

## Sezione 2: chiamate di sistema



os16: _Exit(2) .....	3224
os16: _exit(2) .....	3224
os16: chdir(2) .....	3225
os16: chmod(2) .....	3227
os16: chown(2) .....	3229
os16: clock(2) .....	3231
os16: close(2) .....	3232
os16: dup(2) .....	3233
os16: dup2(2) .....	3235
os16: execve(2) .....	3235
os16: fchmod(2) .....	3237
os16: fchown(2) .....	3237
os16: fcntl(2) .....	3238
os16: fork(2) .....	3241
os16: fstat(2) .....	3242
os16: getcwd(2) .....	3242
os16: geteuid(2) .....	3244
os16: getuid(2) .....	3244
os16: getpgrp(2) .....	3245
os16: getpid(2) .....	3245

os16: getppid(2)	3247
os16: kill(2)	3247
os16: link(2)	3248
os16: lseek(2)	3251
os16: mkdir(2)	3252
os16: mknod(2)	3255
os16: mount(2)	3257
os16: open(2)	3259
os16: read(2)	3264
os16: rmdir(2)	3265
os16: seteuid(2)	3267
os16: setpgrp(2)	3267
os16: setuid(2)	3268
os16: signal(2)	3270
os16: sleep(2)	3272
os16: stat(2)	3273
os16: sys(2)	3279
os16: stime(2)	3281
os16: time(2)	3281
os16: umask(2)	3282
os16: umount(2)	3283
os16: unlink(2)	3284

os16: wait(2)	.....	3285					
os16: write(2)	.....	3286					
os16: z(2)	.....	3288					
os16: z_perror(2)	.....	3290					
os16: z_printf(2)	.....	3290					
os16: z_putchar(2)	.....	3290					
os16: z_puts(2)	.....	3290					
os16: z_vprintf(2)	.....	3290					
chdir()	3225	chmod()	3227	chown()	3229	clock()	3231
close()	3232	dup()	3233	dup2()	3233	execve()	3235
fchmod()	3227	fchown()	3229	fcntl()	3238	fork()	3241
fstat()	3273	getcwd()	3242	geteuid()	3244		
getpgrp()	3245	getpid()	3245	getppid()	3245		
getuid()	3244	kill()	3247	link()	3248	lseek()	3251
mkdir()	3252	mknod()	3255	mount()	3257	open()	3259
read()	3264	rmdir()	3265	seteuid()	3268	setpgrp()	
	3267	setuid()	3268	signal()	3270	sleep()	3272
	3273	stime()	3281	sys()	3279	time()	3281
		umask()	3282				
umount()	3257	unlink()	3284	wait()	3285	write()	3286
z_perror()	3288	z_printf()	3288	z_putchar()	3288		
z_puts()	3288	z_vprintf()	3288	_exit()	3224		
_Exit()	3224						

os16: `_Exit(2)`

« Vedere `_exit(2)` [u0.2].

os16: `_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<sub>16</sub>).

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’ [u0.17]

‘lib/unistd/\_exit.c’ [i161.17.1]

‘lib/stdlib.h’ [u0.10]

‘lib/stdlib/\_Exit.c’ [i161.10.1]

‘lib/sys/os16/sys.s’ [i161.12.15]

‘kernel/proc/\_isr.s’ [i160.9.1]

‘kernel/proc/sysroutine.c’ [i160.9.30]

‘kernel/proc/proc\_sys\_exit.c’ [i160.9.22]

## VEDERE ANCHE

*execve(2)* [u0.10], *fork(2)* [u0.14], *kill(2)* [u0.22], *wait(2)* [u0.43], *atexit(3)* [u0.4], *exit(3)* [u0.4].

os16: *chdir(2)*



## NOME

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

## SINTASSI

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

## DESCRIZIONE

La funzione *chdir()* cambia la directory corrente, in modo che quella nuova corrisponda al percorso annotato nella stringa *path*.

## VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	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.

## FILE SORGENTI

'lib/unistd.h' [[u0.17](#)]

'lib/unistd/chdir.c' [[i161.17.3](#)]

'lib/sys/os16/sys.s' [[i161.12.15](#)]

'kernel/proc/\_isr.s' [[i160.9.1](#)]

'kernel/proc/sysroutine.c' [[i160.9.30](#)]

'kernel/fs/path\_chdir.c' [[i160.4.32](#)]

## VEDERE ANCHE

*rmdir*(2) [[u0.30](#)], *access*(3) [[u0.1](#)].

## 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 <sub>8</sub>	Lettura, scrittura ed esecuzione per l'utente proprietario.
S_IRUSR	00400 <sub>8</sub>	Lettura per l'utente proprietario.
S_IWUSR	00200 <sub>8</sub>	Scrittura per l'utente proprietario.
S_IXUSR	00100 <sub>8</sub>	Esecuzione per l'utente proprietario.
S_IRWXG	00070 <sub>8</sub>	Lettura, scrittura ed esecuzione per il gruppo.
S_IRGRP	00040 <sub>8</sub>	Lettura per il gruppo.
S_IWGRP	00020 <sub>8</sub>	Scrittura per il gruppo.
S_IXGRP	00010 <sub>8</sub>	Esecuzione per il gruppo.
S_IRWXO	00007 <sub>8</sub>	Lettura, scrittura ed esecuzione per gli altri utenti.
S_IROTH	00004 <sub>8</sub>	Lettura per gli altri utenti.
S_IWOTH	00002 <sub>8</sub>	Scrittura per gli altri utenti.
S_IXOTH	00001 <sub>8</sub>	Esecuzione per gli altri utenti.

os16 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 <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	Significato
EACCES	Permesso negato.
EBADF	Il descrittore del file richiesto non è valido.

## FILE SORGENTI

‘lib/sys/stat.h’ [u0.13]

‘lib/sys/stat/chmod.c’ [i161.13.1]

‘lib/sys/stat/fchmod.c’ [i161.13.2]

‘lib/sys/os16/sys.s’ [i161.12.15]

‘kernel/proc/\_isr.s’ [i160.9.1]

‘kernel/proc/sysroutine.c’ [i160.9.30]

‘kernel/fs/path\_chmod.c’ [i160.4.33]

## VEDERE ANCHE

*chmod(1)* [u0.5], *chown(2)* [u0.5], *open(2)* [u0.28], *stat(2)* [u0.36].

os16: *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 <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	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 `os16` non valuta i permessi di accesso ai file, relativi ai gruppi.

## FILE SORGENTI

‘lib/unistd.h’ [u0.17]

‘lib/unistd/chown.c’ [i161.17.4]

‘lib/sys/os16/sys.s’ [i161.12.15]

‘kernel/proc/\_isr.s’ [i160.9.1]

‘kernel/proc/sysroutine.c’ [i160.9.30]

‘kernel/fs/path\_chown.c’ [i160.4.34]

‘kernel/fs/fd\_chown.c’ [i160.4.2]

## VEDERE ANCHE

`chmod(2)` [u0.4].

`os16: 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 `os16`, come dichiarato nel file `'time.h'`, il valore di ***CLOCKS\_PER\_SEC*** è 18, essendo la frequenza di CPU poco più di 18 Hz.

## VALORE RESTITUITO

La funzione restituisce il tempo di CPU, espresso in multipli di 1/18 di secondo.

## FILE SORGENTI

`'lib/time.h'` [[u0.16](#)]

`'lib/time/clock.c'` [[i161.16.2](#)]

`'lib/sys/os16/sys.s'` [[i161.12.15](#)]

`'kernel/proc/_isr.s'` [[i160.9.1](#)]

`'kernel/proc/sysroutine.c'` [[i160.9.30](#)]

`'kernel/k_libc/k_clock.c'` [[i160.6.1](#)]

## VEDERE ANCHE

`time(2)` [[u0.39](#)].

`os16: 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 <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	Significato
EBADF	Il descrittore del file non è valido.

## FILE SORGENTI

'lib/unistd.h' [[u0.17](#)]

'lib/unistd/close.c' [[i161.17.5](#)]

'lib/sys/os16/sys.s' [[i161.12.15](#)]

'kernel/proc/\_isr.s' [[i160.9.1](#)]

'kernel/proc/sysroutine.c' [[i160.9.30](#)]

'kernel/fs/fd\_close.c' [[i160.4.3](#)]

## VEDERE ANCHE

*fcntl*(2) [[u0.13](#)], *open*(2) [[u0.28](#)], *fclose*(3) [[u0.27](#)].

os16: 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 <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	Significato
EBADF	Uno dei descrittori specificati non è valido.
EMFILE	Troppi file aperti per il processo.

## FILE SORGENTI

‘lib/unistd.h’ [[u0.17](#)]

‘lib/unistd/dup.c’ [[i161.17.6](#)]

‘lib/unistd/dup2.c’ [[i161.17.7](#)]

‘lib/sys/os16/sys.s’ [[i161.12.15](#)]

‘kernel/proc/\_isr.s’ [[i160.9.1](#)]

‘kernel/proc/sysroutine.c’ [[i160.9.30](#)]

‘kernel/fs/fd\_dup.c’ [[i160.4.4](#)]

‘kernel/fs/fd\_dup2.c’ [[i160.4.5](#)]

## VEDERE ANCHE

*close(2)* [u0.7], *fcntl(2)* [u0.13], *open(2)* [u0.28].

os16: *dup2(2)*

Vedere *dup(2)* [u0.8].

os16: *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 <i>errno</i>	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 accesso necessari.
ENOEXEC	Il file non può essere un file eseguibile, perché non ne ha le caratteristiche.
EIO	Errore di input-output.



## DIFETTI

os16 non prevede l'interpretazione di script, perché non esiste alcun programma in grado di farlo. Anche la shell di os16 si limita a eseguire i comandi inseriti, ma non può interpretare un file.

## FILE SORGENTI

'lib/unistd.h' [[u0.17](#)]

'lib/unistd/execle.c' [[i161.17.10](#)]

'lib/sys/os16/sys.s' [[i161.12.15](#)]

'kernel/proc/\_isr.s' [[i160.9.1](#)]

'kernel/proc/sysroutine.c' [[i160.9.30](#)]

'kernel/proc/proc\_sys\_exec.c' [[i160.9.21](#)]

## VEDERE ANCHE

*fork(2)* [[u0.14](#)], *exec(3)* [[u0.20](#)], *getopt(3)* [[u0.52](#)], *environ(7)* [[u0.1](#)].

os16: *fchmod(2)*

Vedere *chmod(2)* [[u0.4](#)].

«

os16: *fchown(2)*

Vedere *chown(2)* [[u0.5](#)].

«

## os16: 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
<pre>fcntl (<i>fdn</i>, F_DUPFD,       (int) <i>fdn_min</i>)</pre>	<p>Richiede la duplicazione del descrittore di file <i>fdn</i>, in modo tale che la copia abbia il numero di descrittore minore possibile, ma maggiore o uguale a quello indicato come argomento <i>fdn_min</i>.</p>
<pre>fcntl (<i>fdn</i>, F_GETFD) fcntl (<i>fdn</i>, F_SETFD,       (int) <i>fd_flags</i>)</pre>	<p>Rispettivamente, legge o imposta, gli indicatori del descrittore di file <i>fdn</i> (eventualmente noti come <i>file descriptor flags</i> o solo <i>fd flags</i>). Per il momento, è possibile impostare un solo indicatore, ‘<b>FD_CLOEXEC</b>’, pertanto, al posto di <i>fd_flags</i> si può mettere solo la costante ‘<b>FD_CLOEXEC</b>’.</p>
<pre>fcntl (<i>fdn</i>, F_GETFL) fcntl (<i>fdn</i>, F_SETFL,       (int) <i>fl_flags</i>)</pre>	<p>Rispettivamente, legge o imposta, gli indicatori dello stato del file, relativi al descrittore <i>fdn</i> (eventualmente noti come <i>file flaga</i> o solo <i>fl flags</i>). Per impostare questi indicatori, vanno combinate delle costanti simboliche: ‘<b>O_RDONLY</b>’, ‘<b>O_WRONLY</b>’, ‘<b>O_RDWR</b>’, ‘<b>O_CREAT</b>’, ‘<b>O_EXCL</b>’, ‘<b>O_NOCTTY</b>’, ‘<b>O_TRUNC</b>’.</p>

## 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 ( <i>fd flags</i> ), oppure -1 in caso di errore.
F_GETFL	Si ottiene il valore degli indicatori del file ( <i>fl flags</i> ), 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 os16.
altri tipi di operazione	Si ottiene 0 in caso di successo, oppure -1 in caso di errore.

## ERRORI

Valore di <i>errno</i>	Significato
E_NOT_IMPLEMENTED	È stato richiesto un tipo di operazione che non è disponibile nel caso particolare di os16.
EINVAL	È stato richiesto un tipo di operazione non valido.

## FILE SORGENTI

‘lib/fcntl.h’ [[u0.4](#)]

‘lib/fcntl/fcntl.c’ [[i161.4.2](#)]

## VEDERE ANCHE

*dup(2)* [u0.8], *dup2(2)* [u0.8], *open(2)* [u0.28].

os16: 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 <i>errno</i>	Significato
ENOMEM	Memoria insufficiente per avviare un altro processo.

## FILE SORGENTI

'lib/unistd.h' [[u0.17](#)]

'lib/unistd/fork.c' [[i161.17.17](#)]

'lib/sys/os16/sys.s' [[i161.12.15](#)]

'kernel/proc/\_isr.s' [[i160.9.1](#)]

'kernel/proc/sysroutine.c' [[i160.9.30](#)]

'kernel/proc/proc\_sys\_fork.c' [[i160.9.23](#)]

## VEDERE ANCHE

*execve(2)* [[u0.10](#)], *wait(2)* [[u0.43](#)], *exec(3)* [[u0.20](#)].

os16: *fstat(2)*

« Vedere *stat(2)* [[u0.36](#)].

os16: *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 <i>errno</i>	Significato
EINVAL	Il puntatore <i>buffer</i> non è valido.
E_LIMIT	Il percorso della directory corrente è troppo lungo, rispetto ai limiti realizzativi di os16.

## DIFETTI

La funzione *getcwd()* di os16 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' [[u0.17](#)]

'lib/unistd/getcwd.c' [[i161.17.18](#)]

'lib/sys/os16/sys.s' [[i161.12.15](#)]

'kernel/proc/\_isr.s' [[i160.9.1](#)]

'kernel/proc/sysroutine.c' [i160.9.30]

## VEDERE ANCHE

*chdir(2)* [u0.3].

os16: *geteuid(2)*

« Vedere *getuid(2)* [u0.18].

os16: *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 os16 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' [[u0.17](#)]

'lib/unistd/getuid.c' [[i161.17.24](#)]

'lib/unistd/geteuid.c' [[i161.17.19](#)]

'lib/sys/os16/sys.s' [[i161.12.15](#)]

'kernel/proc/\_isr.s' [[i160.9.1](#)]

'kernel/proc/sysroutine.c' [[i160.9.30](#)]

## VEDERE ANCHE

*setuid(2)* [[u0.33](#)].

os16: *getpgrp(2)*

Vedere *getpid(2)* [[u0.20](#)].

os16: *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 os16 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' [[u0.17](#)]  
'lib/unistd/getpid.c' [[i161.17.22](#)]  
'lib/unistd/getppid.c' [[i161.17.23](#)]  
'lib/unistd/getpgrp.c' [[i161.17.21](#)]  
'lib/sys/os16/sys.s' [[i161.12.15](#)]

‘kernel/proc/\_isr.s’ [i160.9.1]

‘kernel/proc/sysroutine.c’ [i160.9.30]

## VEDERE ANCHE

*getuid(2)* [u0.18] *fork(2)* [u0.14], *execve(2)* [u0.10].

os16: *getppid(2)*

Vedere *getpid(2)* [u0.20].

os16: *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 os16 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 *eu*id), ma se il processo che chiama la funzione lavora con un valore di *eu*id 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 <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	Significato
EPERM	Il processo non ha i permessi per inviare il segnale alla destinazione richiesta.
ESRCH	La ricerca del processo <i>pid</i> è fallita. Nel caso di os16, si ottiene questo errore anche per valori negativi di <i>pid</i> .

## FILE SORGENTI

`lib/sys/types.h` [[u0.14](#)]

`lib/signal.h` [[u0.8](#)]

`lib/signal/kill.c` [[i161.8.1](#)]

`lib/sys/os16/sys.s` [[i161.12.15](#)]

`kernel/proc/_isr.s` [[i160.9.1](#)]

`kernel/proc/sysroutine.c` [[i160.9.30](#)]

`kernel/proc/proc_sys_kill.c` [[i160.9.24](#)]

## VEDERE ANCHE

*signal(2)* [[u0.34](#)].

## 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 <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	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’ [[u0.17](#)]

‘lib/unistd/link.c’ [[i161.17.26](#)]

‘lib/sys/os16/sys.s’ [[i161.12.15](#)]

‘kernel/proc/\_isr.s’ [[i160.9.1](#)]

‘kernel/proc/sysroutine.c’ [[i160.9.30](#)]

‘kernel/fs/path\_link.c’ [[i160.4.40](#)]

## VEDERE ANCHE

*ln(1)* [[u0.11](#)] *open(2)* [[u0.28](#)], *stat(2)* [[u0.36](#)], *unlink(2)* [[u0.42](#)].

## 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 <i>whence</i>	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 <i>errno</i>	Significato
EBADF	Il numero del descrittore di file non è valido.
EINVAL	Il valore di <i>whence</i> non è contemplato, oppure la combinazione tra <i>whence</i> e <i>offset</i> non è valida.

## FILE SORGENTI

'lib/unistd.h' [[u0.17](#)]

'lib/unistd/lseek.c' [[i161.17.27](#)]

'lib/sys/os16/sys.s' [[i161.12.15](#)]

'kernel/proc/\_isr.s' [[i160.9.1](#)]

'kernel/proc/sysroutine.c' [[i160.9.30](#)]

'kernel/fs/fd\_lseek.c' [[i160.4.7](#)]

## VEDERE ANCHE

*dup(2)* [[u0.8](#)] *fork(2)* [[u0.14](#)], *open(2)* [[u0.28](#)], *fseek(3)* [[u0.43](#)].

os16: *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)* [[u0.40](#)]).

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 <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	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'` [[u0.13](#)]

`'lib/sys/stat/mkdir.c'` [[i161.13.4](#)]

`'lib/sys/os16/sys.s'` [[i161.12.15](#)]

`'kernel/proc/_isr.s'` [[i160.9.1](#)]

`'kernel/proc/sysroutine.c'` [[i160.9.30](#)]

`'kernel/fs/path_mkdir.c'` [[i160.4.41](#)]

## VEDERE ANCHE

*mkdir(1)* [[u0.15](#)], *chmod(2)* [[u0.4](#)], *chown(2)* [[u0.5](#)], *mknod(2)* [[u0.26](#)], *mount(2)* [[u0.27](#)], *stat(2)* [[u0.36](#)], *umask(2)* [[u0.40](#)], *unlink(2)* [[u0.42](#)].

## 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)* [[u0.36](#)], tenendo conto che os16 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)* [u0.36]).

## VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
-1	Errore, descritto dal contenuto della variabile <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	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’ [u0.13]

‘lib/sys/stat/mknod.c’ [i161.13.5]

‘lib/sys/os16/sys.s’ [i161.12.15]

‘kernel/proc/\_isr.s’ [i160.9.1]

‘kernel/proc/sysroutine.c’ [i160.9.30]

‘kernel/fs/path\_mknod.c’ [i160.4.42]

## VEDERE ANCHE

*mkdir(2)* [u0.25], *chmod(2)* [u0.4], *chown(2)* [u0.5], *fcntl(2)* [u0.13], *stat(2)* [u0.36], *umask(2)* [u0.40], *unlink(2)* [u0.42].

## os16: mount(2)



## NOME

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

## SINTASSI

```
#include <sys/os16.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 <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	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 innestare 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/os16.h' [[u0.12](#)]

'lib/sys/os16/mount.c' [[i161.12.12](#)]

'lib/sys/os16/umount.c' [[i161.12.16](#)]

'lib/sys/os16/sys.s' [[i161.12.15](#)]

'kernel/proc/\_isr.s' [[i160.9.1](#)]

'kernel/proc/sysroutine.c' [[i160.9.30](#)]

'kernel/fs/path\_mount.c' [i160.4.43]

'kernel/fs/path\_umount.c' [i160.4.45]

## VEDERE ANCHE

*mount(8)* [u0.4], *umount(8)* [u0.4].

os16: 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 'O_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 <i>mode</i> .
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 os16 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 <sub>8</sub>	Lettura, scrittura ed esecuzione per l'utente proprietario.
S_IRUSR	00400 <sub>8</sub>	Lettura per l'utente proprietario.
S_IWUSR	00200 <sub>8</sub>	Scrittura per l'utente proprietario.
S_IXUSR	00100 <sub>8</sub>	Esecuzione per l'utente proprietario.

Modalità simbolica	Modalità numerica	Descrizione
S_IRWXG	00070 <sub>8</sub>	Lettura, scrittura ed esecuzione per il gruppo.
S_IRGRP	00040 <sub>8</sub>	Lettura per il gruppo.
S_IWGRP	00020 <sub>8</sub>	Scrittura per il gruppo.
S_IXGRP	00010 <sub>8</sub>	Esecuzione per il gruppo.

Modalità simbolica	Modalità numerica	Descrizione
S_IRWXO	00007 <sub>8</sub>	Lettura, scrittura ed esecuzione per gli altri utenti.
S_IROTH	00004 <sub>8</sub>	Lettura per gli altri utenti.
S_IWOTH	00002 <sub>8</sub>	Scrittura per gli altri utenti.
S_IXOTH	00001 <sub>8</sub>	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 <i>errno</i>	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' [[u0.14](#)]

'lib/sys/stat.h' [[u0.13](#)]

'lib/fcntl.h' [[u0.4](#)]

'lib/fcntl/open.c' [[i161.4.3](#)]

## VEDERE ANCHE

*chmod(2)* [[u0.4](#)], *chown(2)* [[u0.5](#)], *close(2)* [[u0.7](#)], *dup(2)* [[u0.8](#)], *fcntl(2)* [[u0.13](#)], *link(2)* [[u0.24](#)], *mknod(2)* [[u0.26](#)], *mount(2)*

[u0.27], *read*(2) [u0.29], *stat*(2) [u0.36], *umask*(2) [u0.40], *unlink*(2) [u0.42], *write*(2) [u0.44], *fopen*(3) [u0.35].

os16: *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 <i>errno</i>	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 os16.

## FILE SORGENTI

‘lib/unistd.h’ [u0.17]

‘lib/unistd/read.c’ [i161.17.28]

‘lib/sys/os16/sys.s’ [i161.12.15]

‘kernel/fs/fd\_read.c’ [i160.4.9]

## VEDERE ANCHE

*close(2)* [u0.7] *open(2)* [u0.28], *write(2)* [u0.44].

os16: *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 <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	Significato
EINVAL	Il percorso <i>path</i> 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' [[u0.17](#)]

'lib/unistd/rmdir.c' [[i161.17.29](#)]

'lib/sys/os16/sys.s' [[i161.12.15](#)]

'kernel/fs/path\_unlink.c' [[i160.4.46](#)]

## VEDERE ANCHE

*mkdir(2)* [[u0.25](#)], *unlink(2)* [[u0.42](#)].

os16: seteuid(2)

Vedere *setuid(2)* [[u0.33](#)].

os16: 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’ [[u0.17](#)]

‘lib/unistd/setpgrp.c’ [[i161.17.31](#)]

‘lib/sys/os16/sys.s’ [[i161.12.15](#)]

‘kernel/proc/\_isr.s’ [[i160.9.1](#)]

‘kernel/proc/sysroutine.c’ [[i160.9.30](#)]

## VEDERE ANCHE

`getpgrp(2)` [u0.32], `getuid(2)` [u0.33].

os16: `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 <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	Significato
EPERM	Non si dispone dei permessi necessari a eseguire il cambiamento di identità.

## FILE SORGENTI

'lib/unistd.h' [[u0.17](#)]

`'lib/unistd/setuid.c'` [i161.17.32]

`'lib/unistd/seteuid.c'` [i161.17.30]

`'lib/sys/os16/sys.s'` [i161.12.15]

`'kernel/proc/_isr.s'` [i160.9.1]

`'kernel/proc/sysroutine.c'` [i160.9.30]

## VEDERE ANCHE

`getuid(2)` [u0.33].

## os16: 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 os16 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 <i>errno</i>	Significato
EINVAL	Il numero <i>sig</i> o il valore di <i>handler</i> 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)*’. `os16` 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`’ [[u0.8](#)]

‘lib/signal/signal.c’ [i161.8.2]

‘kernel/proc/\_isr.s’ [i160.9.1]

‘kernel/proc/sysroutine.c’ [i160.9.30]

‘kernel/proc/proc\_sys\_signal.c’ [i160.9.27]

## VEDERE ANCHE

*kill(2)* [u0.22].

os16: 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’ [u0.17]

‘lib/unistd/sleep.c’ [i161.17.33]

‘lib/sys/os16/sys.s’ [i161.12.15]

‘kernel/proc/\_isr.s’ [i160.9.1]

‘kernel/proc/sysroutine.c’ [i160.9.30]

## VEDERE ANCHE

*signal(2)* [u0.34].

os16: 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 os16, 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)* [u0.4], 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 ( <i>m</i> )	È un file di dispositivo a blocchi?
S_ISCHR ( <i>m</i> )	È un file di dispositivo a caratteri?
S_ISFIFO ( <i>m</i> )	È un file FIFO?
S_ISREG ( <i>m</i> )	È un file puro e semplice?
S_ISDIR ( <i>m</i> )	È una directory?
S_ISLNK ( <i>m</i> )	È un collegamento simbolico?
S_ISSOCK ( <i>m</i> )	È un socket di dominio Unix?

Naturalmente, anche se nel file system possono esistere file di ogni tipo, poi os16 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 <sub>8</sub>	Maschera che raccoglie tutti i bit che individuano il tipo di file.
S_IFBLK	0060000 <sub>8</sub>	File di dispositivo a blocchi.
S_IFCHR	0020000 <sub>8</sub>	File di dispositivo a caratteri.
S_IFIFO	0010000 <sub>8</sub>	File FIFO.
S_IFREG	0100000 <sub>8</sub>	File puro e semplice.
S_IFDIR	0040000 <sub>8</sub>	Directory.
S_IFLNK	0120000 <sub>8</sub>	Collegamento simbolico.
S_IFSOCK	0140000 <sub>8</sub>	Socket di dominio Unix.

Modalità simbolica	Modalità numerica	Descrizione
S_ISUID	0004000 <sub>8</sub>	SUID.
S_ISGID	0002000 <sub>8</sub>	SGID.
S_ISVTX	0001000 <sub>8</sub>	Sticky.



Modalità simbolica	Modalità numerica	Descrizione
S_IRWXU	0000700 <sub>8</sub>	Lettura, scrittura ed esecuzione per l'utente proprietario.
S_IRUSR	0000400 <sub>8</sub>	Lettura per l'utente proprietario.
S_IWUSR	0000200 <sub>8</sub>	Scrittura per l'utente proprietario.
S_IXUSR	0000100 <sub>8</sub>	Esecuzione per l'utente proprietario.

Modalità simbolica	Modalità numerica	Descrizione
S_IRWXG	0000070 <sub>8</sub>	Lettura, scrittura ed esecuzione per il gruppo.
S_IRGRP	0000040 <sub>8</sub>	Lettura per il gruppo.
S_IWGRP	0000020 <sub>8</sub>	Scrittura per il gruppo.
S_IXGRP	0000010 <sub>8</sub>	Esecuzione per il gruppo.

Modalità simbolica	Modalità numerica	Descrizione
S_IRWXO	0000007 <sub>8</sub>	Lettura, scrittura ed esecuzione per gli altri utenti.
S_IROTH	0000004 <sub>8</sub>	Lettura per gli altri utenti.
S_IWOTH	0000002 <sub>8</sub>	Scrittura per gli altri utenti.
S_IXOTH	0000001 <sub>8</sub>	Esecuzione per gli altri utenti.

os16 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 <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	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' [[u0.13](#)]

'lib/sys/stat/stat.c' [[i161.13.6](#)]

'lib/sys/stat/fstat.c' [[i161.13.3](#)]

'lib/sys/os16/sys.s' [[i161.12.15](#)]

'kernel/proc/\_isr.s' [[i160.9.1](#)]

'kernel/proc/sysroutine.c' [[i160.9.30](#)]

'kernel/fs/path\_stat.c' [[i160.4.44](#)]

'kernel/fs/fd\_stat.c' [[i160.4.11](#)]

## VEDERE ANCHE

*chmod(2)* [[u0.4](#)], *chown(2)* [[u0.5](#)].

## NOME

‘**sys**’ - chiamata di sistema

## SINTASSI

```
#include <sys/os16.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
<code>char path[PATH_MAX];</code>	Un percorso di file o directory.
<code>... ret;</code>	Serve a contenere il valore restituito dalla funzione che nel kernel compie effettivamente il lavoro. Il tipo del membro varia caso per caso.
<code>int errno;</code>	Serve a contenere il numero dell'errore prodotto dalla funzione che nel kernel compie effettivamente il lavoro.
<code>int errln;</code>	Serve a contenere il numero della riga di codice in cui si è prodotto un errore.
<code>char errfn[PATH_MAX];</code>	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/os16.h' [[u0.12](#)]

'lib/sys/os16/sys.s' [[i161.12.15](#)]

'kernel/proc/\_isr.s' [[i160.9.1](#)]

'kernel/proc/sysroutine.c' [[i160.9.30](#)]

os16: stime(2)

Vedere *time(2)* [[u0.39](#)].

os16: 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 funzione *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’ [u0.16]

‘lib/time/time.c’ [i161.16.6]

‘lib/time/stime.c’ [i161.16.5]

‘lib/sys/os16/sys.s’ [i161.12.15]

‘kernel/proc/\_isr.s’ [i160.9.1]

‘kernel/proc/sysroutine.c’ [i160.9.30]

‘kernel/k\_libc.h’ [u0.6]

‘kernel/k\_libc/k\_time.c’ [i160.6.11]

‘kernel/k\_libc/k\_stime.c’ [i160.6.10]

## VEDERE ANCHE

*date(1)* [u0.8], *ctime(3)* [u0.13].

os16: 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 permessi 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’ [[u0.13](#)]

‘lib/sys/stat/umask.c’ [[i161.13.7](#)]

‘lib/sys/os16/sys.s’ [[i161.12.15](#)]

‘kernel/proc/\_isr.s’ [[i160.9.1](#)]

‘kernel/proc/sysroutine.c’ [[i160.9.30](#)]

## VEDERE ANCHE

*mkdir(2)* [[u0.25](#)], *chmod(2)* [[u0.4](#)], *open(2)* [[u0.28](#)], *stat(2)* [[u0.36](#)].

os16: umount(2)

«  
Vedere *mount(2)* [[u0.27](#)].

os16: 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 <i>errno</i> viene impostata di conseguenza.



## ERRORI

Valore di <i>errno</i>	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'` [[u0.17](#)]

`'lib/unistd/unlink.c'` [[i161.17.35](#)]

`'lib/sys/os16/sys.s'` [[i161.12.15](#)]

`'kernel/proc/_isr.s'` [[i160.9.1](#)]

`'kernel/proc/sysroutine.c'` [[i160.9.30](#)]

`'kernel/fs/path_unlink.c'` [[i160.4.46](#)]

## VEDERE ANCHE

`rm(1)` [[u0.18](#)], `link(2)` [[u0.23](#)], `rmdir(2)` [[u0.30](#)].

os16: `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 <i>errno</i>	Significato
ECHILD	Non ci sono processi figli da attendere.

## FILE SORGENTI

'lib/sys/wait.h' [[u0.15](#)]

'lib/sys/wait/wait.c' [[i161.15.1](#)]

'lib/sys/os16/sys.s' [[i161.12.15](#)]

'kernel/proc/\_isr.s' [[i160.9.1](#)]

'kernel/proc/sysroutine.c' [[i160.9.30](#)]

'kernel/proc/proc\_sys\_wait.c' [[i160.9.28](#)]

## VEDERE ANCHE

*\_exit(2)* [[u0.2](#)], *fork(2)* [[u0.14](#)], *kill(2)* [[u0.22](#)], *signal(2)* [[u0.34](#)].

## 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 <i>errno</i>	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 os16.
EIO	Errore di input-output.

## FILE SORGENTI

'lib/unistd.h' [[u0.17](#)]

'lib/unistd/write.c' [[i161.17.36](#)]

'lib/sys/os16/sys.s' [[i161.12.15](#)]

'kernel/proc/\_isr.s' [[i160.9.1](#)]

'kernel/proc/sysroutine.c' [[i160.9.30](#)]

'kernel/fs/fd\_write.c' [[i160.4.12](#)]

## VEDERE ANCHE

*close*(2) [[u0.7](#)], *lseek*(2) [[u0.24](#)], *open*(2) [[u0.28](#)], *read*(2) [[u0.29](#)], *fwrite*(3) [[u0.48](#)].

os16: *z*(2)

«

## NOME

'z\_...' - funzioni provvisorie

# SINTASSI

```
#include <sys/os16.h>
void z_perror    (const char *string);
int  z_printf   (char *format, ...);
int  z_putchar  (int c);
int  z_puts     (char *string);
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 os16, 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 os16 e, d’altro canto, possono servire se si rende necessario aggirare la gestione dei dispositivi per visualizzare dei messaggi sullo schermo.

## FILE SORGENTI

‘lib/sys/os16.h’ [[u0.12](#)]  
‘lib/sys/os16/z\_perror.c’ [[i161.12.17](#)]  
‘lib/sys/os16/z\_printf.c’ [[i161.12.18](#)]  
‘lib/sys/os16/z\_putchar.c’ [[i161.12.19](#)]  
‘lib/sys/os16/z\_puts.c’ [[i161.12.20](#)]  
‘lib/sys/os16/z\_vprintf.c’ [[i161.12.21](#)]

## VEDERE ANCHE

*perror(3)* [u0.77], *printf(3)* [u0.78], *putchar(3)* [u0.37], *puts(3)* [u0.38], *vprintf(3)* [u0.128], *vsprintf(3)* [u0.128].

os16: *z\_perror(2)*

« Vedere *z(2)* [u0.45].

os16: *z\_printf(2)*

« Vedere *z(2)* [u0.45].

os16: *z\_putchar(2)*

« Vedere *z(2)* [u0.45].

os16: *z\_puts(2)*

« Vedere *z(2)* [u0.45].

os16: *z\_vprintf(2)*

« Vedere *z(2)* [u0.45].