87.1 os32: _Exit(2)

Vedere *_exit(2)* [87.2].

87.2 os32: _exit(2)

NOME

'**_exit**', '**_Exit**' - conclusione del processo chiamante

SINTASSI

```
#include <unistd.h>
void _exit (int status);
```

```
#include <stdlib.h>
void _Exit (int status);
```

DESCRIZIONE

Le funzioni ***_exit()*** e ***_Exit()*** sono equivalenti e servono per concludere il processo chiamante, con un valore pari a quello indicato come argomento (***status***), purché inferiore o uguale 255 (FF_{16}).

La conclusione del processo implica anche la chiusura dei suoi file aperti, e l'affidamento di eventuali processi figli al processo numero uno ('**init**'); inoltre, si ottiene l'invio di un segnale SIGCHLD al processo genitore di quello che viene concluso.

VALORE RESTITUITO

La funzione non può restituire alcunché, dal momento che la sua esecuzione comporta la morte del processo.

FILE SORGENTI

‘`lib/unistd.h`’ [95.30]

‘`lib/unistd/_exit.c`’ [95.30.1]

‘`lib/stdlib.h`’ [95.19]

‘`lib/stdlib/_Exit.c`’ [95.19.1]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

‘`kernel/lib_s/s__exit.c`’ [94.8.1]

VEDERE ANCHE

execve(2) [87.14], *fork(2)* [87.19], *kill(2)* [87.29], *wait(2)* [87.63], *atexit(3)* [88.7], *exit(3)* [88.7].

87.3 os32: accept(2)

«

NOME

‘**`accept`**’ - accetta una richiesta di connessione in un socket

SINTASSI

```
#include <sys/socket.h>
int accept (int sfdn, struct sockaddr *addr,
            socklen_t *addrlen);
```

DESCRIZIONE

La funzione di sistema ***accept()*** viene usata in os32 solo in relazione a socket di tipo ***SOCK_STREAM***, per il protocollo TCP, in ascolto in attesa di connessione. Se il descrittore ***sfdn*** corrisponde a queste caratteristiche, la funzione estrae la prima richiesta in attesa, crea una nuova connessione e restituisce il numero del nuovo descrittore relativo.

Perché la funzione possa svolgere il proprio compito, è necessario che ***sfdn*** sia il descrittore di un socket TCP creato con la funzione ***socket()*** [87.54], collegato a una porta locale con la funzione ***bind()*** [87.4] e in ascolto di richieste di connessione attraverso la funzione ***listen()*** [87.31].

Per poter utilizzare la funzione ***accept()*** è necessario predisporre una memoria tampone in cui collocare una variabile strutturata di tipo '`struct sockaddr`', a cui punta ***addr***. Questa struttura viene popolata da ***accept()*** con l'indirizzo della controparte che richiede di connettersi (il tipo di indirizzo, l'indirizzo IPv4 e la porta).

Il parametro ***addrlen*** deve puntare a una variabile che contiene inizialmente la dimensione massima di ****addr***; la funzione ***accept()*** si limita a utilizzare al massimo lo spazio così indicato, ma il valore di ****addrlen*** viene comunque aggiornato alla dimensione effettiva che ha o che dovrebbe avere la struttura completa. Pertanto, al termine del funzionamento di ***accept()***, se il valore che ****addrlen*** dovesse essere maggiore di quello indicato inizialmente, vuol dire che la struttura ottenuta è incompleta.

Se si usa ***accept()*** quando non ci sono richieste di connessione in attesa, questa blocca il funzionamento del processo chiamante,

fino a che si ottiene effettivamente una richiesta. Tuttavia, prima di usare la funzione è possibile attribuire al descrittore l'opzione ***O_NONBLOCK*** e in tal modo ottenere che la funzione termini ugualmente il proprio lavoro restituendo un errore di tipo ***EAGAIN***.

VALORE RESTITUITO

In caso di successo la funzione restituisce un valore non negativo, corrispondente al descrittore del socket creato per la nuova connessione. In presenza di errori, la funzione restituisce −1 e aggiorna la variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il descrittore ***sfdn*** non è valido; oppure, la richiesta in coda non corrisponde a un descrittore di file.
`ENOTSOCK`	Il descrittore ***sfdn*** non è un socket; oppure, la richiesta in coda non corrisponde a un socket.
`EOPNOTSUPP`	Il descrittore ***sfdn*** non è un socket di tipo ***SOCK_STREAM***, oppure il protocollo relativo non è di tipo ***IPPROTO_TCP***.
`EINVAL`	Il descrittore ***sfdn*** non è un socket in ascolto (la connessione non è nello stato ***TCP_LISTEN***).
`EAGAIN`	Non ci sono richieste di connessione in coda, ma è possibile ritentare.

FILE SORGENTI

'`lib/sys/socket.h`' [95.23]

'`lib/sys/socket/accept.c`' [95.23.1]

‘kernel/lib_s/s_accept.c’ [94.8.2]

VEDERE ANCHE

bind(2) [87.4], *connect(2)* [87.11], *listen(2)* [87.31], *socket(2)* [87.54].

87.4 os32: bind(2)

NOME

‘**bind**’ - associa un indirizzo e una porta locali a un socket

SINTASSI

```
#include <sys/socket.h>
int bind (int sfdn, const struct sockaddr *addr,
          socklen_t addrlen);
```

DESCRIZIONE

Dopo che un socket è stato creato con l’ausilio della funzione ***socket()*** [87.54], questo contiene soltanto l’informazione del tipo (nel caso di os32 può trattarsi soltanto di ***AF_INET***), senza indirizzo e porta. Per attribuire tali informazioni, riferite al lato locale della connessione, si può usare la funzione ***bind()***.

Per os32 la variabile strutturata che fa capo a ***addr** può contenere solo informazioni relative a IPv4.

VALORE RESTITUITO

In caso di successo la funzione restituisce zero; in caso di errore si ottiene invece −1 e l’aggiornamento della variabile ***errno***.

ERRORI

Valore di *errno*	Significato
EBADF	Il descrittore ***sfdn*** non è valido.
ENOTSOCK	Il descrittore ***sfdn*** non è un socket.
EINVAL	I dati contenuti in ****addr*** non sono validi.
EADDRNOTAVAIL	All'interno di ****addr*** manca l'indicazione della porta locale.
EACCES	Si sta tentando di aprire una porta locale minore di 1024, disponendo però di un UID efficace diverso da zero.
EAFNOSUPPORT	Il tipo di indirizzamento del socket è diverso da ***AF_INET***.
EOPNOTSUPP	Il descrittore ***sfdn*** non è un socket di tipo ***SOCK_STREAM***, oppure il protocollo relativo non è di tipo ***IPPROTO_TCP*** e nemmeno ***IPPROTO_UDP***.
EAGAIN	Non ci sono porte disponibili.

FILE SORGENTI

'lib/sys/socket.h' [95.23]

'lib/sys/socket/bind.c' [95.23.2]

'kernel/lib_s/s_bind.c' [94.8.3]

VEDERE ANCHE

accept(2) [87.3], *connect(2)* [87.11], *listen(2)* [87.31], *socket(2)* [87.54].

87.5 os32: brk(2)

«

NOME

'**brk**', '**sbrk**' - modifica della dimensione del segmento dati

SINTASSI

```
#include <unistd.h>
int brk (void *address);
void *sbrk (intptr_t increment);
```

DESCRIZIONE

Le funzione ***brk()*** e ***sbrk()*** permettono di modificare la dimensione del segmento dati del processo, generalmente per aumentarlo. Per os32, l'area di memoria che può essere espansa o contratta, si trova dopo la pila dei dati. L'incremento del segmento dati usato da un processo implica generalmente la copia dello stesso in una nuova posizione.

La funzione ***brk()*** permette di richiedere di fissare la fine del segmento dati in corrispondenza dell'indirizzo fornito come argomento, in qualità di puntatore generico.

La funzione ***sbrk()*** permette di richiedere la modifica del segmento dati specificandone l'incremento e restituendo il valore precedente della fine del segmento dati, in forma di puntatore. In tal modo, la funzione ***sbrk()*** può essere usata anche solo per conoscere la fine attuale del segmento dati, fornendo come argomento un incremento pari a zero.

VALORE RESTITUITO

La funzione ***brk()*** restituisce zero se l'operazione richiesta è completata con successo, diversamente restituisce −1, aggiornando la variabile ***errno***.

La funzione ***sbrk()*** restituisce un puntatore valido, se l'operazione si conclude con successo, diversamente si ottiene l'equivalente del valore −1, tradotto in forma di puntatore generico, con il contestuale aggiornamento della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`ENOMEM`	Memoria insufficiente per completare la richiesta.
`EINVAL`	L'argomento fornito non è valido.

NOTE

Le funzioni ***brk()*** e ***sbrk()*** servono per l'allocazione dinamica della memoria attraverso ***malloc()***, ***free()*** e ***realloc()***. Pertanto, è meglio avvalersi di queste ultime, piuttosto di rischiare di mettere in conflitto le due cose.

FILE SORGENTI

'`lib/unistd.h`' [95.30]

'`lib/unistd/brk.c`' [95.30.3]

'`lib/unistd/sbrk.c`' [95.30.34]

'`lib/sys/os32/sys.s`' [95.21.7]

'`kernel/ibm_i386/isr.s`' [94.6.21]

'`kernel/proc/sysroutine.c`' [94.14.28]

‘kernel/lib_s/s_brk.c’ [94.8.4]

‘kernel/lib_s/s_sbrk.c’ [94.8.33]

VEDERE ANCHE

free(3) [88.76], *malloc(3)* [88.76], *realloc(3)* [88.76].

87.6 os32: chdir(2)

«

NOME

‘**chdir**’, ‘**fchdir**’ - modifica della directory corrente

SINTASSI

```
#include <unistd.h>
int chdir (const char *path);
int chdir (int fdn);
```

DESCRIZIONE

La funzione ***chdir()*** cambia la directory corrente, in modo che quella nuova corrisponda al percorso annotato nella stringa ***path***. La funzione ***fchdir()*** dovrebbe svolgere lo stesso compito, indicando la nuova directory come descrittore già aperto; tuttavia os32 gestisce le directory esclusivamente in forma di percorso.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di *errno*	Significato
`EINVAL`	Argomento non valido.
`EACCES`	Accesso negato.
`ENOTDIR`	Uno dei componenti del percorso non è una directory.
`ENOENT`	Uno dei componenti del percorso non esiste.
`E_NOT_IMPLEMENTED`	La funzione ***fchdir()*** di os32 non può essere realizzata, pertanto risponde con questo errore.

FILE SORGENTI

‘`lib/unistd.h`’ [95.30]

‘`lib/unistd/chdir.c`’ [95.30.4]

‘`lib/unistd/fchdir.c`’ [95.30.16]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

‘`kernel/lib_s/s_chdir.c`’ [94.8.5]

VEDERE ANCHE

rmdir(2) [87.41], *access(3)* [88.4].

87.7 os32: chmod(2)

«

NOME

‘**chmod**’, ‘**fchmod**’ - cambiamento della modalità dei permessi di un file

SINTASSI

```
#include <sys/stat.h>
int chmod  (const char *path, mode_t mode);
int fchmod (int fdn, mode_t mode);
```

DESCRIZIONE

Le funzioni ***chmod()*** e ***fchmod()*** consentono di modificare la modalità dei permessi di accesso di un file. La funzione ***chmod()*** individua il file su cui intervenire attraverso un percorso, ovvero la stringa ***path***; la funzione ***fchmod()***, invece, richiede l’indicazione del numero di un descrittore di file, già aperto. In entrambi i casi, l’ultimo argomento serve a specificare la nuova modalità dei permessi.

Tradizionalmente, i permessi si scrivono attraverso un numero in base otto; in alternativa, si possono usare convenientemente della macro-variabili, dichiarate nel file ‘`sys/stat.h`’, combinate assieme con l’operatore binario OR.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXU`	00700_8	Lettura, scrittura ed esecuzione per l'utente proprietario.
`S_IRUSR`	00400_8	Lettura per l'utente proprietario.
`S_IWUSR`	00200_8	Scrittura per l'utente proprietario.
`S_IXUSR`	00100_8	Esecuzione per l'utente proprietario.
`S_IRWXG`	00070_8	Lettura, scrittura ed esecuzione per il gruppo.
`S_IRGRP`	00040_8	Lettura per il gruppo.
`S_IWGRP`	00020_8	Scrittura per il gruppo.
`S_IXGRP`	00010_8	Esecuzione per il gruppo.
`S_IRWXO`	00007_8	Lettura, scrittura ed esecuzione per gli altri utenti.
`S_IROTH`	00004_8	Lettura per gli altri utenti.
`S_IWOTH`	00002_8	Scrittura per gli altri utenti.
`S_IXOTH`	00001_8	Esecuzione per gli altri utenti.

os32 non considera i permessi SUID (*Set user id*), SGID (*Set group id*) e Sticky, che nella tabella non sono stati nemmeno annotati; inoltre, non tiene in considerazione i permessi legati al gruppo.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EACCES`	Permesso negato.
`EBADF`	Il descrittore del file richiesto non è valido.

FILE SORGENTI

‘`lib/sys/stat.h`’ [95.25]

‘`lib/sys/stat/chmod.c`’ [95.25.1]

‘`lib/sys/stat/fchmod.c`’ [95.25.2]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

‘`kernel/lib_s/s_chmod.c`’ [94.8.6]

‘`kernel/lib_s/s_fchmod.c`’ [94.8.13]

VEDERE ANCHE

chmod(1) [86.7], *chown(2)* [87.8], *open(2)* [87.37], *stat(2)* [87.55].

87.8 os32: chown(2)

«

NOME

‘**chown**’, ‘**fchown**’ - modifica della proprietà dei file

SINTASSI

```
#include <unistd.h>
int chown  (const char *path, uid_t uid, gid_t gid);
int fchown (int fdn, uid_t uid, gid_t gid);
```

DESCRIZIONE

Le funzioni ***chown()*** e ***fchown()***, modificano la proprietà di un file, fornendo il numero UID e il numero GID. Il file viene indicato, rispettivamente, attraverso il percorso scritto in una stringa, oppure come numero di descrittore già aperto.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di *errno*	Significato
`EINVAL`	Argomento non valido.
`EPERM`	Permessi insufficienti per eseguire l'operazione.
`ENOTDIR`	Uno dei componenti del percorso non è una directory.
`ENOENT`	Uno dei componenti del percorso non esiste.
`EBADF`	Il descrittore del file non è valido.

DIFETTI

Le funzioni consentono di attribuire il numero del gruppo, ma os32 non valuta i permessi di accesso ai file, relativi ai gruppi.

FILE SORGENTI

'`lib/unistd.h`' [95.30]

'`lib/unistd/chown.c`' [95.30.5]

'`lib/sys/os32/sys.s`' [95.21.7]

'`kernel/ibm_i386/isr.s`' [94.6.21]

'`kernel/proc/sysroutine.c`' [94.14.28]

'`kernel/lib_s/s_chown.c`' [94.8.7]

'`kernel/lib_s/s_fchown.c`' [94.8.14]

VEDERE ANCHE

chmod(2) [87.7].

87.9 os32: clock(2)

«

NOME

‘**clock**’ - tempo della CPU espresso in unità ‘**clock_t**’

SINTASSI

```
#include <time.h>
clock_t clock (void);
```

DESCRIZIONE

La funzione ***clock()*** restituisce il tempo di utilizzo della CPU, espresso in unità ‘**clock_t**’, corrispondenti a secondi diviso il valore della macro-variabile ***CLOCKS_PER_SEC***. Per os32, come dichiarato nel file ‘`time.h`’, il valore di ***CLOCKS_PER_SEC*** è 100, essendo la frequenza del temporizzatore interno regolata a 100 Hz.

VALORE RESTITUITO

La funzione restituisce il tempo di CPU, espresso in centesimi di secondo.

FILE SORGENTI

‘`lib/time.h`’ [95.29]

‘`lib/time/clock.c`’ [95.29.2]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

‘`kernel/lib_s/s_clock.c`’ [94.8.8]

VEDERE ANCHE

time(2) [87.59].

87.10 os32: close(2)

NOME

'**close**' - chiusura di un descrittore di file

SINTASSI

```
#include <unistd.h>
int close (int fdn);
```

DESCRIZIONE

Le funzioni ***close()*** chiude un descrittore di file.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
EBADF	Il descrittore del file non è valido.

FILE SORGENTI

'lib/unistd.h' [95.30]

'lib/unistd/close.c' [95.30.6]

'lib/sys/os32/sys.s' [95.21.7]

'kernel/ibm_i386/isr.s' [94.6.21]

‘kernel/proc/sysroutine.c’ [94.14.28]

‘kernel/lib_s/s_close.c’ [94.8.9]

VEDERE ANCHE

fcntl(2) [87.18], *open(2)* [87.37], *fclose(3)* [88.28].

87.11 os32: connect(2)

«

NOME

‘**connect**’ - inizia una connessione su un socket

SINTASSI

```
#include <sys/socket.h>
int connect (int sfdn, const struct sockaddr *addr,
             socklen_t addrlen);
```

DESCRIZIONE

Un socket che sia già stato associato a una porta locale, può essere collegato a un socket remoto attraverso la funzione ***connect()***. Si distinguono due casi: se il protocollo è «connesso» (***IPPROTO_TCP***), questo procedimento può essere eseguito una volta sola, se invece non lo è (***IPPROTO_UDP***), ci si può connettere successivamente a socket remoti differenti.

Si intende, pertanto, che ***addr** si riferisca all’indirizzo del socket remoto.

VALORE RESTITUITO

In caso di successo la funzione restituisce zero; in caso di errore si ottiene invece −1 e l’aggiornamento della variabile ***errno***.

ERRORI

Valore di *errno*	Significato
`EBADF`	Il descrittore ***sfdn*** non è valido.
`ENOTSOCK`	Il descrittore ***sfdn*** non è un socket.
`EINVAL`	I dati contenuti in ****addr*** non sono validi.
`EADDRNOTAVAIL`	All'interno di ****addr*** manca l'indicazione dell'indirizzo o della porta da raggiungere.
`EAGAIN`	La porta locale del socket ***sfdn*** non risulta specificata, e inoltre non è possibile attribuirne una in modo automatico.
`EISCONN`	Esiste già una connessione TCP in corso con il socket ***sfdn*** e si sta tentando di cambiarne i parametri.
`EAFNOSUPPORT`	Il tipo di indirizzamento del socket ***sfdn*** è diverso da ***AF_INET***.

FILE SORGENTI

'`lib/sys/socket.h`' [95.23]

'`lib/sys/socket/connect.c`' [95.23.3]

'`kernel/lib_s/s_connect.c`' [94.8.10]

VEDERE ANCHE

accept(2) [87.3], *bind(2)* [87.4], *listen(2)* [87.31], *socket(2)* [87.54].

87.12 os32: dup(2)

«

NOME

‘**dup**’, ‘**dup2**’ - duplicazione di descrittori di file

SINTASSI

```
#include <unistd.h>
int dup  (int fdn_old);
int dup2 (int fdn_old, int fdn_new);
```

DESCRIZIONE

Le funzioni ***dup()*** e ***dup2()*** servono a duplicare un descrittore di file. La funzione ***dup()*** duplica il descrittore ***fdn_old***, utilizzando il numero di descrittore libero più basso che sia disponibile; ***dup2()***, invece, richiede che il nuovo numero di descrittore sia specificato, attraverso il parametro ***fdn_new***. Tuttavia, se il numero di descrittore ***fdn_new*** risulta utilizzato, questo viene chiuso prima di diventare la copia di ***fdn_old***.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Uno dei descrittori specificati non è valido.
`EMFILE`	Troppi file aperti per il processo.

FILE SORGENTI

‘lib/unistd.h’ [95.30]

‘lib/unistd/dup.c’ [95.30.7]

‘lib/unistd/dup2.c’ [95.30.8]

‘lib/sys/os32/sys.s’ [95.21.7]

‘kernel/ibm_i386/isr.s’ [94.6.21]

‘kernel/proc/sysroutine.c’ [94.14.28]

‘kernel/fs/fd_dup.c’ [94.5.1]

‘kernel/lib_s/s_dup.c’ [94.8.11]

‘kernel/lib_s/s_dup2.c’ [94.8.12]

VEDERE ANCHE

close(2) [87.10], *fcntl(2)* [87.18], *open(2)* [87.37].

87.13 os32: dup2(2)

Vedere *dup(2)* [87.12].

87.14 os32: execve(2)

NOME

‘**execve**’ - esecuzione di un file

SINTASSI

```
#include <unistd.h>
int execve (const char *path, char *const argv[],
            char *const envp[]);
```

DESCRIZIONE

La funzione ***execve()*** è quella che avvia effettivamente un programma, mentre le altre funzioni '**exec**...**()**' offrono semplicemente un'interfaccia differente per l'avvio, ma poi si avvalgono di ***execve()*** per svolgere effettivamente quanto loro richiesto.

La funzione ***execve()*** avvia il file il cui percorso è specificato come stringa, nel primo argomento.

Il secondo argomento deve essere un array di stringhe, dove la prima deve rappresentare il nome del programma avviato e le successive sono gli argomenti da passare al programma. L'ultimo elemento di tale array deve essere un puntatore nullo, per poter essere riconosciuto.

Il terzo argomento deve essere un array di stringhe, rappresentanti l'ambiente da passare al nuovo processo. Per ambiente si intende l'insieme delle variabili di ambiente, pertanto queste stringhe devono avere la forma '***nome=valore***', per essere riconoscibili. Anche in questo caso, per poter individuare l'ultimo elemento dell'array, questo deve essere un puntatore nullo.

VALORE RESTITUITO

Se ***execve()*** riesce nel suo compito, non può restituire alcunché, dato che in quel momento, il processo chiamante viene rimpiazzato da quello del file che viene eseguito. Pertanto, se viene restituito qualcosa, può trattarsi solo di un valore che rappresenta un errore, ovvero -1, aggiornando anche la variabile ***errno*** di conseguenza.

ERRORI

Valore di *errno*	Significato
`E2BIG`	Ci sono troppi argomenti.
`ENOMEM`	Memoria insufficiente.
`ENOENT`	Il file richiesto non esiste.
`EACCES`	Il file non può essere avviato per la mancanza dei permessi di acceso necessari.
`ENOEXEC`	Il file non può essere un file eseguibile, perché non ne ha le caratteristiche.
`EIO`	Errore di input-output.

DIFETTI

os32 non prevede l'interpretazione di script, perché non esiste alcun programma in grado di farlo. Anche la shell di os32 si limita a eseguire i comandi inseriti, ma non può interpretare un file.

FILE SORGENTI

'`lib/unistd.h`' [95.30]

'`lib/unistd/execle.c`' [95.30.11]

'`lib/sys/os32/sys.s`' [95.21.7]

'`kernel/ibm_i386/isr.s`' [94.6.21]

'`kernel/proc/sysroutine.c`' [94.14.28]

'`kernel/proc/proc_sys_exec.c`' [94.14.22]

VEDERE ANCHE

fork(2) [87.19], *exec(3)* [88.21], *getopt(3)* [88.56], *environ(7)* [91.1].

87.15 os32: fchdir(2)

«

Vedere *chdir(2)* [87.6].

87.16 os32: fchmod(2)

«

Vedere *chmod(2)* [87.7].

87.17 os32: fchown(2)

«

Vedere *chown(2)* [87.8].

87.18 os32: fcntl(2)

«

NOME

‘**`fcntl`**’ - configurazione e intervento sui descrittori di file

SINTASSI

```
#include <fcntl.h>
int fcntl (int fdn, int cmd, ...);
```

DESCRIZIONE

La funzione ***fcntl()*** esegue un’operazione, definita dal parametro ***cmd***, sul descrittore richiesto come primo parametro (***fdn***). A seconda del tipo di operazione richiesta, potrebbero essere necessari degli argomenti ulteriori, i quali però non possono essere

formalizzati in modo esatto nel prototipo della funzione. Il valore del secondo parametro che rappresenta l'operazione richiesta, va fornito in forma di costante simbolica, come descritto nell'elenco seguente.

Sintassi	Descrizione
`fcntl (`***fdn***`, F_DUPFD, (int)` ***fdn_min***`)`	Richiede la duplicazione del descrittore di file ***fdn***, in modo tale che la copia abbia il numero di descrittore minore possibile, ma maggiore o uguale a quello indicato come argomento ***fdn_min***.
`fcntl (`***fdn***`, F_GETFD)` `fcntl (`***fdn***`, F_SETFD, (int)` ***fd_flags***`)`	Rispettivamente, legge o imposta, gli indicatori del descrittore di file ***fdn*** (eventualmente noti come *file descriptor flags* o solo *fd flags*). Per il momento, è possibile impostare un solo indicatore, '**`FD_CLOEXEC`**', pertanto, al posto di ***fd_flags*** si può mettere solo la costante '**`FD_CLOEXEC`**'.
`fcntl (`***fdn***`, F_GETFL)` `fcntl (`***fdn***`, F_SETFL, (int)` ***fl_flags***`)`	Rispettivamente, legge o imposta, gli indicatori dello stato del file, relativi al descrittore ***fdn*** (eventualmente noti come *file flaga* o solo *fl flags*). Per impostare questi indicatori, vanno combinate delle costanti simboliche: '**`O_RDONLY`**', '**`O_WRONLY`**', '**`O_RDWR`**', '**`O_CREAT`**', '**`O_EXCL`**', '**`O_NOCTTY`**', '**`O_TRUNC`**'.

VALORE RESTITUITO

Il significato del valore restituito dalla funzione dipende dal tipo di operazione richiesta, come sintetizzato dalla tabella successiva.

Operazione richiesta	Significato del valore restituito
`F_DUPFD`	Si ottiene il numero del descrittore prodotto dalla copia, oppure −1 in caso di errore.
`F_GETFD`	Si ottiene il valore degli indicatori del descrittore (*fd flags*), oppure −1 in caso di errore.
`F_GETFL`	Si ottiene il valore degli indicatori del file (*fl flags*), oppure −1 in caso di errore.

`F_GETOWN`
`F_SETOWN`
`F_GETLK`
`F_SETLK`
`F_SETLKW`
	Si ottiene −1, in quanto si tratta di operazioni non realizzate in questa versione della funzione, per os32.

altri tipi di operazione	Si ottiene 0 in caso di successo, oppure −1 in caso di errore.

ERRORI

Valore di ***errno***	Significato
`E_NOT_IMPLEMENTED`	È stato richiesto un tipo di operazione che non è disponibile nel caso particolare di os32.
`EINVAL`	È stato richiesto un tipo di operazione non valido.

FILE SORGENTI

‘`lib/fcntl.h`’ [95.6]

‘`lib/fcntl/fcntl.c`’ [95.6.2]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘kernel/ibm_i386/isr.s’ [94.6.21]

‘kernel/proc/sysroutine.c’ [94.14.28]

‘kernel/lib_s/s_fcntl.c’ [94.8.15]

VEDERE ANCHE

dup(2) [87.12], *dup2(2)* [87.12], *open(2)* [87.37].

87.19 os32: fork(2)

«

NOME

‘**fork**’ - sdoppiamento di un processo, ovvero creazione di un processo figlio

SINTASSI

```
#include <unistd.h>
pid_t fork (void);
```

DESCRIZIONE

La funzione ***fork()*** crea una copia del processo in corso, la quale copia diventa un processo figlio del primo. Il processo figlio eredita una copia dei descrittori di file aperti e di conseguenza dei flussi di file e directory.

Il processo genitore riceve dalla funzione il valore del numero PID del processo figlio avviato; il processo figlio si mette a funzionare dal punto in cui si trova la funzione ***fork()***, restituendo però un valore nullo: in questo modo tale processo figlio può riconoscersi come tale.

VALORE RESTITUITO

La funzione restituisce al processo genitore il numero PID del processo figlio; al processo figlio restituisce zero. In caso di problemi, invece, il valore restituito è −1 e la variabile ***errno*** risulta aggiornata di conseguenza.

ERRORI

Valore di ***errno***	Significato
`ENOMEM`	Memoria insufficiente per avviare un altro processo.

FILE SORGENTI

‘`lib/unistd.h`’ [95.30]

‘`lib/unistd/fork.c`’ [95.30.18]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

‘`kernel/lib_s/s_fork.c`’ [94.8.16]

VEDERE ANCHE

execve(2) [87.14], *wait(2)* [87.63], *exec(3)* [88.21].

87.20 os32: fstat(2)

Vedere *stat(2)* [87.55].

87.21 os32: getcwd(2)

«

NOME

'**getcwd**' - determinazione della directory corrente

SINTASSI

```
#include <unistd.h>
char *getcwd (char *buffer, size_t size);
```

DESCRIZIONE

La funzione ***getcwd()*** modifica il contenuto dell'area di memoria a cui punta ***buffer***, copiandovi al suo interno la stringa che rappresenta il percorso della directory corrente. La scrittura all'interno di ***buffer*** può prolungarsi al massimo per ***size*** byte, incluso il codice nullo di terminazione delle stringhe.

VALORE RESTITUITO

La funzione restituisce il puntatore alla stringa che rappresenta il percorso della directory corrente, il quale deve coincidere con ***buffer***. In caso di errore, invece, la funzione restituisce il puntatore nullo '**NULL**'.

ERRORI

Valore di ***errno***	Significato
EINVAL	Il puntatore ***buffer*** non è valido.
E_LIMIT	Il percorso della directory corrente è troppo lungo, rispetto ai limiti realizzativi di os32.

DIFETTI

La funzione ***getcwd()*** di os32 deve comunicare con il kernel per ottenere l'informazione che le serve, perché la «u-area» (*User area*) è trattenuta all'interno del kernel stesso.

FILE SORGENTI

'`lib/unistd.h`' [95.30]

'`lib/unistd/getcwd.c`' [95.30.19]

'`lib/sys/os32/sys.s`' [95.21.7]

'`kernel/ibm_i386/isr.s`' [94.6.21]

'`kernel/proc/sysroutine.c`' [94.14.28]

VEDERE ANCHE

chdir(2) [87.6].

87.22 os32: getgid(2)

«

NOME

'**`getgid`**', '**`getegid`**' - determinazione del gruppo reale ed efficace

SINTASSI

```
#include <unistd.h>
gid_t getgid  (void);
gid_t getegid (void);
```

DESCRIZIONE

La funzione ***getgid()*** restituisce il numero corrispondente al gruppo reale a cui appartiene il processo; la funzione ***getegid()*** restituisce il numero del gruppo efficace del processo.

VALORE RESTITUITO

Il numero GID, reale o efficace del processo chiamante. Non sono previsti casi di errore.

DIFETTI

Le funzioni ***getgid()*** e ***getegid()*** di os32 devono comunicare con il kernel per ottenere l'informazione che a loro serve, perché la «u-area» (*User area*) è trattenuta all'interno del kernel stesso.

FILE SORGENTI

'`lib/unistd.h`' [95.30]

'`lib/unistd/getgid.c`' [95.30.22]

'`lib/unistd/getegid.c`' [95.30.20]

'`lib/sys/os32/sys.s`' [95.21.7]

'`kernel/ibm_i386/isr.s`' [94.6.21]

'`kernel/proc/sysroutine.c`' [94.14.28]

VEDERE ANCHE

setgid(2) [87.48], *getuid(2)* [87.27], *geteuid(2)* [87.27].

87.23 os32: geteuid(2)

«

Vedere *getuid(2)* [87.27].

87.24 os32: getpgrp(2)

«

Vedere *getpid(2)* [87.25].

87.25 os32: getpid(2)

NOME

‘**getpid**’, ‘**getppid**’, ‘**getpgrp**’ - determinazione del numero del processo o del gruppo di processi

SINTASSI

```
#include <unistd.h>
pid_t getpid  (void);
pid_t getppid (void);
pid_t getpgrp (void);
```

DESCRIZIONE

La funzione ***getpid()*** restituisce il numero del processo chiamante; la funzione ***getppid()*** restituisce il numero del processo genitore rispetto a quello chiamante; la funzione ***getpgrp()*** restituisce il numero attribuito al gruppo di processi a cui appartiene quello chiamante.

VALORE RESTITUITO

Il numero di processo o di gruppo di processi, relativo al contesto della funzione. Non sono previsti casi di errore.

DIFETTI

Le funzioni ***getpid()***, ***getppid()*** e ***getpgrp()*** di os32 devono comunicare con il kernel per ottenere l’informazione che a loro serve, perché la «u-area» (*User area*) è trattenuta all’interno del kernel stesso.

FILE SORGENTI

‘`lib/unistd.h`’ [95.30]

‘lib/unistd/getpid.c’ [95.30.25]

‘lib/unistd/getppid.c’ [95.30.26]

‘lib/unistd/getpgrp.c’ [95.30.24]

‘lib/sys/os32/sys.s’ [95.21.7]

‘kernel/ibm_i386/isr.s’ [94.6.21]

‘kernel/proc/sysroutine.c’ [94.14.28]

VEDERE ANCHE

getuid(2) [87.27] *fork(2)* [87.19], *execve(2)* [87.14].

87.26 os32: getppid(2)

«

Vedere *getpid(2)* [87.25].

87.27 os32: getuid(2)

«

NOME

‘**getuid**’, ‘**geteuid**’ - determinazione dell’identità reale ed efficace

SINTASSI

```
#include <unistd.h>
uid_t getuid  (void);
uid_t geteuid (void);
```

DESCRIZIONE

La funzione ***getuid()*** restituisce il numero corrispondente all’identità reale del processo; la funzione ***geteuid()*** restituisce il numero dell’identità efficace del processo.

VALORE RESTITUITO

Il numero UID, reale o efficace del processo chiamante. Non sono previsti casi di errore.

DIFETTI

Le funzioni ***getuid()*** e ***geteuid()*** di os32 devono comunicare con il kernel per ottenere l'informazione che a loro serve, perché la «u-area» (*User area*) è trattenuta all'interno del kernel stesso.

FILE SORGENTI

'`lib/unistd.h`' [95.30]

'`lib/unistd/getuid.c`' [95.30.27]

'`lib/unistd/geteuid.c`' [95.30.21]

'`lib/sys/os32/sys.s`' [95.21.7]

'`kernel/ibm_i386/isr.s`' [94.6.21]

'`kernel/proc/sysroutine.c`' [94.14.28]

VEDERE ANCHE

setuid(2) [87.51], *getgid(2)* [87.22], *getegid(2)* [87.22].

87.28 os32: ipconfig(2)

NOME

'**`ipconfig`**' - configurazione di un'interfaccia di rete con l'indirizzo IPv4 e la maschera di rete (funzione specifica di os32)

SINTASSI

```
#include <sys/os32.h>
int ipconfig (int n, in_addr_t address, int m);
```

DESCRIZIONE

La funzione di sistema ***ipconfig()***, specifica di os32, permette di configurare un'interfaccia di rete con il suo indirizzo IPv4 e la sua maschera di rete. Il primo parametro, ***n***, individua l'interfaccia di rete, nella tabella ***net_table[]***; pertanto, lo zero è riservato all'indirizzo locale virtuale (*loopback*), pari all'interfaccia '**`net0`**', mentre i valori successivi riguardano le interfacce Ethernet reali. L'ultimo parametro, ***m***, è la maschera di rete, espressa in quantità di bit; per esempio, il valore 16 corrisponderebbe alla maschera 255.255.0.0.

VALORE RESTITUITO

In caso di successo la funzione restituisce zero, altrimenti, in caso di errore, si ottiene −1 e l'aggiornamento della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EPERM`	È possibile usare la funzione soltanto con UID efficace pari a zero; diversamente si ottiene questo errore.
`EINVAL`	È stato richiesto un numero di interfaccia oltre i limiti della tabella ***net_table[]***.
`ENODEV`	È stata richiesta un'interfaccia inesistente.

FILE SORGENTI

'`lib/sys/os32.h`' [95.21]

'`lib/sys/os32/ipconfig.c`' [95.21.2]

'`kernel/lib_s/s_ipconfig.c`' [94.8.18]

VEDERE ANCHE

routeadd(2) [87.42], *routedel(2)* [87.43].

87.29 os32: kill(2)

NOME

'**kill**' - invio di un segnale a un processo

SINTASSI

```
#include <sys/types.h>
#include <signal.h>
int kill (pid_t pid, int sig)
```

DESCRIZIONE

La funzione ***kill()*** invia il segnale ***sig*** al processo numero ***pid***, oppure a un gruppo di processi. Questa realizzazione particolare di os32 comporta come segue:

- se il valore ***pid*** è maggiore di zero, il segnale viene inviato al processo con il numero ***pid***, ammesso di averne il permesso;
- se il valore ***pid*** è pari a zero, il segnale viene inviato a tutti i processi appartenenti allo stesso utente (quelli che hanno la stessa identità efficace, ovvero il valore ***euid***), ma se il processo che chiama la funzione lavora con un valore di ***euid*** pari a zero, il segnale viene inviato a tutti i processi, a partire dal numero due (si salta '**init**');
- valori negativi di ***pid*** non vengono presi in considerazione.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EPERM`	Il processo non ha i permessi per inviare il segnale alla destinazione richiesta.
`ESRCH`	La ricerca del processo ***pid*** è fallita. Nel caso di os32, si ottiene questo errore anche per valori negativi di ***pid***.

FILE SORGENTI

‘`lib/sys/types.h`’ [95.26]

‘`lib/signal.h`’ [95.17]

‘`lib/signal/kill.c`’ [95.17.2]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

‘`kernel/lib_s/s_kill.c`’ [94.8.19]

VEDERE ANCHE

signal(2) [87.52].

87.30 os32: link(2)

«

NOME

‘**`link`**’ - creazione di un collegamento fisico tra un file esistente e un altro nome

SINTASSI

```
#include <unistd.h>
int link (const char *path_old, const char *path_new);
```

DESCRIZIONE

La funzione ***link()*** produce un nuovo collegamento a un file già esistente. Va fornito il percorso del file già esistente, ***path_old*** e quello del file da creare, in qualità di collegamento, ***path_new***. L'operazione può avvenire soltanto se i due percorsi si trovano sulla stessa unità di memorizzazione e se ci sono i permessi di scrittura necessari nella directory di destinazione. Dopo l'operazione di collegamento, fatta in questo modo, non è possibile distinguere quale sia il file originale e quale sia invece il nome aggiunto.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di *errno*	Significato
`EINVAL`	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
`EPERM`	Operazione non consentita.
`EEXIST`	Il nome da creare esiste già.
`EACCES`	Accesso non consentito.
`ENOENT`	Il file non esiste, oppure non esiste il percorso che porta al file da creare.
`EROFS`	Il file system consente soltanto un accesso in lettura.
`ENOTDIR`	Uno dei due percorsi non è valido, in quanto ciò che dovrebbe essere una directory, non lo è.

FILE SORGENTI

‘`lib/unistd.h`’ [95.30]

‘`lib/unistd/link.c`’ [95.30.29]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

‘`kernel/lib_s/s_link.c`’ [94.8.20]

VEDERE ANCHE

ln(1) [86.13] *open(2)* [87.37], *stat(2)* [87.55], *unlink(2)* [87.62].

87.31 os32: listen(2)

«

NOME

'**listen**' - in ascolto attendendo una connessione verso un socket locale

SINTASSI

```
#include <sys/socket.h>
int listen (int sfdn, int backlog);
```

DESCRIZIONE

La funzione di sistema ***listen()*** viene usata per mettere un socket in attesa di connessione dall'esterno. Di norma, prima di usare questa funzione ci si avvale di ***bind()*** [87.4], per impostare le caratteristiche principali del socket.

Il parametro ***backlog*** serve a specificare il numero massimo di richieste di connessione che si possono accodare.

Per mettere in atto effettivamente una nuova connessione, partendo dalla prima richiesta disponibile, accodata da ***listen()***, si usa poi la funzione ***accept()*** [87.3].

VALORE RESTITUITO

In caso di successo la funzione restituisce zero; in presenza di errori restituisce invece -1 e aggiorna la variabile ***errno***.

ERRORI

Valore di *errno*	Significato
EBADF	Il descrittore ***sfdn*** non è valido.
ENOTSOCK	Il descrittore ***sfdn*** non è un socket.
EOPNOTSUPP	Il descrittore ***sfdn*** non è un socket di tipo ***SOCK_STREAM***.
EISCONN	Il descrittore ***sfdn*** corrisponde a un socket già connesso o in un altro stato che non può essere cambiato.
EADDRINUSE	Un altro socket in ascolto sta già utilizzando la stessa porta locale.

FILE SORGENTI

‘lib/sys/socket.h’ [95.23]

‘lib/sys/socket/listen.c’ [95.23.4]

‘kernel/lib_s/s_listen.c’ [94.8.21]

VEDERE ANCHE

bind(2) [87.4], *connect(2)* [87.11], *accept(2)* [87.3], *socket(2)* [87.54].

87.32 os32: longjmp(2)

«

Vedere *setjmp(2)* [87.49].

87.33 os32: lseek(2)

«

NOME

'**lseek**' - riposizionamento dell'indice di accesso a un descrittore di file

SINTASSI

```
#include <unistd.h>
off_t lseek (int fdn, off_t offset, int whence);
```

DESCRIZIONE

La funzione ***lseek()*** consente di riposizionare l'indice di accesso interno al descrittore di file ***fdn***. Per fare questo occorre prima determinare un punto di riferimento, rappresentato dal parametro ***whence***, dove va usata una macro-variabile definita nel file 'unistd.h'. Può trattarsi dei casi seguenti.

Valore di ***whence***	Significato
SEEK_SET	Lo scostamento si riferisce all'inizio del file.
SEEK_CUR	Lo scostamento si riferisce alla posizione che ha già l'indice interno al file.
SEEK_END	Lo scostamento si riferisce alla fine del file.

Lo scostamento indicato dal parametro ***offset*** si applica a partire dalla posizione a cui si riferisce ***whence***, pertanto può avere segno positivo o negativo, ma in ogni caso non è possibile collocare l'indice prima dell'inizio del file.

VALORE RESTITUITO

Se l'operazione avviene con successo, la funzione restituisce il valore dell'indice riposizionato, preso come scostamento a partire dall'inizio del file. In caso di errore, restituisce invece il valore −1, aggiornando di conseguenza anche la variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il numero del descrittore di file non è valido.
`EINVAL`	Il valore di ***whence*** non è contemplato, oppure la combinazione tra ***whence*** e ***offset*** non è valida.

FILE SORGENTI

'`lib/unistd.h`' [95.30]

'`lib/unistd/lseek.c`' [95.30.30]

'`lib/sys/os32/sys.s`' [95.21.7]

'`kernel/ibm_i386/isr.s`' [94.6.21]

'`kernel/proc/sysroutine.c`' [94.14.28]

'`kernel/lib_s/s_lseek.c`' [94.8.23]

VEDERE ANCHE

dup(2) [87.12] *fork(2)* [87.19], *open(2)* [87.37], *fseek(3)* [88.44].

87.34 os32: mkdir(2)

«

NOME

'**`mkdir`**' - creazione di una directory

SINTASSI

```
#include <sys/stat.h>
int mkdir (const char *path, mode_t mode);
```

DESCRIZIONE

La funzione ***mkdir()*** crea una directory, indicata attraverso un percorso, nel parametro ***path***, specificando la modalità dei permessi, con il parametro ***mode***.

Tuttavia, il valore del parametro ***mode*** non viene preso in considerazione integralmente: di questo si considerano solo gli ultimi nove bit, ovvero quelli dei permessi di utenti, gruppi e altri utenti; inoltre, vengono tolti i bit presenti nella maschera dei permessi associata al processo (si veda anche *umask(2)* [87.60]).

La directory che viene creata in questo modo, appartiene all'identità efficace del processo, ovvero all'utente per conto del quale questo sta funzionando.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di *errno*	Significato
`EINVAL`	Il percorso indicato non è valido.
`EEXIST`	Esiste già un file o una directory con lo stesso nome.
`ENOTDIR`	Una porzione del percorso della directory da creare, non è una directory.
`ENOENT`	Una porzione del percorso della directory da creare non esiste.
`EACCES`	Permesso negato.

FILE SORGENTI

‘`lib/sys/stat.h`’ [95.25]

‘`lib/sys/stat/mkdir.c`’ [95.25.4]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

‘`kernel/lib_s/s_mkdir.c`’ [94.8.24]

VEDERE ANCHE

mkdir(1) [86.17], *chmod(2)* [87.7], *chown(2)* [87.8], *mknod(2)* [87.35], *mount(2)* [87.36], *stat(2)* [87.55], *umask(2)* [87.60], *unlink(2)* [87.62].

87.35 os32: mknod(2)

«

NOME

'**mknod**' - creazione di un file vuoto di qualunque tipo

SINTASSI

```
#include <sys/stat.h>
int mknod (const char *path, mode_t mode, dev_t device);
```

DESCRIZIONE

La funzione ***mknod()*** crea un file vuoto, di qualunque tipo. Potenzialmente può creare anche una directory, ma priva di qualunque voce, rendendola così non adeguata al suo scopo (una directory richiede almeno le voci '`.`' e '`..`', per potersi considerare tale).

Il parametro ***path*** specifica il percorso del file da creare; il parametro ***mode*** serve a indicare il tipo di file da creare, oltre ai permessi comuni.

Il parametro ***device***, con il quale va indicato il numero di un dispositivo (completo di numero primario e secondario), viene preso in considerazione soltanto se nel parametro ***mode*** si richiede la creazione di un file di dispositivo a caratteri o a blocchi.

Il valore del parametro ***mode*** va costruito combinando assieme delle macro-variabili definite nel file '`sys/stat.h`', come descritto nella pagina di manuale *stat(2)* [87.55], tenendo conto che os32 non può gestire file FIFO, collegamenti simbolici e socket di dominio Unix.

Il valore del parametro ***mode***, per la porzione che riguarda i permessi di accesso al file, viene comunque filtrato con la maschera dei permessi (*umask(2)* [87.55]).

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il percorso indicato non è valido.
`EEXIST`	Esiste già un file o una directory con lo stesso nome.
`ENOTDIR`	Una porzione del percorso del file da creare, non è una directory.
`ENOENT`	Una porzione del percorso del file da creare non esiste.
`EACCES`	Permesso negato.

FILE SORGENTI

‘`lib/sys/stat.h`’ [95.25]

‘`lib/sys/stat/mknod.c`’ [95.25.5]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

‘`kernel/lib_s/s_mknod.c`’ [94.8.25]

VEDERE ANCHE

mkdir(2) [87.34], *chmod(2)* [87.7], *chown(2)* [87.8], *fcntl(2)* [87.18], *stat(2)* [87.55], *umask(2)* [87.60], *unlink(2)* [87.62].

87.36 os32: mount(2)

NOME

‘**mount**’, ‘**umount**’ - innesto e distacco di unità di memorizzazione

SINTASSI

```
#include <sys/os32.h>
int mount  (const char *path_dev, const char *path_mnt,
            int options);
int umount (const char *path_mnt);
```

DESCRIZIONE

La funzione ***mount()*** permette l’innesto di un’unità di memorizzazione individuata attraverso il percorso del file di dispositivo nel parametro ***path_dev***, nella directory corrispondente al percorso ***path_mnt***, con le opzioni indicate numericamente nell’ultimo argomento ***options***. Le opzioni di innesto, rappresentate attraverso delle macro-variabili, sono solo due:

Opzione	Descrizione
`MOUNT_DEFAULT`	Innesto normale, in lettura e scrittura.
`MOUNT_RO`	Innesto in sola lettura.

La funzione ***umount()*** consente di staccare un innesto fatto precedentemente, specificando il percorso della directory in cui questo è avvenuto.

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
−1	Errore: va verificato il contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EPERM`	Problema di accesso dovuto alla mancanza dei permessi necessari.
`ENOTDIR`	Ciò che dovrebbe essere una directory, non lo è.
`EBUSY`	La directory innesta già un file system e non può innestarne un altro.
`ENOENT`	La directory non esiste.
`E_NOT_MOUNTED`	La directory non innesta un file system da staccare.
`EUNKNOWN`	Si è verificato un problema non previsto e sconosciuto.

FILE SORGENTI

‘`lib/sys/os32.h`’ [95.21]

‘`lib/sys/os32/mount.c`’ [95.21.3]

‘`lib/sys/os32/umount.c`’ [95.21.8]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

‘kernel/lib_s/s_mount.c’ [94.8.26]

‘kernel/lib_s/s_umount.c’ [94.8.47]

VEDERE ANCHE

mount(8) [92.7], *umount(8)* [92.7].

87.37 os32: open(2)

«

NOME

‘**open**’ - apertura di un file puro e semplice oppure di un file di dispositivo

SINTASSI

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open (const char *path, int oflags);
int open (const char *path, int oflags, mode_t mode);
```

DESCRIZIONE

La funzione ***open()*** apre un file, indicato attraverso il percorso ***path***, in base alle opzioni rappresentate dagli indicatori ***oflags***. A seconda del tipo di indicatori specificati, potrebbe essere richiesto il parametro ***mode***.

Quando la funzione porta a termine correttamente il proprio compito, restituisce il numero del descrittore del file associato, il quale è sempre quello di valore più basso disponibile per il processo elaborativo in corso.

Il descrittore di file ottenuto inizialmente con la funzione ***open()***, è legato al processo elaborativo in corso; tuttavia, se successivamente il processo si sdoppia attraverso la funzione ***fork()***, tale descrittore, se ancora aperto, viene duplicato nella nuova copia del processo. Inoltre, se per il descrittore aperto non viene impostato l'indicatore '**FD_CLOEXEC**' (con l'ausilio della funzione ***fcntl()***), se il processo viene rimpiazzato con la funzione ***execve()***, il descrittore aperto viene ereditato dal nuovo processo.

Il parametro ***oflags*** richiede necessariamente la specificazione della modalità di accesso, attraverso la combinazione appropriata dei valori: '**O_RDONLY**', '**O_WRONLY**', '**O_RDWR**'. Inoltre, si possono combinare altri indicatori: '**O_CREAT**', '**O_TRUNC**', '**O_APPEND**'.

Opzione	Descrizione
`O_RDONLY`	Richiede un accesso in lettura.
`O_WRONLY`	Richiede un accesso in scrittura.
`O_RDWR` `O_RDONLY|O_WRONLY`	Richiede un accesso in lettura e scrittura (la combinazione di '**R_RDONLY**' e di '**O_WRONLY**' è equivalente all'uso di '**O_RDWR**').
`O_CREAT`	Richiede di creare contestualmente il file, ma in tal caso va usato anche il parametro ***mode***.
`O_TRUNC`	Se file da aprire esiste già, richiede che questo sia ridotto preventivamente a un file vuoto.
`O_APPEND`	Fa in modo che le operazioni di scrittura avvengano sempre partendo dalla fine del file.

Quando si utilizza l'opzione '**O_CREAT**', è necessario stabilire la modalità dei permessi, cosa che va fatta preferibilmente attraverso la combinazione di costanti simboliche appropriate, come elencato nella tabella successiva. Tale combinazione va fatta con l'uso dell'operatore OR binario; per esempio: '**S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH**'. Va osservato che os32 non gestisce i gruppi di utenti, pertanto, la definizione dei permessi relativi agli utenti appartenenti al gruppo proprietario di un file, non ha poi effetti pratici nel controllo degli accessi per tale tipo di contesto.

Modalità simbolica	Modalità numerica	Descrizione
S_IRWXU	00700_8	Lettura, scrittura ed esecuzione per l'utente proprietario.
S_IRUSR	00400_8	Lettura per l'utente proprietario.
S_IWUSR	00200_8	Scrittura per l'utente proprietario.
S_IXUSR	00100_8	Esecuzione per l'utente proprietario.

Modalità simbolica	Modalità numerica	Descrizione
S_IRWXG	00070_8	Lettura, scrittura ed esecuzione per il gruppo.
S_IRGRP	00040_8	Lettura per il gruppo.
S_IWGRP	00020_8	Scrittura per il gruppo.
S_IXGRP	00010_8	Esecuzione per il gruppo.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXO`	00007_8	Lettura, scrittura ed esecuzione per gli altri utenti.
`S_IROTH`	00004_8	Lettura per gli altri utenti.
`S_IWOTH`	00002_8	Scrittura per gli altri utenti.
`S_IXOTH`	00001_8	Esecuzione per gli altri utenti.

VALORE RESTITUITO

La funzione restituisce il numero del descrittore del file aperto, se l'operazione ha avuto successo, altrimenti dà semplicemente -1, impostando di conseguenza il valore della variabile ***errno***.

ERRORI

Valore di *errno*	Significato
`EINVAL`	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
`EPERM`	Operazione non consentita.
`EEXIST`	Il file da creare esiste già.
`EACCES`	Accesso non consentito.
`ENOENT`	Il file non esiste, oppure non esiste il percorso che porta al file da creare.
`EROFS`	Avendo richiesto un accesso in scrittura, si ottiene che il file system che lo contiene consente soltanto un accesso in lettura.
`ENOTDIR`	Il percorso che porta al file da aprire non è valido, in quanto ciò che dovrebbe essere una directory, non lo è.
`ENFILE`	Non si possono aprire altri file nell'ambito del sistema operativo (il sistema ha raggiunto il limite).
`EMFILE`	Non si possono aprire altri file nell'ambito del processo in corso.

FILE SORGENTI

'`lib/sys/types.h`' [95.26]

'`lib/sys/stat.h`' [95.25]

'`lib/fcntl.h`' [95.6]

'`lib/fcntl/open.c`' [95.6.3]

VEDERE ANCHE

chmod(2) [87.7], *chown(2)* [87.8], *close(2)* [87.10], *dup(2)* [87.12], *fcntl(2)* [87.18], *link(2)* [87.33], *mknod(2)* [87.35],

mount(2) [87.36], *read(2)* [87.39], *stat(2)* [87.55], *umask(2)* [87.60], *unlink(2)* [87.62], *write(2)* [87.64], *fopen(3)* [88.36].

87.38 os32: pipe(2)

«

NOME

'**pipe**' - creazione di un condotto senza nome

SINTASSI

```
#include <unistd.h>
int pipe (int pipefd[2]);
```

DESCRIZIONE

La funzione ***pipe()*** crea, nella tabella degli inode, un condotto, ovvero un inode speciale con questa caratteristica. All'inode associa due descrittori, uno per la lettura e l'altro per la scrittura, restituendone il numero, rispettivamente in ***pipefd[0]*** e ***pipefd[1]***. Ciò che viene scritto attraverso il descrittore ***pipefd[1]*** viene accumulato in una memoria tampone (costituita dallo spazio di memoria inutilizzato nell'inode che lo rappresenta) e viene prelevato con la lettura dal descrittore ***pipefd[0]***.

VALORE RESTITUITO

La funzione restituisce zero se la creazione si è conclusa con successo, oppure -1 in caso di problemi, aggiornando di conseguenza il valore di ***errno***.

ERRORI

Valore di *errno*	Significato
EINVAL	L'argomento fornito non è valido.
EMFILE	Troppi file aperti.
ENFILE	Troppi file aperti nel sistema.

FILE SORGENTI

'lib/unistd.h' [95.30]

'lib/unistd/pipe.c' [95.30.31]

'lib/sys/os32/sys.s' [95.21.7]

'kernel/lib_s/s_pipe.c' [94.8.28]

VEDERE ANCHE

close(2) [87.10] *open(2)* [87.37], *write(2)* [87.64], *read(2)* [87.39].

87.39 os32: read(2)

«

NOME

'**read**' - lettura di descrittore di file

SINTASSI

```
#include <unistd.h>
ssize_t read (int fdn, void *buffer, size_t count);
```

DESCRIZIONE

La funzione ***read()*** cerca di leggere il file rappresentato dal descrittore ***fdn***, partendo dalla posizione in cui si trova l'indice interno di accesso, per un massimo di ***count*** byte, collocando i dati letti in memoria a partire dal puntatore ***buffer***. L'indice interno al file viene fatto avanzare della quantità di byte letti effettivamente, se invece si incontra la fine del file, viene aggiornato l'indicatore interno per segnalare tale fatto.

VALORE RESTITUITO

La funzione restituisce la quantità di byte letti effettivamente, oppure zero se è stata raggiunta la fine del file e non si può proseguire oltre. Va osservato che la lettura effettiva di una quantità inferiore di byte rispetto a quanto richiesto non costituisce un errore: in quel caso i byte mancanti vanno richiesti con successive operazioni di lettura. In caso di errore, la funzione restituisce il valore -1, aggiornando contestualmente la variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il numero del descrittore di file non è valido.
`EINVAL`	Il file non è aperto in lettura.
`E_FILE_TYPE_UNSUPPORTED`	Il file è di tipo non gestibile con os32.

FILE SORGENTI

'`lib/unistd.h`' [95.30]

'`lib/unistd/read.c`' [95.30.32]

‘lib/sys/os32/sys.s’ [95.21.7]

‘kernel/lib_s/s_read.c’ [94.8.29]

VEDERE ANCHE

close(2) [87.10] *open(2)* [87.37], *write(2)* [87.64].

87.40 os32: recvfrom(2)

NOME

‘**recvfrom**’ - ricezione di un messaggio da un socket

SINTASSI

```
#include <sys/socket.h>
ssize_t recvfrom (int sfdn, void *buffer, size_t count,
                  int flags, struct sockaddr *addrfrom,
                  socklen_t *addrlen);
```

DESCRIZIONE

La funzione ***recvfrom()*** consente di ricevere un «messaggio» da un socket, collocandolo in memoria a partire dall’indirizzo ***buffer***, utilizzando al massimo ***count*** byte. La funzione restituisce poi la quantità di byte letti, o un valore negativo in caso di errore.

Se la funzione riceve un puntatore valido in corrispondenza di ***addrfrom*** e di ***addrlen***, significa che si vuole annotare in corrispondenza di ****addrfrom*** l’indirizzo di origine del messaggio ricevuto, in forma di variabile strutturata di tipo ‘**struct sockaddr**’. In tal caso, alla chiamata della funzione il valore di ****addrlen*** deve indicare la dimensione massima disponibile in

memoria per annotare tale informazione, sapendo che questo valore viene poi modificato per contenere la dimensione originale effettiva: in pratica, se questa dimensione è maggiore di quella della chiamata, vuol dire che tale informazione è stata annotata ma solo in parzialmente, troncandola.

Nella realizzazione di os32, il parametro ***flags*** viene semplicemente ignorato, non essendo previsti indicatori che possano modificare la modalità di ricezione-lettura dei dati.

È importante osservare che la ricezione di un messaggio può risultare troncata, se la memoria tampone che parte da ***buffer*** non ha una dimensione sufficiente. Questo succede per esempio se si riceve da un socket relativo a una connessione UDP. Quando invece si sta operando con un socket TCP, il flusso di ricezione avviene in modo continuo, senza troncamenti.

La lettura avviene normalmente bloccando il processo chiamante, fino a che si ottiene qualcosa. Diversamente, con l'ausilio della funzione ***fcntl()*** è possibile attribuire l'opzione ***O_NONBLOCK*** per lasciare che la funzione termini ugualmente segnalando l'errore ***EAGAIN***.

VALORE RESTITUITO

In caso di successo la funzione restituisce la dimensione del messaggio ricevuto; altrimenti si ottiene -1 e l'aggiornamento della variabile ***errno***.

ERRORI

Valore di *errno*	Significato
EBADF	Il descrittore ***sfdn*** non è valido.
ENOTSOCK	Il descrittore ***sfdn*** non è un socket.
EINVAL	Il puntatore ***buffer*** non è valido.
EPROTONOSUPPORT	Il protocollo del socket non è gestito da os32 in questo contesto.
EAFNOSUPPORT	Il tipo di indirizzamento del socket ***sfdn*** è diverso da ***AF_INET***.
EAGAIN	Non è disponibile alcun messaggio per il momento.

FILE SORGENTI

‘lib/sys/socket.h’ [95.23]

‘lib/sys/socket/recvfrom.c’ [95.23.5]

‘kernel/lib_s/s_recvfrom.c’ [94.8.30]

VEDERE ANCHE

accept(2) [87.3], *bind(2)* [87.4], *connect(2)* [87.11], *listen(2)* [87.31], *socket(2)* [87.54].

87.41 os32: rmdir(2)

«

NOME

‘**rmdir**’ - eliminazione di una directory vuota

SINTASSI

```
#include <unistd.h>
int rmdir (const char *path);
```

DESCRIZIONE

La funzione ***rmdir()*** cancella la directory indicata come percorso, nella stringa ***path***, purché sia vuota.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il percorso ***path*** non è valido o è semplicemente un puntatore nullo.
`ENOTDIR`	Il nome indicato o le posizioni intermedie del percorso si riferiscono a qualcosa che non è una directory.
`ENOTEMPTY`	La directory che si vorrebbe cancellare non è vuota.
`EROFS`	La directory si trova in un'unità innestata in sola lettura.
`EPERM`	Mancano i permessi necessari per eseguire l'operazione.
`EUNKNOWN`	Si è verificato un errore imprevisto e sconosciuto.

FILE SORGENTI

'`lib/unistd.h`' [95.30]

‘lib/unistd/rmdir.c’ [95.30.33]

‘lib/sys/os32/sys.s’ [95.21.7]

‘kernel/lib_s/s_unlink.c’ [94.8.48]

VEDERE ANCHE

mkdir(2) [87.34], *unlink(2)* [87.62].

87.42 os32: routeadd(2)

NOME

‘**routeadd**’ - aggiunta di un instradamento nella tabella degli instradamenti (funzione specifica di os32)

SINTASSI

```
#include <sys/os32.h>
int routeadd (in_addr_t destination, int m,
              in_addr_t router, int device);
```

DESCRIZIONE

La funzione di sistema ***routeadd()***, specifica di os32, permette di aggiungere un instradamento IPv4 che richiede l’attraversamento di un router. Infatti, gli instradamenti nella rete locale sono definiti automaticamente dalla funzione ***ipconfig()*** [87.28], contestualmente alla configurazione dell’interfaccia.

I parametri della funzione sono rappresentati rispettivamente da: indirizzo di destinazione; maschera in forma di quantità di bit; indirizzo del router da interpellare; numero dell’interfaccia di rete locale, attraverso la quale eseguire la comunicazione.

VALORE RESTITUITO

In caso di successo la funzione restituisce zero, altrimenti, in caso di errore, si ottiene −1 e l'aggiornamento della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EPERM`	È possibile usare la funzione soltanto con UID efficace pari a zero; diversamente si ottiene questo errore.
`EINVAL`	È stata indicata un'interfaccia di rete impossibile (sono ammissibili solo valori da 0 a 32).
`ENOMEM`	Non c'è spazio per aggiungere l'instradamento nella tabella relativa.

FILE SORGENTI

'`lib/sys/os32.h`' [95.21]

'`lib/sys/os32/routeadd.c`' [95.21.5]

'`kernel/lib_s/s_routeadd.c`' [94.8.31]

VEDERE ANCHE

ipconfig(2) [87.28], *routedel(2)* [87.43].

87.43 os32: routedel(2)

«

NOME

'**`routedel`**' - eliminazione di un instradamento nella tabella degli instradamenti (funzione specifica di os32)

SINTASSI

```
#include <sys/os32.h>
int routedel (in_addr_t destination, int m);
```

DESCRIZIONE

La funzione di sistema ***routedel()***, specifica di os32, permette di eliminare un instradamento IPv4, individuato dall'indirizzo e dalla maschera di rete (espressa in quantità di bit).

VALORE RESTITUITO

In caso di successo la funzione restituisce zero, altrimenti, in caso di errore, si ottiene −1 e l'aggiornamento della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EPERM`	È possibile usare la funzione soltanto con UID efficace pari a zero; diversamente si ottiene questo errore.
`EINVAL`	È stata indicata un'interfaccia di rete impossibile (sono ammissibili solo valori da 0 a 32), oppure l'instradamento da cancellare non esiste.

FILE SORGENTI

'`lib/sys/os32.h`' [95.21]

'`lib/sys/os32/routedel.c`' [95.21.6]

'`kernel/lib_s/s_routedel.c`' [94.8.32]

VEDERE ANCHE

ipconfig(2) [87.28], *routeadd(2)* [87.42].

87.44 os32: sbrk(2)

«

Vedere *brk(2)* [87.5].

87.45 os32: send(2)

«

NOME

‘**send**’ - invio di un messaggio attraverso un socket

SINTASSI

```
#include <sys/socket.h>
ssize_t send (int sfdn, const void *buffer, size_t count,
              int flags);
```

DESCRIZIONE

La funzione ***recvfrom()*** consente di inviare un «messaggio» attraverso un socket, prelevandolo dalla memoria a partire dall’indirizzo ***buffer***, utilizzando da lì al massimo ***count*** byte. La funzione restituisce poi la quantità di byte trasmessi effettivamente, o un valore negativo in caso di errore.

Nella realizzazione di os32, il parametro ***flags*** viene semplicemente ignorato, non essendo previsti indicatori che possano modificare la modalità di ricezione-lettura dei dati. D’altra parte, se non si considera il parametro ***flags***, la funzione si comporta nello stesso modo di ***write()****write(2)* [87.64].

La trasmissione avviene normalmente bloccando il processo chiamante, fino a che si ottiene il risultato. Diversamente, con l’ausilio della funzione ***fcntl()****fcntl(2)* [87.18] è possibile attribuire l’opzione ***O_NONBLOCK*** per lasciare che la funzione termini ugualmente segnalando l’errore ***EAGAIN***.

VALORE RESTITUITO

In caso di successo la funzione restituisce la dimensione del messaggio trasmesso effettivamente; altrimenti si ottiene −1 e l'aggiornamento della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il descrittore ***sfdn*** non è valido.
`ENOTSOCK`	Il descrittore ***sfdn*** non è un socket.
`EINVAL`	Il puntatore ***buffer*** non è valido.
`ECONNREFUSED`	Non è possibile contattare la controparte alla porta desiderata.
`ENOPROTOOPT`	Protocollo non disponibile.
`EHOSTUNREACH`	Non è possibile contattare la controparte all'indirizzo desiderato.
`ENETUNREACH`	Rete irraggiungibile.
`EDESTADDRREQ`	Indirizzo di destinazione mancante.
`EPROTONOSUPPORT`	Il protocollo del socket non è gestito da os32 in questo contesto.
`EPIPE`	La trasmissione in un flusso TCP si è interrotta prematuramente.
`EAGAIN`	Temporaneamente non è possibile trasmettere.
`EAFNOSUPPORT`	Il tipo di indirizzamento non è ***AF_INET***, l'unico attualmente ammissibile per os32.

FILE SORGENTI

‘`lib/sys/socket.h`’ [95.23]

‘`lib/sys/socket/send.c`’ [95.23.6]

‘`kernel/lib_s/s_send.c`’ [94.8.34]

VEDERE ANCHE

accept(2) [87.3], *bind(2)* [87.4], *connect(2)* [87.11], *listen(2)* [87.31], *socket(2)* [87.54], *recvfrom(2)* [87.40], *write(2)* [87.64].

87.46 os32: setegid(2)

«

Vedere *setgid(2)* [87.48].

87.47 os32: seteuid(2)

«

Vedere *setuid(2)* [87.51].

87.48 os32: setgid(2)

«

NOME

‘**`setgid`**’, ‘**`setegid`**’ - impostazione dell’identità del gruppo

SINTASSI

```
#include <unistd.h>
int setgid  (gid_t gid);
int setegid (gid_t gid);
```

DESCRIZIONE

Ogni processo viene associato a un gruppo di utenti, rappresentato da un numero, noto come GID, ovvero *group identity*. Tuttavia

si distinguono tre tipi di numeri GID: l'identità reale, l'identità efficace e un'identità salvata in precedenza. L'identità efficace di gruppo (EGID) è quella con cui opera sostanzialmente il processo; l'identità salvata è quella che ha avuto il processo in un altro momento in qualità di identità efficace e che per qualche motivo non ha più.

La funzione ***setgid()*** riceve come argomento un numero GID e si comporta diversamente a seconda della personalità del processo, come descritto nell'elenco successivo:

- se l'identità efficace del processo, EUID o EGID, corrisponde a zero, trattandosi di un caso particolarmente privilegiato, tutte le identità di gruppo del processo (reale, efficace e salvata) vengono inizializzate con il valore fornito alla funzione ***setgid()***;
- se l'identità efficace di gruppo del processo corrisponde a quella fornita come argomento a ***setgid()***, nulla cambia nella gestione delle identità del processo;
- se l'identità di gruppo reale o quella salvata in precedenza corrispondono a quella fornita come argomento di ***setgid()***, viene aggiornato il valore dell'identità efficace, senza cambiare le altre;
- in tutti gli altri casi, l'operazione non è consentita e si ottiene un errore.

La funzione ***setegid()*** riceve come argomento un numero GID e imposta con tale valore l'identità di gruppo efficace del processo, purché si verifichi almeno una delle condizioni seguenti:

- l'identità EUID o EGID del processo è zero;

- l'identità di gruppo reale o quella salvata del processo corrisponde all'identità efficace che si vuole impostare;
- l'identità di gruppo efficace del processo corrisponde già all'identità efficace che si vuole impostare.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EPERM`	Non si dispone dei permessi necessari a eseguire il cambiamento di identità.

FILE SORGENTI

'`lib/unistd.h`' [95.30]

'`lib/unistd/setgid.c`' [95.30.37]

'`lib/unistd/setegid.c`' [95.30.35]

'`lib/sys/os32/sys.s`' [95.21.7]

'`kernel/ibm_i386/isr.s`' [94.6.21]

'`kernel/proc/sysroutine.c`' [94.14.28]

'`kernel/lib_s/setgid.c`' [94.8.37]

'`kernel/lib_s/setegid.c`' [94.8.35]

VEDERE ANCHE

getuid(2) [87.27], *geteuid(2)* [87.27], *getgid(2)* [87.22], *getegid(2)* [87.22], *setuid(2)* [87.51].

87.49 os32: setjmp(2)

NOME

'**setjmp**', '**longjmp**' - salvataggio e recupero della pila per i «salti non locali»

SINTASSI

```
#include <setjmp.h>
int setjmp (jmp_buf env);
void longjmp (jmp_buf env, int val);
```

DESCRIZIONE

La funzione ***setjmp()*** consente di salvare, in corrispondenza dei una posizione di memoria rappresentata da ***env***, il contesto della pila dei dati per poterne recuperare lo stato in un momento successivo, attraverso ***longjmp()***. Quando la funzione ***longjmp()*** viene chiamata, la sua uscita si manifesta al posto della chiamata di ***setjmp()*** a cui è stato fatto riferimento con ***env***. Pertanto, quando la funzione ***setjmp()*** viene chiamata realmente, restituisce sempre il valore zero, mentre quando l'uscita di ***setjmp()*** deve in realtà manifestare il salto ottenuto con ***longjmp()***, il valore restituito è quanto corrisponde a ***val*** (il secondo parametro di ***longjmp()***.

os32 realizza il meccanismo del salto con ripristino del contesto, attraverso chiamate di funzione, per facilitare la comprensibilità del codice.

Figura 87.52. Lo stato della pila durante le varie fasi che riguardano la chiamata di ***setjmp()***, a confronto con i tipi '**`jmp_stack_t`**' e '**`jmp_env_t`**'.

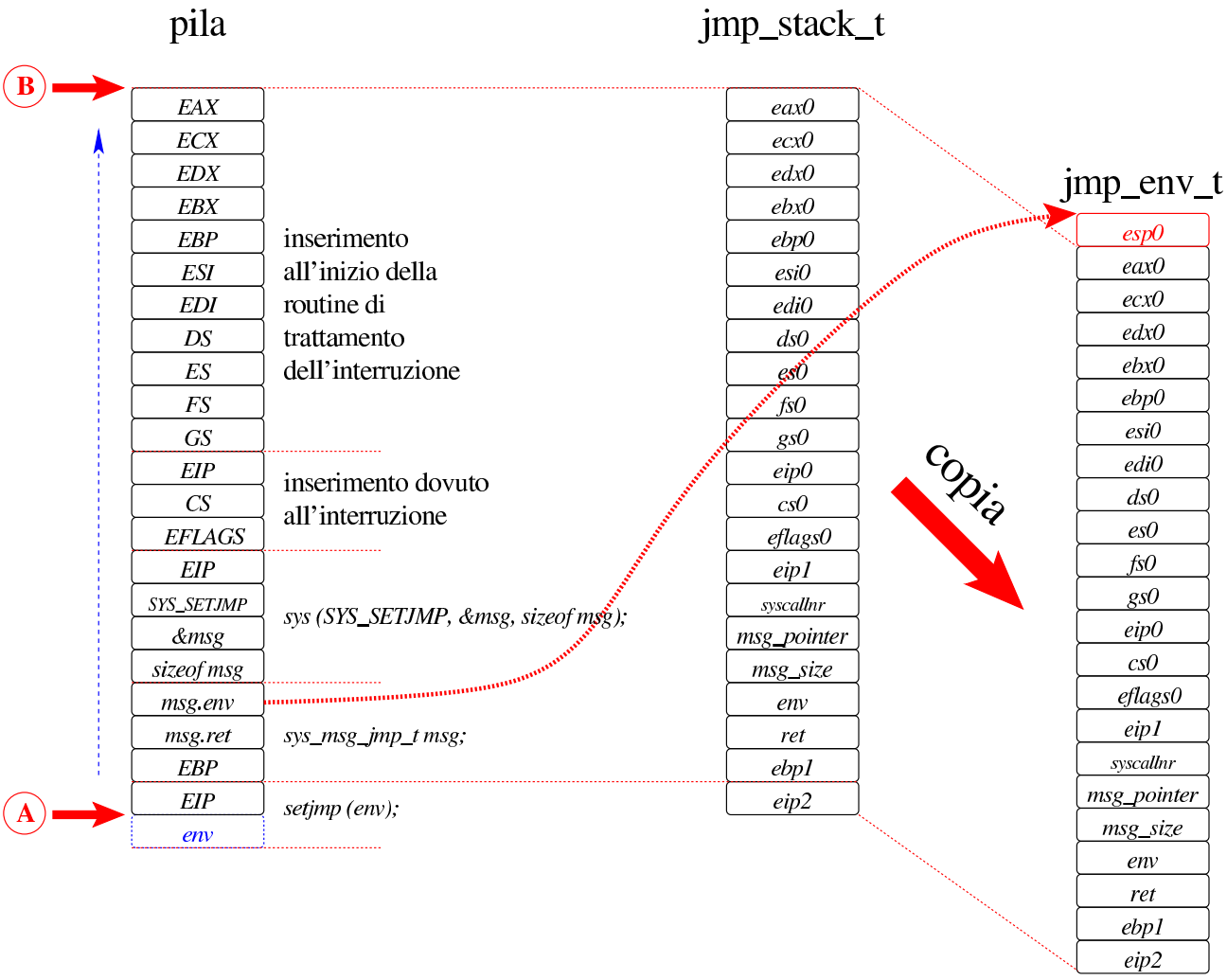
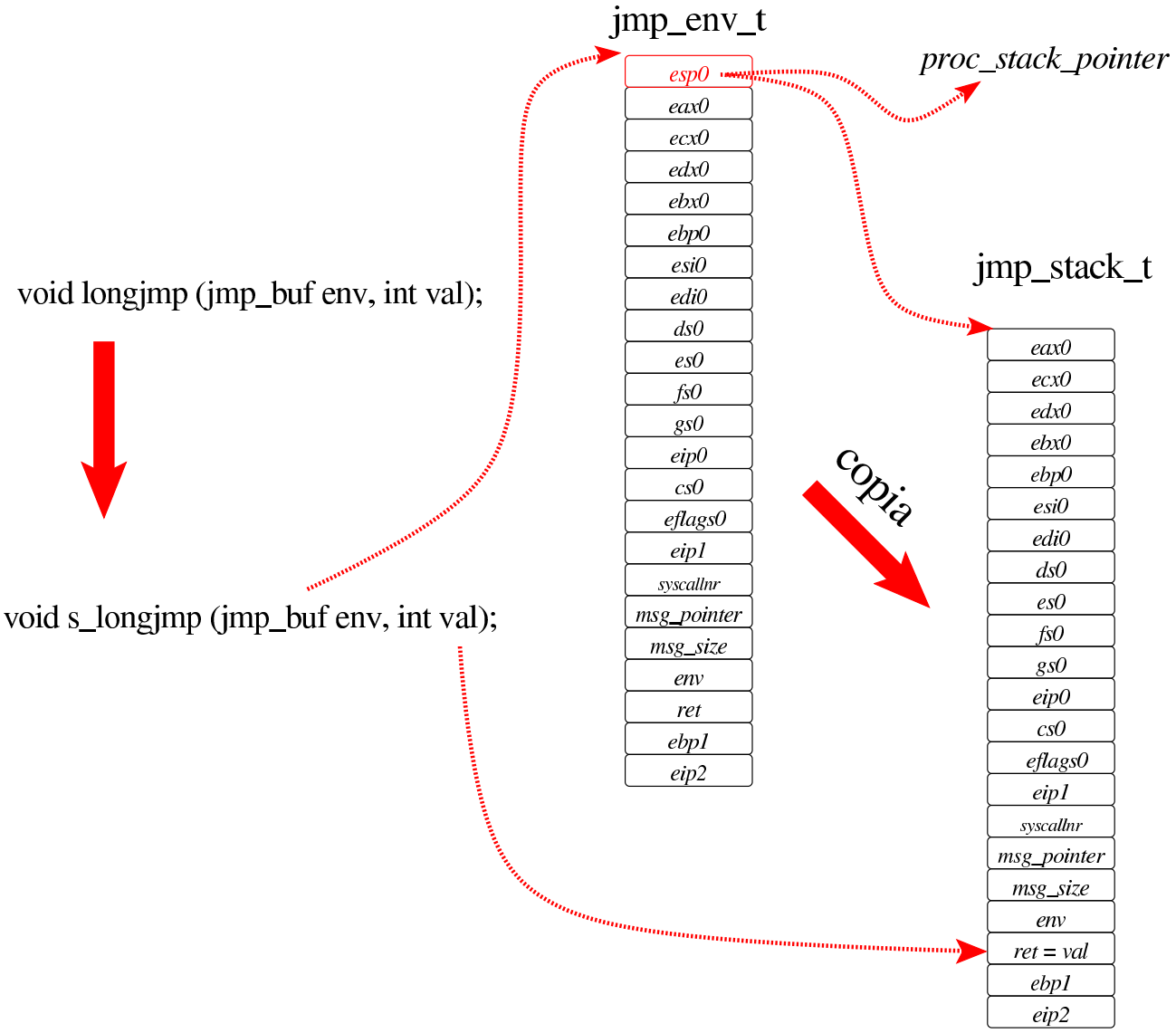


Figura 87.53. La chiamata di ***longjmp()*** ricostruisce la vecchia pila di ***setjmp()***, nella posizione in cui si trovava, ricollocando l'indice della pila e modificando il valore che poi ***setjmp()*** rediviva va a restituire.



VALORE RESTITUITO

La funzione ***setjmp()*** restituisce zero quando viene chiamata per salvare il contesto operativo, mentre restituisce ***val*** quando rappresenta il ripristino del contesto derivante dall'uso della funzione ***longjmp()***.

NOTE

Le funzioni ***setjmp()*** e ***longjmp()*** fanno parte dello standard per motivi storici, ma il loro uso è decisamente sconsigliabile.

Quando si ripristina il contesto con ***longjmp()***, è necessario che la pila precedente alla chiamata di ***setjmp()*** non sia stata compromessa. Pertanto, come condizione necessaria, ma non sufficiente, ***longjmp()*** può essere usato soltanto per ripristinare un contesto che nella pila attuale si trovi in una posizione antecedente (più interna).

FILE SORGENTI

‘`lib/setjmp.h`’ [95.16]

‘`lib/setjmp/setjmp.s`’ [95.16.2]

‘`lib/setjmp/longjmp.c`’ [95.16.1]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

‘`kernel/lib_s/s_setjmp.c`’ [94.8.38]

‘`kernel/lib_s/s_longjmp.c`’ [94.8.22]

VEDERE ANCHE

signal(2) [87.52].

87.50 os32: setpgrp(2)

«

NOME

‘**`setpgrp`**’ - impostazione del gruppo a cui appartiene il processo

SINTASSI

```
#include <unistd.h>
int setpgrp (void);
```

DESCRIZIONE

La funzione ***setpgrp()*** fa sì che il processo in corso costituisca un proprio gruppo autonomo, corrispondente al proprio numero PID. In altri termini, la funzione serve per iniziare un nuovo gruppo di processi, a cui i processi figli creati successivamente vengano associati in modo predefinito.

VALORE RESTITUITO

La funzione termina sempre con successo e restituisce sempre zero.

FILE SORGENTI

‘`lib/unistd.h`’ [95.30]

‘`lib/unistd/setpgrp.c`’ [95.30.38]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

VEDERE ANCHE

getpgrp(2) [87.50], *getuid(2)* [87.27].

87.51 os32: setuid(2)

«

NOME

‘**setuid**’, ‘**seteuid**’ - impostazione dell’identità dell’utente

SINTASSI

```
#include <unistd.h>
int setuid  (uid_t uid);
int seteuid (uid_t uid);
```

DESCRIZIONE

A ogni processo viene attribuita l’identità di un utente, rappresentata da un numero, noto come UID, ovvero *user identity*. Tuttavia si distinguono tre tipi di numeri UID: l’identità reale, l’identità efficace e un’identità salvata in precedenza. L’identità efficace (EUID) è quella con cui opera sostanzialmente il processo; l’identità salvata è quella che ha avuto il processo in un altro momento in qualità di identità efficace e che per qualche motivo non ha più.

La funzione ***setuid()*** riceve come argomento un numero UID e si comporta diversamente a seconda della personalità del processo, come descritto nell’elenco successivo:

- se l’identità efficace del processo corrisponde a zero, trattandosi di un caso particolarmente privilegiato, tutte le identità del processo (reale, efficace e salvata) vengono inizializzate con il valore fornito alla funzione ***setuid()***;

- se l'identità efficace del processo corrisponde a quella fornita come argomento a ***setuid()***, nulla cambia nella gestione delle identità del processo;
- se l'identità reale o quella salvata in precedenza corrispondono a quella fornita come argomento di ***setuid()***, viene aggiornato il valore dell'identità efficace, senza cambiare le altre;
- in tutti gli altri casi, l'operazione non è consentita e si ottiene un errore.

La funzione ***seteuid()*** riceve come argomento un numero UID e imposta con tale valore l'identità efficace del processo, purché si verifichi almeno una delle condizioni seguenti:

- l'identità efficace del processo è zero;
- l'identità reale o quella salvata del processo corrisponde all'identità efficace che si vuole impostare;
- l'identità efficace del processo corrisponde già all'identità efficace che si vuole impostare.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EPERM`	Non si dispone dei permessi necessari a eseguire il cambiamento di identità.

FILE SORGENTI

'`lib/unistd.h`' [95.30]

‘lib/unistd/setuid.c’ [95.30.39]

‘lib/unistd/seteuid.c’ [95.30.36]

‘lib/sys/os32/sys.s’ [95.21.7]

‘kernel/ibm_i386/isr.s’ [94.6.21]

‘kernel/proc/sysroutine.c’ [94.14.28]

‘kernel/lib_s/s_setuid.c’ [94.8.39]

‘kernel/lib_s/s_seteuid.c’ [94.8.36]

VEDERE ANCHE

getuid(2) [87.27], *geteuid(2)* [87.27], *getgid(2)* [87.22], *getegid(2)* [87.22], *setgid(2)* [87.48], *setegid(2)* [87.48].

VEDERE ANCHE

getuid(2) [87.51].

87.52 os32: signal(2)

«

NOME

‘**signal**’ - abilitazione e disabilitazione dei segnali

SINTASSI

```
#include <signal.h>
sighandler_t signal (int sig, sighandler_t handler);
```

DESCRIZIONE

La funzione ***signal()*** di os32 consente soltanto di abilitare o disabilitare un segnale, il quale, se abilitato, può essere gestito solo in modo predefinito dal sistema. Pertanto, il valore che

può assumere ***handler*** sono solo: '**SIG_DFL**' (gestione predefinita) e '**SIG_IGN**' (ignora il segnale). Il primo parametro, ***sig***, rappresenta il segnale a cui applicare ***handler***.

Il tipo '**sighandler_t**' è definito nel file '`signal.h`', nel modo seguente:

```
typedef void (*sighandler_t) (int);
```

Rappresenta il puntatore a una funzione avente un solo parametro di tipo '**int**', la quale non restituisce alcunché.

VALORE RESTITUITO

La funzione restituisce il tipo di «gestione» impostata precedentemente per il segnale richiesto, ovvero '**SIG_ERR**' in caso di errore.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il numero ***sig*** o il valore di ***handler*** non sono validi.

NOTE

Lo scopo della funzione ***signal()*** dovrebbe essere quello di consentire l'associazione di un evento, manifestato da un segnale, all'esecuzione di un'altra funzione, avente una forma del tipo '***nome*** **(int)**'. os32 non consente tale gestione, pertanto lascia soltanto la possibilità di attribuire un comportamento predefinito al segnale scelto, oppure di disabilitarlo, ammesso che ciò sia consentito. Sotto questo aspetto, il fatto di dover gestire i valori '**SIG_ERR**', '**SIG_DFL**' e '**SIG_IGN**', come se fossero puntatori

a una funzione, diventa superfluo, ma rimane utile soltanto per mantenere un minimo di conformità con quello che è lo standard della funzione ***signal()***.

FILE SORGENTI

‘`lib/signal.h`’ [95.17]

‘`lib/signal/signal.c`’ [95.17.3]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

‘`kernel/lib_s/s_signal.c`’ [94.8.40]

VEDERE ANCHE

kill(2) [87.29].

87.53 os32: sleep(2)

«

NOME

‘**`sleep`**’ - pausa volontaria del processo chiamante

SINTASSI

```
#include <unistd.h>
unsigned int sleep (unsigned int seconds);
```

DESCRIZIONE

La funzione ***sleep()*** chiede di mettere a riposo il processo chiamante per la quantità di secondi indicata come argomento. Il processo può però essere risvegliato prima della conclusione di tale durata, ma in tal caso la funzione restituisce la quantità di secondi che non sono stati usati per il «riposo» del processo.

VALORE RESTITUITO

La funzione restituisce zero se la pausa richiesta è trascorsa completamente; altrimenti, restituisce quanti secondi mancano ancora per completare il tempo di riposo chiesto originariamente. Non si prevede il manifestarsi di errori.

FILE SORGENTI

‘`lib/unistd.h`’ [95.30]

‘`lib/unistd/sleep.c`’ [95.30.40]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

VEDERE ANCHE

signal(2) [87.52].

87.54 os32: socket(2)

NOME

‘**`socket`**’ - crea un socket, definendo solo il tipo e il protocollo

SINTASSI

```
#include <sys/socket.h>
int socket (int family, int type, int protocol);
```

DESCRIZIONE

La funzione ***socket()*** crea un socket, ovvero un terminale di una comunicazione, restituendone il descrittore numerico, per poi potervi fare riferimento.

Il primo parametro della funzione (***family***)), noto anche come «dominio» del socket, può essere soltanto ***AF_INET*** per os32, corrispondente a un socket di dominio Internet IPv4; pertanto non è possibile creare socket di dominio Unix.

Il secondo parametro, ***type***, definisce la modalità con cui avviene la comunicazione attraverso il socket. Per os32 questa può essere:

Tipo	Significato
`SOCK_RAW`	Definisce una comunicazione generica che per os32 riguarda soltanto i protocolli ICMP.
`SOCK_DGRAM`	Definisce una comunicazione a pacchetti indipendenti, di una certa dimensione massima, senza controlli. Per os32, questo tipo di modalità riguarda esclusivamente il protocollo UDP.
`SOCK_STREAM`	Definisce un flusso di byte ordinato, affidabile, a due vie (trasmissione e ricezione indipendenti). Per os32, questo tipo di modalità riguarda esclusivamente il protocollo TCP.

Il terzo parametro, ***protocol*** definisce il protocollo di comunicazione. Per os32 può essere:

Protocollo	Descrizione
`IPPROTO_ICMP`	Protocollo ICMP, da abbinare necessariamente a un tipo ***SOCK_RAW***.
`IPPROTO_UDP`	Protocollo UDP, da abbinare necessariamente a un tipo ***SOCK_DGRAM***.
`IPPROTO_TCP`	Protocollo TCP, da abbinare necessariamente a un tipo ***SOCK_STREAM***.

VALORE RESTITUITO

In caso di successo la funzione restituisce il descrittore del socket creato; altrimenti si ottiene −1 e l'aggiornamento della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EACCES`	È stato richiesto di creare un socket di tipo ***SOCK_RAW*** senza i privilegi necessari (serve un UID efficace pari a zero).
`EAFNOSUPPORT`	Il tipo di indirizzamento non è ***AF_INET***, l'unico attualmente ammissibile per os32.
`EPROTONOSUPPORT`	Il protocollo del socket non è gestito da os32 in questo contesto.
`ENFILE`	Troppi file aperti nel sistema.
`EMFILE`	Troppi file aperti.

FILE SORGENTI

'`lib/sys/socket.h`' [95.23]

'`lib/sys/socket/socket.c`' [95.23.7]

'`kernel/lib_s/s_socket.c`' [94.8.41]

VEDERE ANCHE

socket(7) [91.2], *accept(2)* [87.3], *bind(2)* [87.4], *connect(2)* [87.11], *listen(2)* [87.31].

87.55 os32: stat(2)

«

NOME

‘**stat**’, ‘**fstat**’ - interrogazione dello stato di un file

SINTASSI

```
#include <sys/stat.h>
int stat  (const char *path, struct stat *buffer);
int fstat (int fdn, struct stat *buffer);
```

DESCRIZIONE

Le funzioni *stat()* e *fstat()* interrogano il sistema su di un file, per ottenerne le caratteristiche in forma di variabile strutturata di tipo ‘**struct stat**’.

La funzione *stat()* individua il file attraverso una stringa contenente il suo percorso (***path***); la funzione ‘**fstat**’ si riferisce a un file aperto di cui si conosce il numero del descrittore (***fdn***). In entrambi i casi, la struttura che deve accogliere l’esito dell’interrogazione, viene indicata attraverso un puntatore, come ultimo argomento (***buffer***).

La struttura ‘**struct stat**’ è definita nel file ‘`sys/stat.h`’ nel modo seguente:

```
struct stat {
  dev_t     st_dev;     // Device containing the file.
  ino_t     st_ino;     // File serial number (inode
                        // number).
  mode_t    st_mode;    // File type and permissions.
  nlink_t   st_nlink;   // Links to the file.
  uid_t     st_uid;     // Owner user id.
  gid_t     st_gid;     // Owner group id.
  dev_t     st_rdev;    // Device number if it is a device
                        // file.
  off_t     st_size;    // File size.
  time_t    st_atime;   // Last access time.
  time_t    st_mtime;   // Last modification time.
  time_t    st_ctime;   // Last inode modification.
  blksize_t st_blksize; // Block size for I/O operations.
  blkcnt_t  st_blocks;  // File size / block size.
};
```

Va osservato che il file system Minix 1, usato da os32, riporta esclusivamente la data e l'ora di modifica, pertanto le altre due date previste sono sempre uguali a quella di modifica.

Il membro ***st_mode***, oltre alla modalità dei permessi che si cambiano con *chmod(2)* [87.7], serve ad annotare altre informazioni. Nel file '`sys/stat.h`' sono definite delle macroistruzioni, utili per individuare il tipo di file. Queste macroistruzioni si risolvono in un valore numerico diverso da zero, solo se la condizione che rappresentano è vera:

Macroistruzione	Significato
`S_ISBLK (`***m***`)`	È un file di dispositivo a blocchi?
`S_ISCHR (`***m***`)`	È un file di dispositivo a caratteri?
`S_ISFIFO (`***m***`)`	È un file FIFO?
`S_ISREG (`***m***`)`	È un file puro e semplice?
`S_ISDIR (`***m***`)`	È una directory?
`S_ISLNK (`***m***`)`	È un collegamento simbolico?
`S_ISSOCK (`***m***`)`	È un socket di dominio Unix?

Naturalmente, anche se nel file system possono esistere file di ogni tipo, poi os32 non è in grado di gestire i file FIFO, i collegamenti simbolici e nemmeno i socket di dominio Unix.

Nel file ‘`sys/stat.h`’ sono definite anche delle macro-variabili per individuare e facilitare la selezione dei bit che compongono le informazioni del membro ***st_mode***:

Modalità simbolica	Modalità numerica	Descrizione
`S_IFMT`	0170000_8	Maschera che raccoglie tutti i bit che individuano il tipo di file.
`S_IFBLK`	0060000_8	File di dispositivo a blocchi.
`S_IFCHR`	0020000_8	File di dispositivo a caratteri.
`S_IFIFO`	0010000_8	File FIFO.
`S_IFREG`	0100000_8	File puro e semplice.
`S_IFDIR`	0040000_8	Directory.
`S_IFLNK`	0120000_8	Collegamento simbolico.
`S_IFSOCK`	0140000_8	Socket di dominio Unix.

Modalità simbolica	Modalità numerica	Descrizione
`S_ISUID`	0004000_8	SUID.
`S_ISGID`	0002000_8	SGID.
`S_ISVTX`	0001000_8	Sticky.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXU`	0000700_8	Lettura, scrittura ed esecuzione per l'utente proprietario.
`S_IRUSR`	0000400_8	Lettura per l'utente proprietario.
`S_IWUSR`	0000200_8	Scrittura per l'utente proprietario.
`S_IXUSR`	0000100_8	Esecuzione per l'utente proprietario.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXG`	0000070_8	Lettura, scrittura ed esecuzione per il gruppo.
`S_IRGRP`	0000040_8	Lettura per il gruppo.
`S_IWGRP`	0000020_8	Scrittura per il gruppo.
`S_IXGRP`	0000010_8	Esecuzione per il gruppo.

Modalità simbolica	Modalità numerica	Descrizione
`S_IRWXO`	0000007_8	Lettura, scrittura ed esecuzione per gli altri utenti.
`S_IROTH`	0000004_8	Lettura per gli altri utenti.
`S_IWOTH`	0000002_8	Scrittura per gli altri utenti.
`S_IXOTH`	0000001_8	Esecuzione per gli altri utenti.

os32 non considera i permessi SUID (*Set user id*), SGID (*Set group id*) e Sticky; inoltre, non tiene in considerazione i permessi legati al gruppo, perché non ne tiene traccia.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dal contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`ENFILE`	Troppi file aperti nel sistema.
`ENOENT`	File non trovato.
`EACCES`	Permesso negato.
`EBADF`	Il descrittore del file richiesto non è valido.

FILE SORGENTI

‘`lib/sys/stat.h`’ [95.25]

‘`lib/sys/stat/stat.c`’ [95.25.6]

‘`lib/sys/stat/fstat.c`’ [95.25.3]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

‘`kernel/lib_s/s_stat.c`’ [94.8.42]

‘`kernel/lib_s/s_fstat.c`’ [94.8.17]

VEDERE ANCHE

chmod(2) [87.7], *chown(2)* [87.8].

87.56 os32: sys(2)

«

NOME

‘**sys**’ - chiamata di sistema

SINTASSI

```
#include <sys/os32.h>
void sys (int syscallnr, void *message, size_t size);
```

DESCRIZIONE

Attraverso la funzione ***sys()*** si effettuano tutte le chiamate di sistema, passando al kernel un messaggio, a cui punta ***message***, lungo ***size*** byte. A seconda dei casi, il messaggio può essere modificato dal kernel, come risposta alla chiamata.

Il messaggio è in pratica una variabile strutturata, la cui articolazione cambia a seconda del tipo di chiamata, pertanto si rende necessario specificarne ogni volta la dimensione.

Le strutture usate per comporre i messaggi hanno alcuni membri ricorrenti frequentemente:

Membro	Descrizione
`char path[PATH_MAX];`	Un percorso di file o directory.
`… ret;`	Serve a contenere il valore restituito dalla funzione che nel kernel compie effettivamente il lavoro. Il tipo del membro varia caso per caso.
`int errno;`	Serve a contenere il numero dell'errore prodotto dalla funzione che nel kernel compie effettivamente il lavoro.
`int errln;`	Serve a contenere il numero della riga di codice in cui si è prodotto un errore.
`char errfn[PATH_MAX];`	Serve a contenere il nome del file in cui si è prodotto un errore.

Le funzioni che si avvalgono di ***sys()***, prima della chiamata devono provvedere a compilare il messaggio con i dati necessari, dopo la chiamata devono acquisire i dati di ritorno, contenuti nel messaggio aggiornato dal kernel, e in particolare devono aggiornare le variabili ***errno***, ***errln*** e ***errfl***, utilizzando i membri con lo stesso nome presenti nel messaggio.

FILE SORGENTI

'`lib/sys/os32.h`' [95.21]

'`lib/sys/os32/sys.s`' [95.21.7]

'`kernel/ibm_i386/isr.s`' [94.6.21]

'`kernel/proc/sysroutine.c`' [94.14.28]

87.57 os32: stime(2)

«

Vedere *time(2)* [87.59].

87.58 os32: tcgetattr(2)

«

NOME

‘**tcgetattr**’, ‘**tcsetattr**’ - lettura o impostazione della configurazione del terminale

SINTASSI

```
#include <termios.h>
int tcgetattr (int fdn, struct termios *termios_p);
int tcsetattr (int fdn, int action,
               struct termios *termios_p);
```

DESCRIZIONE

Le funzioni che fanno capo al file di intestazione ‘`termios.h`’ consentono di gestire la configurazione del terminale, per ciò che riguarda l’input e l’output dello stesso. Va comunque osservato che os32 gestisce il terminale esclusivamente in modalità canonica, sostanzialmente equivalente a quella della vecchia telescrivente.

La configurazione del terminale viene letta o scritta attraverso una variabile strutturata di tipo ‘**struct termios**’, organizzata nel modo seguente:

```
struct termios {
    tcflag_t c_iflag;
    tcflag_t c_oflag;
    tcflag_t c_cflag;
    tcflag_t c_lflag;
    cc_t     c_cc[NCCS];
};
```

Il membro ***c_cc[]*** è un array di caratteri di controllo, a cui viene attribuita una definizione. Il membro ***c_iflag*** serve a contenere opzioni sull'inserimento, ovvero sul controllo della digitazione. Il membro ***c_lflag*** serve a contenere delle opzioni definite come «locali», le quali si occupano in pratica di controllare la visualizzazione della digitazione introdotta e di decidere se l'interruzione ricevuta da tastiera debba produrre l'invio di un segnale di interruzione al processo con cui si sta interagendo. Gli altri due membri della struttura non vengono utilizzati da os32.

Tabella 84.91. Caratteri di controllo riconosciuti da os32, secondo le definizioni del file '`termios.h`'.

Definizione	Corrispondenza	Descrizione
VEOF	04_{16} <*EOT*>	Carattere di fine file.
VERASE	08_{16} <*BS*>	Carattere di cancellazione.
VINTR	03_{16} <*ETX*>	Carattere di interruzione.
VQUIT	$1C_{16}$ <*FS*>	Carattere di abbandono.

Tabella 84.92. Opzioni del membro ***c_iflag*** riconosciute da os32.

Opzione	Descrizione
BRKINT	Se questa opzione è attiva e, nel contempo, non è attiva ***IGNBRK***, si intendono recepire i codici di interruzione ***VINTR***. Se l'opzione ***ISIG*** del membro ***c_lflag*** è attiva, il processo più interno del gruppo a cui appartiene il terminale viene concluso; in ogni caso, viene annullato il contenuto della riga di inserimento in corso.
ICRNL	Se si riceve il carattere *<CR>*, questo viene convertito in *<NL>*.
IGNBRK	Se questa opzione è attiva, fa sì che il carattere definito come ***VINTR*** sia ignorato.
IGNCR	Se si riceve il carattere *<CR>*, questo viene ignorato semplicemente.
INLCR	Se si riceve il carattere *<NL>*, questo viene convertito in *<CR>*.

Tabella 84.93. Opzioni del membro ***c_lflag*** riconosciute da os32.

Opzione	Descrizione
ECHO	Abilita la visualizzazione sullo schermo del testo inserito da tastiera.
ECHOE	Ammesso che sia attiva l'opzione ***ECHO***, questa abilita il recepimento del carattere definito come ***VERASE*** per cancellare l'ultimo carattere inserito, indietreggiando di una posizione.
ECHONL	Indipendentemente dall'opzione ***ECHO***, questa abilita il recepimento del carattere *<NL>* per fare avanzare il cursore alla riga successiva, con l'eventuale scorrimento in avanti se si trova già sull'ultima.
ISIG	Ammesso che sia recepito e accettato un codice di interruzione, definito come ***VINTR***, con questa opzione si ottiene l'invio di un segnale di interruzione al processo più interno del gruppo collegato al terminale (il processo più interno dovrebbe corrispondere a quello in primo piano al momento della digitazione).

La funzione ***tcgetattr()*** legge la configurazione del terminale connesso al descrittore ***fdn***, mettendo i dati ottenuti nella struttura a cui punta ***termios_p***; al contrario, ***tcsetattr()*** configura il terminale del descrittore ***fdn***, in base ai dati contenuti nella struttura a cui punta ***termios_p***.

VALORE RESTITUITO

La funzioni ***tcgetattr()*** e ***tcsetattr()*** restituiscono zero in caso di successo, oppure il valore -1 in caso contrario, aggiornando di conseguenza la variabile ***errno***.

ERRORI

Valore di *errno*	Significato
`EINVAL`	Il puntatore ***termios_p*** non è valido.
`EBADF`	Il numero del descrittore di file non è valido.
`ENOTTY`	Il numero del descrittore di file non corrisponde a un terminale.

DIFETTI

È disponibile soltanto una modalità canonica di funzionamento del terminale.

FILE SORGENTI

‘`lib/termios.h`’ [95.28]

‘`lib/termios/tcgetattr.c`’ [95.28.1]

‘`lib/termios/tcsetattr.c`’ [95.28.2]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

‘`kernel/lib_s.h`’ [94.8]

‘`kernel/lib_s/s_tcgetattr.c`’ [94.8.44]

‘`kernel/lib_s/s_tcsetattr.c`’ [94.8.45]

VEDERE ANCHE

tty(1) [86.27], *isatty(3)* [88.69].

87.59 os32: time(2)

«

NOME

'**time**', '**stime**' - lettura o impostazione della data e dell'ora del sistema

SINTASSI

```
#include <time.h>
time_t time  (time_t *timer);
int    stime (time_t *timer);
```

DESCRIZIONE

La funzioni ***time()*** legge la data e l'ora attuale del sistema, espressa in secondi; se il puntatore ***timer*** è valido (non è '**NULL**'), il risultato dell'interrogazione viene salvato anche in ciò a cui questo punta.

La funzione ***stime()*** consente di modificare la data e l'ora attuale del sistema, fornendo il puntatore alla variabile contenente la quantità di secondi trascorsi a partire dall'ora zero del 1 gennaio 1970.

VALORE RESTITUITO

La funzione ***time()*** restituisce la data e l'ora attuale del sistema, espressa in secondi trascorsi a partire dall'ora zero del 1 gennaio 1970.

La funzione ***stime()*** restituisce zero in caso di successo e, teoricamente, −1 in caso di errore, ma attualmente nessun tipo di errore è previsto.

DIFETTI

La funzione ***stime()*** dovrebbe essere riservata a un utente privilegiato, mentre attualmente qualunque utente può servirsene per cambiare la data di sistema.

FILE SORGENTI

‘`lib/time.h`’ [95.29]

‘`lib/time/time.c`’ [95.29.6]

‘`lib/time/stime.c`’ [95.29.5]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

‘`kernel/lib_k.h`’ [94.7]

‘`kernel/lib_s/s_time.c`’ [94.8.46]

‘`kernel/lib_s/s_stime.c`’ [94.8.43]

VEDERE ANCHE

date(1) [86.10], *ctime(3)* [88.15].

87.60 os32: umask(2)

«

NOME

‘**`umask`**’ - maschera dei permessi

SINTASSI

```
#include <sys/stat.h>
mode_t umask (mode_t mask);
```

DESCRIZIONE

La funzione ***umask()*** modifica la maschera dei permessi associata al processo in corso. Nel contenuto del parametro ***mask*** vengono presi in considerazione soltanto i nove bit meno significativi, i quali rappresentano i premessi di accesso di utente proprietario, gruppo e altri utenti.

La maschera dei permessi viene usata dalle funzioni che creano dei file, per limitare i permessi in modo automatico: ciò che appare attivo nella maschera è quello che non viene consentito nella creazione del file.

VALORE RESTITUITO

La funzione restituisce il valore che aveva la maschera dei permessi prima della chiamata.

FILE SORGENTI

‘`lib/sys/stat.h`’ [95.25]

‘`lib/sys/stat/umask.c`’ [95.25.7]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

VEDERE ANCHE

mkdir(2) [87.34], *chmod(2)* [87.7], *open(2)* [87.37], *stat(2)* [87.55].

87.61 os32: umount(2)

«

Vedere *mount(2)* [87.36].

87.62 os32: unlink(2)

«

NOME

‘**unlink**’ - cancellazione di un nome

SINTASSI

```
#include <unistd.h>
int unlink (const char *path);
```

DESCRIZIONE

La funzione ***unlink()*** cancella un nome da una directory, ma se si tratta dell’ultimo collegamento che ha quel file, allora libera anche l’inode corrispondente.

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
−1	Errore: la variabile ***errno*** viene impostata di conseguenza.

ERRORI

Valore di *errno*	Significato
ENOTEMPTY	È stata tentata la cancellazione di una directory, ma questa non è vuota.
ENOTDIR	Una delle directory del percorso, non è una directory.
ENOENT	Il nome richiesto non esiste.
EROFS	Il file system è in sola lettura.
EPERM	Mancano i permessi necessari.
EUNKNOWN	Si è verificato un errore imprevisto e sconosciuto.

FILE SORGENTI

‘lib/unistd.h’ [95.30]

‘lib/unistd/unlink.c’ [95.30.42]

‘lib/sys/os32/sys.s’ [95.21.7]

‘kernel/ibm_i386/isr.s’ [94.6.21]

‘kernel/proc/sysroutine.c’ [94.14.28]

‘kernel/lib_s/s_unlink.c’ [94.8.48]

VEDERE ANCHE

rm(1) [86.22], *link(2)* [87.30], *rmdir(2)* [87.41].

87.63 os32: wait(2)

«

NOME

‘**wait**’ - attesa della morte di un processo figlio

SINTASSI

```
#include <sys/wait.h>
pid_t wait (int *status);
```

DESCRIZIONE

La funzione ***wait()*** mette il processo in pausa, in attesa della morte di un processo figlio; quando ciò dovesse accadere, il valore di ****status*** verrebbe aggiornato con quanto restituito dal processo defunto e il processo sospeso riprenderebbe l'esecuzione.

VALORE RESTITUITO

La funzione restituisce il numero del processo defunto, oppure −1 se non ci sono processi figli.

ERRORI

Valore di ***errno***	Significato
`ECHILD`	Non ci sono processi figli da attendere.

FILE SORGENTI

'`lib/sys/wait.h`' [95.27]

'`lib/sys/wait/wait.c`' [95.27.1]

'`lib/sys/os32/sys.s`' [95.21.7]

'`kernel/ibm_i386/isr.s`' [94.6.21]

'`kernel/proc/sysroutine.c`' [94.14.28]

'`kernel/lib_s/s_wait.c`' [94.8.49]

VEDERE ANCHE

_exit(2) [87.2], *fork(2)* [87.19], *kill(2)* [87.29], *signal(2)* [87.52].

87.64 os32: write(2)

«

NOME

‘**write**’ - scrittura di un descrittore di file

SINTASSI

```
#include <unistd.h>
ssize_t write (int fdn, const void *buffer, size_t count);
```

DESCRIZIONE

La funzione ***write()*** consente di scrivere fino a un massimo di ***count*** byte, tratti dall’area di memoria che inizia all’indirizzo ***buffer***, presso il file rappresentato dal descrittore ***fdn***. La scrittura avviene a partire dalla posizione in cui si trova l’indice interno.

VALORE RESTITUITO

La funzione restituisce la quantità di byte scritti effettivamente e in tal caso è possibile anche ottenere una quantità pari a zero. Se si verifica invece un errore, la funzione restituisce -1 e aggiorna la variabile ***errno***.

ERRORI

Valore di *errno*	Significato
`EBADF`	Il numero del descrittore di file non è valido.
`EINVAL`	Il file non è aperto in scrittura.
`EISDIR`	Il file è una directory.
`E_FILE_TYPE_UNSUPPORTED`	Il file è di tipo non gestibile con os32.
`EIO`	Errore di input-output.

FILE SORGENTI

‘`lib/unistd.h`’ [95.30]

‘`lib/unistd/write.c`’ [95.30.43]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

‘`kernel/proc/sysroutine.c`’ [94.14.28]

‘`kernel/lib_s/s_write.c`’ [94.8.50]

VEDERE ANCHE

close(2) [87.10], *lseek(2)* [87.33], *open(2)* [87.37], *read(2)* [87.39], *fwrite(3)* [88.49].

87.65 os32: z(2)

«

NOME

‘**z**_...’ - funzioni provvisorie

SINTASSI

```
#include <sys/os32.h>
void z_perror   (const char *string);
int  z_printf   (char *format, ...);
int  z_vprintf  (char *format, va_list arg);
```

DESCRIZIONE

Le funzioni del gruppo '**z_...()**' eseguono compiti equivalenti a quelli delle funzioni di libreria con lo stesso nome, ma prive del prefisso '**z_**'. Queste funzioni '**z_...()**' si avvalgono, per il loro lavoro, di chiamate di sistema particolari; la loro realizzazione si è resa necessaria durante lo sviluppo di os32, prima che potesse essere disponibile un sistema di gestione centralizzato dei dispositivi.

Queste funzioni non sono più utili, ma rimangono per documentare le fasi realizzative iniziali di os32 e, d'altro canto, possono servire se si rende necessario raggirare la gestione dei dispositivi per visualizzare dei messaggi sullo schermo.

FILE SORGENTI

'`lib/sys/os32.h`' [95.21]

'`lib/sys/os32/z_perror.c`' [95.21.9]

'`lib/sys/os32/z_printf.c`' [95.21.10]

'`lib/sys/os32/z_vprintf.c`' [95.21.11]

VEDERE ANCHE

perror(3) [88.90], *printf(3)* [88.91], *putchar(3)* [88.38], *puts(3)* [88.39], *vprintf(3)* [88.137], *vsprintf(3)* [88.137].

87.66 os32: z_perror(2)

«

Vedere *z(2)* [87.65].

87.67 os32: z_printf(2)

«

Vedere *z(2)* [87.65].

87.68 os32: z_vprintf(2)

«

Vedere *z(2)* [87.65].