

## Sezione 3: funzioni di libreria

### 88.1 os32: \_gcc(3)

#### NOME

‘\_gcc’ - libreria per il compilatore

#### DESCRIZIONE

Le funzioni descritte dal file di intestazione ‘lib/gcc.h’ e contenute nella directory ‘lib/\_gcc/’, servono al compilatore GNU C per compiere il proprio lavoro correttamente con valori da 64 bit.

#### FILE SORGENTI

‘lib/\_gcc.h’ [95.2].

### 88.2 os32: abort(3)

#### NOME

‘abort’ - conclusione anormale del processo

#### SINTASSI

```
#include <stdlib.h>
void abort (void);
```

#### DESCRIZIONE

La funzione *abort()* verifica lo stato di configurazione del segnale ‘SIGABRT’ e, se risulta bloccato, lo sblocca, quindi invia questo segnale per il processo in corso. Ciò provoca la conclusione del

processo, secondo la modalità prevista per tale segnale, a meno che il segnale sia stato ridiretto a una funzione, nel qual caso, dopo l'invio del segnale, potrebbe esserci anche una ripresa del controllo da parte della funzione *abort()*. Tuttavia, se così fosse, il segnale '**SIGABRT**' verrebbe poi riconfigurato alla sua impostazione normale e verrebbe inviato nuovamente lo stesso segnale per provocare la conclusione del processo. Pertanto, la funzione *abort()* non restituisce il controllo.

Va comunque osservato che os32 non è in grado di associare una funzione a un segnale, pertanto, i segnali possono solo avere una gestione predefinita, o al massimo risultare bloccati.

## FILE SORGENTI

'lib/stdlib.h' [95.19]

'lib/stdlib/abort.c' [95.19.2]

## VEDERE ANCHE

*signal(2)* [87.52].

## 88.3 os32: abs(3)

«

### NOME

'**abs**', '**labs**', '**llabs**', '**imaxabs**' - valore assoluto di un numero intero

### SINTASSI

```
#include <stdlib.h>
int abs (int j);
long int labs (long int j);
long long int llabs (long long int j);
```

```
#include <inttypes.h>
intmax_t imaxabs (intmax_t j);
```

## DESCRIZIONE

Le funzioni ‘...**abs** ()’ restituiscono il valore assoluto del loro argomento, distinguendosi per tipo di intero.

## VALORE RESTITUITO

Il valore assoluto del numero intero fornito come argomento.

## FILE SORGENTI

‘lib/stdlib.h’ [95.19]

‘lib/stdlib/abs.c’ [95.19.3]

‘lib/stdlib/labs.c’ [95.19.11]

‘lib/stdlib/llabs.c’ [95.19.13]

‘lib/inttypes.h’ [95.8]

‘lib/inttypes/imaxabs.c’ [95.8.1]

## VEDERE ANCHE

*div(3)* [88.17], *ldiv(3)* [88.17], *lldiv(3)* [88.17], *imaxdiv(3)* [88.17], *rand(3)* [88.97].

## 88.4 os32: access(3)

### NOME

‘**access**’ - verifica dei permessi di accesso dell’utente

## SINTASSI

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

## DESCRIZIONE

La funzione *access()* verifica lo stato di accessibilità del file indicato nella stringa *path*, secondo i permessi stabiliti con il parametro *mode*.

L'argomento corrispondente al parametro *mode* può assumere un valore corrispondente alla macro-variabile *F\_OK*, per verificare semplicemente l'esistenza del file specificato; altrimenti, può avere un valore composto dalla combinazione (con l'operatore OR binario) di *R\_OK*, *W\_OK* e *X\_OK*, per verificare, rispettivamente, l'accessibilità in lettura, in scrittura e in esecuzione. Queste macro-variabili sono dichiarate nel file 'unistd.h'.

## VALORE RESTITUITO

Valore	Significato
0	I permessi di accesso richiesti sono tutti disponibili.
-1	I permessi non sono tutti disponibili, oppure si è verificato un errore di altro genere, da chiarire analizzando la variabile <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	Significato
ENFILE	Troppi file aperti nel sistema.
ENOENT	File non trovato.
EACCES	Permesso negato.

## DIFETTI

Questa realizzazione della funzione *access()* determina l'accessibilità a un file attraverso le informazioni che può trarre autonomamente; pertanto, si tratta di una valutazione presunta e non reale.

## FILE SORGENTI

'lib/unistd.h' [95.30]

'lib/unistd/access.c' [95.30.2]

## VEDERE ANCHE

*stat(2)* [87.55].

88.5 os32: *asctime(3)*

Vedere *ctime(3)* [88.15].

88.6 os32: *assert(3)*

## NOME

'**assert**' - verifica diagnostica del risultato di un'espressione

## SINTASSI

```
#include <assert.h>
void assert (tipo_scalare espressione);
```

## DESCRIZIONE

Il file ‘`assert.h`’ della libreria standard definisce la macroistruzione ***assert()***, da usare per generare informazioni diagnostiche, sulla base dell’esito della valutazione di un’espressione.

La macroistruzione ***assert()*** viene definita in due modi alternativi, in base alla presenza o meno della macro-variabile ***NDEBUG***. Per la precisione, in presenza della macro-variabile ***NDEBUG*** la macroistruzione ***assert()*** risulta inerte.

La macroistruzione ***assert()*** va usata con la sintassi indicata, dove il parametro indica un’espressione di tipo non specificato, purché di tipo scalare. Se l’espressione si traduce in un valore *Falso*, ovvero pari a zero, la macroistruzione emette, attraverso lo standard error, un messaggio contenente l’espressione stessa e altre indicazioni. Precisamente, oltre all’espressione appaiono: il nome della funzione in cui ci si trova, il nome del file (sorgente) e il numero della riga.

## FILE SORGENTI

‘`lib/assert.h`’ [[95.1.3](#)]

## 88.7 os32: atexit(3)

«

## NOME

‘***atexit***’, ‘***exit***’ - gestione della chiusura dei processi

## SINTASSI

```
#include <stdlib.h>
typedef void (*atexit_t) (void);
int  atexit (atexit_t function);
void exit   (int status);
```

## DESCRIZIONE

La funzione *exit()* conclude il processo in corso, avvalendosi della chiamata di sistema *\_exit(2)* [87.2], ma prima di farlo, scandisce un array contenente un elenco di funzioni, da eseguire prima di tale chiamata finale. Questo array viene popolato eventualmente con l'aiuto della funzione *atexit()*, sapendo che l'ultima funzione di chiusura aggiunta, è la prima a dover essere eseguita alla conclusione.

La funzione *atexit()* riceve come argomento il puntatore a una funzione che non prevede argomenti e non restituisce alcunché. Per facilitare la dichiarazione del prototipo e, di conseguenza, dell'array usato per accumulare tali puntatori, il file di intestazione 'stdlib.h' di os32 dichiara un tipo speciale, non standard, denominato 'atexit\_t', definito come:

```
typedef void (*atexit_t) (void);
```

Si possono annotare un massimo di *ATEXIT\_MAX* funzioni da eseguire prima della conclusione di un processo. Tale macro-variabile è definita nel file 'limits.h'.

## VALORE RESTITUITO

Solo la funzione *atexit()* restituisce un valore, perché *exit()* non può nemmeno restituire il controllo.

Valore	Significato
0	Operazione conclusa con successo.
-1	Errore, dovuto all'esaurimento dello spazio nell'array usato per accumulare le funzioni di chiusura.

## FILE SORGENTI

'lib/limits.h' [95.1.6]

'lib/stdlib.h' [95.19]

'lib/stdlib/atexit.c' [95.19.4]

'lib/stdlib/exit.c' [95.19.9]

## VEDERE ANCHE

`_exit(2)` [87.2], `_Exit(2)` [87.2].

## 88.8 os32: atoi(3)

«

### NOME

'atoi', 'atol' - conversione da stringa a numero intero

### SINTASSI

```
#include <stdlib.h>
int atoi (const char *string);
long int atol (const char *string);
```

### DESCRIZIONE

Le funzioni 'ato... ()' convertono una stringa, fornita come argomento, in un numero intero. La conversione avviene escludendo gli spazi iniziali, considerando eventualmente un segno («+» o «-») e poi soltanto i caratteri che rappresentano cifre numeriche.



La scansione della stringa e l'interpretazione del valore numerico contenuto terminano quando si incontra un carattere diverso dalle cifre numeriche.

## VALORE RESTITUITO

Il valore numerico ottenuto dall'interpretazione della stringa.

## FILE SORGENTI

'lib/stdlib.h' [95.19]

'lib/stdlib/atoi.c' [95.19.5]

'lib/stdlib/atol.c' [95.19.6]

## VEDERE ANCHE

*strtol(3)* [88.130], *strtoul(3)* [88.130].

## 88.9 os32: atol(3)

Vedere *atoi(3)* [88.8].

## 88.10 os32: basename(3)

## NOME

'**basename**', '**dirname**' - elaborazione dei componenti di un percorso

## SINTASSI

```
#include <libgen.h>
char *basename (char *path);
char *dirname (char *path);
```

## DESCRIZIONE

Le funzioni *basename()* e *dirname()*, restituiscono un percorso, estratto da quello fornito come argomento (*path*). Per la precisione, *basename()* restituisce l'ultimo componente del percorso, mentre *dirname()* restituisce ciò che precede l'ultimo componente. Valgono gli esempi seguenti:

Contenuto originale di <i>path</i>	Risultato prodotto da ' <i>dirname (path)</i> '	Risultato prodotto da ' <i>basename (path)</i> '
"/usr/bin/	"/usr"	"bin"
"/usr/bin	"/usr"	"bin"
"/usr	"/"	"usr"
"usr	"."	"usr"
"/"	"/"	"/"
"."	"."	"."
".."	".."	".."

È importante considerare che le due funzioni alterano il contenuto di *path*, in modo da isolare i componenti che servono.

## VALORE RESTITUITO

Le due funzioni restituiscono il puntatore alla stringa contenente il risultato dell'elaborazione, trattandosi di una porzione della stringa già usata come argomento della chiamata e modificata per l'occasione. Non è previsto il manifestarsi di alcun errore.

## FILE SORGENTI

‘lib/libgen.h’ [95.9]

‘lib/libgen/basename.c’ [95.9.1]

‘lib/libgen/dirname.c’ [95.9.2]

## 88.11 os32: byteorder(3)



### NOME

‘htonl’, ‘htons’, ‘ntohl’, ‘ntohs’ - conversione dell’ordine dei byte da *host* a *network* e viceversa

### SINTASSI

```
#include <arpa/inet.h>
uint32_t htonl (uint32_t host32);
uint16_t htons (uint16_t host16);
uint32_t ntohl (uint32_t net32);
uint16_t ntohs (uint16_t net16);
```

### DESCRIZIONE

La funzione *htonl()* converte un valore a 32 bit, dall’ordinamento di byte usato dal sistema, nell’ordinamento adatto alla trasmissione in rete: *Host to network long*.

La funzione *htons()* converte un valore a 16 bit, dall’ordinamento di byte usato dal sistema, nell’ordinamento adatto alla trasmissione in rete: *Host to network short*.

La funzione *ntohl()* converte un valore a 32 bit, dall’ordinamento di byte adatto alla trasmissione in rete, nell’ordinamento usato nel sistema: *Network to host long*.

La funzione *ntohs()* converte un valore a 16 bit, dall'ordinamento di byte adatto alla trasmissione in rete, nell'ordinamento usato nel sistema: *Network to host short*.

In un sistema os32, l'ordine dei byte è tale da avere prima il byte meno significativo, mentre l'ordine usato nella trasmissione in rete richiede di avere prima il byte più significativo.

## FILE SORGENTI

'lib/arpa/inet.h' [95.3]

'lib/arpa/inet/htonl.c' [95.3.1]

'lib/arpa/inet/htons.c' [95.3.2]

'lib/arpa/inet/ntohl.c' [95.3.5]

'lib/arpa/inet/ntohs.c' [95.3.6]

## VEDERE ANCHE

*inet\_ntop(3)* [88.66], *inet\_pton(3)* [88.67].

## 88.12 os32: clearerr(3)

«

### NOME

'**clearerr**' - azzeramento degli indicatori di errore e di fine file di un certo flusso di file

### SINTASSI

```
#include <stdio.h>
void clearerr (FILE *fp);
```

### DESCRIZIONE

La funzione *clearerr()* azzerava gli indicatori di errore e di fine file, del flusso di file indicato come argomento.

## FILE SORGENTI

‘lib/stdio.h’ [95.18]

‘lib/stdio/clearerr.c’ [95.18.2]

## VEDERE ANCHE

*feof*(3) [88.29], *ferror*(3) [88.30], *fileno*(3) [88.35], *stdio*(3) [88.112].

## 88.13 os32: closedir(3)



## NOME

‘**closedir**’ - chiusura di una directory

## SINTASSI

```
#include <sys/types.h>
#include <dirent.h>
int closedir (DIR *dp);
```

## DESCRIZIONE

La funzione *closedir*() chiude la directory rappresentata da *dp*.

## 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
EBADF	La directory rappresentata da <i>dp</i> , non è valida.

## FILE SORGENTI

‘lib/sys/types.h’ [95.26]

‘lib/dirent.h’ [95.4]

‘lib/dirent/DIR.c’ [95.4.1]

‘lib/dirent/closedir.c’ [95.4.2]

## VEDERE ANCHE

*close(2)* [87.10], *opendir(3)* [88.89], *readdir(3)* [88.98],  
*rewinddir(3)* [88.101].

## 88.14 os32: creat(3)

«

### NOME

‘**creat**’ - creazione di un file puro e semplice

### SINTASSI

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

### DESCRIZIONE

La funzione *creat()* equivale esattamente all’uso della funzione *open()*, con le opzioni ‘O\_WRONLY|O\_CREAT|O\_TRUNC’:

```
open (path, O_WRONLY|O_CREAT|O_TRUNC, mode)
```

Per ogni altra informazione, si veda la pagina di manuale *open(2)* [87.37].

## FILE SORGENTI

‘lib/sys/types.h’ [95.26]

‘lib/sys/stat.h’ [95.25]

‘lib/fcntl.h’ [95.6]

‘lib/fcntl/creat.c’ [95.6.1]

## 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], *open(2)* [87.37], *read(2)* [87.39], *stat(2)* [87.55], *umask(2)* [87.60], *unlink(2)* [87.62], *write(2)* [87.64], *fopen(3)* [88.36].

## 88.15 os32: ctime(3)



### NOME

‘**asctime**’, ‘**ctime**’, ‘**gmtime**’, ‘**localtime**’, ‘**mktime**’ -  
conversione di informazioni data-orario

### SINTASSI

```
#include <time.h>
char      *asctime      (const struct tm *timeptr);
char      *ctime        (const time_t *timer);
struct tm *gmtime       (const time_t *timer);
struct tm *localtime    (const time_t *timer);
time_t     mktime       (const struct tm *timeptr);
```

## DESCRIZIONE

Queste funzioni hanno in comune il compito di convertire delle informazioni data-orario, da un formato a un altro, eventualmente anche testuale. Una data e un orario possono essere rappresentati con il tipo `'time_t'`, nel qual caso si tratta del numero di secondi trascorsi dall'ora zero del 1 gennaio 1970; in alternativa potrebbe essere rappresentata in una variabile strutturata di tipo `'struct tm'`, dichiarato nel file `'time.h'`:

```
struct tm {
    int tm_sec;      // secondi
    int tm_min;     // minuti
    int tm_hour;    // ore
    int tm_mday;    // giorno del mese
    int tm_mon;     // mese, da 1 a 12
    int tm_year;    // anno
    int tm_wday;    // giorno della settimana,
                    // da 0 (domenica) a 6
    int tm_yday;    // giorno dell'anno
    int tm_isdst;   // informazioni sull'ora estiva
};
```

In alcuni casi, la conversione dovrebbe tenere conto della configurazione locale, ovvero del fuso orario ed eventualmente del cambiamento di orario nel periodo estivo. `os32` non considera alcunché e gestisce il tempo in un modo assoluto, senza nozione della convenzione locale.

La funzione `asctime()` converte quanto contenuto in una variabile strutturata di tipo `'struct tm'` in una stringa che descrive la data e l'ora in inglese. La stringa in questione è allocata staticamente e viene sovrascritta se la funzione viene usata più volte.

La funzione `ctime()` è in realtà soltanto una macroistruzione, la



quale, complessivamente, converte il tempo indicato come quantità di secondi, restituendo il puntatore a una stringa che descrive la data attuale, tenendo conto della configurazione locale. La stringa in questione utilizza la stessa memoria statica usata per *asctime()*; inoltre, dato che `os32` non distingue tra ora locale e tempo universale, la funzione non esegue alcuna conversione temporale.

La funzione *gmtime()* converte il tempo espresso in secondi in una forma suddivisa, secondo il tipo `'struct tm'`. La funzione restituisce quindi il puntatore a una variabile strutturata di tipo `'struct tm'`, la quale però è dichiarata in modo statico, internamente alla funzione, e viene sovrascritta nelle chiamate successive della stessa.

La funzione *localtime()* converte una data espressa in secondi, in una data suddivisa in campi (`'struct tm'`), tenendo conto (ma non succede con `os32`) della configurazione locale.

La funzione *mktime()* converte una data contenute in una variabile strutturata di tipo `'struct tm'` nella quantità di secondi corrispondente.

## VALORE RESTITUITO

Le funzioni che restituiscono un puntatore, se incontrano un errore, restituiscono il puntatore nullo, `'NULL'`. Nel caso particolare di *mktime()*, se il valore restituito è pari a `-1`, si tratta di un errore.

## FILE SORGENTI

`'lib/time.h'` [95.29]

`'lib/time/asctime.c'` [95.29.1]

`'lib/time/gmtime.c'` [95.29.3]

'lib/time/mktime.c' [95.29.4]

## VEDERE ANCHE

*date(1)* [86.10], *clock(2)* [87.9], *time(2)* [87.59].

### 88.16 os32: *dirname(3)*

&lt;&lt;

Vedere *basename(3)* [88.10].

### 88.17 os32: *div(3)*

&lt;&lt;

## NOME

'**div**', '**ldiv**', '**lldiv**', '**imaxdiv**' - calcolo del quoziente e del resto di una divisione intera

## SINTASSI

```
#include <stdlib.h>
div_t div (int numer, int denom);
ldiv_t ldiv (long int numer, long int denom);
lldiv_t lldiv (long long int numer, long long int denom);
```

```
#include <inttypes.h>
imaxdiv_t imaxdiv (intmax_t numer, intmax_t denom);
```

## DESCRIZIONE

Le funzioni '**...div()**' calcolano la divisione tra numeratore e denominatore, forniti come argomenti della chiamata, restituendo un risultato, composto di divisione intera e resto, in una variabile strutturata.

I tipi `'div_t'`, `'ldiv_t'` e `'lldiv_t'`, sono dichiarati nel file `'stdlib.h'` nel modo seguente:

```
typedef struct {
    int    quot;
    int    rem;
} div_t;
```

```
typedef struct {
    long long int quot;
    long long int rem;
} lldiv_t;
```

```
typedef struct {
    long long int quot;
    long long int rem;
} lldiv_t;
```

Il tipo `'imaxdiv_t'` è dichiarati nel file `'inttypes.h'` in maniera analoga:

```
typedef struct {
    intmax_t quot;
    intmax_t rem;
} imaxdiv_t;
```

I membri *quot* contengono il quoziente, ovvero il risultato intero; i membri *rem* contengono il resto della divisione.

## VALORE RESTITUITO

Il risultato della divisione, strutturato in quoziente e resto.

## FILE SORGENTI

`'lib/stdlib.h'` [[95.19](#)]

`'lib/stdlib/div.c'` [[95.19.7](#)]

`'lib/stdlib/ldiv.c'` [[95.19.12](#)]

'lib/stdlib/lldiv.c' [95.19.14]

'lib/inttypes.h' [95.8]

'lib/inttypes/imaxdiv.c' [95.8.2]

## VEDERE ANCHE

*abs(3)* [88.3].

88.18 os32: *endgrent(3)*

«

Vedere *getgrent(3)* [88.53].

88.19 os32: *endpwent(3)*

«

Vedere *getpwent(3)* [88.57].

88.20 os32: *errno(3)*

«

## NOME

'**errno**' - numero dell'ultimo errore riportato

## SINTASSI

```
#include <errno.h>
```

## DESCRIZIONE

Attraverso l'inclusione del file 'errno.h', si ottiene la dichiarazione della variabile *errno*. In pratica, per os32 viene dichiarata così:

```
extern int errno;
```

Per annotare un errore si assegna un valore numerico a questa variabile. Il valore numerico in questione rappresenta sinteticamente la descrizione dell'errore; pertanto, si utilizzano per questo delle macro-variabili, dichiarate tutte nel file `errno.h`. Nell'esempio seguente si annota l'errore ***ENOMEM***, corrispondente alla descrizione «Not enough space», ovvero «spazio insufficiente»:

```
errno = ENOMEM;
```

Dal momento che in questo modo viene comunque a mancare un riferimento al sorgente e alla posizione in cui l'errore si è manifestato, la libreria di `os32` aggiunge la macroistruzione ***errset()***, la quale però non fa parte dello standard. Si usa così:

```
errset (ENOMEM);
```

La macroistruzione ***errset()*** aggiorna la variabile ***errln*** e l'array ***errfn[]***, rispettivamente con il numero della riga di codice in cui si è manifestato il problema e il nome della funzione che lo contiene. Con queste informazioni, la funzione ***perror(3)*** [88.90] può visualizzare più dati.

Pertanto, nel codice di `os32`, si usa sempre la macroistruzione ***errset()***, invece di assegnare semplicemente un valore alla variabile ***errno***.

## ERRORI

Gli errori previsti dalla libreria di `os32` sono riassunti dalla tabella successiva. La prima parte contiene gli errori definiti dallo standard POSIX, ma solo alcuni di questi vengono usati effettivamente nella libreria, data la limitatezza di `os32`.

Valore di <i>errno</i>	Definizione
E2BIG	Argument list too long.
EACCES	Permission denied.
EADDRINUSE	Address in use.
EADDRNOTAVAIL	Address not available.
EAFNOSUPPORT	Address family not supported.

Valore di <i>errno</i>	Definizione
EAGAIN	Resource unavailable, try again.
EALREADY	Connection already in progress.
EBADF	Bad file descriptor.
EBADMSG	Bad message.
EBUSY	Device or resource busy.

<b>Valore di <i>errno</i></b>	<b>Definizione</b>
ECANCELED	Operation canceled.
ECHILD	No child processes.
ECONNABORTED	Connection aborted.
ECONNREFUSED	Connection refused.
ECONNRESET	Connection reset.

<b>Valore di <i>errno</i></b>	<b>Definizione</b>
EDEADLK	Resource deadlock would occur.
EDESTADDRREQ	Destination address required.
EDOM	Mathematics argument out of domain of function.
EDQUOT	Reserved.
EEXIST	File exists.

Valore di <i>errno</i>	Definizione
EFAULT	Bad address.
EFBIG	File too large.
EHOSTUNREACH	Host is unreachable.
EIDRM	Identifier removed.
EILSEQ	Illegal byte sequence.

Valore di <i>errno</i>	Definizione
EINPROGRESS	Operation in progress.
EINTR	Interrupted function.
EINVAL	Invalid argument.
EIO	I/O error.
EISCONN	Socket is connected.



Valore di <i>errno</i>	Definizione
EISDIR	Is a directory.
ELOOP	Too many levels of symbolic links.
EMFILE	Too many open files.
EMLINK	Too many links.
EMSGSIZE	Message too large.

Valore di <i>errno</i>	Definizione
EMULTIHOP	Reserved.
ENAMETOOLONG	Filename too long.
ENETDOWN	Network is down.
ENETRESET	Connection aborted by network.
ENETUNREACH	Network unreachable.

Valore di <i>errno</i>	Definizione
ENFILE	Too many files open in system.
ENOBUFFS	No buffer space available.
ENODATA	No message is available on the stream head read queue.
ENODEV	No such device.
ENOENT	No such file or directory.

Valore di <i>errno</i>	Definizione
ENOEXEC	Executable file format error.
ENOLCK	No locks available.
ENOLINK	Reserved.
ENOMEM	Not enough space.
ENOMSG	No message of the desired type.

Valore di <i>errno</i>	Definizione
ENOPROTOPT	Protocol not available.
ENOSPC	No space left on device.
ENOSR	No stream resources.
ENOSTR	Not a stream.
ENOSYS	Function not supported.

Valore di <i>errno</i>	Definizione
ENOTCONN	The socket is not connected.
ENOTDIR	Not a directory.
ENOTEMPTY	Directory not empty.
ENOTSOCK	Not a socket.
ENOTSUP	Not supported.

Valore di <i>errno</i>	Definizione
ENOTTY	Inappropriate I/O control operation.
ENXIO	No such device or address.
EOPNOTSUPP	Operation not supported on socket.
EOVERFLOW	Value too large to be stored in data type.
EPERM	Operation not permitted.

Valore di <i>errno</i>	Definizione
EPIPE	Broken pipe.
EPROTO	Protocol error.
EPROTONOSUPPORT	Protocol not supported.
EPROTOTYPE	Protocol wrong type for socket.
ERANGE	Result too large.

Valore di <i>errno</i>	Definizione
EROFS	Read-only file system.
ESPIPE	Invalid seek.
ESRCH	No such process.
ESTALE	Reserved.
ETIME	Stream ioctl() timeout.

Valore di <i>errno</i>	Definizione
ETIMEDOUT	Connection timed out.
ETXTBSY	Text file busy.
EWOULDBLOCK	Operation would block (may be the same as EAGAIN).
EXDEV	Cross-device link.

La tabella successiva raccoglie le definizioni degli errori aggiuntivi, specifici di os32.

Valore di <i>errno</i>	Definizione
EUNKNOWN	Unknown error.
E_NO_MEDIUM	No medium found.
E_MEDIUM	Medium reported error.
E_FILE_TYPE	File type not compatible.
E_ROOT_INODE_NOT_CACHED	The root directory inode is not cached.

Valore di <i>errno</i>	Definizione
E_CANNOT_READ_SUPERBLOCK	Cannot read super block.
E_MAP_INODE_TOO_BIG	Map inode too big.
E_MAP_ZONE_TOO_BIG	Map zone too big.
E_DATA_ZONE_TOO_BIG	Data zone too big.
E_CANNOT_FIND_ROOT_DEVICE	Cannot find root device.

Valore di <i>errno</i>	Definizione
E_CANNOT_FIND_ROOT_INODE	Cannot find root inode.
E_FILE_TYPE_UNSUPPORTED	File type unsupported.
E_ENV_TOO_BIG	Environment too big.
E_LIMIT	Exceeded implementation limits.
E_NOT_MOUNTED	Not mounted.

Valore di <i>errno</i>	Definizione
E_NOT_IMPLEMENTED	Not implemented.
E_HARDWARE_FAULT	Hardware fault.
E_DRIVER_FAULT	Driver fault.
E_PIPE_FULL	Pipe full.
E_PIPE_EMPTY	Pipe empty.

## FILE SORGENTI

‘lib/errno.h’ [95.5]

‘lib/errno/errno.c’ [95.5.1]

## VEDERE ANCHE

*perror(3)* [88.90], *strerror(3)* [88.120].

## 88.21 os32: exec(3)



### NOME

‘**execl**’, ‘**execle**’, ‘**execlp**’, ‘**execv**’, ‘**execvp**’ - esecuzione di un file

### SINTASSI

```
#include <unistd.h>
extern char **environ;
int execl (const char *path, const char *arg, ...);
int execle (const char *path, const char *arg, ...);
int execlp (const char *path, const char *arg, ...);
int execv (const char *path, char *const argv());
int execvp (const char *path, char *const argv());
```

### DESCRIZIONE

Le funzioni ‘**exec... ()**’ rimpiazzano il processo chiamante con un altro processo, ottenuto dal caricamento di un file eseguibile in memoria. Tutte queste funzioni si avvalgono, direttamente o indirettamente di *execve(2)* [87.14].

Il primo parametro delle funzioni descritte qui rappresenta il percorso, costituito dalla stringa *path*, di un file da eseguire.

Le funzioni ‘**execl... ()**’, dopo il percorso del file eseguibile, richiedono l’indicazione di una quantità variabile di argomenti, a cominciare da *arg*, costituiti da stringhe, tenendo conto che dopo l’ultimo di questi argomenti, occorre fornire il puntatore nullo, ‘**NULL**’, per chiarire che questi sono terminati. L’argomento corrispondente al parametro *arg* deve essere una stringa contenente



il nome del file da avviare, mentre quelli successivi sono a loro volta gli argomenti da passare al programma stesso.

Rispetto al gruppo di funzioni ‘**exec1... ()**’, la funzione *execle()*, dopo il puntatore nullo che conclude la sequenza di argomenti per il programma da avviare, si attende una sequenza di altre stringhe, anche questa conclusa da un ultimo e definitivo puntatore nullo. Questa ulteriore sequenza di stringhe va a costituire l’ambiente del processo da avviare, pertanto il contenuto di tali stringhe deve essere del tipo ‘*nome=valore*’.

Le funzioni ‘**execv... ()**’ hanno come ultimo parametro il puntatore a un array di stringhe, dove *argv[0]* deve essere il nome del file da avviare, mentre da *argv[1]* in poi, si tratta degli argomenti da passare al programma. Anche in questo caso, per riconoscere l’ultimo elemento di questo array, gli si deve assegnare il puntatore nullo.

Le funzioni *execlp()* e *execvp()*, se ricevono solo il nome del file da eseguire, senza altre indicazioni del percorso, cercano questo file tra i percorsi indicati nella variabile di ambiente *PATH*, ammesso che sia dichiarata per il processo in corso.

Tutte le funzioni qui descritte, a eccezione di *execle()*, trasmettono al nuovo processo lo stesso ambiente (le stesse variabili di ambiente) del processo chiamante.

## VALORE RESTITUITO

Queste funzioni, se hanno successo nel loro compito, non possono restituire alcunché, dato che in quel momento, il processo chiamante viene rimpiazzato da quello del file che viene eseguito. Pertanto, in caso contrario, queste funzioni possono re-

stituire soltanto 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.

## FILE SORGENTI

'lib/unistd.h' [[95.30](#)]

'lib/unistd/execl.c' [[95.30.10](#)]

'lib/unistd/execl.c' [[95.30.11](#)]

'lib/unistd/execlp.c' [[95.30.12](#)]

'lib/unistd/execv.c' [[95.30.13](#)]

'lib/unistd/execvp.c' [[95.30.15](#)]

'lib/unistd/execve.c' [[95.30.14](#)]

## VEDERE ANCHE

*execve(2)* [[87.14](#)], *fork(2)* [[87.19](#)], *environ(7)* [[91.1](#)].

## 88.22 os32: `execl(3)`

Vedere *exec(3)* [88.21].

## 88.23 os32: `execle(3)`

Vedere *exec(3)* [88.21].

## 88.24 os32: `execlp(3)`

Vedere *exec(3)* [88.21].

## 88.25 os32: `execv(3)`

Vedere *exec(3)* [88.21].

## 88.26 os32: `execvp(3)`

Vedere *exec(3)* [88.21].

## 88.27 os32: `exit(3)`

Vedere *atexit(3)* [88.7].

## 88.28 os32: `fclose(3)`

### **NOME**

‘**fclose**’ - chiusura di un flusso di file

### **SINTASSI**

```
#include <stdio.h>
int fclose (FILE *fp);
```

## DESCRIZIONE

La funzione *fclose()* chiude il flusso di file specificato tramite il puntatore *fp*. Questa realizzazione particolare di os32, si limita a richiamare la funzione *close()*, con l'indicazione del descrittore di file corrispondente al flusso.

## VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
'EOF'	Errore: la variabile <i>errno</i> viene impostata di conseguenza.

## ERRORI

Valore di <i>errno</i>	Significato
EBADF	Il descrittore di file associato al flusso da chiudere, non è valido.

## FILE SORGENTI

'lib/stdio.h' [95.18]

'lib/stdio/fclose.c' [95.18.3]

## VEDERE ANCHE

*close(2)* [87.10], *fopen(3)* [88.36], *stdio(3)* [88.112].

88.29 os32: feof(3)

«

## NOME

'feof' - verifica dello stato dell'indicatore di fine file

## SINTASSI

```
#include <stdio.h>
int feof (FILE *fp);
```

## DESCRIZIONE

La funzione *feof()* restituisce il valore dell'indicatore di fine file, riferito al flusso di file rappresentato da *fp*.

## VALORE RESTITUITO

Valore	Significato
diverso da zero	Significa che l'indicatore di fine file è impostato.
0	Significa che l'indicatore di fine file non è impostato.

## FILE SORGENTI

'lib/stdio.h' [95.18]

'lib/stdio/feof.c' [95.18.4]

## VEDERE ANCHE

*clearerr(3)* [88.12], *ferror(3)* [88.30], *fileno(3)* [88.35], *stdio(3)* [88.112].

88.30 os32: *ferror(3)*



## NOME

'**ferror**' - verifica dello stato dell'indicatore di errore

## SINTASSI

```
#include <stdio.h>
int ferror (FILE *fp);
```

## DESCRIZIONE

La funzione *ferror()* restituisce il valore dell'indicatore di errore, riferito al flusso di file rappresentato da *fp*.

## VALORE RESTITUITO

Valore	Significato
diverso da zero	Significa che l'indicatore di errore è impostato.
0	Significa che l'indicatore di errore non è impostato.

## FILE SORGENTI

'lib/stdio.h' [95.18]

'lib/stdio/ferror.c' [95.18.5]

## VEDERE ANCHE

*clearerr(3)* [88.12], *feof(3)* [88.29], *fileno(3)* [88.35], *stdio(3)* [88.112].

## 88.31 os32: fflush(3)

«

## NOME

'**fflush**' - fissaggio dei dati ancora sospesi nella memoria tampone

## SINTASSI

```
#include <stdio.h>
int fflush (FILE *fp);
```

## DESCRIZIONE

La funzione *fflush()* di *os32*, non fa alcunché, dato che non è prevista alcuna gestione della memoria tampone per i flussi di file.

## VALORE RESTITUITO

Valore	Significato
0	Rappresenta il successo dell'operazione.

## FILE SORGENTI

'lib/stdio.h' [95.18]

'lib/stdio/fflush.c' [95.18.6]

## VEDERE ANCHE

*fclose(3)* [88.28], *fopen(3)* [88.36].

## 88.32 os32: fgetc(3)



## NOME

'*fgetc*', '*getc*', '*getchar*' - lettura di un carattere da un flusso di file

## SINTASSI

```
#include <stdio.h>

int fgetc (FILE *fp);
int getc (FILE *fp);
int getchar (void);
```

## DESCRIZIONE

Le funzioni *fgetc()* e *getc()* sono equivalenti e leggono il carattere successivo dal flusso di file rappresentato da *fp*. La

funzione *getchar()* esegue la lettura di un carattere, ma dallo standard input.

## VALORE RESTITUITO

In caso di successo, il carattere letto viene restituito in forma di intero positivo (il carattere viene inteso inizialmente senza segno, quindi viene trasformato in un intero, il quale rappresenta così un valore positivo). Se la lettura non può avere luogo, la funzione restituisce ‘**EOF**’, corrispondente a un valore negativo.

## ERRORI

La variabile *errno* potrebbe risultare aggiornata nel caso la funzione restituisca ‘**EOF**’. Ma per saperlo, occorre azzerare la variabile *errno* prima della chiamata di *fgetc()*.

Valore di <i>errno</i>	Significato
EBADF	Il descrittore di file associato al flusso da cui leggere un carattere, non è valido.
EINVAL	Il descrittore di file associato al flusso da cui leggere un carattere, non consente un accesso in lettura.

## FILE SORGENTI

‘lib/stdio.h’ [[95.18](#)]

‘lib/stdio/fgetc.c’ [[95.18.7](#)]

‘lib/stdio/getchar.c’ [[95.18.24](#)]

## VEDERE ANCHE

*fgets(3)* [[88.34](#)], *gets(3)* [[88.34](#)].



## 88.33 os32: fgetpos(3)



### NOME

‘**fgetpos**’, ‘**fsetpos**’ - lettura e impostazione della posizione corrente di un flusso di file

### SINTASSI

```
#include <stdio.h>

int fgetpos (FILE *restrict fp, fpos_t *restrict pos);
int fsetpos (FILE *restrict fp, fpos_t *restrict pos);
```

### DESCRIZIONE

Le funzioni *fgetpos()* e *fsetpos()*, rispettivamente, leggono o impostano la posizione corrente di un flusso di file.

Per os32, il tipo ‘**fpos\_t**’ è dichiarato nel file ‘`stdio.h`’ come equivalente al tipo ‘**off\_t**’ (file ‘`sys/types.h`’); tuttavia, la modifica di una variabile di tipo ‘**fpos\_t**’ va fatta utilizzando una funzione apposita, perché lo standard consente che possa trattarsi anche di una variabile strutturata, i cui membri non sono noti.

L’uso della funzione *fgetpos()* comporta una modifica dell’informazione contenuta all’interno di *\*pos*, mentre la funzione *fsetpos()* usa il valore contenuto in *\*pos* per cambiare la posizione corrente del flusso di file. In pratica, si può usare *fsetpos()* solo dopo aver salvato una certa posizione con l’aiuto di *fgetpos()*.

Si comprende che il valore restituito dalle funzioni è solo un indice del successo o meno dell’operazione, dato che l’informazione sulla posizione viene ottenuta dalla modifica di una variabile di cui si fornisce il puntatore negli argomenti.

## VALORE RESTITUITO

Valore	Significato
0	Rappresenta il successo dell'operazione.
-1	Indica il verificarsi di un errore, il quale può essere interpretato leggendo la variabile <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	Significato
EBADF	Il descrittore di file associato al flusso, non è valido.

## FILE SORGENTI

'lib/stdio.h' [95.18]

'lib/stdio/fgetpos.c' [95.18.8]

'lib/stdio/fsetpos.c' [95.18.20]

## VEDERE ANCHE

*fseek(3)* [88.44], *ftell(3)* [88.47], *rewind(3)* [88.100].

## 88.34 os32: fgets(3)

«

## NOME

'**fgets**', '**gets**' - lettura di una stringa da un flusso di file

## SINTASSI

```
#include <stdio.h>
char *fgets (char *restrict string, int n,
             FILE *restrict fp);
char *gets  (char *string);
```

## DESCRIZIONE

La funzione *fgets()* legge una «riga» dal flusso di file *fp*, purché non più lunga di  $n-1$  caratteri, collocandola a partire da ciò a cui punta *stringa*. La lettura termina al raggiungimento del carattere ‘\n’ (*new line*), oppure alla fine del file, oppure a  $n-1$  caratteri. In ogni caso, viene aggiunto al termine, il codice nullo di terminazione di stringa: ‘\0’.

La funzione *gets()*, in modo analogo a *fgets()*, legge una riga dallo standard input, ma senza poter porre un limite massimo alla lunghezza della lettura.

## VALORE RESTITUITO

In caso di successo, viene restituito il puntatore alla stringa contenente la riga letta, ovvero restituiscono *string*. In caso di errore, o comunque in caso di una lettura nulla, si ottiene ‘NULL’. La variabile *errno* viene aggiornata solo se si presenta un errore di accesso al file, mentre una lettura nulla, perché il flusso si è concluso, non comporta tale aggiornamento.

## ERRORI

La variabile *errno* potrebbe risultare aggiornata nel caso le funzioni restituiscano ‘NULL’. Ma per saperlo, occorre azzerare la variabile *errno* prima della chiamata di queste.

Valore di <i>errno</i>	Significato
EBADF	Il descrittore di file associato al flusso da cui leggere un carattere, non è valido.
EINVAL	Il descrittore di file associato al flusso da cui leggere un carattere, non consente un accesso in lettura.

## FILE SORGENTI

'lib/stdio.h' [95.18]

'lib/stdio/fgets.c' [95.18.9]

'lib/stdio/gets.c' [95.18.25]

## VEDERE ANCHE

*fgetc(3)* [88.32], *getc(3)* [88.32].

## 88.35 os32: fileno(3)

«

## NOME

'**fileno**' - traduzione di un flusso di file nel numero di descrittore corrispondente

## SINTASSI

```
#include <stdio.h>
int fileno (FILE *fp);
```

## DESCRIZIONE

La funzione *fileno()* traduce il flusso di file, rappresentato da *fp*, nel numero del descrittore corrispondente. Tuttavia, il risultato è valido solo se il flusso di file specificato è aperto effettivamente.

## VALORE RESTITUITO

Se *fp* punta effettivamente a un flusso di file aperto, il valore restituito corrisponde al numero di descrittore del file stesso; diversamente, si potrebbe ottenere un numero privo di senso. Se come argomento si indica il puntatore nullo, si ottiene un errore, rappresentato dal valore -1.

## ERRORI

Valore di <i>errno</i>	Significato
EBADF	È stato richiesto di risolvere il puntatore nullo.

## FILE SORGENTI

‘lib/stdio.h’ [95.18]

‘lib/stdio/FILE.c’ [95.18.1]

‘lib/stdio/fileno.c’ [95.18.10]

## VEDERE ANCHE

*clearerr(3)* [88.12], *feof(3)* [88.29], *ferror(3)* [88.30], *stdio(3)* [88.112].

## 88.36 os32: fopen(3)

«

## NOME

‘**fopen**’, ‘**freopen**’ - apertura di un flusso di file

## SINTASSI

```
#include <stdio.h>
FILE *fopen (const char *path, const char *mode);
FILE *freopen (const char *restrict path,
               const char *restrict mode,
               FILE *restrict fp);
```

## DESCRIZIONE

La funzione *fopen()* apre il file indicato nella stringa a cui punta *path*, secondo le modalità di accesso contenute in *mode*, as-

sociandovi un flusso di file. In modo analogo agisce anche la funzione *freopen()*, la quale però, prima, chiude il flusso *fp*.

La modalità di accesso al file si specifica attraverso una stringa, come sintetizzato dalla tabella successiva.

<i>mode</i>	Significato
"r" "rb"	Si richiede un accesso in lettura, di un file già esistente. L'indice interno per l'accesso ai dati viene posizionato all'inizio del file.
"r+" "r+b" "rb+"	Si richiede un accesso in lettura e scrittura, di un file già esistente. L'indice interno per l'accesso ai dati viene posizionato all'inizio del file.
"w" "wb"	Si richiede un accesso in scrittura, di un file che viene troncato se esiste oppure viene creato. L'indice interno per l'accesso ai dati viene posizionato all'inizio del file.
"w+" "w+b" "wb+"	Si richiede un accesso in lettura e scrittura, di un file che viene troncato se esiste oppure viene creato. L'indice interno per l'accesso ai dati viene posizionato all'inizio del file.
"a" "ab"	Si richiede la creazione o il troncamento di un file, con accesso in aggiunta. L'indice interno per l'accesso ai dati viene posizionato alla fine del file, prima di ogni operazione di scrittura.
"a+" "a+b" "ab+"	Si richiede la creazione o il troncamento di un file, con accesso in lettura e scrittura. L'indice interno per l'accesso ai dati viene posizionato alla fine del file, prima di ogni operazione di scrittura.

## VALORE RESTITUITO

Se l'operazione si conclude con successo, viene restituito il puntatore a ciò che rappresenta il flusso di file aperto. Se però si ot-

tiene un puntatore nullo (**‘NULL’**), si è verificato un errore che può essere interpretato dal contenuto della variabile *errno*.

## ERRORI

Valore di <i>errno</i>	Significato
EINVAL	È stato fornito un argomento non valido.
EPERM	Operazione non consentita.
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/stdio.h'` [[95.18](#)]

`'lib/stdio/FILE.c'` [[95.18.1](#)]

`'lib/stdio/fopen.c'` [[95.18.11](#)]

`'lib/stdio/freopen.c'` [[95.18.16](#)]

## VEDERE ANCHE

*open*(2) [[87.37](#)], *fclose*(3) [[88.28](#)], *stdio*(3) [[88.112](#)].



## 88.37 os32: fprintf(3)

Vedere *printf(3)* [88.91].

## 88.38 os32: fputc(3)

### NOME

‘**fputc**’, ‘**putc**’, ‘**putchar**’ - emissione di un carattere attraverso un flusso di file

### SINTASSI

```
#include <stdio.h>
int fputc (int c, FILE *fp);
int putc (int c, FILE *fp);
int putchar (int c);
```

### DESCRIZIONE

Le funzioni *fputc()* e *putc()* sono equivalenti e scrivono il carattere *c* nel flusso di file rappresentato da *fp*. La funzione *putchar()* esegue la scrittura di un carattere, ma nello standard output.

### VALORE RESTITUITO

In caso di successo, il carattere scritto viene restituito in forma di intero positivo (il carattere viene inteso inizialmente senza segno, quindi viene trasformato in un intero, il quale rappresenta così un valore positivo). Se la scrittura non può avere luogo, la funzione restituisce ‘**EOF**’, corrispondente a un valore negativo.

## ERRORI

Valore di <i>errno</i>	Significato
EBADF	Il descrittore di file associato al flusso presso cui scrivere un carattere, non è valido.
EINVAL	Il descrittore di file associato al flusso presso cui scrivere un carattere, non consente un accesso in scrittura.

## FILE SORGENTI

‘lib/stdio.h’ [95.18]

‘lib/stdio/fputc.c’ [95.18.13]

## VEDERE ANCHE

*fputs(3)* [88.39], *puts(3)* [88.39].

## 88.39 os32: fputs(3)

«

## NOME

‘**fputs**’, ‘**puts**’ - scrittura di una stringa attraverso un flusso di file

## SINTASSI

```
#include <stdio.h>
int fputs (const char *restrict string, FILE *restrict fp);
int puts (const char *string);
```

## DESCRIZIONE

La funzione *fputs()* scrive una stringa nel flusso di file *fp*, ma senza il carattere nullo di terminazione; la funzione *puts()* scrive

una stringa, aggiungendo anche il codice di terminazione ‘\n’, attraverso lo standard output.

## VALORE RESTITUITO

Valore	Significato
$\geq 0$	Rappresenta il successo dell'operazione.
'EOF'	Indica il verificarsi di un errore, il quale può essere interpretato eventualmente leggendo la variabile <i>errno</i> .

## ERRORI

La variabile *errno* potrebbe risultare aggiornata nel caso le funzioni restituiscano 'NULL'. Ma per saperlo, occorre azzerare la variabile *errno* prima della chiamata di queste.

Valore di <i>errno</i>	Significato
EBADF	Il descrittore di file associato al flusso verso cui si deve scrivere, non è valido.
EINVAL	Il descrittore di file associato al flusso verso cui si deve scrivere, non consente un accesso in scrittura.

## FILE SORGENTI

'lib/stdio.h' [[95.18](#)]

'lib/stdio/fputs.c' [[95.18.14](#)]

'lib/stdio/puts.c' [[95.18.29](#)]

## VEDERE ANCHE

*fputc(3)* [[88.38](#)], *putc(3)* [[88.38](#)], *putchar(3)* [[88.38](#)].

## 88.40 os32: fread(3)

**NOME**

‘**fread**’ - lettura di dati da un flusso di file

**SINTASSI**

```
#include <stdio.h>
size_t fread (void *restrict buffer, size_t size,
              size_t nmemb,
              FILE *restrict fp);
```

**DESCRIZIONE**

La funzione *fread()* legge *size*×*nmemb* byte dal flusso di file *fp*, trascrivendoli in memoria a partire dall’indirizzo a cui punta *buffer*.

**VALORE RESTITUITO**

La funzione restituisce la quantità di byte letta, diviso la dimensione del blocco *nmemb* (byte/*nmemb*). Se il valore ottenuto è inferiore a quello richiesto, occorre verificare eventualmente se ciò deriva dalla conclusione del file o da un errore, con l’aiuto di *feof(3)* [88.29] e di *ferror(3)* [88.30].

**FILE SORGENTI**

‘lib/stdio.h’ [95.18]

‘lib/stdio/fread.c’ [95.18.15]

**VEDERE ANCHE**

*read(2)* [87.39], *write(2)* [87.64], *feof(3)* [88.29], *ferror(3)* [88.30], *fwrite(3)* [88.49].

## 88.41 os32: free(3)

Vedere *malloc(3)* [88.76].

## 88.42 os32: freopen(3)

Vedere *fopen(3)* [88.36].

## 88.43 os32: fscanf(3)

Vedere *scanf(3)* [88.102].

## 88.44 os32: fseek(3)

## NOME

‘**fseek**’, ‘**fseeko**’ - riposizionamento dell’indice di accesso di un flusso di file

## SINTASSI

```
#include <stdio.h>
int fseek (FILE *fp, long int offset, int whence);
int fseeko (FILE *fp, off_t offset, int whence);
```

## DESCRIZIONE

Le funzioni *fseek()* e *fseeko()* cambiano l’indice della posizione interna a un flusso di file, specificato dal parametro *fp*. L’indice viene collocato secondo lo scostamento rappresentato da *offset*, rispetto al riferimento costituito dal parametro *whence*. Il parametro *whence* può assumere solo tre valori, come descritto nello schema successivo.

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.

La differenza tra le due funzioni sta solo nel tipo del parametro *offset*, il quale, da `'long int'` passa a `'off_t'`.

## VALORE RESTITUITO

Valore	Significato
0	Rappresenta il successo dell'operazione.
-1	Indica il verificarsi di un errore, il quale può essere interpretato leggendo la variabile <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	Significato
EBADF	Il descrittore di file associato al flusso, non è valido.
EINVAL	Gli argomenti non sono validi, come succede se la combinazione di scostamento e riferimento non è ammissibile (per esempio uno scostamento negativo, quando il riferimento è l'inizio del file).

## FILE SORGENTI

`'lib/stdio.h'` [[95.18](#)]

`'lib/stdio/FILE.c'` [[95.18.1](#)]

`'lib/stdio/fseek.c'` [[95.18.18](#)]

## VEDERE ANCHE

*lseek(2)* [87.33], *fgetpos(3)* [88.33], *fsetpos(3)* [88.33], *ftell(3)* [88.47], *rewind(3)* [88.100].

### 88.45 os32: *fseeko(3)*

Vedere *fseek(3)* [88.44].

### 88.46 os32: *fsetpos(3)*

Vedere *fgetpos(3)* [88.33].

### 88.47 os32: *ftell(3)*

## NOME

‘**ftell**’, ‘**ftello**’ - interrogazione dell’indice di accesso relativo a un flusso di file

## SINTASSI

```
#include <stdio.h>
long int ftell (FILE *fp);
off_t ftello (FILE *fp);
```

## DESCRIZIONE

Le funzioni *ftell()* e *ftello()* restituiscono il valore dell’indice interno di accesso al file specificato in forma di flusso, con il parametro *fp*. La differenza tra le due funzioni consiste nel tipo restituito, il quale, nel primo caso è ‘**long int**’, mentre nel secondo è ‘**off\_t**’. L’indice ottenuto è riferito all’inizio del file.

## VALORE RESTITUITO

Valore	Significato
$\geq 0$	Rappresenta l'indice interno al file.
-1	Indica il verificarsi di un errore, il quale può essere interpretato leggendo la variabile <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	Significato
EBADF	Il descrittore di file associato al flusso, non è valido.
EINVAL	Il flusso di file specificato non è valido.

## FILE SORGENTI

'lib/stdio.h' [95.18]

'lib/stdio/FILE.c' [95.18.1]

'lib/stdio/ftell.c' [95.18.21]

'lib/stdio/ftello.c' [95.18.22]

## VEDERE ANCHE

*lseek(2)* [87.33], *fgetpos(3)* [88.33], *fsetpos(3)* [88.33], *ftell(3)* [88.44], *rewind(3)* [88.100].

88.48 os32: *ftello(3)*

« Vedere *ftell(3)* [88.47].



## 88.49 os32: fwrite(3)



### NOME

‘**fwrite**’ - scrittura attraverso un flusso di file

### SINTASSI

```
#include <stdio.h>

size_t fwrite (const void *restrict buffer, size_t size,
               size_t nmemb,
               FILE *restrict fp);
```

### DESCRIZIONE

La funzione *fwrite()* scrive *size*×*nmemb* byte nel flusso di file *fp*, traendoli dalla memoria, a partire dall’indirizzo a cui punta *buffer*.

### VALORE RESTITUITO

La funzione restituisce la quantità di byte scritta, diviso la dimensione del blocco rappresentato da *nmemb* (byte/*nmemb*). Se il valore ottenuto è inferiore a quello richiesto, si tratta presumibilmente di un errore, ma per accertarsene conviene usare *ferror(3)* [88.30].

### FILE SORGENTI

‘lib/stdio.h’ [95.18]

‘lib/stdio/fwrite.c’ [95.18.23]

### VEDERE ANCHE

*read(2)* [87.39], *write(2)* [87.64], *feof(3)* [88.29], *ferror(3)* [88.30], *fread(3)* [88.40].

88.50 os32: `getc(3)`

« Vedere *fgetc(3)* [88.32].

88.51 os32: `getchar(3)`

« Vedere *fgetc(3)* [88.32].

88.52 os32: `getenv(3)`

«

## NOME

‘**getenv**’ - lettura del valore di una variabile di ambiente

## SINTASSI

```
#include <stdlib.h>
char *getenv (const char *name);
```

## DESCRIZIONE

La funzione *getenv()* richiede come argomento una stringa contenente il nome di una variabile di ambiente, per poter restituire la stringa che rappresenta il contenuto di tale variabile.

## VALORE RESTITUITO

Il puntatore alla stringa con il contenuto della variabile di ambiente richiesta, oppure il puntatore nullo (‘**NULL**’), se la variabile in questione non esiste.

## FILE SORGENTI

‘lib/stdlib.h’ [95.19]

‘applic/crt0.mer.s’ [96.1.12]

‘`applic/crt0.sep.s`’ [96.1.13]

‘`lib/stdlib/environment.c`’ [95.19.8]

‘`lib/stdlib/getenv.c`’ [95.19.10]

## VEDERE ANCHE

*environ*(7) [91.1], *putenv*(3) [88.94], *setenv*(3) [88.104],  
*unsetenv*(3) [88.104].

## 88.53 os32: getgrent(3)



### NOME

‘**getgrent**’, ‘**setgrent**’, ‘**endgrent**’ - accesso alle voci del file  
‘`/etc/group`’

### SINTASSI

```
#include <sys/types.h>
#include <grp.h>
struct group *getgrent (void);
void          setgrent (void);
void          endgrent (void);
```

### DESCRIZIONE

La funzione *getgrent()* restituisce il puntatore a una variabile strutturata, di tipo ‘**struct group**’, come definito nel file ‘`grp.h`’, in cui si possono trovare le stesse informazioni contenute nelle voci (righe) del file ‘`/etc/group`’, separate in campi. La prima volta, nella variabile struttura a cui punta la funzione si ottiene il contenuto della prima voce, ovvero del primo utente dell’elenco; nelle chiamate successive si ottengono le altre.

Si utilizza la funzione *setgrent()* per ripartire dalla prima voce del file `/etc/group`; si utilizza invece la funzione *endgrent()* per chiudere il file `/etc/group` quando non serve più.

Il tipo `struct group` è definito nel file `grp.h` nel modo seguente:

```
struct group {
    char   *gr_name;
    char   *gr_passwd;
    gid_t  gr_gid;
    char   *gr_mem[];
};
```

La sequenza dei membri della struttura corrisponde a quella dei campi delle righe del file `/etc/group`; in particolare, l'ultimo membro raccoglie i riferimenti alle stringhe che descrivono i nomi degli utenti aggregati al gruppo.

## VALORE RESTITUITO

La funzione *getgrent()* restituisce il puntatore a una variabile strutturata di tipo `struct group`, se l'operazione ha avuto successo. Se la scansione del file `/etc/group` ha raggiunto il termine, oppure se si è verificato un errore, restituisce invece il valore `NULL`. Per poter distinguere tra la conclusione del file o il verificarsi di un errore, prima della chiamata della funzione occorre azzerare il valore della variabile *errno*, verificando successivamente se ha acquisito un valore differente.

## ERRORI

Valore di <i>errno</i>	Significato
EINVAL	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
EPERM	Operazione non consentita.
EACCES	Accesso non consentito.
ENOENT	Il file non esiste, oppure non esiste il percorso che porta al file da creare.
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.

## DIFETTI

Nel file `‘/etc/group’` non è possibile lasciare campi vuoti, a parte l'ultimo, anche se questi non sono usati, perché la scansione avviene con la funzione *strtok()*. Pertanto, benché il campo della parola d'ordine non sia usato con `os32`, occorre metterci ugualmente qualcosa.

## FILE SORGENTI

`‘lib/sys/types.h’` [[95.26](#)]

`‘lib/pwd.h’` [[95.15](#)]

`‘lib/grp/grent.c’` [[95.7.1](#)]

## VEDERE ANCHE

*getgrnam(3)* [[88.54](#)], *getgrgid(3)* [[88.54](#)], *group(5)* [[90.1](#)].

## 88.54 os32: getgrnam(3)

&lt;&lt;

### NOME

‘**getgrnam**’, ‘**getgrgid**’ - selezione di una voce dal file ‘/etc/passwd’

### SINTASSI

```
#include <sys/types.h>
#include <grp.h>
struct group *getgrnam (const char *name);
struct group *getgrgid (gid_t gid);
```

### DESCRIZIONE

La funzione *getgrnam()* restituisce il puntatore a una variabile strutturata, di tipo ‘**struct group**’, come definito nel file ‘grp.h’, contenente le informazioni sul gruppo specificato per nome, dal ‘/etc/group’. La funzione *getgrgid()* si comporta in modo analogo, individuando però il gruppo da selezionare in base al numero **GID**.

Il tipo ‘**struct group**’ è definito nel file ‘grp.h’ nel modo seguente:

```
struct group {
    char   *gr_name;
    char   *gr_passwd;
    gid_t  gr_gid;
    char   *gr_mem[];
};
```

La sequenza dei membri della struttura corrisponde a quella dei campi delle righe del file ‘/etc/group’; in particolare, l’ulti-

mo membro raccoglie i riferimenti alle stringhe che descrivono i nomi degli utenti aggregati al gruppo.

## VALORE RESTITUITO

Le funzioni *getgrnam()* e *getgrgid()* restituiscono il puntatore a una variabile strutturata di tipo `'struct group'`, se l'operazione ha avuto successo. Se il nome o il numero del gruppo non si trovano nel file `'/etc/group'`, oppure se si presenta un errore, il valore restituito è `'NULL'`. Per poter distinguere tra una voce non trovata o il verificarsi di un errore di accesso al file `'/etc/group'`, prima della chiamata della funzione occorre azzerare il valore della variabile *errno*, verificando successivamente se ha acquisito un valore differente.

## ERRORI

Valore di <i>errno</i>	Significato
EINVAL	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
EPERM	Operazione non consentita.
EACCES	Accesso non consentito.
ENOENT	Il file non esiste, oppure non esiste il percorso che porta al file da creare.
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/grp.h' [95.7]

'lib/grp/grent.c' [95.7.1]

## VEDERE ANCHE

*getgrent(3)* [88.53], *setgrent(3)* [88.53], *endgrent(3)* [88.53],  
*group(5)* [90.1].

88.55 os32: *getpwuid(3)*

«

Vedere *getgrnam(3)* [88.54].

88.56 os32: *getopt(3)*

«

## NOME

'**getopt**' - scansione delle opzioni della riga di comando

## SINTASSI

```
#include <unistd.h>
extern *char optarg;
extern int optind;
extern int opterr;
extern int optopt;
int getopt (int argc, char *const argv[],
            const char *optstring);
```

## DESCRIZIONE

La funzione *getopt()* riceve, come primi due argomenti, gli stessi parametri *argc* e *argv*[], che sono già della funzione *main()* del



programma in cui *getopt()* si usa. In altri termini, *getopt()* deve conoscere la quantità degli argomenti usati per l'avvio del programma e deve poterli scandire. L'ultimo argomento di *getopt()* è una stringa contenente l'elenco delle lettere delle opzioni che ci si attende di trovare nella scansione delle stringhe dell'array *argv[]*, con altre sigle eventuali per sapere se tali opzioni sono singole o si attendono un proprio argomento.

Per poter usare la funzione *getopt()* proficuamente, è necessario che la sintassi di utilizzo del programma del quale si vuole scandire la riga di comando, sia uniforme con l'uso comune:

```
programma [-x [ argomento ] ] ... [ argomento ] ...
```

Pertanto, dopo il nome del programma possono esserci delle opzioni, riconoscibili perché composte da una sola lettera, preceduta da un trattino. Tali opzioni potrebbero richiedere un proprio argomento. Dopo le opzioni e i relativi argomenti, ci possono essere altri argomenti, al di fuori della competenza di *getopt()*. Vale anche la considerazione che più opzioni, prive di argomento, possono essere unite assieme in un'unica parola, con un solo trattino iniziale.

La funzione *getopt()* si avvale di variabili pubbliche, di cui occorre conoscere lo scopo.

La variabile *optind* viene usata da *getopt()* come indice per scandire l'array *argv[]*. Quando con gli utilizzi successivi di *optarg()* si determina che è stata completata la scansione delle opzioni (in quanto *optarg()* restituisce il valore -1), la variabile *optind* diventa utile per conoscere qual è il prossimo elemento di *argv[]*

da prendere in considerazione, trattandosi del primo argomento della riga di comando che non è un'opzione.

La variabile *opterr* serve per configurare il comportamento di *getopt()*. Questa variabile contiene inizialmente il valore 1. Quando *getopt()* incontra un'opzione per la quale si richiede un argomento, il quale risulta però mancante, se la variabile *opterr* risulta avere un valore diverso da zero, visualizza un messaggio di errore attraverso lo standard error. Pertanto, per evitare tale visualizzazione, è sufficiente assegnare preventivamente il valore zero alla variabile *opterr*.

Quando un'opzione individuata da *getopt()* risulta errata per qualche ragione (perché non prevista o perché si attende un argomento che invece non c'è), la variabile *optopt* riceve il valore (tradotto da carattere senza segno a intero) della lettera corrispondente a quell'opzione. Pertanto, in tal modo è possibile conoscere cosa ha provocato il problema.

Il puntatore *optarg* viene modificato quando *getopt()* incontra un'opzione che chiede un argomento. In tal caso, *optarg* viene modificato in modo da puntare alla stringa che rappresenta tale argomento.

La compilazione della stringa corrispondente a *optstring* deve avvenire secondo una sintassi precisa:

```
[:] [x [:]] ...
```

La stringa *optstring* può iniziare con un simbolo di due punti, quindi seguono le lettere che rappresentano le opzioni possibili, tenendo conto che quelle per cui si attende un argomento devono

anche essere seguite da due punti. Per esempio, ‘**ab:cd:**’ significa che ci può essere un’opzione ‘**-a**’, un’opzione ‘**-b**’ seguita da un argomento, un’opzione ‘**-c**’ e un’opzione ‘**-d**’ seguita da un argomento.

Per comprendere l’uso della funzione *getopt()* si propone una versione ultraridotta di *kill(1)* [86.12], dove si ammette solo l’invio dei segnali SIGTERM e SIGQUIT.

```
#include <sys/os32.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <libgen.h>
//-----
int
main (int argc, char *argv[], char *envp[])
{
    int          signal = SIGTERM;
    int          pid;
    int          a;           // Index inside arguments.
    int          opt;
    extern char *optarg;
    extern int   optopt;
    //
    while ((opt = getopt (argc, argv, ":ls:")) != -1)
    {
        switch (opt)
```

```
{
  case 'l':
    printf ("TERM ");
    printf ("KILL ");
    printf ("\n");
    return (0);
    break;
  case 's':
    if (strcmp (optarg, "KILL") == 0)
      {
        signal = SIGKILL;
      }
    else if (strcmp (optarg, "TERM") == 0)
      {
        signal = SIGTERM;
      }
    break;
  case '?':
    fprintf (stderr, "Unknown option -%c.\n",
            optopt);
    return (1);
    break;
  case ':':
    fprintf (stderr, "Missing argument for option "
            "-%c\n", optopt);
    return (1);
    break;
  default:
    fprintf (stderr, "Getopt problem: unknown "
            "option %c\n", opt);
    return (1);
}
}
//
// Scan other command line arguments.
```

```
//  
for (a = optind; a < argc; a++)  
  {  
    pid = atoi (argv[a]);  
    if (pid > 0)  
      {  
        if (kill (pid, signal) < 0)  
          {  
            perror (argv[a]);  
          }  
      }  
  }  
return (0);  
}
```

Come si vede nell'esempio, la funzione *getopt()* viene chiamata sempre nello stesso modo, all'interno di un ciclo iterativo.

Alla prima chiamata della funzione, questa esamina il primo argomento della riga di comando, verificando se si tratta di un'opzione o meno. Se si tratta di un'opzione, benché possa essere errata per qualche ragione, restituisce un carattere (convertito a intero), il quale può corrispondere alla lettera dell'opzione se questa è valida, oppure a un simbolo differente in caso di problemi. Nelle chiamate successive, *getopt()* considera di volta in volta gli argomenti successivi della riga di comando, fino a quando si accorge che non ci sono più opzioni e restituisce semplicemente il valore  $-1$ .

Durante la scansione delle opzioni, se *getopt()* restituisce il carattere '?', significa che ha incontrato un'opzione errata: potrebbe trattarsi di un'opzione non prevista, oppure di un'opzione che attende un argomento che non c'è. Tuttavia, la stringa *optstring*

potrebbe iniziare opportunamente con il simbolo di due punti, così come si vede nell'esempio. In tal caso, se *getopt()* incontra un'opzione errata in quanto mancante di un'opzione necessaria, invece di restituire '?', restituisce ':', così da poter distinguere il tipo di errore.

È il caso di osservare che le chiamate successive di *getopt()* fanno progredire la scansione della riga di comando e generalmente non c'è bisogno di tornare indietro per ripeterla. Tuttavia, nel caso lo si volesse, basterebbe reinizializzare la variabile *optind* a uno (il primo argomento della riga di comando).

## FILE SORGENTI

'lib/unistd.h' [95.30]

'lib/unistd/getopt.c' [95.30.23]

## 88.57 os32: getpwent(3)

«

### NOME

'getpwent', 'setpwent', 'endpwent' - accesso alle voci del file  
'/etc/passwd'

### SINTASSI

```
#include <sys/types.h>
#include <pwd.h>
struct passwd *getpwent (void);
void          setpwent (void);
void          endpwent (void);
```

## DESCRIZIONE

La funzione *getpwent()* restituisce il puntatore a una variabile strutturata, di tipo `'struct passwd'`, come definito nel file `'pwd.h'`, in cui si possono trovare le stesse informazioni contenute nelle voci (righe) del file `'/etc/passwd'`, separate in campi. La prima volta, nella variabile struttura a cui punta la funzione si ottiene il contenuto della prima voce, ovvero del primo utente dell'elenco; nelle chiamate successive si ottengono le altre.

Si utilizza la funzione *setpwent()* per ripartire dalla prima voce del file `'/etc/passwd'`; si utilizza invece la funzione *endpwent()* per chiudere il file `'/etc/passwd'` quando non serve più.

Il tipo `'struct passwd'` è definito nel file `'pwd.h'` nel modo seguente:

```
struct passwd {
    char    *pw_name;
    char    *pw_passwd;
    uid_t   pw_uid;
    gid_t   pw_gid;
    char    *pw_gecos;
    char    *pw_dir;
    char    *pw_shell;
};
```

La sequenza dei campi della struttura corrisponde a quella contenuta nel file `'/etc/passwd'`.

## VALORE RESTITUITO

La funzione *getpwent()* restituisce il puntatore a una variabile strutturata di tipo `'struct passwd'`, se l'operazione ha avuto successo. Se la scansione del file `'/etc/passwd'` ha raggiunto

il termine, oppure se si è verificato un errore, restituisce invece il valore **'NULL'**. Per poter distinguere tra la conclusione del file o il verificarsi di un errore, prima della chiamata della funzione occorre azzerare il valore della variabile *errno*, verificando successivamente se ha acquisito un valore differente.

## ERRORI

Valore di <i>errno</i>	Significato
EINVAL	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
EPERM	Operazione non consentita.
EACCES	Accesso non consentito.
ENOENT	Il file non esiste, oppure non esiste il percorso che porta al file da creare.
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.

## DIFETTI

Nel file `‘/etc/passwd’` non è possibile lasciare campi vuoti, anche se questi non sono usati, perché la scansione avviene con la funzione *strtok()*.

## FILE SORGENTI

`‘lib/sys/types.h’` [95.26]

`‘lib/pwd.h’` [95.15]



‘lib/pwd/pwent.c’ [95.15.1]

## VEDERE ANCHE

*getpwnam(3)* [88.58], *getpwuid(3)* [88.58], *passwd(5)* [90.4].

## 88.58 os32: getpwnam(3)



### NOME

‘**getpwnam**’, ‘**getpwuid**’ - selezione di una voce dal file ‘/etc/passwd’

### SINTASSI

```
#include <sys/types.h>
#include <pwd.h>
struct passwd *getpwnam (const char *name);
struct passwd *getpwuid (uid_t uid);
```

### DESCRIZIONE

La funzione *getpwnam()* restituisce il puntatore a una variabile strutturata, di tipo ‘**struct passwd**’, come definito nel file ‘pwd.h’, contenente le informazioni sull’utenza specificata per nome, dal ‘/etc/passwd’. La funzione *getpwuid()* si comporta in modo analogo, individuando però l’utenza da selezionare in base al numero UID.

Il tipo ‘**struct passwd**’ è definito nel file ‘pwd.h’ nel modo seguente:

```
struct passwd {
    char    *pw_name;
    char    *pw_passwd;
    uid_t   pw_uid;
    gid_t   pw_gid;
    char    *pw_gecos;
    char    *pw_dir;
    char    *pw_shell;
};
```

La sequenza dei campi della struttura corrisponde a quella contenuta nel file `‘/etc/passwd’`.

## VALORE RESTITUITO

Le funzioni *getpwnam()* e *getpwuid()* restituiscono il puntatore a una variabile strutturata di tipo `‘struct passwd’`, se l’operazione ha avuto successo. Se il nome o il numero dell’utente non si trovano nel file `‘/etc/passwd’`, oppure se si presenta un errore, il valore restituito è `‘NULL’`. Per poter distinguere tra una voce non trovata o il verificarsi di un errore di accesso al file `‘/etc/passwd’`, prima della chiamata della funzione occorre azzerare il valore della variabile *errno*, verificando successivamente se ha acquisito un valore differente.

## 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' [[95.26](#)]

'lib/pwd.h' [[95.15](#)]

'lib/pwd/pwent.c' [[95.15.1](#)]

## VEDERE ANCHE

*getpwent(3)* [[88.57](#)], *setpwent(3)* [[88.57](#)], *endpwent(3)* [[88.57](#)], *passwd(5)* [[90.4](#)].

88.59 os32: `getpwuid(3)`

«

Vedere *getpwnam(3)* [88.58].

88.60 os32: `gets(3)`

«

Vedere *fgets(3)* [88.34].

88.61 os32: `gmtime(3)`

«

Vedere *ctime(3)* [88.15].

88.62 os32: `htonl(3)`

«

Vedere *byteorder(3)* [88.11].

88.63 os32: `htons(3)`

«

Vedere *byteorder(3)* [88.11].

88.64 os32: `imaxabs(3)`

«

Vedere *abs(3)* [88.3].

88.65 os32: `imaxdiv(3)`

«

Vedere *div(3)* [88.17].

88.66 os32: `inet_ntop(3)`

«

## **NOME**

‘**inet\_ntop**’ - conversione di un indirizzo IP dalla sua forma numerica a quella testuale

## SINTASSI

```
#include <arpa/inet.h>

const char *inet_ntop (int family, const void *src,
                      char *dst, socklen_t size);
```

## DESCRIZIONE

La funzione *inet\_ntop()* converte un indirizzo IPv4, contenuto in una variabile strutturata di tipo ‘**struct in\_addr**’, a cui punta *src*, in una stringa, annotandola nella memoria tampone a cui punta *dst*, la cui dimensione massima è di *size* byte.

Il parametro *family* indica il tipo di indirizzo da convertire, ma per os32 è ammesso soltanto il tipo *AF\_INET*, corrispondente a IPv4.

## VALORE RESTITUITO

Se l’operazione si conclude con successo, la funzione restituisce il puntatore a *dst*; altrimenti, si ottiene il valore *NULL* e va verificato il contenuto della variabile *errno*.

## ERRORI

Valore di <i>errno</i>	Significato
EAFNOSUPPORT	È stata richiesta la conversione di un tipo di indirizzo diverso da <i>AF_INET</i> .
EINVAL	In corrispondenza del parametro <i>src</i> o di <i>dst</i> si ha un puntatore nullo.

## FILE SORGENTI

‘lib/arpa/inet.h’ [95.3]

‘lib/arpa/inet/inet\_ntop.c’ [95.3.3]

## VEDERE ANCHE

*inet\_pton(3)* [88.67].

## 88.67 os32: inet\_pton(3)

«

### NOME

‘**inet\_pton**’ - conversione di un indirizzo IP dalla sua forma testuale a quella numerica

### SINTASSI

```
#include <arpa/inet.h>
int inet_pton (int family, const char *src, void *dst);
```

### DESCRIZIONE

La funzione *inet\_pton()* converte un indirizzo IPv4, contenuto nella stringa *src*, in una variabile strutturata di tipo ‘**struct in\_addr**’, a cui punta *dst*.

Il parametro *family* indica il tipo di indirizzo da convertire, ma per os32 è ammesso soltanto il tipo *AF\_INET*, corrispondente a IPv4.

### VALORE RESTITUITO

Se l’operazione si conclude con successo, la funzione restituisce il valore numerico 1; se invece la stringa *src* non contiene un indirizzo valido, la funzione restituisce 0; se il valore di *family* non è valido o non è gestito, restituisce -1.

## ERRORI

La variabile *errno* non viene modificata, perché tutti i casi che possono presentarsi sono espressi attraverso il valore restituito dalla funzione.

## FILE SORGENTI

'lib/arpa/inet.h' [95.3]

'lib/arpa/inet/inet\_pton.c' [95.3.4]

## VEDERE ANCHE

*inet\_ntop(3)* [88.66].

## 88.68 os32: input\_line(3)



## NOME

'input\_line' - riga di comando

## SINTASSI

```
#include <sys/os32.h>
void input_line (char *line, char *prompt, size_t size,
                int type);
```

## DESCRIZIONE

La funzione *input\_line()* consente di inserire un'informazione da tastiera, interpretando in modo adeguato i codici usati per cancellare. Si tratta del mezzo principale di inserimento di un dato da tastiera per os32.

Il parametro *line* è il puntatore a un'area di memoria, da modificare con l'inserimento che si intende fare; questa area di memoria

deve essere in grado di contenere tanti byte quanto indicato con il parametro *size*. Il parametro *prompt* indica una stringa da usare come invito, a sinistra della riga da inserire. Il parametro *type* serve a specificare il tipo di visualizzazione sullo schermo di ciò che si inserisce. Si utilizzano della macro-variabili dichiarate nel file ‘`sys/os32.h`’:

Macro-variabile	Descrizione
<code>INPUT_LINE_ECHO</code>	Produce un comportamento «normale», per cui ciò che viene digitato è rappresentato conformemente sullo schermo del terminale.
<code>INPUT_LINE_HIDDEN</code>	Fa sì che quanto digitato non appaia: si usa per esempio per l’inserimento di una parola d’ordine.

La funzione conclude il suo funzionamento quando si preme [*Invio*].

## VALORE RESTITUITO

La funzione non restituisce alcunché, ma ciò che viene digitato è disponibile nella memoria tampone rappresentata dal puntatore *line*, da intendere come stringa terminata correttamente.

## FILE SORGENTI

‘`lib/sys/os32.h`’ [[95.21](#)]

‘`lib/sys/os32/input_line.c`’ [[95.21.1](#)]

## VEDERE ANCHE

*shell(1)* [[86.24](#)].

*login(1)* [[86.14](#)].



## 88.69 os32: isatty(3)

**NOME**

‘**isatty**’ - verifica che un certo descrittore di file si riferisca a un terminale

**SINTASSI**

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

**DESCRIZIONE**

La funzione *isatty()* verifica se il descrittore di file specificato con il parametro *fdn* si riferisce a un dispositivo di terminale.

**VALORE RESTITUITO**

Valore	Significato
1	Si tratta effettivamente del file di dispositivo di un terminale.
0	Il descrittore di file non riguarda un terminale, oppure si è verificato un errore; in ogni caso la variabile <i>errno</i> viene aggiornata.

**ERRORI**

Valore di <i>errno</i>	Significato
EINVAL	Il descrittore di file non si riferisce a un terminale.
EBADF	Il descrittore di file indicato non è valido.

**FILE SORGENTI**

‘lib/unistd.h’ [95.30]

‘lib/unistd/isatty.c’ [95.30.28]

## VEDERE ANCHE

*stat(2)* [87.55], *ttynname(3)* [88.133].

88.70 os32: *labs(3)*

« Vedere *abs(3)* [88.3].

88.71 os32: *ldiv(3)*

« Vedere *div(3)* [88.17].

88.72 os32: *llabs(3)*

« Vedere *abs(3)* [88.3].

88.73 os32: *lldiv(3)*

« Vedere *div(3)* [88.17].

88.74 os32: *major(3)*

« Vedere *makedev(3)* [88.75].

88.75 os32: *makedev(3)*

«

## NOME

**'makedev', 'major', 'minor'** - gestione dei numeri di dispositivo

## SINTASSI

```
#include <sys/types.h>
dev_t makedev (int major, int minor);
int major (dev_t device);
int minor (dev_t device);
```

## DESCRIZIONE

La funzione *makedev()* restituisce il numero di dispositivo complessivo, partendo dal numero primario (*major*) e dal numero secondario (*minor*), presi separatamente.

Le funzioni *major()* e *minor()*, rispettivamente, restituiscono il numero primario o il numero secondario, partendo da un numero di dispositivo completo.

Si tratta di funzioni non previste dallo standard, ma ugualmente diffuse.

## FILE SORGENTI

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

'lib/sys/types/makedev.c' [95.26.2]

'lib/sys/types/major.c' [95.26.1]

'lib/sys/types/minor.c' [95.26.3]

## 88.76 os32: malloc(3)



## NOME

'**malloc**', '**free**', '**realloc**' - allocazione e rilascio dinamico di memoria

## SINTASSI

```
#include <stdlib.h>
void *malloc (size_t size);
void free (void *address);
void *realloc (void *address, size_t size);
```

## DESCRIZIONE

Le funzioni ‘...**alloc()**’ e *free()* consentono di allocare, riallocare e liberare delle aree di memoria, in modo dinamico.

La funzione *malloc()* (*memory allocation*) si usa per richiedere l’allocazione di una dimensione di almeno *size* byte di memoria. Se l’allocazione avviene con successo, la funzione restituisce il puntatore generico di tale area allocata.

Quando un’area di memoria allocata precedentemente non serve più, va liberata espressamente con l’ausilio della funzione *free()*, la quale richiede come argomento il puntatore generico all’inizio di tale area. Naturalmente, si può liberare la memoria una volta sola e un’area di memoria liberata non può più essere raggiunta.

Quando un’area di memoria già allocata richiede una modifica nella sua estensione, si può usare la funzione *realloc()*, la quale necessita di conoscere il puntatore precedente e la nuova estensione. La funzione restituisce un nuovo puntatore, il quale potrebbe eventualmente, ma non necessariamente, coincidere con quello dell’area originale.

Se le funzioni *malloc()* e *realloc()* falliscono nel loro intento, restituiscono un puntatore nullo.

## VALORE RESTITUITO

Le funzioni *malloc()* e *realloc()* restituiscono il puntatore generico all’area di memoria allocata; se falliscono, restituiscono invece un puntatore nullo.

## ERRORI

Valore di <i>errno</i>	Significato
ENOMEM	Memoria insufficiente.

## FILE SORGENTI

‘lib/limits.h’ [95.1.6]

‘lib/stdlib.h’ [95.19]

‘lib/stdlib\_alloc/malloc.c’ [95.19.24]

‘lib/stdlib\_alloc/realloc.c’ [95.19.25]

‘lib/stdlib\_alloc/free.c’ [95.19.23]

‘lib/stdlib\_alloc/\_alloc\_list.c’ [95.19.22]

## 88.77 os32: memccpy(3)

«

## NOME

‘**memccpy**’ - copia di un’area di memoria

## SINTASSI

```
#include <string.h>
void *memccpy (void *restrict dst, const void *restrict org, int
                size_t n);
```

## DESCRIZIONE

La funzione *memccpy()* copia al massimo *n* byte a partire dall’area di memoria a cui punta *org*, verso l’area che inizia da *dst*, fermandosi se si incontra il carattere *c*, il quale viene copiato regolarmente, fermo restando il limite massimo di *n* byte.

Le due aree di memoria, origine e destinazione, non devono sovrapporsi.

## VALORE RESTITUITO

Nel caso in cui la copia sia avvenuta con successo, fino a incontrare il carattere *c*, la funzione restituisce il puntatore al carattere successivo a *c*, nell'area di memoria di destinazione. Se invece tale carattere non viene trovato nei primi *n* byte, restituisce il puntatore nullo 'NULL'. La variabile *errno* non viene modificata.

## FILE SORGENTI

'lib/string.h' [95.20]

'lib/string/memccpy.c' [95.20.1]

## VEDERE ANCHE

*memcpy(3)* [88.80], *memmove(3)* [88.81], *strcpy(3)* [88.117], *strncpy(3)* [88.117].

## 88.78 os32: memchr(3)

«

## NOME

'**memchr**' - scansione della memoria alla ricerca di un carattere

## SINTASSI

```
#include <string.h>
void *memchr (const void *memory, int c, size_t n);
```

## DESCRIZIONE

La funzione *memchr()* scandisce l'area di memoria a cui punta *memory*, fino a un massimo di *n* byte, alla ricerca del carattere *c*.

## VALORE RESTITUITO

Se la funzione trova il carattere, restituisce il puntatore al carattere trovato, altrimenti restituisce il puntatore nullo **'NULL'**.

## FILE SORGENTI

`'lib/string.h'` [95.20]

`'lib/string/memchr.c'` [95.20.2]

## VEDERE ANCHE

*strchr(3)* [88.114], *strrchr(3)* [88.114], *strpbrk(3)* [88.125].

## 88.79 os32: memcmp(3)



## NOME

**'memcmp'** - confronto di due aree di memoria

## SINTASSI

```
#include <string.h>
int memcmp (const void *memory1, const void *memory2,
           size_t n);
```

## DESCRIZIONE

La funzione *memcmp()* confronta i primi *n* byte di memoria delle aree che partono, rispettivamente, da *memory1* e da *memory2*.

## VALORE RESTITUITO

Valore	Esito del confronto.
-1	<i>memory1</i> < <i>memory2</i>
0	<i>memory1</i> == <i>memory2</i>
+1	<i>memory1</i> > <i>memory2</i>

## FILE SORGENTI

'lib/string.h' [95.20]

'lib/string/memcmp.c' [95.20.3]

## VEDERE ANCHE

*strcmp(3)* [88.115], *strncmp(3)* [88.115].

## 88.80 os32: memcpy(3)

<<

## NOME

'**memcpy**' - copia di un'area di memoria

## SINTASSI

```
#include <string.h>
void *memcpy (void *restrict dst, const void *restrict org,
              size_t n);
```

## DESCRIZIONE

La funzione *memcpy()* copia al massimo *n* byte a partire dall'area di memoria a cui punta *org*, verso l'area che inizia da *dst*.

Le due aree di memoria, origine e destinazione, non devono sovrapporsi.

## VALORE RESTITUITO

La funzione restituisce *dst*.

## FILE SORGENTI

'lib/string.h' [95.20]

'lib/string/memcpy.c' [95.20.4]



## VEDERE ANCHE

*memccpy(3)* [88.77], *memmove(3)* [88.81], *strcpy(3)* [88.117], *strncpy(3)* [88.117].

## 88.81 os32: memmove(3)



## NOME

‘**memmove**’ - copia di un’area di memoria

## SINTASSI

```
#include <string.h>
void *memmove (void *dst, const void *org, size_t n);
```

## DESCRIZIONE

La funzione *memmove()* copia al massimo *n* byte a partire dall’area di memoria a cui punta *org*, verso l’area che inizia da *dst*. A differenza di quanto fa *memcpy()*, la funzione *memmove()* esegue la copia correttamente anche se le due aree di memoria sono sovrapposte.

## VALORE RESTITUITO

La funzione restituisce *dst*.

## FILE SORGENTI

‘lib/string.h’ [95.20]

‘lib/string/memmove.c’ [95.20.5]

## VEDERE ANCHE

*memccpy(3)* [88.77], *memcpy(3)* [88.80], *strcpy(3)* [88.117], *strncpy(3)* [88.117].

## 88.82 os32: memset(3)

&lt;&lt;

### NOME

‘**memset**’ - scrittura della memoria con un byte sempre uguale

### SINTASSI

```
#include <string.h>
void *memset (void *memory, int c, size_t n);
```

### DESCRIZIONE

La funzione *memset()* scrive *n* byte, contenenti il valore di *c*, ridotto a un carattere, a partire dal ciò a cui punta *memory*.

### FILE SORGENTI

‘lib/string.h’ [95.20]

‘lib/string/memset.c’ [95.20.6]

### VEDERE ANCHE

*memcpy(3)* [88.80].

## 88.83 os32: minor(3)

&lt;&lt;

Vedere *makedev(3)* [88.75].

## 88.84 os32: mktime(3)

&lt;&lt;

Vedere *ctime(3)* [88.15].

## 88.85 os32: namep(3)



### NOME

‘**namep**’ - ricerca del percorso di un programma utilizzando la variabile di ambiente **PATH**

### SINTASSI

```
#include <sys/os32.h>
int namep (const char *name, char *path, size_t size);
```

### DESCRIZIONE

La funzione **namep()** trova il percorso di un programma, tenendo conto delle informazioni contenute nella variabile di ambiente **PATH**.

Il parametro **name** rappresenta una stringa con il nome del comando da cercare nel file system; il parametro **path** deve essere il puntatore di un'area di memoria, da sovrascrivere con il percorso assoluto del programma da avviare, una volta trovato, con l'accortezza di far sì che risulti una stringa terminata correttamente; il parametro **size** specifica la dimensione massima che può avere la stringa **path**.

Questa funzione viene utilizzata in particolare da **execvp()**.

### VALORE RESTITUITO

Valore	Significato
0	Operazione riuscita.
-1	Operazione fallita. Va verificato l'errore indicato dalla variabile <b>errno</b> .

## ERRORI

Valore di <i>errno</i>	Significato
EINVAL	Uno degli argomenti non è valido.
ENOENT	La variabile di ambiente <i>PATH</i> non è dichiarata, oppure il comando richiesto non si trova nei percorsi previsti.
ENAMETOOLONG	Il percorso per l'avvio del programma è troppo lungo.

## FILE SORGENTI

'lib/sys/os32.h' [[95.21](#)]

'lib/sys/os32/namep.c' [[95.21.4](#)]

## VEDERE ANCHE

*shell(1)* [[86.24](#)], *execvp(3)* [[88.21](#)], *execlp(3)* [[88.21](#)].

88.86 os32: ntohl(3)

«

Vedere *byteorder(3)* [[88.11](#)].

88.87 os32: ntohs(3)

«

Vedere *byteorder(3)* [[88.11](#)].

88.88 os32: offsetof(3)

«

## NOME

'**offsetof**' - posizione di un membro di una struttura, dall'inizio della stessa

## SINTASSI

```
#include <stddef.h>
size_t offsetof (type, member);
```

## DESCRIZIONE

La macroistruzione *offsetof()* consente di determinare la collocazione relativa di un membro di una variabile strutturata, restituendo la quantità di byte che la struttura occupa prima dell'inizio del membro richiesto. Per ottenere questo risultato, il primo argomento deve essere il nome del tipo del membro cercato, mentre il secondo argomento deve essere il nome del membro stesso.

## VALORE RESTITUITO

La macroistruzione restituisce lo scostamento del membro specificato, rispetto all'inizio della struttura a cui appartiene, espresso in byte.

## FILE SORGENTI

'lib/stddef.h' [[95.1.12](#)]

88.89 os32: opendir(3)

## NOME

'**opendir**' - apertura di una directory

## SINTASSI

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir (const char *name);
```



## DESCRIZIONE

La funzione *opendir()* apre la directory rappresentata da *name*, posizionando l'indice interno per le operazioni di accesso alla prima voce della directory stessa.

## VALORE RESTITUITO

La funzione restituisce il puntatore al flusso aperto; in caso di errore, restituisce 'NULL' e aggiorna la variabile *errno*.

## ERRORI

Valore di <i>errno</i>	Significato
EINVAL	Il valore di <i>name</i> non è valido.
EMFILE	Troppi file aperti dal processo.
ENFILE	Troppi file aperti complessivamente nel sistema.
ENOTDIR	Il percorso indicato in <i>name</i> non corrisponde a una directory.

## NOTE

La funzione *opendir()* attiva il bit *close-on-exec*, rappresentato dalla macro-variabile *FD\_CLOEXEC*, per il descrittore del file che rappresenta la directory. Ciò serve a garantire che la directory venga chiusa quando si utilizzano le funzioni 'exec... ()'.

## FILE SORGENTI

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

'lib/dirent.h' [95.4]

'lib/dirent/DIR.c' [95.4.1]

'lib/dirent/opendir.c' [95.4.3]

## VEDERE ANCHE

*open(2)* [87.37], *closedir(3)* [88.13], *readdir(3)* [88.98],  
*rewinddir(3)* [88.101].

## 88.90 os32: perror(3)

### NOME

‘**perror**’ - emissione di un messaggio di errore di sistema

### SINTASSI

```
#include <stdio.h>
void perror (const char *string);
```

### DESCRIZIONE

La funzione *perror()* legge il valore della variabile *errno* e, se questo è diverso da zero, emette attraverso lo standard output la stringa fornita come argomento, ammesso che non si tratti del puntatore nullo, quindi continua con l'emissione della descrizione dell'errore.

La funzione *perror()* di os32, nell'emettere il testo dell'errore, mostra anche il nome del file sorgente e il numero della riga in cui si è verificato. Ma questi dati sono validi soltanto se l'annotazione dell'errore è avvenuta, a suo tempo, con l'ausilio della funzione *errset(3)* [88.20], la quale non è prevista dagli standard.

### FILE SORGENTI

‘lib/errno.h’ [95.5]

‘lib/stdio.h’ [95.18]

‘lib/stdio/perror.c’ [95.18.26]

## VEDERE ANCHE

*errno(3)* [88.20], *strerror(3)* [88.120].

## 88.91 os32: printf(3)

«

### NOME

‘**printf**’, ‘**fprintf**’, ‘**sprintf**’, ‘**snprintf**’ - composizione dei dati per la visualizzazione

### SINTASSI

```
#include <stdio.h>
int printf (char *restrict format, ...);
int fprintf (FILE *fp, char *restrict format, ...);
int sprintf (char *restrict string,
             const char *restrict format, ...);
int snprintf (char *restrict string, size_t size,
             const char *restrict format, ...);
```

### DESCRIZIONE

Le funzioni del gruppo ‘**...printf()**’ hanno in comune lo scopo di comporre dei dati in forma di stringa, generalmente per la visualizzazione, o comunque per la fruizione a livello umano.

I dati in ingresso possono essere vari e si collocano come argomenti finali, di tipo e quantità non noti nel prototipo delle funzioni. Per quantificare e qualificare questi argomenti aggiuntivi, la stringa a cui punta il parametro *format*, deve contenere degli *specificatori di conversione*, oltre eventualmente ad altri caratteri. Pertanto, queste funzioni, prendono la stringa a cui punta



*format*, la interpretano e determinano quali argomenti variabili sono presenti, quindi producono un'altra stringa, composta dalla stringa precedente, sostituendo gli specificatori di conversione con i dati a cui questi si riferiscono, secondo una forma di conversione definita dagli specificatori stessi. Si osservi l'esempio seguente:

```
printf ("Valore: %x %i %o\n", 123, 124, 125);
```

In questo modo si ottiene la visualizzazione, attraverso lo standard output, della stringa **Valore: 7b 124 175**. Infatti: **'%x'** è uno specificatore di conversione che richiede di interpretare il proprio parametro (in questo caso il primo) come intero normale e di rappresentarlo in esadecimale; **'%i'** legge un numero intero normale e lo rappresenta nella forma decimale consueta; **'%o'** legge un intero e lo mostra in ottale.

La funzione *printf()* emette il risultato della composizione attraverso lo standard output; la funzione *fprintf()* lo fa attraverso il flusso di file *fp*; le funzioni *sprintf()* e *snprintf()* si limitano a scrivere il risultato a partire da ciò a cui punta *string*, con la particolarità di *snprintf()* che si dà comunque un limite da non superare, per evitare che la scrittura vada a sovrascrivere altri dati in memoria.

Gli specificatori di conversione devono rispettare la sintassi seguente per la libreria di `os32`:

```
% [ simbolo ] [ n_ampiezza ] [ .n_precision ] [ hh | h | l | ll | j | z | t ] tipo
```

La prima cosa da individuare in uno specificatore di conversione è il tipo di argomento che viene interpretato e, di conseguenza, il

genere di rappresentazione che se ne vuole produrre. Il tipo viene espresso da una lettera alfabetica, alla fine dello specificatore di conversione.

Simbolo	Tipo di argomento	Conversione applicata
%...d %...i	int	Numero intero con segno da rappresentare in base dieci.
%...u	unsigned int	Numero intero senza segno da rappresentare in base dieci.
%...o	unsigned int	Numero intero senza segno da rappresentare in ottale (senza lo zero iniziale che viene usato spesso per caratterizzare un tale tipo di rappresentazione).
%...x %...X	unsigned int	Numero intero senza segno da rappresentare in esadecimale (senza il prefisso '0x' o '0X' che viene usato spesso per caratterizzare un tale tipo di rappresentazione).
%...c	int	Un carattere singolo, dopo la conversione in ' <b>unsigned char</b> '.
%...b	unsigned int	Numero intero senza segno da rappresentare in binario (si tratta di un'estensione non prevista nello standard).
%...s	char *	Una stringa.
%%		Questo specificatore si limita a produrre un carattere di percentuale ('%') che altrimenti non sarebbe rappresentabile.

Nel modello sintattico che descrive lo specificatore di conversio-

ne, si vede che subito dopo il segno di percentuale può apparire un simbolo (*flag*).

Simbolo	Corrispondenza
%+... %+0 <i>ampiezza</i> ...	Il segno «+» fa sì che i numeri con segno lo mostrino anche se è positivo. Può combinarsi con lo zero.
%0 <i>ampiezza</i> ... %+0 <i>ampiezza</i> ...	Lo zero fa sì che siano inseriti degli zeri a sinistra per allineare a destra il valore, nell'ambito dell'ampiezza specificata. Può combinarsi con il segno «+».
% <i>ampiezza</i> ... % <i> </i> <i>ampiezza</i> ...	In mancanza di uno zero iniziale, in presenza dell'indicazione dell'ampiezza, il valore viene allineato a destra usando degli spazi. È possibile esprimere esplicitamente l'intenzione di usare gli spazi mettendo proprio uno spazio, ma in generale non è richiesto. Se si mette lo spazio letteralmente, questo non è poi compatibile con lo zero, mentre le combinazioni con gli altri simboli sono ammissibili.
%- <i>ampiezza</i> ... %-+ <i>ampiezza</i> ...	Il segno meno, usato quando la conversione prevede l'uso di una quantità fissa di caratteri con un valore che appare di norma allineato a destra, fa sì che il risultato sia allineato a sinistra. Il segno meno si può combinare il segno «+» e il cancelletto.

Subito prima della lettera che definisce il tipo di conversione, possono apparire una o due lettere che modificano la lunghezza del valore da interpretare (per lunghezza si intende qui la quantità di byte usati per rappresentarlo). Per esempio, '%...li' indica che la conversione riguarda un valore di tipo 'long int'. Tra questi specificatori della lunghezza del dato in ingresso ce ne sono

alcuni che indicano un rango inferiore a quello di **'int'**, come per esempio **'%...hhd'** che si riferisce a un numero intero della dimensione di un **'signed char'**; in questi casi occorre comunque considerare che nella trasmissione degli argomenti alle funzioni interviene sempre la promozione a intero, pertanto viene letto il dato della dimensione specificata, ma viene «consumato» il risultato ottenuto dalla promozione.

Simbolo	Tipo	Simbolo	Tipo
%...hhd %...hhi	signed char	%...hhu %...hho %...hhx   %...hhX %...hbb	unsigned char
%...hd %...hi	short int	%...hu %...ho %...hx   %...hX %...hb	unsigned short int

Simbolo	Tipo	Simbolo	Tipo
<code>%ld</code> <code>%li</code>	<code>long int</code>	<code>%lu</code> <code>%lo</code> <code>%lx</code>   <code>%LX</code> <code>%lb</code>	<code>unsigned long int</code>
<code>%lld</code> <code>%lli</code>	<code>long long int</code>	<code>%llu</code> <code>%lo</code> <code>%llx</code>   <code>%LX</code> <code>%lb</code>	<code>unsigned long long int</code>

Simbolo	Tipo	Simbolo	Tipo
<code>%jd</code> <code>%ji</code>	<code>intmax_t</code>	<code>%ju</code> <code>%jo</code> <code>%jx</code>   <code>%jX</code> <code>%jb</code>	<code>uintmax_t</code>

Simbolo	Tipo	Simbolo	Tipo
%...zd %...zi	size_t	%...zu %...zo %...zx   %...zX %...zb	size_t

Simbolo	Tipo	Simbolo	Tipo
%...td %...ti	ptrdiff_t	%...tu %...to %...tx   %...tX %...tb	ptrdiff_t

Tra il simbolo (*flag*) e il modificatore di lunghezza può apparire un numero che rappresenta l'ampiezza da usare nella trasformazione ed eventualmente la precisione: '*ampiezza* [*.precisione*]' . Per os32, la precisione si applica esclusivamente alle stringhe, la quale specifica la quantità di caratteri da considerare, troncando il resto.

## VALORE RESTITUITO

Le funzioni restituiscono la quantità di caratteri utilizzati nella composizione della nuova stringa, escluso il carattere nullo di terminazione.

## FILE SORGENTI

'lib/stdio.h' [95.18]

'lib/stdio/FILE.c' [95.18.1]

'lib/stdio/fprintf.c' [95.18.12]

'lib/stdio/printf.c' [95.18.27]

'lib/stdio/sprintf.c' [95.18.35]

'lib/stdio/snprintf.c' [95.18.34]

## VEDERE ANCHE

*vfprintf(3)* [88.137], *vprintf(3)* [88.137], *vsprintf(3)* [88.137],  
*vsnprintf(3)* [88.137], *scanf(3)* [88.102].

88.92 os32: *putc(3)*

Vedere *fputc(3)* [88.38].

«

88.93 os32: *putchar(3)*

Vedere *fputc(3)* [88.38].

«

88.94 os32: *putenv(3)*

«

## NOME

'**putenv**' - assegnamento di una variabile di ambiente

## SINTASSI

```
#include <stdlib.h>
int putenv (const char *string);
```

## DESCRIZIONE

La funzione `putenv()` assegna una variabile di ambiente. Se questa esiste già, va a rimpiazzare il valore assegnatole in precedenza, altrimenti la crea contestualmente.

La funzione richiede un solo parametro, costituito da una stringa in cui va specificato il nome della variabile e il contenuto da assegnargli, usando la forma `'nome=valore'`. Per esempio, `'PATH=/bin:/usr/bin'`.

## VALORE RESTITUITO

Valore	Significato
0	Operazione riuscita.
-1	Operazione fallita. Va verificato l'errore indicato dalla variabile <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	Significato
EINVAL	Argomento non valido.
ENOMEM	Memoria insufficiente.

## FILE SORGENTI

'lib/stdlib.h' [95.19]

'applic/crt0.mer.s' [96.1.12]

'applic/crt0.sep.s' [96.1.13]

'lib/stdlib/environment.c' [95.19.8]

'lib/stdlib/putenv.c' [95.19.15]



## VEDERE ANCHE

*environ*(7) [91.1], *getenv*(3) [88.52], *setenv*(3) [88.104],  
*unsetenv*(3) [88.104].

88.95 os32: *puts*(3)

Vedere *fputs*(3) [88.39].

88.96 os32: *qsort*(3)

## NOME

‘**qsort**’ - riordino di un array

## SINTASSI

```
#include <stdlib.h>
void qsort (void *base, size_t nmemb, size_t size,
            int (*compare)(const void *, const void *));
```

## DESCRIZIONE

La funzione *qsort*() riordina un array composto da *nmemb* elementi da *size* byte ognuno. Il primo argomento, ovvero il parametro *base*, è il puntatore all’indirizzo iniziale di questo array in memoria.

Il riordino avviene comparando i vari elementi con l’ausilio di una funzione, passata tramite il suo puntatore, la quale deve ricevere due argomenti, costituiti dai puntatori agli elementi dell’array da confrontare. Tale funzione deve restituire un valore minore di zero per un confronto in cui il suo primo argomento deve essere collocato prima del secondo; un valore pari a zero se

gli argomenti sono uguali ai fini del riordino; un valore maggiore di zero se il suo primo argomento va collocato dopo il secondo nel riordino.

Segue un esempio di utilizzo della funzione *qsort()*:

```
#include <stdio.h>
#include <stdlib.h>

int confronta (const void *a, const void *b)
{
    int x = *((int *) a);
    int y = *((int *) b);
    return x - y;
}

int main (void)
{
    int a[] = {3, 1, 5, 2};

    qsort (&a[0], 4, sizeof (int), confronta);
    printf ("%d %d %d %d\n", a[0], a[1], a[2], a[3]);

    return 0;
}
```

## FILE SORGENTI

‘lib/stdlib.h’ [95.19]

‘lib/stdlib/qsort.c’ [95.19.16]

## 88.97 os32: rand(3)

**NOME**

‘**rand**’ - generazione di numeri pseudo-casuali

**SINTASSI**

```
#include <stdlib.h>
int  rand  (void);
void srand (unsigned int seed);
```

**DESCRIZIONE**

La funzione *rand()* produce un numero intero casuale, sulla base di un seme, il quale può essere cambiato in ogni momento, con l’ausilio di *srand()*. A ogni chiamata della funzione *rand()*, il risultato ottenuto, viene utilizzato anche come seme per la chiamata successiva. Se inizialmente non viene assegnato alcun seme, il primo valore predefinito è pari a 1.

**VALORE RESTITUITO**

La funzione *rand()* restituisce un numero intero casuale, determinato sulla base del seme accumulato in precedenza.

**FILE SORGENTI**

‘lib/stdlib.h’ [[95.19](#)]

‘lib/stdlib/rand.c’ [[95.19.17](#)]

## 88.98 os32: readdir(3)

**NOME**

‘**readdir**’ - lettura di una directory

**SINTASSI**

```
#include <sys/types.h>
#include <dirent.h>
struct dirent *readdir (DIR *dp);
```

**DESCRIZIONE**

La funzione *readdir()* legge una voce dalla directory rappresentata da *dp* e restituisce il puntatore a una variabile strutturata di tipo ‘**struct dirent**’, contenente le informazioni tratte dalla voce letta. La variabile strutturata in questione si trova in memoria statica e viene sovrascritta con le chiamate successive della funzione *readdir()*.

Il tipo ‘**struct dirent**’ è definito nel file di intestazione ‘*dirent.h*’, nel modo seguente:

```
struct dirent {
    ino_t      d_ino;
    char      d_name[NAME_MAX+1];
};
```

Il membro *d\_ino* è il numero di inode del file il cui nome appare nel membro *d\_name*. La macro-variabile *NAME\_MAX* è dichiarata a sua volta nel file di intestazione ‘*limits.h*’. La dimensione del membro *d\_name* è tale da permettere di includere anche il valore zero di terminazione delle stringhe.

## VALORE RESTITUITO

La funzione restituisce il puntatore a una variabile strutturata di tipo ‘**struct dirent**’; se la lettura ha già raggiunto la fine della directory, oppure per qualunque altro tipo di errore, la funzione restituisce ‘**NULL**’ e aggiorna eventualmente la variabile *errno*.

## ERRORI

Valore di <i>errno</i>	Significato
EBADF	La directory rappresentata da <i>dp</i> non è valida.

## FILE SORGENTI

‘lib/sys/types.h’ [95.26]

‘lib/dirent.h’ [95.4]

‘lib/dirent/DIR.c’ [95.4.1]

‘lib/dirent/readdir.c’ [95.4.4]

## VEDERE ANCHE

*read(2)* [87.39], *closedir(3)* [88.13], *opendir(3)* [88.89], *rewinddir(3)* [88.101].

88.99 os32: *realloc(3)*

Vedere *malloc(3)* [88.76].

88.100 os32: *rewind(3)*

## NOME

‘**rewind**’ - riposizionamento all’inizio dell’indice di accesso a un flusso di file

## SINTASSI

```
#include <stdio.h>
void rewind (FILE *fp);
```

## DESCRIZIONE

La funzione *rewind()* azzera l'indice della posizione interna del flusso di file specificato con il parametro *fp*; inoltre azzera anche l'indicatore di errore dello stesso flusso. In pratica si ottiene la stessa cosa di:

```
(void) fseek (fp, 0L, SEEK_SET);
clearerr (fp);
```

## FILE SORGENTI

'lib/stdio.h' [95.18]

'lib/stdio/FILE.c' [95.18.1]

'lib/stdio/rewind.c' [95.18.30]

## VEDERE ANCHE

*lseek(2)* [87.33], *fgetpos(3)* [88.33], *fsetpos(3)* [88.33], *ftell(3)* [88.47], *fseek(3)* [88.44], *rewind(3)* [88.100].

88.101 os32: *rewinddir(3)*

«

## NOME

'**rewinddir**' - riposizionamento all'inizio del riferimento per l'accesso a una directory

## SINTASSI

```
#include <sys/types.h>
#include <dirent.h>
void rewinddir (DIR *dp);
```

## DESCRIZIONE

La funzione *rewinddir()* riposiziona i riferimenti per l'accesso alla directory indicata, in modo che la prossima lettura o scrittura avvenga dalla prima posizione.

## VALORE RESTITUITO

La funzione non restituisce alcunché e non si presenta nemmeno la possibilità di segnalare errori attraverso la variabile *errno*.

## FILE SORGENTI

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

'lib/dirent.h' [95.4]

'lib/dirent/DIR.c' [95.4.1]

'lib/dirent/rewinddir.c' [95.4.5]

## VEDERE ANCHE

*rewind(3)* [88.100], *closedir(3)* [88.13], *opendir(3)* [88.89], *rewinddir(3)* [88.98].

88.102 os32: scanf(3)



## NOME

'**scanf**', '**fscanf**', '**sscanf**' - interpretazione dell'input e conversione

## SINTASSI

```
#include <stdio.h>

int scanf (const char *restrict format, ...);

int fscanf (FILE *restrict fp,
            const char *restrict format, ...);

int sscanf (char *restrict string,
            const char *restrict format, ...);
```

## DESCRIZIONE

Le funzioni del gruppo ‘...**scanf()**’ hanno in comune lo scopo di interpretare dei dati, forniti in forma di stringa, convertendoli opportunamente.

I dati in ingresso sono costituiti da una sequenza di caratteri, la quale viene fornita tramite lo standard input per *scanf()*, tramite il flusso di file *fp* per *fscanf()*, oppure tramite la stringa *string* per *sscanf()*. L’interpretazione dei dati in ingresso viene guidata da una stringa di formato, costituita dal parametro *format*, per le tre funzioni. La stringa di formato contiene degli *specificatori di conversione*, con cui si determina il tipo degli argomenti variabili che non sono esplicitati nel prototipo delle funzioni.

Per ogni specificatore di conversione contenuto nella stringa di formato, deve esistere un argomento, successivo al parametro *format*, costituito dal puntatore a una variabile di tipo conforme a quanto indicato dallo specificatore relativo. La conversione per quello specificatore, comporta la memorizzazione del risultato in memoria, in corrispondenza del puntatore relativo. Si osservi l’esempio seguente:



```
int valore;  
...  
scanf ("%i", &valore);
```

In questo modo si attende l'inserimento, attraverso lo standard input, di un numero intero, da convertire e assegnare così alla variabile *valore*; Infatti, lo specificatore di conversione '*%i*', consente di interpretare un numero intero.

Gli specificatori di conversione devono rispettare la sintassi seguente per la libreria di `os32`:

```
% [*] [n_ampiezza] [hh | h | l | j | z | t] tipo
```

Come si può vedere, all'inizio può apparire un asterisco, il cui scopo è quello di annullare l'assegnamento del valore a una variabile. In pratica, con l'asterisco il dato corrispondente allo specificatore viene interpretato, ma poi non viene salvato.

Successivamente può apparire un numero che rappresenta l'ampiezza del dato da interpretare, in byte, il cui scopo è quello di limitare la lettura fino a un certo carattere.

Dopo può apparire una sigla, composta da una o più lettere, il cui scopo è quello di modificare la dimensione predefinita della variabile di destinazione. In altri termini, senza questo modificatore si intende che la variabile ricevente debba essere di una certa grandezza, ma con l'aggiunta del «modificatore di lunghezza» si precisa invece qualcosa di diverso. In pratica, il modificatore di lunghezza usato da queste funzioni è equivalente a quello delle funzioni di composizione dell'output.

Al termine dello specificatore di conversione appare una lette-

ra che dichiara come deve essere interpretato il dato in ingresso e, in mancanza del modificatore di lunghezza, indica anche la dimensione della variabile ricevente.

### Tipi di conversione.

Simbolo	Tipo di argomento	Conversione applicata
<code>%...d</code>	<code>int *</code>	Numero intero con segno rappresentato in base dieci.
<code>%...i</code>	<code>int *</code>	Numero intero con segno rappresentare in base dieci o in base otto, avendo come prefisso uno zero, oppure in base sedici, avendo come prefisso <code>'0x'</code> o <code>'0X'</code> .
<code>%...u</code>	<code>unsigned int *</code>	Numero intero senza segno rappresentato in base dieci.
<code>%...o</code>	<code>unsigned int *</code>	Numero intero senza segno rappresentato in ottale (con o senza lo zero iniziale).

Simbolo	Tipo di argomento	Conversione applicata
<code>%...x</code>	<code>unsigned int *</code>	Numero intero senza segno rappresentato in esadecimale (con o senza il prefisso <code>'0x'</code> o <code>'0X'</code> ).
<code>%...c</code>	<code>char *</code>	Interpreta un solo carattere, o più caratteri se si specifica l'ampiezza. Nella lettura contano anche gli spazi o qualunque altro carattere e non viene aggiunto il carattere nullo di terminazione.
<code>%...s</code>	<code>char *</code>	Interpreta una sequenza di caratteri che non siano spazi, aggiungendo alla fine il carattere nullo di terminazione.
<code>%p</code>	<code>void *</code>	Interpreta il valore di un puntatore che sia rappresentato nello stesso modo in cui farebbe la funzione <code>'printf("%p", puntatore)'</code> .

Simbolo	Tipo di argomento	Conversione applicata
<code>%n</code>	<code>int *</code>	Questo specificatore non esegue alcuna conversione e si limita a memorizzare la quantità di caratteri ( <b>'char'</b> ) letti fino a quel punto.
<code>%... [...]</code>	<code>char *</code>	Interpreta una stringa non vuota contenente solo i caratteri elencati tra parentesi quadre, aggiungendo alla fine il carattere nullo di terminazione. Se tra i caratteri si cerca anche la parentesi quadra chiusa, questa va messa all'inizio dell'elenco: <code>'%... [...]'</code> .
<code>%... [^...]</code>	<code>char *</code>	Interpreta una stringa non vuota contenente solo caratteri diversi da quelli elencati tra parentesi quadre, aggiungendo alla fine il carattere nullo di terminazione. Se tra i caratteri da escludere si vuole indicare anche la parentesi quadra chiusa, questa va messa all'inizio dell'elenco: <code>'%... [^] ...'</code> .
<code>%%</code>		Interpreta un carattere di percentuale tra i dati in ingresso, ma senza memorizzare alcunché.

### Modificatori della lunghezza del dato in uscita.

Simbolo	Tipo	Simbolo	Tipo
%...hhd %...hhi	signed char *	%...hhu %...hho %...hhx %...hhn	unsigned char *
%...hd %...hi	short int *	%...hu %...ho %...hx %...hn	unsigned short int *
%...ld %...li	long int *	%...lu %...lo %...lx %...ln	unsigned long int *

Simbolo	Tipo	Simbolo	Tipo
%...jd %...ji	intmax_t *	%...ju %...jo %...jx %...jn	uintmax_t *
%...zd %...zi	size_t *	%...zu %...zo %...zx %...zn	size_t *
%...td %...ti	ptrdiff_t *	%...tu %...to %...tx %...tn	ptrdiff_t *

La stringa di conversione è composta da *direttive*, ognuna delle quali è formata da: uno o più spazi (spazi veri e propri o caratteri di tabulazione orizzontale); un carattere diverso da ‘%’ e diverso dai caratteri che rappresentano spazi, oppure uno specificatore di conversione.

[*spazi*] *carattere* | %...

Dalla sequenza di caratteri che costituisce i dati in ingresso da

interpretare, vengono eliminati automaticamente gli spazi iniziali e finali (tutto ciò che si può considerare spazio, anche il codice di interruzione di riga), quando all'inizio o alla fine non ci sono corrispondenze con specificatori di conversione che possono interpretarli.

Quando la direttiva di interpretazione inizia con uno o più spazi orizzontali, significa che si vogliono ignorare gli spazi a partire dalla posizione corrente nella lettura dei dati in ingresso; inoltre, la presenza di un carattere che non fa parte di uno specificatore di conversione indica che quello stesso carattere deve essere incontrato nell'interpretazione dei dati in ingresso, altrimenti il procedimento di lettura e valutazione si deve interrompere. Se due specificatori di conversione appaiono adiacenti, i dati in ingresso corrispondenti possono essere separati da spazi orizzontali o da spazi verticali (il codice di interruzione di riga).

## **VALORE RESTITUITO**

Le funzioni restituiscono la quantità di elementi in ingresso interpretati e assegnati correttamente: una quantità inferiore al previsto indica pertanto un errore. Se le funzioni restituiscono il valore **'EOF'**, si tratta di un errore, dovuto eventualmente a un problema di interpretazione del formato o a un problema di accesso al flusso di file da cui deve provenire l'input.

## ERRORI

Valore di <i>errno</i>	Significato
EINVAL	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
EPERM	Operazione di accesso non consentita.
EACCES	Accesso non consentito.
EBADF	Il descrittore del file a cui si riferisce il flusso, non è valido.
ERANGE	Il risultato della conversione di un intero, non può essere memorizzato nel tipo di variabile a cui si riferisce lo specificatore di conversione.

## FILE SORGENTI

‘lib/stdio.h’ [95.18]

‘lib/stdio/fscanf.c’ [95.18.17]

‘lib/stdio/scanf.c’ [95.18.31]

‘lib/stdio/sscanf.c’ [95.18.36]

## VEDERE ANCHE

*vfscanf(3)* [88.138], *vscanf(3)* [88.138], *vsscanf(3)* [88.138], *printf(3)* [88.91].

88.103 os32: *setbuf(3)*

«

## NOME

‘**setbuf**’, ‘**setvbuf**’ - modifica della memoria tampone per i flussi di file



## SINTASSI

```
#include <stdio.h>
void setbuf (FILE *restrict fp, char *restrict buffer);
int setvbuf (FILE *restrict fp, char *restrict buffer,
            int buf_mode, size_t size);
```

## DESCRIZIONE

Le funzioni *setbuf()* e *setvbuf()* della libreria di os32, non fanno alcunché, perché os32 non gestisce una memoria tampone per i flussi di file.

## VALORE RESTITUITO

La funzione *setvbuf()* restituisce, in tutti i casi, il valore zero.

## FILE SORGENTI

‘lib/stdio.h’ [95.18]

‘lib/stdio/setbuf.c’ [95.18.32]

‘lib/stdio/setvbuf.c’ [95.18.33]

## VEDERE ANCHE

*fflush(3)* [88.31].

88.104 os32: setenv(3)



## NOME

‘**setenv**’, ‘**unsetenv**’ - assegnamento o cancellazione di una variabile di ambiente

## SINTASSI

```
#include <stdlib.h>

int setenv (const char *name, const char *value,
           int overwrite);

int unsetenv (const char *name);
```

## DESCRIZIONE

La funzione *setenv()* crea o assegna un valore a una variabile di ambiente. Se questa variabile esiste già, la modifica del valore assegnatole può avvenire soltanto se l'argomento corrispondente al parametro *overwrite* risulta essere diverso da zero; in caso contrario, la modifica non ha luogo.

La funzione *unsetenv()* si limita a cancellare la variabile di ambiente specificata come argomento.

## VALORE RESTITUITO

Valore	Significato
0	Operazione riuscita.
-1	Operazione fallita. Va verificato l'errore indicato dalla variabile <i>errno</i> .

## ERRORI

Valore di <i>errno</i>	Significato
EINVAL	Argomento non valido.
ENOMEM	Memoria insufficiente.

## FILE SORGENTI

'lib/stdlib.h' [95.19]

`'applic/crt0.mer.s'` [96.1.12]

`'applic/crt0.sep.s'` [96.1.13]

`'lib/stdlib/environment.c'` [95.19.8]

`'lib/stdlib/setenv.c'` [95.19.18]

`'lib/stdlib/unsetenv.c'` [95.19.21]

## VEDERE ANCHE

*environ*(7) [91.1], *getenv*(3) [88.52], *putenv*(3) [88.94].

88.105 os32: *setgrent*(3)

Vedere *getgrent*(3) [88.53].

88.106 os32: *setpwent*(3)

Vedere *getpwent*(3) [88.57].

88.107 os32: *setvbuf*(3)

Vedere *setbuf*(3) [88.103].

88.108 os32: *snprintf*(3)

Vedere *printf*(3) [88.91].

88.109 os32: *sprintf*(3)

Vedere *printf*(3) [88.91].

88.110 os32: srand(3)

«  
Vedere *rand(3)* [88.97].

88.111 os32: sscanf(3)

«  
Vedere *scanf(3)* [88.102].

88.112 os32: stdio(3)

«

## NOME

‘**stdio**’ - libreria per la gestione dei file in forma di flussi di file (*stream*)

## SINTASSI

```
#include <stdio.h>
```

## DESCRIZIONE

Le funzioni di libreria che fanno capo al file di intestazione ‘`stdio.h`’, consentono di gestire i file in forma di «flussi», rappresentati da puntatori al tipo ‘**FILE**’. Questa gestione si sovrappone a quella dei file in forma di «descrittori», la quale avviene tramite chiamate di sistema. Lo scopo della sovrapposizione dovrebbe essere quello di gestire i file con l’ausilio di una memoria tampone, cosa che però la libreria di os32 non fornisce. Nella libreria di os32, il tipo ‘**FILE \***’ è un puntatore a una variabile strutturata che contiene solo tre informazioni: il numero del descrittore del file a cui il flusso si associa; lo stato di errore; lo stato di raggiungimento della fine del file.

```
typedef struct {
    int         fdn;        // File descriptor number.
    char        error;     // Error indicator.
    char        eof;       // End of file indicator.
} FILE;
```

Le variabili strutturate necessarie per questa gestione, sono raccolte in un array, dichiarato nel file `lib/stdio/FILE.c`, con il nome `_stream[]`, dove per il descrittore di file *n*, si associano sempre i dati di `_stream[n]`.

```
FILE _stream[FOPEN_MAX];
```

Così come sono previsti tre descrittori (zero, uno e due) per la gestione di standard input, standard output e standard error, tutti i processi inizializzano l'array `_stream[]` con l'abbinamento a tali descrittori, per i primi tre flussi.

```
void
_stdio_stream_setup (void)
{
    _stream[0].fdn    = 0;
    _stream[0].error = 0;
    _stream[0].eof    = 0;

    _stream[1].fdn    = 1;
    _stream[1].error = 0;
    _stream[1].eof    = 0;

    _stream[2].fdn    = 2;
    _stream[2].error = 0;
    _stream[2].eof    = 0;
}
```

Ciò avviene attraverso il codice contenuto nel file `crt0.s`, dove si chiama la funzione che provvede a tale inizializzazione, conte-

nuta nel file `'lib/stdio/FILE.c'`. Per fare riferimento ai flussi predefiniti, si usano i nomi `'stdin'`, `'stdout'` e `'stderr'`, i quali sono dichiarati nel file `'stdio.h'`, come puntatori ai primi tre elementi dell'array `_stream[]`:

```
#define stdin    (&_stream[0])
#define stdout   (&_stream[1])
#define stderr   (&_stream[2])
```

## FILE SORGENTI

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

`'lib/stdio.h'` [95.18]

`'lib/stdio/FILE.c'` [95.18.1]

`'applic/crt0.mer.s'` [96.1.12]

`'applic/crt0.sep.s'` [96.1.13]

## VEDERE ANCHE

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

## 88.113 os32: strcat(3)

«

### NOME

`'strcat'`, `'strncat'` - concatenamento di una stringa a un'altra già esistente

## SINTASSI

```
#include <string.h>
char *strcat (char *restrict dst,
             const char *restrict org);
char *strncat (char *restrict dst,
              const char *restrict org,
              size_t n);
```

## DESCRIZIONE

Le funzioni *strcat()* e *strncat()* copiano la stringa di origine *org*, aggiungendola alla stringa di destinazione *dst*, nel senso che la scrittura avviene a partire dal codice di terminazione ‘\0’ che viene così sovrascritto. Al termine della copia, viene aggiunto nuovamente il codice di terminazione di stringa ‘\0’, nella nuova posizione conclusiva.

Nel caso particolare di *strncat()*, la copia si arresta al massimo dopo il trasferimento di *n* caratteri. Pertanto, la stringa di origine per *strncat()* potrebbe anche non essere terminata correttamente, se raggiunge o supera la dimensione di *n* caratteri. In ogni caso, nella destinazione viene aggiunto il codice nullo di terminazione di stringa, dopo la copia del carattere *n*-esimo.

## VALORE RESTITUITO

Le due funzioni restituiscono *dst*.

## FILE SORGENTI

‘lib/string.h’ [95.20]

‘lib/string/strcat.c’ [95.20.7]

‘lib/string/strncat.c’ [95.20.16]

## VEDERE ANCHE

*memccpy(3)* [88.77], *memcpy(3)* [88.80], *strcpy(3)* [88.117], *strncpy(3)* [88.117].

## 88.114 os32: strchr(3)

&lt;&lt;

## NOME

‘**strchr**’, ‘**strrchr**’ - ricerca di un carattere all’interno di una stringa

## SINTASSI

```
#include <string.h>
char *strchr (const char *string, int c);
char *strrchr (const char *string, int c);
```

## DESCRIZIONE

Le funzioni *strchr()* e *strrchr()* scandiscono la stringa *string* alla ricerca di un carattere uguale al valore di *c*. La funzione *strchr()* scandisce a partire da «sinistra», ovvero ricerca la prima corrispondenza con il carattere *c*, mentre la funzione *strrchr()* cerca l’ultima corrispondenza con il carattere *c*, pertanto è come se scandisse da «destra».

## VALORE RESTITUITO

Se le due funzioni trovano il carattere che cercano, ne restituiscono il puntatore, altrimenti restituiscono ‘**NULL**’.

## FILE SORGENTI

‘lib/string.h’ [95.20]



'lib/string/strchr.c' [95.20.8]

'lib/string/strrchr.c' [95.20.20]

## VEDERE ANCHE

*memchr(3)* [88.78], *strlen(3)* [88.121], *strpbrk(3)* [88.125], *strspn(3)* [88.127].

## 88.115 os32: strcmp(3)



### NOME

'**strcmp**', '**strncmp**' - confronto di due stringhe

### SINTASSI

```
#include <string.h>
int strcmp (const char *string1, const char *string2);
int strncmp (const char *string1, const char *string2,
             size_t n);
int strcoll (const char *string1, const char *string2);
```

### DESCRIZIONE

Le funzioni *strcmp()* e *strncmp()* confrontano due stringhe, nel secondo caso, il confronto avviene al massimo fino al *n*-esimo carattere.

La funzione *strcoll()* dovrebbe eseguire il confronto delle due stringhe tenendo in considerazione la configurazione locale. Tuttavia, os32 non è in grado di gestire le configurazioni locali, pertanto questa funzione coincide esattamente con *strcmp()*.

## VALORE RESTITUITO

Valore	Esito del confronto.
-1	<i>string1</i> < <i>string2</i>
0	<i>string1</i> == <i>string2</i>
+1	<i>string1</i> > <i>string2</i>

## FILE SORGENTI

'lib/string.h' [95.20]

'lib/string/strcmp.c' [95.20.9]

'lib/string/strncmp.c' [95.20.17]

'lib/string/strcoll.c' [95.20.10]

## VEDERE ANCHE

*memcmp(3)* [88.79].

88.116 os32: *strcoll(3)*

«

Vedere *strcmp(3)* [88.115].

88.117 os32: *strcpy(3)*

«

## NOME

'*strcpy*', '*strncpy*' - copia di una stringa

## SINTASSI

```
#include <string.h>
char *strcpy (char *restrict dst,
              const char *restrict org);
char *strncpy (char *restrict dst,
              const char *restrict org,
              size_t n);
```

## DESCRIZIONE

Le funzioni *strcpy()* e *strncpy()*, copiano la stringa *org*, completa di codice nullo di terminazione, nella destinazione *dst*. Eventualmente, nel caso di *strncpy()*, la copia non supera i primi *n* caratteri, con l'aggravante che in tal caso, se nei primi *n* caratteri non c'è il codice nullo di terminazione delle stringhe, nella destinazione *dst* si ottiene una stringa non terminata.

## VALORE RESTITUITO

Le funzioni restituiscono *dst*.

## FILE SORGENTI

'lib/string.h' [95.20]

'lib/string/strcpy.c' [95.20.11]

'lib/string/strncpy.c' [95.20.18]

## VEDERE ANCHE

*memccpy(3)* [88.77], *memcpy(3)* [88.80], *memmove(3)* [88.81].

88.118 os32: *strcspn(3)*

Vedere *strspn(3)* [88.127].

88.119 os32: *strdup(3)*

## NOME

'*strdup*' - duplicazione di una stringa

## SINTASSI

```
#include <string.h>
char *strdup (const char *string);
```

## DESCRIZIONE

La funzione *strdup()*, alloca dinamicamente una quantità di memoria, necessaria a copiare la stringa *string*, quindi esegue tale copia e restituisce il puntatore alla nuova stringa allocata. Tale puntatore può essere usato successivamente per liberare la memoria, con l'ausilio della funzione *free()*.

## VALORE RESTITUITO

La funzione restituisce il puntatore alla nuova stringa ottenuta dalla copia, oppure **NULL** nel caso non fosse possibile allocare la memoria necessaria.

## ERRORI

Valore di <i>errno</i>	Significato
ENOMEM	Memoria insufficiente.

## FILE SORGENTI

'lib/string.h' [95.20]

'lib/string/strdup.c' [95.20.13]

## VEDERE ANCHE

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

88.120 os32: *strerror(3)*

«

## NOME

'**strerror**' - descrizione di un errore in forma di stringa

## SINTASSI

```
#include <string.h>
char *strerror (int errnum);
```

## DESCRIZIONE

La funzione *strerror()* interpreta il valore *errnum* come un errore, di quelli che può rappresentare la variabile *errno* del file ‘*errno.h*’.

## VALORE RESTITUITO

La funzione restituisce il puntatore a una stringa contenente la descrizione dell’errore, oppure soltanto ‘**Unknown error**’, se l’argomento ricevuto non è traducibile.

## FILE SORGENTI

‘*lib/errno.h*’ [[95.5](#)]

‘*lib/string.h*’ [[95.20](#)]

‘*lib/string/strerror.c*’ [[95.20.14](#)]

## VEDERE ANCHE

*errno(3)* [[88.20](#)], *perror(3)* [[88.90](#)].

88.121 os32: *strlen(3)*

## NOME

‘**strlen**’ - lunghezza di una stringa

## SINTASSI

```
#include <string.h>
size_t strlen (const char *string);
```

## DESCRIZIONE

La funzione *strlen()* calcola la lunghezza della stringa, ovvero la quantità di caratteri che la compone, escludendo il codice nullo di conclusione.

## VALORE RESTITUITO

La funzione restituisce la quantità di caratteri che compone la stringa, escludendo il codice ‘\0’ finale.

## FILE SORGENTI

‘lib/string.h’ [95.20]

‘lib/string/strlen.c’ [95.20.15]

88.122 os32: strncat(3)

« Vedere *strcat(3)* [88.113].

88.123 os32: strncmp(3)

« Vedere *strcmp(3)* [88.115].

88.124 os32: strncpy(3)

« Vedere *strcpy(3)* [88.117].

88.125 os32: strpbrk(3)

«

## NOME

‘**strpbrk**’ - scansione di una stringa alla ricerca di un carattere

## SINTASSI

```
#include <string.h>
char *strpbrk (const char *string, const char *accept);
```

## DESCRIZIONE

La funzione *strpbrk()* cerca il primo carattere, nella stringa *string*, che corrisponda a uno di quelli contenuti nella stringa *accept*.

## VALORE RESTITUITO

Restituisce il puntatore al primo carattere che, nella stringa *string* corrisponde a uno di quelli contenuti nella stringa *accept*. In mancanza di alcuna corrispondenza, restituisce 'NULL'.

## FILE SORGENTI

'lib/string.h' [95.20]

'lib/string/strpbrk.c' [95.20.19]

## VEDERE ANCHE

*memchr(3)* [88.78], *strchr(3)* [88.114], *strstr(3)* [88.128], *strtok(3)* [88.129].

88.126 os32: *strrchr(3)*

Vedere *strchr(3)* [88.114].



## 88.127 os32: strspn(3)

**NOME**

‘**strspn**’, ‘**strcspn**’ - scansione di una stringa, limitatamente a un certo insieme di caratteri

**SINTASSI**

```
#include <string.h>
size_t strspn (const char *string, const char *accept);
size_t strcspn (const char *string, const char *reject);
```

**DESCRIZIONE**

La funzione *strspn()* scandisce la stringa *string*, calcolando la lunghezza di questa che contiene, a partire dall’inizio, soltanto caratteri che si trovano nella stringa *accept*.

La funzione *strcspn()* scandisce la stringa *string*, calcolando la lunghezza di questa che contiene, a partire dall’inizio, soltanto caratteri che non si trovano nella stringa *reject*.

**VALORE RESTITUITO**

La funzione *strspn()* restituisce la lunghezza della stringa che contiene soltanto caratteri contenuti in *accept*.

La funzione *strcspn()* restituisce la lunghezza della stringa che contiene soltanto caratteri che non sono contenuti in *reject*.

**FILE SORGENTI**

‘lib/string.h’ [95.20]

‘lib/string/strspn.c’ [95.20.21]

‘lib/string/strcspn.c’ [95.20.12]



## VEDERE ANCHE

*memchr(3)* [88.78], *strchr(3)* [88.114], *strpbrk(3)* [88.125], *strstr(3)* [88.128], *strtok(3)* [88.129].

## 88.128 os32: strstr(3)



## NOME

‘**strstr**’ - ricerca di una sottostringa

## SINTASSI

```
#include <string.h>
char *strstr (const char *string, const char *substring);
```

## DESCRIZIONE

La funzione *strstr()* scandisce la stringa *string*, alla ricerca della prima corrispondenza con la stringa *substring*, restituendo eventualmente il puntatore all’inizio di tale corrispondenza.

## VALORE RESTITUITO

Se la ricerca termina con successo, viene restituito il puntatore all’inizio della sottostringa contenuta in *string*; diversamente viene restituito il puntatore nullo ‘**NULL**’.

## FILE SORGENTI

‘lib/string.h’ [95.20]

‘lib/string/strstr.c’ [95.20.22]

## VEDERE ANCHE

*memchr(3)* [88.78], *strchr(3)* [88.114], *strpbrk(3)* [88.125], *strtok(3)* [88.129].

## 88.129 os32: strtok(3)

**NOME**

‘**strtok**’ - *string token*, ovvero estrazione di pezzi da una stringa

**SINTASSI**

```
#include <string.h>
char *strtok (char *restrict string,
              const char *restrict delim);
```

**DESCRIZIONE**

La funzione *strtok()* serve a suddividere una stringa in unità, definite *token*, specificando un elenco di caratteri da intendere come delimitatori, in una seconda stringa. La funzione va usata in fasi successive, fornendo solo inizialmente la stringa da suddividere che continua poi a essere utilizzata se al suo posto viene fornito il puntatore nullo. La funzione restituisce, di volta in volta, il puntatore alla sottostringa contenente l'unità individuata, oppure il puntatore nullo, se non può trovarla.

La funzione deve tenere memoria di un puntatore in modo persistente e deve isolare le unità modificando la stringa originale, inserendo il carattere nullo di terminazione alla fine delle unità individuate.

Quando la funzione viene chiamata indicando al posto della stringa da scandire il puntatore nullo, l'insieme dei delimitatori può essere diverso da quello usato nelle fasi precedenti.

Per comprendere lo scopo della funzione viene utilizzato lo stesso esempio che appare nel documento *ISO/IEC 9899:TC2*, al para-

grafo 7.21.5.7, con qualche piccola modifica per poterlo rendere un programma autonomo:

```
#include <stdio.h>
#include <string.h>
int
main (void)
{
    char str[] = "?a???b,,,#c";
    char *t;

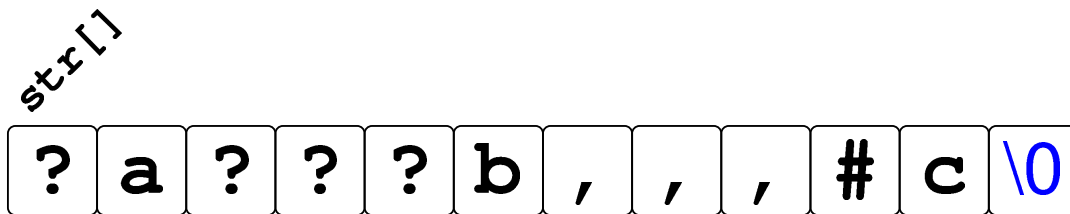
    t = strtok (str, "?");           // t punta all'unità "a"
    printf ("strtok: \"%s\"\n", t);
    t = strtok (NULL, ",");         // t punta all'unità "??b"
    printf ("strtok: \"%s\"\n", t);
    t = strtok (NULL, "#,");        // t punta all'unità "c"
    printf ("strtok: \"%s\"\n", t);
    t = strtok (NULL, "?");         // t è un puntatore nullo
    printf ("strtok: \"%s\"\n", t);

    return 0;
}
```

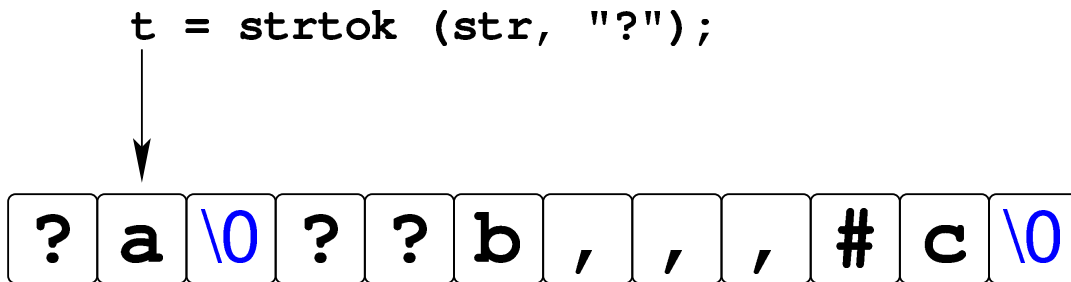
Avviando il programma si ottiene quanto già descritto dai commenti inseriti nel codice:

```
strtok: "a"
strtok: "??b"
strtok: "c"
strtok: "(null)"
```

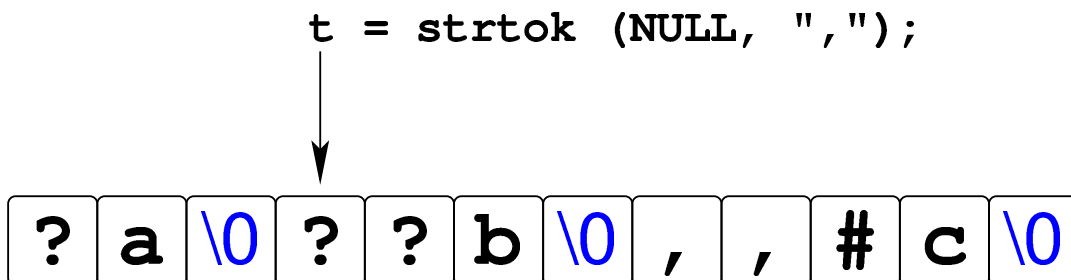
Ciò che avviene nell'esempio può essere schematizzato come segue. Inizialmente la stringa 'str' ha in memoria l'aspetto seguente:



Dopo la prima chiamata della funzione *strtok()* la stringa risulta alterata e il puntatore ottenuto raggiunge la lettera 'a':

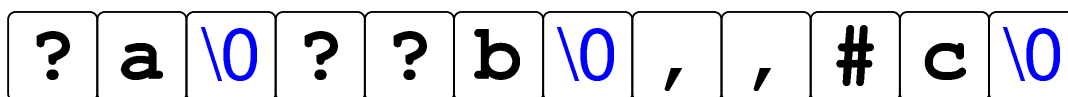


Dopo la seconda chiamata della funzione, in cui si usa il puntatore nullo per richiedere una scansione ulteriore della stringa originale, si ottiene un nuovo puntatore che, questa volta, inizia a partire dal quarto carattere, rispetto alla stringa originale, dal momento che il terzo è già stato sovrascritto da un carattere nullo:



La penultima chiamata della funzione *strtok()* raggiunge la lettera 'c' che è anche alla fine della stringa originale:

```
t = strtok (NULL, "#,");
```



L'ultimo tentativo di chiamata della funzione non può dare alcun esito, perché la stringa originale si è già conclusa.

## VALORE RESTITUITO

La funzione restituisce il puntatore al prossimo «pezzo», oppure `'NULL'` se non ce ne sono più.

## FILE SORGENTI

`'lib/string.h'` [95.20]

`'lib/string/strtok.c'` [95.20.23]

## VEDERE ANCHE

*memchr(3)* [88.78], *strchr(3)* [88.114], *strpbrk(3)* [88.125], *strspn(3)* [88.127].

88.130 os32: *strtol(3)*

«

## NOME

`'strtol'`, `'strtoul'` - conversione di una stringa in un numero

## SINTASSI

```
#include <stdlib.h>

long int strtol (const char *restrict string,
                char **restrict endptr,
                int base);

unsigned long int strtoul (const char *restrict string,
                          char **restrict endptr,
                          int base);
```

## DESCRIZIONE

Le funzioni *strtol()* e *strtoul()*, convertono la stringa *string* in un numero, intendendo la sequenza di caratteri nella base di numerazione indicata come ultimo argomento (*base*). Tuttavia, la base di numerazione potrebbe essere omessa (valore zero) e in tal caso la stringa deve essere interpretata ugualmente in qualche modo: se (dopo un segno eventuale) inizia con zero seguito da un'altra cifra numerica, deve trattarsi di una sequenza ottale; se inizia con zero, quindi appare una lettera «x» deve trattarsi di un numero esadecimale; se inizia con una cifra numerica diversa da zero, deve trattarsi di un numero in base dieci.

La traduzione della stringa ha luogo progressivamente, arrestandosi quando si incontra un carattere incompatibile con la base di numerazione selezionata o stabilita automaticamente. Il valore convertito viene restituito; inoltre, se il puntatore *endptr* è valido (diverso da 'NULL'), si assegna a *\*endptr* la posizione raggiunta nella stringa, corrispondente al primo carattere che non può essere convertito. Pertanto, nello stesso modo, se la stringa non può

essere convertita affatto e si può assegnare qualcosa a *\*endptr*, alla fine, *\*endptr* corrisponde esattamente a *string*.

## VALORE RESTITUITO

Le funzioni restituiscono il valore tratto dall'interpretazione della stringa, ammesso che sia rappresentabile, altrimenti si ottiene 'LONG\_MIN' o 'LONG\_MAX', a seconda dei casi, sapendo che occorre consultare la variabile *errno* per maggiori dettagli.

## ERRORI

Valore di <i>errno</i>	Significato
EINVAL	Argomento non valido.
ERANGE	Il valore risultante è al di fuori dell'intervallo ammissibile per la rappresentazione.

## DIFETTI

La realizzazione di *strtoul()* è incompleta, in quanto si limita a utilizzare *strtol()*, convertendo il risultato in un valore senza segno.

## FILE SORGENTI

'lib/stdlib.h' [95.19]

'lib/stdlib/strtol.c' [95.19.19]

'lib/stdlib/strtoul.c' [95.19.20]

88.131 os32: strtoul(3)

Vedere *strtol(3)* [88.130].

## 88.132 os32: strxfrm(3)

&lt;&lt;

**NOME**

‘**strxfrm**’ - *string transform*, ovvero trasformazione di una stringa

**SINTASSI**

```
#include <string.h>
size_t strxfrm (char *restrict dst,
                const char *restrict org,
                size_t n);
```

**DESCRIZIONE**

Lo scopo della funzione *strxfrm()* sarebbe quello di copiare la stringa *org*, sovrascrivendo *dst*, fino a un massimo di *n* caratteri nella destinazione, ma applicando una trasformazione relativa alla configurazione locale.

os32 non gestisce la configurazione locale, pertanto questa funzione si comporta in modo simile a *strncpy()*, con una differenza in ciò che viene restituito.

**VALORE RESTITUITO**

La funzione restituisce la quantità di byte utilizzati per contenere la trasformazione in *dst*, senza però contare il carattere nullo di terminazione.

**FILE SORGENTI**

‘lib/string.h’ [95.20]

‘lib/string/strxfrm.c’ [95.20.24]



## VEDERE ANCHE

*memcmp(3)* [88.79], *strcmp(3)* [88.115], *strcoll(3)* [88.115].

## 88.133 os32: ttyname(3)



### NOME

‘**ttyname**’ - determinazione del percorso del file di dispositivo di un terminale aperto

### SINTASSI

```
#include <unistd.h>
char *ttyname (int fdn);
```

### DESCRIZIONE

La funzione *ttyname()* richiede come unico argomento il numero che identifica il descrittore di un file. Ammesso che tale descrittore si riferisca a un terminale, la funzione restituisce il puntatore a una stringa che rappresenta il percorso del file di dispositivo corrispondente.

La stringa in questione viene modificata se si usa la funzione in altre occasioni.

### VALORE RESTITUITO

La funzione restituisce il puntatore a una stringa che descrive il percorso del file di dispositivo, presunto, del terminale aperto con il numero *fdn*. Se non si tratta di un terminale, si ottiene un errore. In ogni caso, se la funzione non può restituire un'informazione corretta, produce semplicemente il puntatore nullo e aggiorna la variabile *errno*.

## ERRORI

Valore di <i>errno</i>	Significato
EBADF	Il descrittore di file indicato non è valido.
ENOTTY	Il descrittore di file indicato non riguarda un terminale.

## FILE SORGENTI

‘lib/unistd.h’ [95.30]

‘lib/unistd/ttyname.c’ [95.30.41]

## VEDERE ANCHE

*stat(2)* [87.55], *isatty(3)* [88.69].

88.134 os32: *unsetenv(3)*

«

Vedere *setenv(3)* [88.104].

88.135 os32: *vfprintf(3)*

«

Vedere *vprintf(3)* [88.137].

88.136 os32: *vfscanf(3)*

«

Vedere *vfscanf(3)* [88.138].

88.137 os32: *vprintf(3)*

«

## NOME

‘**vprintf**’, ‘**vfprintf**’, ‘**vsprintf**’, ‘**vsnprintf**’ - composizione dei dati per la visualizzazione

## SINTASSI

```
#include <stdarg.h>
#include <stdio.h>

int vprintf (char *restrict format, va_list arg);
int vfprintf (FILE *fp, char *restrict format,
             va_list arg);
int vsnprintf (char *restrict string, size_t size,
              const char *restrict format, va_list ap);
int vsprintf (char *string, char *restrict format,
             va_list arg);
```

## DESCRIZIONE

Le funzioni del gruppo ‘**v...printf()**’ hanno in comune lo scopo di comporre dei dati in forma di stringa, generalmente per la visualizzazione, o comunque per la fruizione a livello umano.

I dati in ingresso possono essere vari e vengono comunicati attraverso un puntatore di tipo ‘**va\_list**’. Per quantificare e qualificare questi dati in ingresso, la stringa a cui punta il parametro *format*, deve contenere degli *specificatori di conversione*, oltre eventualmente ad altri caratteri. Pertanto, queste funzioni, prendono la stringa a cui punta *format*, la interpretano e determinano come scandire gli argomenti a cui fa riferimento il puntatore *arg*, quindi producono un’altra stringa, composta dalla stringa precedente, sostituendo gli specificatori di conversione con i dati a cui questi si riferiscono, secondo una forma di conversione definita dagli specificatori stessi.

In generale, le funzioni ‘**v...printf()**’ servono per realizzare le altre funzioni ‘**...printf()**’, le quali invece ricevono gli argo-

menti variabili direttamente. Per esempio, la funzione *printf()* può essere realizzata utilizzando in pratica *vprintf()*:

```
#include <stdio.h>
#include <stdarg.h>
int
printf (char *restrict format, ...)
{
    va_list ap;
    va_start (ap, format);
    return (vprintf (format, ap));
}
```

Si veda *printf(3)* [88.91], per la descrizione di come va predisposta la stringa *format*. Nella realizzazione di os32, di tutte queste funzioni, quella che compie effettivamente il lavoro di interpretazione della stringa di formato e che in qualche modo viene chiamata da tutte le altre, è soltanto *vsnprintf()*.

## VALORE RESTITUITO

Le funzioni restituiscono la quantità di caratteri utilizzati nella composizione della nuova stringa, escluso il carattere nullo di terminazione.

## FILE SORGENTI

‘lib/stdarg.h’ [95.1.10]

‘lib/stdio.h’ [95.18]

‘lib/stdio/FILE.c’ [95.18.1]

‘lib/stdio/vfprintf.c’ [95.18.37]

‘lib/stdio/vprintf.c’ [95.18.40]

‘lib/stdio/vsprintf.c’ [95.18.43]

‘lib/stdio/vsnprintf.c’ [95.18.42]

## VEDERE ANCHE

*fprintf(3)* [88.91], *printf(3)* [88.91], *sprintf(3)* [88.91],  
*snprintf(3)* [88.91], *scanf(3)* [88.102].

## 88.138 os32: vscanf(3)



## NOME

‘**vscanf**’, ‘**vfscanf**’, ‘**vsscanf**’ - interpretazione dell’input e conversione

## SINTASSI

```
#include <stdarg.h>
#include <stdio.h>

int vscanf (const char *restrict format, va_list ap);
int vfscanf (FILE *restrict fp, const char *restrict format,
             va_list ap);
int vsscanf (const char *string, const char *restrict format,
             va_list ap);
```

## DESCRIZIONE

Le funzioni del gruppo ‘**v...scanf()**’ hanno in comune lo scopo di interpretare dei dati, forniti in forma di stringa, convertendoli opportunamente.

I dati in ingresso sono costituiti da una sequenza di caratteri, la quale viene fornita tramite lo standard input per *vscanf()*, tramite il flusso di file *fp* per *vfscanf()*, oppure tramite la stringa *string* per *vsscanf()*. L’interpretazione dei dati in ingresso viene guidata da una stringa di formato, costituita dal parametro *format*, per le

tre funzioni. La stringa di formato contiene degli *specificatori di conversione*, con cui si determina il tipo degli argomenti variabili a cui punta inizialmente *ap*.

Queste funzioni servono per realizzare in pratica quelle corrispondenti che hanno nomi privi della lettera «v» iniziale. Per esempio, per ottenere *scanf()* si può utilizzare *vscanf()*:

```
#include <stdio.h>
#include <stdarg.h>
int
scanf (const char *restrict format, ...)
{
    va_list ap;
    va_start (ap, format);
    return vscanf (format, ap);
}
```

Il modo in cui va predisposta la stringa di formato (*format*) è descritto in *scanf(3)* [88.102]. La funzione più importante di questo gruppo, in quanto svolge effettivamente il lavoro di interpretazione e viene chiamata, più o meno indirettamente, da tutte le altre, è *vfscanf()*, la quale però non è standard.

## VALORE RESTITUITO

Le funzioni restituiscono la quantità di elementi in ingresso interpretati e assegnati correttamente: una quantità inferiore al previsto indica pertanto un errore. Se le funzioni restituiscono il valore ‘**EOF**’, si tratta di un errore, dovuto eventualmente a un problema di interpretazione del formato o a un problema di accesso al flusso di file da cui deve provenire l’input.

## ERRORI

Valore di <i>errno</i>	Significato
EINVAL	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
EPERM	Operazione di accesso non consentita.
EACCES	Accesso non consentito.
EBADF	Il descrittore del file a cui si riferisce il flusso, non è valido.
ERANGE	Il risultato della conversione di un intero, non può essere memorizzato nel tipo di variabile a cui si riferisce lo specificatore di conversione.

## FILE SORGENTI

'lib/stdio.h' [95.18]

'lib/stdio/vfscanf.c' [95.18.38]

'lib/stdio/vscanf.c' [95.18.41]

'lib/stdio/vsscanf.c' [95.18.44]

'lib/stdio/vfsscanf.c' [95.18.39]

## VEDERE ANCHE

*fscanf(3)* [88.102], *scanf(3)* [88.102], *sscanf(3)* [88.102],  
*printf(3)* [88.91].

88.139 os32: vsnprintf(3)

Vedere *vprintf(3)* [88.137].



88.140 os32: vsprintf(3)

«

Vedere *vprintf(3)* [[88.137](#)].

88.141 os32: vsscanf(3)

«

Vedere *vsscanf(3)* [[88.138](#)].