS camino se refiere a un sistema de ficheros de lectura exclusiva.

CONFORME A

POSIX.1

VÉASE TAMBIÉN

mkfifo(1), read(2), write(2), open(2), close(2), stat(2), umask(2).

UNLINK

UNLINK(2) Manual del programador de Linux UNLINK(2)

NOMBRE

unlink - borra un nombre y posiblemente el fichero al que hace referencia

SINOPSIS

```
#include <unistd.h>

int unlink(const char *pathname);
```

DESCRIPCIÓN

unlink borra un nombre del sistema de ficheros. Si dicho nombre era el último enlace a un fichero, y ningún proceso tiene el fichero abierto, el fichero es borrado y el espacio que ocupaba vuelve a estar disponible.

Si el nombre era el último enlace a un fichero, pero algún proceso sigue teniendo el fichero abierto, el fichero seguirá existiendo hasta que el último descriptor de fichero referente a él sea cerrado.

Si el nombre hacía referencia a un enlace simbólico, el enlace es eliminado.

Si el nombre hacía referencia a un socket, fifo o dispositivo, el nombre es eliminado, pero los procesos que tengan el objeto abierto pueden continuar usándolo.

VALOR DEVUELTO

En caso de éxito, se devuelve cero. En caso de error, se devuelve -1 y se establece el errno apropiado.

ERRORES

EFAULT pathname apunta fuera del espacio de direcciones accesible.

EACCES No se otorga permiso de escritura para el directorio contenido en pathname al identificador de usuario efectivo del proceso, o uno de los directorios de pathname no permite búsquedas (no tiene permiso de ejecución).

EPERM El directorio contenido en pathname tiene puesto el sticky-bit (S_ISVTX), y el identificador de usuario efectivo del proceso no es el identificador de usuario del fichero a borrar ni el del directorio que lo contiene, o pathname es un directorio.

ENAMETOOLONG
pathname es demasiado largo.

ENOENT Un elemento usado como directorio en pathname no existe o es un enlace simbólico colgado.

ENOTDIR Un elemento usado como directorio en pathname no es en realidad un directorio.

EISDIR pathname hace referencia a un directorio.

ENOMEM No hay suficiente memoria disponible en el núcleo.

EROFS pathname hace referencia a un fichero de un sistema de ficheros de sólo lectura.

ELOOP Se encontraron demasiados enlaces simbólicos al traducir pathname.

EIO Ocurrió un error de E/S.

```
CONFORME A
       SVr4,  SVID,  POSIX,  X/OPEN,  4.3BSD.  SVr4 documenta las
       condiciones de error adicionales EBUSY, EINTR,  EMULTIHOP,
       ETXTBUSY, ENOLINK.

FALLOS
       Algunos  hechos  desafortunados en el protocolo NFS pueden
       causar la desaparición inesperada de ficheros  que  siguen
       en uso.

VÉASE TAMBIÉN
       link(2),    rename(2),    open(2),   rmdir(2),   mknod(2),
       mkfifo(3), remove(3), rm(1)
```

PIPE

```
PIPE(2)          Manual del Programador de Linux          PIPE(2)


NOMBRE
       pipe - crea una tubería o interconexión

SINOPSIS
       #include <unistd.h>

       int pipe(int descf[2]);

DESCRIPCIÓN
       pipe  crea un par de descriptores de ficheros, que apuntan
       a un nodo-í de una tubería, y los pone en el vector de dos
       elementos  apuntado  por descf.  descf[0] es para lectura,
       descf[1] es para escritura.

VALOR DEVUELTO
       En caso de éxito, se devuelve cero. En caso  de  error  se
       devuelve -1 y se pone un valor apropiado en errno.

ERRORES
       EMFILE  El proceso tiene en uso demasiados descriptores de
               ficheros.

       ENFILE  La tabla de ficheros del sistema está llena.

       EFAULT  descf no es válido.

CONFORME A
       SVr4, SVID, AT&T, POSIX, X/OPEN, BSD 4.3

VÉASE TAMBIÉN
       read(2), write(2), fork(2), socketpair(2)
```

Sockets - TCP UDP

SOCKET

```
SOCKET(2)        Manual del Programador de Linux        SOCKET(2)


NOMBRE
```

socket - crea un extremo de una comunicación

SINOPSIS

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

int socket(int dominio, int tipo, int protocolo);
```

DESCRIPCIÓN

Socket crea un extremo de una comunicación y devuelve un descriptor.

El parámetro dominio especifica un dominio de comunicaciones dentro del cual tendrá lugar la comunicación; esto selecciona la familia de protocolo que deberá emplearse. Estas familias se definen en el fichero de cabecera sys/socket.h. Los formatos actualmente reconocidos son

AF_UNIX (protocolos internos de UNIX)

AF_INET (protocolos de ARPA Internet)

AF_ISO (protocolos ISO)

AF_NS (protocolos de Xerox Network Systems)

AF_IMPLINK
(capa de enlace IMP "anfitrión en IMP")

El zócalo tiene el tipo indicado, que especifica la semántica de la comunicación. Los tipos definidos en la actualidad son:

SOCK_STREAM
SOCK_DGRAM
SOCK_RAW
SOCK_SEQPACKET
SOCK_RDM

Un tipo SOCK_STREAM proporciona flujos de bytes basados en una conexión bidireccional secuenciada, confiable. Se puede admitir un mecanismo de transmisión de datos fuera-de-banda. Un zócalo SOCK_DGRAM admite datagramas (mensajes no confiables, sin conexión, de una longitud máxima (normalmente pequeña) fija). Un zócalo SOCK_SEQPACKET puede proporcionar un camino de transmisión de datos basado en conexión bidireccional secuenciado, confiable, para datagramas de longitud máxima fija; se puede requerir un consumidor para leer un paquete entero con cada llamada al sistema de lectura. Esta facilidad es específica del protocolo, y está actualmente implementada solamente para AF_NS. Zócalos SOCK_RAW proporcionan acceso a interfaces y protocolos de red internos. Los tipos SOCK_RAW, que sólo está disponible para el súper-usuario, y SOCK_RDM, que está planificado, pero no aún implementado, no se describen aquí.

El protocolo especifica un protocolo particular para ser usado con el zócalo. Normalmente sólo existe un protocolo que admita un tipo particular de zócalo dentro de una familia de protocolos dada. Sin embargo, es posible que puedan existir varios protocolos, en cuyo caso un protocolo particular puede especificarse de esta manera. El número de protocolo a emplear es particular al "dominio de comunicación" en el que la comunicación va a tener lugar; vea protocols(5).

Los zócalos del tipo SOCK_STREAM son flujos de bytes bidireccionales, similar a tuberías. Un zócalo de flujo debe estar en un estado conectado antes de que cualquier dato pueda ser enviado o recibido en él. Se crea una conexión con otro zócalo mediante la llamada connect(2). Una vez hecha la conexión, los datos pueden transferirse utilizando llamadas read(2) y write(2) o alguna variante de las llamadas send(2) y recv(2). Cuando una sesión se ha completado, se puede efectuar un close(2). Los datos fuera-de-banda pueden transmitirse también como se describe en send(2) y recibirse según se describe en recv(2).

Los protocolos de comunicaciones usados para implementar un SOCK_STREAM aseguran que los datos no se pierden ni se duplican. Si un trozo de dato para el cual el protocolo de la pareja tiene espacio de búfer no puede ser transmitido satisfactoriamente en un período razonable de tiempo, entonces la conexión se considera rota y las llamadas indicarán un error devolviendo -1 y con el código ETIMEDOUT en la variable global errno.

Los protocolos opcionalmente mantienen los zócalos calientes forzando transmisiones más o menos cada minuto en ausencia de otra actividad. Entonces se induce un error si no se descubre una respuesta en una conexión en otro caso inactiva en un período extendido de tiempo (p. ej. 5 minutos). Se lanza una señal SIGPIPE si un proceso envía en un flujo roto; esto provoca que procesos simples, que no manejan la señal, acaben.

Los zócalos SOCK_SEQPACKET emplean las mismas llamadas al sistema que los SOCK_STREAM. La única diferencia es que las llamadas a read(2) devolverán solamente la cantidad de datos pedidos, y los que queden en el paquete que llega se perderán.

Los zócalos SOCK_DGRAM y SOCK_RAW permiten el envío de datagramas a los correspondientes nombrados en llamadas a send(2). Los datagramas se reciben generalmente con recvfrom(2), que devuelve el siguiente datagrama con su dirección de retorno.

Una llamada a fcntl(2) puede utilizarse para especificar que un grupo de proceso reciba una señal SIGURG cuando lleguen los datos fuera-de-banda. También puede activar la E/S no bloqueante y la notificación asíncrona de los eventos a través de SIGIO.

La operación de los zócalos se controla por opciones en el nivel de los zócalos. Estas opciones se definen en el fichero de cabecera sys/socket.h. Setsockopt(2) y getsockopt(2) se emplean para establecer y obtener opciones, respectivamente.

VALOR DEVUELTO

Se devuelve un -1 si ocurre un error; en otro caso el valor devuelto es un descriptor para referenciar el zócalo.

ERRORES

EPROTONOSUPPORT
El tipo de protocolo, o el protocolo especificado, no es reconocido dentro de este dominio.

EMFILE La tabla de descriptores por proceso está llena.

ENFILE La tabla de ficheros del sistema está llena.

```
       EACCES  Se deniega el permiso para  crear  un  zócalo  del
               tipo o protocolo especificado.

       ENOBUFS No  hay  bastante  espacio  de búfer. El zócalo no
               puede crearse  hasta  que  no  queden  libres  los
               recursos suficientes.

CONFORME A
       4.4BSD  (la  llamada a función socket apareció en 4.2BSD).
       Generalmente transportable a o desde sistemas no  BSD  que
       admitan  clones  de  la capa de zócalos de BSD (incluyendo
       variantes System V).

VÉASE TAMBIÉN
       accept(2), bind(2), connect(2),  getprotoent(3),  getsock-
       name(2),   getsockopt(2),  ioctl(2),  listen(2),  read(2),
       recv(2), select(2), send(2),  shutdown(2),  socketpair(2),
       write(2)

       "An  Introductory 4.3 BSD Interprocess Communication Tuto-
       rial" está reimpreso en  UNIX  Programmer's  Supplementary
       Documents Volume 1
```

BIND

```
BIND(2)          Manual del Programador de Linux          BIND(2)


NOMBRE
       bind - enlaza un nombre a un zócalo (socket)

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

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

DESCRIPCIÓN
       bind da al zócalo  sockfd,  la  dirección  local  my_addr.
       my_addr tiene una longitud de addrlen bytes.  Tradicional-
       mente, esto se conoce como "asignar un nombre a un zócalo"
       (cuando  un  zócalo  se  crea  con socket(2), existe en un
       espacio de nombres (familia de direcciones) pero carece de
       nombre).

NOTAS
       Enlazando  un  nombre en el dominio UNIX crea un zócalo en
       el sistema de ficheros que debe ser borrado por  el  invo-
       cador cuando ya no es necesario (usando unlink(2)).

       Las  reglas  usadas en el enlace de nombre varian según el
       dominio de comunicacion.  Consultese las  entradas  en  la
       sección 4 para una información más detallada.

VALOR DEVUELTO
       Cero  es  el resultado en caso de éxito. En caso de error,
       -1 es el valor regresado y a errno se le asigna  un  valor
       apropiado.

ERRORES
       EBADF   sockfd no es un descriptor válido.
```

```
        EINVAL  El  zócalo  ya está enlazado a una dirección. Esto
                puede    cambiar    en    el     futuro:     véase
                linux/unix/sock.c para más detalles.

        EACCES  La dirección está protegida, y el usuario no es el
                superusuario.

        Los siguientes errores son específicos a los  zócalos  del
        dominio UNIX (AF_UNIX):

        EINVAL  La  dirección  addr_len es incorrecta, o el zócalo
                no pertenecia a la familia AF_UNIX.

        EROFS   El nodo-i (inode) del zócalo reside en un  sistema
                de ficheros de solo-lectura.

        EFAULT  my_addr  señala fuera de su espacio de direcciones
                accesible.

        ENAMETOOLONG
                my_addr es demasiado larga.

        ENOENT  El fichero no existe.

        ENOMEM  No hay suficiente memoria en el kernel disponible.

        ENOTDIR Un componente del camino no es un directorio.

        EACCES  El  permiso de búsqueda ha sido denegado en uno de
                los componentes del camino.

        ELOOP   Se han encontrado demasiados enlaces simbólicos al
                resolver my_addr.

CONFORME A
        SVr4,  4.4BSD (la función bind apareció por primera vez en
        BSD 4.2). SVr4 documenta condiciones  generales  de  error
        adicionales:  EADDRNOTAVAIL,  EADDRINUSE,  ENOSR, y condi-
        ciones de error específicas del dominio UNIX  adicionales:
        EIO, EISDIR, EROFS.

VÉASE TAMBIÉN
        accept(2),   connect(2),  listen(2),  socket(2),  getsock-
        name(2)
```

RECV, RECVFROM

```
RECV(2)          Manual del Programador de Linux          RECV(2)


NOMBRE
        recv,  recvfrom,  recvmsg  -  reciben  un mensaje desde un
        zócalo

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


        int recv(int s, void *buf, int lon, unsigned int flags);

        int recvfrom(int s, void *buf, int lon, unsigned int flags
        , struct sockaddr *desde, int *londesde);
```

int recvmsg(int s, struct msghdr *msg, unsigned int flags);

DESCRIPCIÓN

Las primitivas recvfrom y recvmsg se emplean para recibir mensajes desde un zócalo (``socket''), y pueden utilizarse para recibir datos de un zócalo sea orientado a conexión o no.

Si desde no es nulo, y el zócalo no es orientado a conexión, la dirección fuente del mensaje se llena. Londesde es un parámetro por referencia, inicializado al tamaño del búfer asociado con desde, y modificado cuando la función regresa para indicar el tamaño real de la dirección guardada ahí.

La llamada a recv se utiliza normalmente sólo en un zócalo conectado (vea connect(2)) y es idéntica a recvfrom con un parámetro desde nulo. Como es redundante, puede que en futuras distribuciones no esté mantenida.

Las tres rutinas devuelven la longitud del mensaje cuando terminan bien. Si un mensaje es demasiado largo como para caber en el búfer suministrado, los bytes que sobran pueden descartarse dependiendo del tipo de zócalo del que se reciba el mensaje (vea socket(2)).

Si no hay mensajes disponibles en el zócalo, la llamada de recepción espera que llegue un mensaje, a menos que el zócalo sea no bloqueante (vea fcntl(2)) en cuyo caso se devuelve el valor -1 y la variable externa errno toma el valor EWOULDBLOCK. Las llamadas de recepción devuelven normalmente cualquier dato disponible, hasta la cantidad pedida, en vez de esperar la recepción de la cantidad pedida completa; este comportamiento se ve afectado por las opciones del nivel de zócalos SO_RCVLOWAT y SO_RCVTIMEO descritas en getsockopt(2).

La llamada select(2) puede emplearse para determinar cuándo llegan más datos.

El argumento flags de una llamada a recv se forma aplicando el operador de bits OR a uno o más de los valores:

MSG_OOB procesar datos fuera-de-banda

MSG_PEEK
mirar el mensaje entrante

MSG_WAITALL
esperar a que se complete la petición u ocurra un error

La opción MSG_OOB pide la recepción de datos fuera-de-banda que no se recibirían en el flujo de datos normal. Algunos protocolos ponen datos despachados con prontitud en la cabeza de la cola de datos normales, y así, esta opción no puede emplearse con tales protocolos.

La opción MSG_PEEK hace que la operación de recepción devuelva datos del principio de la cola de recepción sin quitarlos de allí. Así, una próxima llamada de recepción devolverá los mismos datos. La opción MSG_WAITALL pide que la operación se bloquee hasta que se satisfaga la petición completamente. Sin embargo, la llamada puede aún

devolver menos datos de los pedidos si se captura una señal, si ocurre un error o una desconexión, o si los próximos datos que se van a recibir son de un tipo diferente del que se ha devuelto.

La llamada recvmsg utiliza una estructura msghdr para minimizar el número de parámetros suministrados directamente. Esta estructura tiene la forma siguiente, según se define en sys/socket.h:

```
struct msghdr {
     caddr_t   msg_name; /* dirección opcional */
     u_int     msg_namelen;   /* tamaño de dirección */
     struct    iovec *msg_iov;     /* vector
disperso/reunido */
     u_int     msg_iovlen;    /* Nº de elementos en
msg_iov */
     caddr_t   msg_control;   /* datos auxiliares, vea
abajo */
     u_int     msg_controllen; /* long. búfer datos
auxiliares */
     int  msg_flags;     /* opciones en mensaje recibido
*/
};
```

Aquí msg_name y msg_namelen especifican la dirección de destino si el zócalo está desconectado; msg_name puede darse como un puntero nulo si no se desean o requieren nombres. Msg_iov y msg_iovlen describen localizaciones recogidas dispersas, como se discute en readv(2). Msg_control, que tiene de longitud msg_controllen, apunta a un búfer para otros mensajes relacionados con control de protocolo o para otros datos auxiliares diversos. Los mensajes son de la forma:

```
struct cmsghdr {
     u_int     cmsg_len; /* Nº de byte de datos, incluye
cab. */
     int  cmsg_level;    /* protocolo originante */
     int  cmsg_type;     /* tipo específico del protocolo
*/
/* seguido por
     u_char    cmsg_data[]; */
};
```

Como ejemplo, uno podría usar esto para saber los cambios en el flujo de datos en XNS/SSP, o en ISO, para obtener datos de petición-de-conexión-de-usuario mediante la llamada a recvmsg sin proporcionar un búfer de datos, inmediatamente tras una llamada a accepct.

Los descriptores de ficheros abiertos se pasan ya como datos auxiliares para zócalos de dominio AF_UNIX, con cmsg_level puesto a SOL_SOCKET y cmsg_type puesto a SCM_RIGHTS.

El campo msg_flags toma un valor al regresar dependiendo del mensaje recibido. MSG_EOR indica fin-de-registro; los datos devueltos completaron un registro (generalmente empleado con zócalos del tipo SOCK_SEQPACKET). MSG_TRUNC indica que la porción trasera de un datagrama ha sido descartada porque el datagrama era más grande que el búfer suministrado. MSG_CTRUNC indica que algún dato de control ha sido descartado debido a la falta de espacio en el búfer para datos auxiliares. MSG_OOB se devuelve para indicar que se han recibido datos despachados prontamente o fuera-de-banda.

```
VALOR DEVUELTO
       Estas llamadas devuelven el número de bytes  recibidos,  o
       bien -1 en caso de que ocurriera un error.

ERRORES
       EBADF   El argumento s es un descriptor inválido.

       ENOTCONN
               El zócalo está asociado con un protocolo orientado
               a la conexión y no ha  sido  conectado  (vea  con-
               nect(2) y accept(2)).

       ENOTSOCK
               El argumento s no se refiere a un zócalo.

       EWOULDBLOCK
               El  zócalo  está  marcado como no-bloqueante, y la
               operación de recepción produciría un bloqueo, o se
               ha puesto un límite de tiempo en la recepción, que
               ha expirado antes de que se recibieran datos.

       EINTR   La recepción ha sido interrumpida por  la  llegada
               de  una  señal  antes  de  que  hubiera algún dato
               disponible.

       EFAULT  El puntero  a  búfer  de  recepción  (o  punteros)
               apunta  afuera del espacio de direcciones del pro-
               ceso.

CONFORME A
       4.4BSD (estas funciones aparecieron  por  primera  vez  en
       4.2BSD).

VÉASE TAMBIÉN
       fcntl(2), read(2), select(2), getsockopt(2), socket(2)
```

SEND

```
SEND(2)          Manual del Programador de Linux          SEND(2)


NOMBRE
       send, sendto, sendmsg - envía un mensaje de un socket

SINTAXIS
       #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);

DESCRIPCIÓN
       ATENCIÓN: Esta es una pagina de manual BSD. Hasta el Linux
       0.99.11, sendmsg no ha sido implementado.

       Send,  sendto, y sendmsg son utilizados para transmitir un
```

mensaje a otro socket. Send solo puede ser usado cuando un socket está en un estado connected mientras sendto y sendmsg pueden ser utilizados en cualquier momento.

La dirección de destino viene dada por to con tolen especificando su tamaño. La longitud del mensaje viene dada por len. Si el mensaje es demasiado largo para pasar automáticamente a través del protocolo inferior, es devuelto el error EMSGSIZE y el mensaje no es transmitido.

La no indicación de un error viene implícito en una llamada a send. Errores detectados localmente son indicados medieante un valor de retorno -1.

Si no existe espacio disponible en el socket para contener el mensaje a enviar, entonces send se bloquea, a no ser que el socket ha sido habilitado en un modo de no bloqueo para la E/S (non-blocking I/O). La llamada select(2) puede ser utilizada para determinar cuando es posible enviar más información.

El parámetro flags puede contener uno o más de los siguientes valores:

```
#define   MSG_OOB        0x1  /* process out-of-band data
*/
#define   MSG_DONTROUTE  0x4  /* bypass routing, use
direct interface */
```

El parámetro MSG_OOB es utilizado para enviar out-of-band datos en sockets que soportan esta noción (p.ej. SOCK_STREAM); el protocolo inferior debe soportar información del tipo out-of-band MSG_DONTROUTE es utilizado solo para el diagnóstico de programas de enrutado.

Vea recv(2) para una descripción de la estructura msghdr

VALOR DEVUELTO

La llamada retorna el numero de caracteres enviados, o -1 si ha ocurrido un error.

ERRORES

EBADF Un descriptor no válido ha sido especificado.

ENOTSOCK
El argumento s no es un socket.

EFAULT Una dirección del espacio de usuario incorrecta ha sido especificada como paráme tro.

EMSGSIZE
El socket requiere que este mensaje sea enviado automáticamente, y el tamaño del mensaje lo hace imposible.

EWOULDBLOCK
El socket está marcado como "non-blocking" y la operación solicitada debe bloque ar.

ENOBUFS El sistema ha sido incapaz de habilitar un buffer interno. La operación se ejecutará cuando los buffers estén disponibles.

ENOBUFS La cola de salida del interface de red está lleno. Esto generalmente indica que el interfaz ha parado de enviar, pero puede ser causado por una congestión temporal.

COMPATIBLE CON

4.4BSD, SVr4 (estas llamadas al sistema aparecion en 4.2BSD). Las versiones SV4r docomentan las condiciones de error EINVAL, EINTR, EMSGSIZE, ENOSR, ENOMEM.

VÉASE TAMBIÉN

fcntl(2), recv(2), select(2), getsockopt(2), socket(2), write(2)

LISTEN

LISTEN(2) Manual del Programador de Linux LISTEN(2)

NOMBRE

listen - espera conexiones en un zócalo

SINOPSIS

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

int listen(int s, int backlog);
```

DESCRIPCIÓN

Para aceptar conexiones, primero se crea un zócalo con socket(2), luego se especifica con listen el deseo de aceptar conexiones entrantes, y un límite de la cola para dichas conexiones, y por último las conexiones son aceptadas mediante accept(2). La llamada listen se aplica solamente a zócalos de tipo SOCK_STREAM o SOCK_SEQPACKET.

El parámetro backlog define la longitud máxima a la que puede llegar la cola de conexiones pendientes. Si una petición de conexión llega estando la cola llena, el cliente puede recibir un error con una indicación de ECONNREFUSED, o, si el protocolo subyacente acepta retransmisiones, la petición puede no ser tenida en cuenta, de forma que un reintento pueda llegar a tener éxito.

VALOR DEVUELTO

En caso de éxito, se devuelve cero. En caso de error, se devuelve -1 y se pone en errno un valor apropiado.

ERRORS

EBADF El argumento s no es un descriptor válido.

ENOTSOCK
El argumento s no es un zócalo.

EOPNOTSUPP
El zócalo no es de un tipo que admita la operación listen.

CONFORME A

SVr4, 4.4BSD (la llamada a función listen apareció por 1ª vez en 4.2BSD).

FALLOS

Si el zócalo es de tipo af_inet, y el argumento backlog es mayor que la constante SO_MAXCONN (128 en 2.0.23), se trunca silenciosamente a SO_MAXCONN. Para aplicaciones transportables, no confíe en este valor puesto que BSD (y algunos sistemas derivados de BSD) limitan backlog a 5.

VÉASE TAMBIÉN

accept(2), connect(2), socket(2)

ACCEPT

```
ACCEPT(2)        Manual del programador de Linux        ACCEPT(2)


NOMBRE
       accept - acepta una conexión sobre un enchufe (socket).

SINOPSIS
       #include <sys/types.h>

       #include <sys/socket.h>

       int accept(int s, struct sockaddr *addr, int *addrlen);

DESCRIPCIÓN
       El argumento s es un enchufe que ha sido creado con
       socket(2), ligado a una dirección con bind(2) y que se
       encuentra a la escucha tras un listen(2). La función
       accept extrae la primera petición de conexión de la cola
       de conexiones pendientes, crea un nuevo enchufe con las
       mismas propiedades de s y asigna un nuevo descriptor de
       fichero para el enchufe. Si no hay conexiones pendientes
       en la cola y el enchufe no está marcado como "no blo-
       queante", accept se bloquea hasta que se presente una
       conexión. Si el enchufe está marcado como "no bloqueante"
       y no hay conexiones pendientes en la cola, accept devuelve
       un error tal como se describe abajo. El enchufe aceptado
       (cuyo descriptor es el valor de retorno de la función) no
       se puede usar para aceptar más conexiones. El enchufe
       original, s, permanece abierto.

       El argumento addr es un parámetro de salida al que se
       asigna la dirección de la entidad asociada, tal como se
       conoce en la capa de comunicaciones. El formato exacto del
       parámetro addr viene determinado por el dominio en el que
       se produce la comunicación. addrlen es un parámetro de
       entrada/salida; al efectuar la llamada debe contener la
       cantidad de espacio apuntado por addr; a la salida, con-
       tendrá la longitud real (en bytes) de la dirección
       devuelta. Esta llamada se usa con tipos de enchufes orien-
       tados a conexión, por el momento con SOCK_STREAM.

       Es posible elegir mediante select(2) un enchufe con la
       intención de hacer un accept seleccionándolo para lectura.

       Para determinados protocolos que necesitan una confir-
       mación explícita, tales como ISO o DATAKIT, accept se
       puede interpretar como una función que, simplemente, des-
       encola la siguiente petición de conexión sin que ello
       implique la confirmación. Se sobreentiende la confir-
       mación cuando se produce una lectura o escritura normal
       sobre el nuevo descriptor de fichero, y el rechazo puede
       ser de igual manera implícito cerrando el nuevo enchufe.

       Se puede obtener datos de la petición de conexión del
       usuario sin confirmar la conexión, llamando a la función
       recvmsg(2) con un valor 0 en el parámetro msg_iovlen y un
       valor no cero en msg_controllen, o llamando a la función
       getsockopt(2) De forma similar, se puede proporcionar
       información de rechazo de la conexión de usuario llamando
       a la función sendmsg(2) dando sólo la información de con-
       trol, o llamando a setsockopt(2).

VALOR DEVUELTO
```

La llamada devuelve -1 ante un error. Si tiene éxito, devuelve un entero no negativo que es el descriptor del enchufe aceptado.

ERRORES

La página de manual de BSD documenta cinco casos de error posibles.

EBADF El descriptor es inválido.

ENOTSOCK
El descriptor referencia a un fichero, no a un enchufe.

EOPNOTSUPP
El enchufe referenciado no es del tipo SOCK_STREAM.

EFAULT El parámetro addr no se encuentra en una zona accesible para escritura por el usuario.

EWOULDBLOCK
El enchufe está marcado como no bloqueante y no hay conexiones que aceptar.

Diferentes núcleos de Linux pueden devolver otros errores diferentes como EMFILE, EINVAL, ENOSR, ENOBUFS, EAGAIN, EPERM, ECONNABORTED, ESOCKTNOSUPPORT, EPROTONOSUPPORT, ETIMEDOUT, ERESTARTSYS.

CONFORME A

SVr4, 4.4BSD (la función accept apareció por primera vez en BSD 4.2). IRIX documenta además los errores EMFILE y ENFILE. Solaris documenta adicionalmente los errores EINTR, ENODEV, ENOMEM, ENOSR y EPROTO.

VÉASE TAMBIÉN

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

CONNECT

CONNECT(2) Manual del Programador de Linux CONNECT(2)

NOMBRE

connect - inicia una conexión en un socket (enchufe)

SINOPSIS

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

int connect(int sockfd, struct sockaddr *serv_addr, int
addrlen );
```

DESCRIPCIÓN

El parámetro sockfd es un socket. Si es del tipo SOCK_DGRAM, esta llamada especifica el interlocutor (peer) con el cual el socket va a ser asociado; esta es la dirección a la que se van a enviar datagramas, y la única dirección de la que se van a recibir datagramas. Si el socket es del tipo SOCK_STREAM , esta llamada intenta hacer una conexión a otro socket. El otro socket está especificado por serv_addr, la cual es una dirección en el espacio de comunicaciones del socket. Cada espacio de comunicaciones interpreta el parámetro serv_addr, a su

manera. Generalmente, sockets de corriente (stream sockets) pueden conectarse con éxito mediante connect una vez solamente; sockets de datagramas pueden usar connect múltiples veces para cambiar sus asociaciones. Los sockets de datagramas pueden disolver la asociación conectando a una dirección inválida, tal como una dirección nula.

VALOR DEVUELTO

Si la conexión o enlace tiene éxito, se devuelve 0. En caso de error, se devuelve -1, y la variable errno se asigna apropiadamente.

ERRORES

Los siguientes sólo son errores generales de socket. Puede haber otros códigos de error específicos del dominio.

EBADF Descriptor de fichero está mal.

EFAULT La estructura de dirección del socket está fuera de su espacio de direcciones.

ENOTSOCK
El descriptor no está asociado con un socket.

EISCONN El socket ya está conectado.

ECONNREFUSED
Conexión rehusada por el servidor.

ETIMEDOUT
Plazo límite de tiempo alcanzado (timeout) mientras se intentaba la conexión.

ENETUNREACH
Red inaccesible.

EADDRINUSE
La dirección ya está en uso.

EINPROGRESS
El socket es no bloqueante (non-blocking) y la conexión no puede completarse inmediatamente. Es posible usar select(2) para completarla seleccionando el socket para escritura. Después que select indique que la escritura es posible, use getsockopt(2) para leer la opción SO_ERROR al nivel SOL_SOCKET para determinar si connect se completó con éxito (SO_ERROR será cero) o sin éxito (SO_ERROR será uno de los códigos de error usuales listados arriba, explicando la razón del fallo).

EALREADY
El socket es no bloqueante (non-blocking) y un intento de conexión anterior aún no se ha completado.

CONFORME A

SVr4, 4.4BSD (la función connect apareció por primera vez en BSD 4.2). SVr4 documenta adicionalmente los códigos de error generales EADDRNOTAVAIL, EINVAL, EAFNOSUPPORT, EALREADY, EINTR, EPROTOTYPE, ENOSR. También documenta muchas condiciones de error adicionales que no se describen aquí.

VÉASE TAMBIÉN

accept(2), bind(2), listen(2), socket(2), getsockname(2)

HTONL, NTOHL – Direciones de host a red

```
BYTEORDER(3)     Manual del Programador de Linux     BYTEORDER(3)


NOMBRE
       htonl,  htons,  ntohl,  ntohs  -  convierten valores cuyos
       bytes se encuentran en orden de host a valores cuyos bytes
       se encuentran en orden de red y viceversa

SINOPSIS
       #include <netinet/in.h>

       unsigned long int htonl(unsigned long int hostlong);

       unsigned short int htons(unsigned short int hostshort);

       unsigned long int ntohl(unsigned long int netlong);

       unsigned short int ntohs(unsigned short int netshort);

DESCRIPCIÓN
       La  función  htonl()  convierte  el  entero largo hostlong
       desde el orden de bytes del host al de la red.

       La función htons() convierte  el  entero  corto  hostshort
       desde el orden de bytes del host al de la red.

       La función ntohl() convierte el entero largo netlong desde
       el orden de bytes de la red al del host.

       La función ntohs()  convierte  el  entero  corto  netshort
       desde el orden de bytes de la red al del host.

       En  los  i80x86 en el orden de bytes del host está primero
       el byte menos significativo (LSB), mientras que  el  orden
       de  bytes  de  la  red, tal como se usa en Internet, tiene
       primero el byte más significativo (MSB).

CONFORME A
       BSD 4.3

VÉASE TAMBIÉN
       gethostbyname(3), getservent(3)
```

GETHOSTBYNAME – Resolucion DNS

```
GETHOSTBYNAME(3) Manual del Programador de Linux GETHOSTBYNAME(3)


NOMBRE
       gethostbyname, gethostbyaddr, sethostent, endhostent, her-
       ror - obtienen una entrada de anfitrión de red

SINOPSIS
       #include <netdb.h>
       extern int h_errno;

       struct hostent *gethostbyname(const char *name);

       #include <sys/socket.h>          /* para AF_INET */
```

```
       struct hostent *gethostbyaddr(const char *addr, int len, int
type);

       void sethostent(int stayopen);

       void endhostent(void);

       void herror(const char *s);
```

DESCRIPCIÓN

La función gethostbyname() devuelve una estructura del tipo hostent para el anfitrión (host) dado name. Aquí, name es ora un nombre de anfitrión, ora una dirección IPv4 en la notación normal de puntos, ora una dirección IPv6 en la notación de dos puntos (y posiblemente de puntos). (Vea la RFC 1884 para una descripción de las direcciones en IPv6). Si name es una dirección IPv4 o IPv6, no se realiza ninguna búsqueda y gethostbyname() simplemente copia name en el campo h_name y su equivalente struct in_addr en el campo h_addr_list[0] de la estructura hostent devuelta. Si name no termina con un punto y la variable de ambiente HOSTALIASES está asignada, se buscará primero name en el fichero de alias señalado por HOSTALIASES. (Vea hostname(7) para saber cómo es el formato del fichero.) Se buscan el dominio actual y sus ancestros a menos que name termine en punto.

La función gethostbyaddr() devuelve una estructura del tipo hostent para la dirección de anfitrión dada addr de longitud len y de tipo type. El único tipo de dirección válido actualmente es AF_INET.

La función sethostent() especifica, si stayopen es true (1), que se debería emplear un conector (socket) TCP para las interrogaciones al servidor de nombres y que la conexión debería permanecer abierta durante sucesivas preguntas. De otro modo, las peticiones al servidor de nombres utilizarán datagramas UDP.

La función endhostent() termina el uso de una conexión TCP para las peticiones al servidor de nombres.

La función herror() muestra en stderr un mensaje de error asociado con el valor actual de h_errno.
Las preguntas al servidor de nombres llevadas a cabo por gethostbyname() y gethostbyaddr() usan una combinación de uno o todos los servidores de nombres named(8), una declaración en /etc/hosts, y el Servicio de Información de Red (NIS, antes Páginas Amarillas, YP), dependiendo de los contenidos de la línea order en /etc/host.conf. (Vea resolv+(8)). La acción predeterminada es preguntar a named(8), seguido por /etc/hosts.

La estructura hostent se define en <netdb.h> como sigue:

```
              struct hostent {
                      char    *h_name;        /* nombre oficial del
anfitrión */
                      char    **h_aliases;    /* lista de alias */
                      int     h_addrtype;     /* tipo dirección
anfitrión */
                      int     h_length;       /* longitud de la
dirección */
                      char    **h_addr_list;  /* lista de direcciones */
              }
              #define h_addr  h_addr_list[0]  /* por compatibilidad
atrás */
```

```
        Los miembros de la estructura hostent son:

        h_name El nombre oficial de un anfitrión.

        h_aliases
                Una cadena terminada en el  carácter  nulo  de  los
                nombres alternativos para el anfitrión.

        h_addrtype
                El tipo de dirección; siempre AF_INET de momento.

        h_length
                La longitud de la dirección en bytes.

        h_addr_list
                Una  cadena terminada en nulo de direcciones de red
                para el anfitrión en orden de bytes de red.

        h_addr La primera dirección en h_addr_list por compatibil-
                idad hacia atrás.

VALOR DEVUELTO
        Las  funciones gethostbyname() y gethostbyaddr() devuelven
        la estructura hostent, o un puntero NULL si ha ocurrido un
        error.  En  caso de error, la variable h_errno contiene un
        número de error.

ERRORES
        La variable h_errno puede tener los siguientes valores:

        HOST_NOT_FOUND
                El anfitrión especificado es desconocido.

GETHOSTBYNAME(3) Manual del Programador de Linux GETHOSTBYNAME(3)


        NO_ADDRESS
                El nombre  pedido  es  válido  pero  no  tiene  una
                dirrección IP.

        NO_RECOVERY
                Ha ocurrido un error no recuperable del servidor de
                nombres.

        TRY_AGAIN
                Ha ocurrido un error temporal sobre un servidor  de
                nombres con autoridad. Intente luego más tarde.

FICHEROS
        /etc/host.conf
                fichero de configuración del resolvedor

        /etc/hosts
                fichero de base de datos de anfitriones

CONFORME A
        BSD 4.3

VÉASE TAMBIÉN
        resolver(3),  hosts(5), hostname(7), resolv+(8), named(8).
```

INET – Manipulacion Direcciones Internet

NOMBRE

inet_aton, inet_addr, inet_network, inet_ntoa, inet_makeaddr, inet_lnaof, inet_netof - Rutinas de manipulación de direcciones de Internet

SINOPSIS

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

int inet_aton(const char *cp, struct in_addr *inp);

unsigned long int inet_addr(const char *cp);

unsigned long int inet_network(const char *cp);

char *inet_ntoa(struct in_addr in);

struct in_addr inet_makeaddr(int net, int host);

unsigned long int inet_lnaof(struct in_addr in);

unsigned long int inet_netof(struct in_addr in);
```

DESCRIPCIÓN

inet_aton() convierte la dirección de Internet cp desde la notación estándar de números y puntos a la representación binaria, y la guarda en la estructura a la que apunte inp. inet_aton devuelve no-cero si la dirección es válida, cero si no.

La función inet_addr() convierte la dirección de Internet cp desde la notación de números y puntos a la de datos binarios en orden de bytes del ordenador local. Si la entrada no es válida, devuelve -1. Ésta es una interfaz obsoleta a inet_aton, descrita anteriormente; es obsoleta porque -1 es una dirección válida (255.255.255.255), e inet_aton porporciona una manera más clara para indicar que ha ocurrido un error.

La función inet_network() extrae el número de red en orden de bytes de red desde la dirección cp a la notación de números y puntos. Si la entrada es inválida, devuelve -1.

La función inet_ntoa() convierte la dirección de Internet in dada en orden de bytes de red a una cadena de caracteres en la notación estándar de números y puntos. La cadena se devuelve en un búfer reservado estáticamente, que será sobreescrito en siguientes llamadas.

La función inet_makeaddr() construye una dirección de Internet en orden de bytes de red combinando el número de red net con la dirección local host en la red net, ambas en orden de bytes de ordenador local.

La función inet_lnaof() devuelve la parte del ordenador local de la dirección de Internet in. La dirección de ordenador local se devuelve en orden de bytes de ordenador local.

La función inet_netof() devuelve la parte de número de red de la dirección de Internet in. El número de red se devuelve en orden de bytes de ordenador local.

La estructura in_addr, empleada en inet_ntoa(),

inet_makeaddr(), inet_lnoaf() e inet_netof() se define en netinet/in.h como:

```
struct in_addr {
        unsigned long int s_addr;
}
```

Observe que en el i80x86 el orden de bytes de ordenador es: primero el Byte Menos Significativo (LSB), mientras que el orden de bytes de red es, como se usa en la Internet, el Byte Más Significativo (MSB) primero.

CONFORME A
BSD 4.3

VÉASE TAMBIÉN
gethostbyname(3), getnetent(3), hosts(5), networks(5)

SELECT

SELECT(2) Manual del Programador de Linux SELECT(2)

NOMBRE
select, FD_CLR, FD_ISSET, FD_SET, FD_ZERO - multiplexación de E/S síncrona

SINOPSIS

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int select(int n, fd_set *readfds, fd_set *writefds,
fd_set *exceptfds, struct timeval *timeout);

FD_CLR(int fd, fd_set *set);
FD_ISSET(int fd, fd_set *set);
FD_SET(int fd, fd_set *set);
FD_ZERO(fd_set *set);
```

DESCRIPCIÓN
select espera a que una serie de descriptores de ficheros cambien su estado.

Se miran tres conjuntos independientes de descriptores. Aquéllos listados en readfds serán observados para ver si hay caracteres que llegan a estar disponibles para lectura, aquéllos en writefds serán observados para ver si es correcto escribir inmediatamente en ellos, y aquéllos en exceptfds serán observados para ver si ocurren excepciones. En caso de éxito, los conjuntos se modifican en marcha para indicar qué descriptores cambiaron realmente su estado.

Se proporcionan cuatro macros para manipular los conjuntos. FD_ZERO limpiará un conjunto. FD_SET y FD_CLR añaden o borran un descriptor dado a o de un conjunto. FD_ISSET mira a ver si un descriptor es parte del conjunto; esto es útil después de que select regrese.

n es el descriptor con el número más alto en cualquiera de los tres conjuntos, más 1.

timeout es un límite superior de la cantidad de tiempo

transcurrida antes de que select regrese. Puede ser cero, causando que select regrese inmediatamente. Si timeout es NULL (no hay tiempo de espera), select puede bloquear indefinidamente.

VALOR DEVUELTO

En caso de éxito, select devuelve el número de descriptores contenidos en los conjuntos de descriptores, que puede ser cero si el tiempo de espera expira antes de que ocurra algo interesante. En caso de error, se devuelve -1, y se pone un valor apropiado en errno; los conjuntos y timeout estarán indefinidos, así que no confíe en sus contenidos tras un error.

ERRORES

EBADF Se ha dado un descriptor de fichero inválido en uno de los conjuntos.

EINTR Se ha capturado una señal no bloqueante.

EINVAL n es negativo.

ENOMEM select no ha sido capaz de reservar memoria para las tablas internas.

OBSERVACIONES

Hay algún código por ahí que llama a select con los tres conjuntos vacíos, n cero, y un timeout distinto de cero como una forma transportable y curiosa de dormir con una precisión por debajo del segundo.

En Linux, timeout se modifica para reflejar la cantidad de tiempo no dormido; la mayoría de otras implementaciones no hacen esto. Esto produce problemas cuando el código de Linux que lee timeout se transporta a otros sistemas operativos, y cuando se transporta a Linux código que reutiliza una struct timeval para varias selects en un bucle sin reinicializarla. Considere que timeout está indefinido después de que select regrese.

EJEMPLO

```
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int
main(void)
{
    fd_set rfds;
    struct timeval tv;
    int valret;

    /* Mirar stdin (df 0) para ver si tiene entrada */
    FD_ZERO(&rfds);
    FD_SET(0, &rfds);
    /* Esperar hasta 5 s */
    tv.tv_sec = 5;
    tv.tv_usec = 0;

    valret = select(1, &rfds, NULL, NULL, &tv);
    /* ¡No confiar ahora en el valor de tv! */

    if (valret)
        printf("Los datos ya están disponibles.\n");
        /* FD_ISSET(0, &rfds) será verdadero */
    else
        printf("Ningún dato en 5 segundos.\n");
```

```
            return(0);
        }
```

CONFORME A

4.4BSD (la función select apareció por primera vez en 4.2BSD). Generalmente es transportable a o desde sistemas no-BSD que admitan clones de la capa de zócalos de BSD (incluyendo variantes System V). Sin embargo, observe que la variante System V normalmente pone la variable de espera antes de salir, pero la variante BSD no.

VÉASE TAMBIÉN

accept(2), connect(2), read(2), recv(2), send(2), write(2)

Signals

SIGACTION

SIGACTION(2) Manual del Programador de Linux SIGACTION(2)

NOMBRE

sigaction, sigprocmask, sigpending, sigsuspend - funciones POSIX de manejo de señales

SINOPSIS

```
#include <signal.h>


int sigaction(int signum, const struct sigaction *act,
struct sigaction *oldact);

int sigprocmask(int how, const sigset_t *set, sigset_t
*oldset);

int sigpending(sigset_t *set);

int sigsuspend(const sigset_t *mask);
```

DESCRIPCIÓN

La llamad al sistema sigaction se emplea para cambiar la acción tomada por un proceso cuando recibe una determinada señal.

signum especifica la señal y puede ser cualquiera válida salvo SIGKILL o SIGSTOP.

Si act no es nulo, la nueva acción para la señal signum se instala como act. Si oldact no es nulo, la acción anterior se guarda en oldact.

La estructura sigaction se define como

```
struct sigaction {
    void (*sa_handler)(int);
    sigset_t sa_mask;
    int sa_flags;
    void (*sa_restorer)(void);
}
```

sa_handler especifica la acción que se va a asociar con signum y puede ser SIG_DFL para la acción predeterminada, SIG_IGN para no tener en cuenta la señal, o un puntero a una función manejadora para la señal.

sa_mask da una máscara de señales que deberían bloquearse durante la ejecución del manejador de señal. Además, la señal que lance el manejador será bloqueada, a menos que se activen las opciones SA_NODEFER o SA_NOMASK.

sa_flags especifica un conjunto de opciones que modifican el comportamiento del proceso de manejo de señal. Se forma por la aplicación del operador de bits OR a cero o más de las siguientes constantes:

SA_NOCLDSTOP
: Si signum es SIGCHLD, no se reciba notificación cuando los procesos hijos se paren (esto es, cuando los procesos hijos reciban una de las señales SIGSTOP, SIGTSTP, SIGTTIN o SIGTTOU).

SA_ONESHOT o SA_RESETHAND
: Restáurese la acción para la señal al estado predeterminado una vez que el manejador de señal haya sido llamado. (Esto es el comportamiento predeterminado de la llamada al sistema signal(2).)

SA_RESTART
: Proporciona un comportamiento compatible con la semántica de señales de BSD haciendo reejecutables algunas llamadas al sistema entre señales.

SA_NOMASK o SA_NODEFER
: No se impida que se reciba la señal desde su propio manejador.

El elemento sa_restorer está anticuado y no debería emplearse.

La llamada sigprocmask se emplea para cambiar la lista de señales bloqueadas actualmente. El comportamiento de la llamada depende del valor de how, como sigue:

SIG_BLOCK
: El conjunto de señales bloqueadas es la unión del conjunto actual y el argumento set.

SIG_UNBLOCK
: Las señales en set se quitan del conjunto actual de señales bloqueadas. Es legal intentar el desbloqueo de una señal que no está bloqueada.

SIG_SETMASK
: El conjunto de señales bloqueadas se pone según el argumento set.

Si oldset no es nulo, el valor anterior de la máscara de señal se guarda en oldset.

La llamada sigpending permite el examen de señales pendientes (las que han sido producidas mientras estaban bloqueadas). La máscara de señal de las señales pendientes se guarda en set.

La llamada sigsuspend reemplaza temporalmente la máscara de señal para el proceso con la dada por mask y luego suspende el proceso hasta que se recibe una señal.

VALOR DEVUELTO

sigaction, sigprocmask, sigpending y sigsuspend devuelven 0 en caso de éxito y -1 en caso de error.

ERRORES

EINVAL Se ha especificado una señal inválida. Esto también se genera si se hace un intento de cambiar la acción para SIGKILL o SIGSTOP, que no pueden ser capturadas.

EFAULT act, oldact, set u oldset apuntan a una zona de memoria que no forma parte válida del espacio de direcciones del proceso.

EINTR La llamada al sistema ha sido interrumpida.

OBSERVACIONES

No es posible bloquear SIGKILL ni SIGSTOP con una llamada a sigprocmask. Los intentos de hacerlo no serán tenidos en cuenta, silenciosamente.

De acuerdo con POSIX, el comportamiento de un proceso está indefinido después de que no haga caso de una señal SIGFPE, SIGILL o SIGSEGV que no haya sido generada por las funciones kill() o raise(). La división entera por cero da un resultado indefinido. En algunas arquitecturas generará una señal SIGFPE. (También, el dividir el entero más negativo por -1 puede generar una señal SIGFPE.) No hacer caso de esta señal puede llevar a un bucle infinito.

POSIX (B.3.3.1.3) anula el establecimiento de SIG_IGN como acción para SIGCHLD. Los comportamientos de BSD y SYSV difieren, provocando el fallo en Linux de aquellos programas BSD que asignan SIG_IGN como acción para SIGCHLD.

La especificación POSIX sólo define SA_NOCLDSTOP. El empleo de otros valores en sa_flags no es transportable.

La opción SA_RESETHAND es compatible con la de SVr4 del mismo nombre.

La opción SA_NODEFER es compatible con la de SVr4 del mismo nombre bajo los núcleos 1.3.9 y posteriores. En núcleos más viejos la implementación de Linux permitía la recepción de cualquier señal, no sólo la que estábamos instalando (sustituyendo así efectivament cualquier valor de sa_mask).

Los nombres SA_RESETHAND y SA_NODEFER para compatibilidad con SVr4 están presentes solamente en las versiones de la biblioteca 3.0.9 y mayores.

sigaction puede llamarse con un segundo argumento nulo para saber el manejador de señal en curso. También puede emplearse para comprobar si una señal dada es válida para la máquina donde se está, llamándola con el segundo y el tercer argumento nulos.

Vea sigsetops(3) para detalles sobre manipulación de conjuntos de señales.

```
CONFORME A
       POSIX, SVr4.  SVr4 no documenta la condición EINTR.


VÉASE TAMBIÉN
       kill(1), kill(2), killpg(2), pause(2), raise(3), siginter-
       rupt(3), signal(2), signal(7), sigsetops(3), sigvec(2)
```

SIGSETOPS – Manejo de mascaras

```
SIGSETOPS(3)     Manual del Programador de Linux     SIGSETOPS(3)


NOMBRE
       sigemptyset, sigfillset, sigaddset, sigdelset, sigismember
       - operaciones POSIX con conjuntos de señales


SINOPSIS
       #include <signal.h>


       int sigemptyset(sigset_t *conjunto);

       int sigfillset(sigset_t *conjunto);

       int sigaddset(sigset_t *conjunto, int numse);

       int sigdelset(sigset_t *conjunto, int numse);

       int sigismember(const sigset_t *conjunto, int numse);


DESCRIPCIÓN
       La función sigsetops(3) permite la manipulación de conjun-
       tos de señales, según la norma POSIX.

       sigemptyset  inicia  el  conjunto de señales dado por con-
       junto al conjunto vacío, con todas las señales  fuera  del
       conjunto.

       sigfillset inicia conjunto al conjunto completo, con todas
       las señales incluidas en el conjunto.

       sigaddset y sigdelset añaden y quitan  respectivamente  la
       señal numse de conjunto.

       sigismember mira a ver si numse pertenece a conjunto.


VALOR DEVUELTO
       sigemptyset, sigfullset, sigaddset y sigdelset devuelven 0
       si acaban bien y -1 en caso de error.

       sigismember devuelve 1 si numse es un miembro de conjunto,
       0 si numse no lo es, y -1 en caso de error.


ERRORES
       EINVAL sig no es una señal válida.


CONFORME A
```

```
       POSIX

VÉASE TAMBIÉN
       sigaction(2), sigpending(2), sigprocmask(2), sigsuspend(2)
```

Lista de SIGNALs

```
SIGNAL(7)        Manual del Programador de Linux        SIGNAL(7)


NOMBRE
       signal - lista de las señales disponibles

DESCRIPCIÓN
       Linux  permite el uso de las señales dadas a continuación.
       Los números de varias de las señales dependen de la arqui-
       tectura  del  sistema.  Primero,  las señales descritas en
       POSIX.1.


       Señal      Valor     Acción   Comentario
       ------------------------------------------------------------
-----
       SIGHUP        1        A      Cuelgue detectado en la terminal de
                                     control o muerte del proceso de
control
       SIGINT        2        A      Interrupción procedente del teclado
       SIGQUIT       3        A      Terminación procedente del teclado
       SIGILL        4        A      Instrucción ilegal
       SIGABRT       6        C      Señal de aborto procedente de
abort(3)
       SIGFPE        8        C      Excepción de coma flotante
       SIGKILL       9       AEF     Señal de matar
       SIGSEGV      11        C      Referencia inválida a memoria
       SIGPIPE      13        A      Tubería rota: escritura sin
lectores
       SIGALRM      14        A      Señal de alarma de alarm(2)
       SIGTERM      15        A      Señal de terminación
       SIGUSR1   30,10,16     A      Señal definida por usuario 1
       SIGUSR2   31,12,17     A      Señal definida por usuario 2
       SIGCHLD   20,17,18     B      Proceso hijo terminado o parado
       SIGCONT   19,18,25            Continuar si estaba parado
       SIGSTOP   17,19,23    DEF     Parar proceso
       SIGTSTP   18,20,24     D      Parada escrita en la tty
       SIGTTIN   21,21,26     D      E. de la tty para un proc. de fondo
       SIGTTOU   22,22,27     D      S. a la tty para un proc. de fondo

       A continuación otras señales.


       Señal        Valor     Acción   Comentario
       ------------------------------------------------------------
--------
       SIGTRAP        5         CG     Trampa para rastreo/punto de
ruptura
       SIGIOT         6         CG     Trampa IOT. Un sinónimo de
SIGABRT
       SIGEMT       7,-,7       G
       SIGBUS      10,7,10      AG     Error del bus.
       SIGSYS      12,-,12      G      Argumento incorrecto para función
(SVID)
       SIGSTKFLT    -,16,-      AG     Fallo de la pila en el
coprocesador
```

```
       SIGURG      16,23,21     BG     Condición urgente en zócalo
(socket)
                                       (4.2 BSD)
       SIGIO       23,29,22     AG     E/S permitida ya (4.2 BSD)
       SIGPOLL                  AG     Un sinónimo de SIGIO (System V)
       SIGCLD       -,-,18      G      Un sinónimo de SIGCHLD
       SIGXCPU     24,24,30     AG     Tiempo límite de la CPU excedido
                                       (4.2 BSD)
       SIGXFSZ     25,25,31     AG     Tamaño límite de fichero excedido
                                       (4.2 BSD)
       SIGVTALRM   26,26,28     AG     Reloj-despertador virtual
                                       (4.2 BSD)
       SIGPROF     27,27,29     AG     Perfilar Reloj-despertador
       SIGPWR      29,30,19     AG     Fallo de corriente eléctrica
(System V)
       SIGINFO      29,-,-      G      Un sinónimo para SIGPWR
       SIGLOST      -,-,-       AG     Fichero de bloqueo perdido.
       SIGWINCH    28,28,20     BG     Señal de reescalado de la ventana
                                       (4.3 BSD, Sun)
       SIGUNUSED    -,31,-      AG     Señal no usada.
```

(Aquí, - denota que una señal está ausente; allí donde se indican tres valores, el primero es comúnmente válido para alpha y sparc, el segundo para i386 y ppc, y el último para mips. La señal 29 es SIGINFO /SIGPWR en un alpha pero SIGLOST en una sparc.)

Las letras en la columna "Acción" tienen los siguientes significados:

A La acción por omisión es terminar el proceso.

B La acción por omisión es no hacer caso de la señal.

C La acción por omisión es hacer un volcado de memoria.

D La acción por omisión es parar el proceso.

E La señal no puede ser capturada.

F La señal no puede ser pasada por alto.

G Señal no conforme con POSIX.1.

CONFORME A

POSIX.1

ERRORES

SIGIO y SIGLOST tienen el mismo valor. Este último está comentado en las fuentes del núcleo, pero el proceso de construcción de algunos programas aún piensa que la señal 29 es SIGLOST.

VÉASE TAMBIÉN

kill(1), kill(2), setitimer(2)

WAIT

```
WAIT(2)          Manual del Programador de Linux          WAIT(2)
```

NOMBRE

wait, waitpid - espera por el final de un proceso

SINOPSIS

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

pid_t wait(int *status)
pid_t waitpid(pid_t pid, int *status, int options);
```

DESCRIPCIÓN

La función wait suspende la ejecución del proceso actual haste que un proceso hijo ha terminado, o hasta que se produce una señal cuya acción es terminar el proceso actual o llamar a la función manejadora de la señal. Si un hijo ha salido cuando se produce la llamada (lo que se entiende por proceso "zombie"), la función vuelve inmediatamente. Todos los recursos del sistema reservados por el hijo son liberados.

La función waitpid suspende la ejecución del proceso en curso hasta que un hijo especificado por el argumento pid ha terminado, o hasta que se produce una señal cuya acción es finalizar el proceso actual o llamar a la función manejadora de la señal.

Si el hijo especificado por pid ha terminado cuando se produce la llamada (un proceso "zombie"), la función vuelve inmediatamente. Todos los recursos del sistema reservados por el hijo son liberados.

El valor de pid puede ser uno de los siguientes:

< -1 lo que significa esperar a que cualquier proceso hijo cuyo ID del proceso es igual al valor absoluto de pid.

-1 lo que significa que espera por cualquier proceso hijo; este es el mismo comportamiento que tiene wait.

0 lo que significa que espera por cualquier proceso hijo cuyo ID es igual al del proceso llamante.

> 0 lo que significa que espera por el proceso hijo cuyo ID es igual al valor de pid.

El valor de options es un OR de cero o más de las siguientes constantes:

WNOHANG que significa que vuelve inmediatamente si ningún hijo ha terminado.

WUNTRACED
que significa que también vuelve si hay hijos parados, y de cuyo estado no ha recibido notificación.

Si status no es NULL, wait o waitpid almacena la información de estado en la memoria apuntada por status.

Si el estado puede ser evaluado con las siguientes macros (dichas macros toman el buffer stat (un int) como argumento -- ¡no un puntero al buffer!):

WIFEXITED(status)
es distinto de cero si el hijo terminó normalmente.

WEXITSTATUS(status)
evalúa los ocho bits menos significativos del código de retorno del hijo que terminó, que podrían estar activados como el argumento de una llamada a exit() o como el argumento de unñ return en el programa principal. Esta macro solamente puede ser tenida en cuenta si WIFEXITED devuelve un valor distinto de cero.

WIFSIGNALED(status)
devuelve true si el proceso hijo terminó a causa de una señal no capturada.

WTERMSIG(status)
devuelve el número de la señal que provocó la muerte del proceso hijo. Esta macro sólo puede ser evaluada si WIFSIGNALED devolvió un valor distinto de cero.

WIFSTOPPED(status)
devuelve true si el proceso hijo que provocó el retorno está actualmente pardo; esto solamente es posible si la llamada se hizo usando WUNTRACED.

WSTOPSIG(status)
devuelve el número de la señal que provocó la parada del hijo. Esta macro solamente puede ser evaluada si WIFSTOPPED devolvió un valor distinto de cero.

VALOR DEVUELTO

El ID del proceso del hijo que terminó, -1 en caso de error o cero si se utilizó WNOHANG y no hay hijo disponible (en este caso, errno se pone a un valor apropiado).

ERRORES

ECHILD si el proceso especificado en pid no termina o no es hijo del proceso llamante. (Esto puede ocurrir para nuestros propios hijos si se asigna SIG_IGN como acción de SIGCHLD.)

EINVAL si el argumento options no fue valido.

ERESTARTSYS
si no se activó WNOHANG y si no se ha capturado una señal no bloqueada o SIGCHLD El interfaz de la biblioteca no tiene permitido devolver ERESTARTSYS, pero devolverá EINTR.

NOTAS

The Single Unix Specification (Especificación para un Unix Único) describe un modificador SA_NOCLDWAIT (no presente en Linux) tal que si este modificador está activo, o bien se ha asignado SIG_IGN como acción para SIGCHLD (que, por cierto, no está permitido por POSIX), entonces los hijos que terminan no se convierten en zombies y una llamada a wait() o waitpid() se bloqueará hasta que todos los hijos hayan terminado y, a continuación, fallará asignando a errno el valor ECHILD.

CONFORME A

SVr4, POSIX.1

VÉASE TAMBIÉN

signal(2), wait4(2), signal(7)

KILL Envia señal a proceso

```
KILL(2)          Manual del Programador de Linux          KILL(2)


NOMBRE
       kill - enviar una señal a un proceso

SINOPSIS
       #include <sys/types.h>
       #include <signal.h>

       int kill(pid_t pid, int sig);

DESCRIPCIÓN
       La  llamada kill se puede usar para enviar cualquier señal
       a un proceso o grupo de procesos.

       Si pid es positivo, entonces la señal  sig  es  enviada  a
       pid.  En este caso, se devuelve 0 si hay éxito, o un valor
       negativo si hay error.

       Si pid es 0, entonces sig se envía a cada  proceso  en  el
       grupo de procesos del proceso actual.

       Si  pid  es  igual a -1, entonces se envía sig a cada pro-
       ceso, excepto al primero, desde los números más  altos  en
       la tabla de procesos, hasta los más bajos.

       Si  pid es menor que -1, entonces se envía sig a cada pro-
       ceso en el grupo de procesos -pid.

       Si sig es 0, entonces  no  se  envía  ninguna  señal  pero
       todavía se realiza la comprobación de errores.


VALOR DEVUELTO
       Si  hay éxito, se devuelve cero. Si hay error, se devuelve
       -1, y se actualiza errno apropiadamente.

ERRORES
       EINVAL Se especificó una señal inválida.

       ESRCH  El pid o grupo de procesos no existe. Nótese que un
              proceso  existente  podría ser un zombi, un proceso
              que ya ha sido terminado, pero que aún no  ha  sido
              "wait()eado".

       EPERM  El  proceso no tiene permiso para enviar la señal a
              alguno de los procesos que la recibirán.  Para  que
              un  proceso  tenga permiso para enviar una señal al
              proceso pid debe, o bien tener privilegios de root,
              o bien el ID de usuario real o efectivo del proceso
              que envía la señal ha de ser igual  al  set-user-ID
              real o guardado del proceso que la recibe.

FALLOS
       Es  imposible  enviar  una señal a la tarea número uno, el
       proceso  init,  para  el  que  no  ha  sido  instalado  un
       manejador  de señales. Esto se hace para asegurarse de que
       el sistema no se venga abajo accidentalmente.

CONFORME A
       SVr4, SVID, POSIX.1, X/OPEN y BSD 4.3
```

```
VÉASE TAMBIÉN
       _exit(2), exit(3), signal(2), signal(7)
```

ALARM

```
ALARM(2)         Manual del Programador de Linux         ALARM(2)


NOMBRE
       alarm - activa una alarma para el envio de una señal

SINOPSIS
       #include <unistd.h>

       unsigned int alarm(unsigned int sec);

DESCRIPCIÓN
       alarm se encarga de enviar una señal SIGALRM al proceso en
       sec segundos.

       Si sec es cero, no se prepara una  nueva  alarma.  Además,
       cualquier alarm previamente preparada se cancela.

VALOR DEVUELTO
       alarm devuelve el número de segundos que quedaban para que
       cualquier alarma previa se disparase, o cero si  no  había
       ninguna alarma pendiente.

NOTAS
       alarm  y   setitimer comparten el mismo cronómetro; el uso
       de una interferirá con el de la otra.

       Los retardos en la  planificación  pueden,  como  siempre,
       provocar el retardo de la ejecución del proceso una canti-
       dad arbitraria de tiempo.

CONFORME A
       SVID, AT&T, POSIX, X/OPEN, BSD 4.3

VÉASE TAMBIÉN
       setitimer(2),  signal(2),  sigaction(2),  gettimeofday(2),
       select(2), pause(2), sleep(3)
```

PAUSE

```
PAUSE(2)         Manual del Programador de Linux         PAUSE(2)


NOMBRE
       pause - espera una señal

SINOPSIS
       #include <unistd.h>

       int pause(void);

DESCRIPCIÓN
       La  llamada  al sistema pause hace que el proceso en curso
       se duerma hasta que reciba una señal.

VALOR DEVUELTO
       pause siempre devuelve -1, y errno toma el valor ERESTART-
```

```
       NOHAND.

ERRORES
       EINTR  se ha recibido una señal.

CONFORME A
       SVr4, SVID, POSIX, X/OPEN, BSD 4.3

VÉASE TAMBIÉN
       kill(2), select(2), signal(2)
```

SIGNAL – Según estándar ANSI C

```
SIGNAL(2)        Manual del Programador de Linux        SIGNAL(2)


NOMBRE
       signal - manejo de señales según C ANSI


SINOPSIS
       #include <signal.h>


       void (*signal(int signum, void (*manejador)(int)))(int);


DESCRIPCIÓN
       La llamada al sistema signal instala un nuevo manejador de
       señal para la señal cuyo número es signum.   El  manejador
       de  señal  se  establece como manejador, que puede ser una
       función especificada por el usuario, o una  de  las  sigu-
       ientes macros:

              SIG_IGN
                     No tener en cuenta la señal.

              SIG_DFL
                     Dejar  la  señal  con su comportamiento pre-
                     definido.

       El argumento entero que se pasa a la rutina de  manejo  de
       señal  es el número de la señal. Esto hace posible emplear
       un mismo manejador de señal para varias de ellas.

       Los manejadores de señales son rutinas que  se  llaman  en
       cualquier  momento  en  el  que el proceso recibe la señal
       correspondiente. Usando la función alarm(2), que envia una
       señal  SIGALRM  al  proceso, es posible manejar fácilmente
       trabajos regulares.  A un  proceso  también  se  le  puede
       decir  que  relea  sus ficheros de configuración usando un
       manejador de señal (normalmente, la señal es SIGHUP).


VALOR DEVUELTO
       signal devuelve el valor anterior del manejador de  señal,
       o SIG_ERR si ocurre un error.


OBSERVACIONES
       No se pueden instalar manejadores para las señales SIGKILL
       ni SIGSTOP.
```

Desde libc6, signal usa la semántica BSD y el comportamiento por defecto es no reasignar una señal a su comportamiento por defecto. Puede usar sysv_signal para obtener la semántica SysV.

Ambas, signal and sysv_signal son rutinas de biblioteca construidas sobre sigaction(2).

Si usted no entiende bien el prototipo del principio de esta página, puede ayudar el verlo separado así:

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t manejador);
```

De acuerdo con POSIX, el comportamiento de un proceso es indefinido después de no hacer caso a una señal SIGFPE, SIGILL o SIGSEGV que no ha sido generada por las funciones kill(2) ni raise(2). La división entera por cero tiene un resultado indefinido. En algunas arquitecturas generará una señal SIGFPE. (También, el dividir el entero más negativo por -1 puede generar SIGFPE.) No hacer caso a esta señal puede conducir a un bucle infinito.

De acuerdo con POSIX (B.3.3.1.3) no debe asignar SIG_IGN como acción para SIGCHLD. Aquí los comportamiento de BSD y SYSV difieren, provocando el fallo en Linux de aquellos programas BSD que asignan SIG_IGN como acción para SIGCHLD.

CONFORME A

C ANSI

VÉASE TAMBIÉN

kill(1), kill(2), killpg(2), pause(2), raise(3), sigaction(2), signal(7), sigsetops(3), sigvec(2), alarm(2).

ERRNO

ERRNO(3) Funciones de biblioteca ERRNO(3)

NOMBRE

errno - número del último error

SINOPSIS

```
#include <errno.h>

extern int errno;
```

DESCRIPCIÓN

Algunas llamadas al sistema (y algunas funciones de biblioteca) dan un valor al entero errno para indicar qué ha ido mal. Su valor sólo es significativo cuando la llamada devolvió un error (usualmente -1) y cuando una función de biblioteca que acabe bien tenga permitido cambiar errno.

Algunas veces, cuando una función tambíen puede devolver -1 como valor válido, se debe poner errno a cero antes de la llamada, para poder detectar posibles errores.

errno se define en el estándar ISO de C como un lvalor

(valor de lado izquierdo) modificable de tipo int y no se debe declarar de forma explícita; errno puede ser una macro. errno es "local en hilo"; asignarle un valor en un hilo no afecta al valor que tenga en ningún otro hilo.

Los números de error válidos son todos no cero; ninguna función de biblioteca asigna nunca un valor cero a errno. Todos los nombres de error especificados por POSIX.1 deben tener valores distintos.

POSIX.1 (edición de 1996) lista los siguientes nombres de error simbólicos. De éstos, EDOM y ERANGE existen en el estándar ISO de C. La enmienda 1 del ISO C define el número de error adicional EILSEQ para codificar los errores en múltiples bytes o caracteres extendidos.

E2BIG Lista de argumentos demasiado larga

EACCES Permiso denegado

EAGAIN Recurso temporalmente no disponible

EBADF Descriptor de fichero incorrecto

EBADMSG
Mensaje incorrecto

EBUSY Recurso ocupado

ECANCELED
Operación cancelada

ECHILD No hay procesos hijos

EDEADLK
Interbloqueo de recurso evitado

EDOM Error de dominio

EEXIST El fichero existe

EFAULT Dirección incorrecta

EFBIG Fichero demasiado grande

EINPROGRESS
Operación en progreso

EINTR Llamada a función interrumpida

EINVAL Argumento inválido

EIO Error de Entrada/Salida

EISDIR Es un directorio

EMFILE Demasiados ficheros abiertos

EMLINK Demasiados enlaces

EMSGSIZE
Longitud de buffer de mensaje inapropiada

ENAMETOOLONG
Nombre de fichero demasiado largo

ENFILE Demasiados ficheros abiertos en el sistema

ENODEV No existe tal dispositivo

ENOENT No existe ese fichero o directorio

ENOEXEC
Error en el formato del ejecutable

ENOLCK No hay bloqueos disponibles

ENOMEM No hay bastante espacio

ENOSPC No queda espacio en el dispositivo

ENOSYS Función no implementada

ENOTDIR
No es un directorio

ENOTEMPTY
El directorio no está vacío

ENOTSUP
Operación no soportada

ENOTTY Operación de control de E/S inapropiada

ENXIO No existe tal dispositivo o dirección

EPERM Operación no permitida

EPIPE Interconexión rota

ERANGE Resultado demasiado grande

EROFS Sistema de ficheros de sólo lectura

ESPIPE Posicionamiento inválido

ESRCH No existe tal proceso

ETIMEDOUT
La operación ha excedido su plazo de tiempo

EXDEV Enlace inapropiado

Otras implementaciones de Unix devuelven muchos otros tipos de error. System V devuelve ETXTBSY (fichero de código ocupado) si se intenta ejecutar una llamada exec() sobre un fichero que actualmente está abierto para escritura. Linux también devuelve este error si se intenta tener un fichero tanto asociado en memoria con VM_DENYWRITE como abierto para escritura.

VÉASE TAMBIÉN
perror(3), strerror(3)