Sezione 3: funzioni di libreria

os16: access(3) 3298

os16: abort(3) 3300

os16: abs(3) 3301

os16: atexit(3) 3302

os16: atoi(3) 3303

os16: atol(3) 3304

os16: basename(3) 3305

os16: bp(3) 3307

os16: clearerr(3) 3307

os16: closedir(3) 3308

os16: creat(3) 3309

os16: cs(3) 3310

os16: ctime(3) 3311

os16: dirname(3) 3314

os16: div(3) 3314

os16: ds(3) 3315

os16: endpwent(3) 3315

os16: errno(3) 3316

os16: es(3) 3327

os16: exec(3) 3327

«a2» 2013.11.11 -- Copyright © Daniele Giacomini -- appunti2@gmail.com http://informaticalibera.net

os16: execl(3) 3329
os16: execle(3) 3330
os16: execlp(3) 3330
os16: execv(3) 3330
os16: execvp(3) 3330
os16: exit(3) 3330
os16: fclose(3) 3330
os16: feof(3) 3331
os16: ferror(3) 3332
os16: fflush(3) 3333
os16: fgetc(3) 3334
os16: fgetpos(3) 3336
os16: fgets(3) 3337
os16: fileno(3) 3339
os16: fopen(3) 3340
os16: fprintf(3) 3343
os16: fputc(3) 3344
os16: fputs(3) 3345
os16: fread(3) 3347
os16: free(3) 3348
os16: freopen(3) 3348
os16: fscanf(3) 3348

os16: fseek(3) 3348
os16: fseeko(3) 3350
os16: fsetpos(3) 3350
os16: ftell(3) 3350
os16: ftello(3) 3351
os16: fwrite(3) 3352
os16: getc(3) 3353
os16: getchar(3) 3353
os16: getenv(3) 3353
os16: getopt(3) 3354
os16: getpwent(3) 3360
os16: getpwnam(3) 3363
os16: getpwuid(3) 3366
os16: gets(3) 3366
os16: heap(3) 3366
os16: heap_clear(3) 3367
os16: heap_min(3) 3368
os16: input_line(3) 3368
os16: isatty(3) 3370
os16: labs(3) 3371
os16: ldiv(3) 3371
os16: major(3) 3371

os16: makedev(3) 3371
os16: malloc(3) 3372
os16: memccpy(3) 3374
os16: memchr(3) 3375
os16: memcmp(3) 3376
os16: memcpy(3) 3377
os16: memmove(3) 3378
os16: memset(3) 3378
os16: minor(3) 3379
os16: namep(3) 3379
os16: offsetof(3) 3381
os16: opendir(3) 3382
os16: perror(3) 3384
os16: printf(3) 3385
os16: process_info(3) 3391
os16: putc(3) 3392
os16: putchar(3) 3392
os16: putenv(3) 3392
os16: puts(3) 3394
os16: qsort(3) 3394
os16: rand(3) 3395
os16: readdir(3) 3396

os16: realloc(3) 3398
os16: rewind(3) 3398
os16: rewinddir(3) 3399
os16: scanf(3) 3400
os16: seg_d(3) 3409
os16: seg_i(3) 3411
os16: setbuf(3) 3411
os16: setenv(3) 3411
os16: setpwent(3) 3413
os16: setvbuf(3) 3413
os16: snprintf(3) 3413
os16: sp(3) 3414
os16: sprintf(3) 3414
os16: srand(3) 3414
os16: ss(3) 3414
os16: sscanf(3) 3414
os16: stdio(3) 3414
os16: strcat(3) 3417
os16: strchr(3) 3418
os16: strcmp(3) 3419
os16: strcoll(3) 3420
os16: strcpy(3) 3421

os16: strcspn(3) 3422
os16: strdup(3) 3422
os16: strerror(3) 3423
os16: strlen(3) 3424
os16: strncat(3) 3425
os16: strncmp(3) 3425
os16: strncpy(3) 3425
os16: strpbrk(3) 3425
os16: strrchr(3) 3426
os16: strspn(3) 3426
os16: strstr(3) 3427
os16: strtok(3) 3428
os16: strtol(3) 3432
os16: strtoul(3) 3434
os16: strxfrm(3) 3434
os16: ttyname(3) 3435
os16: unsetenv(3) 3437
os16: vfprintf(3) 3437
os16: vfscanf(3) 3437
os16: vprintf(3) 3437
os16: vscanf(3) 3439
os16: vsnprintf(3) 3442

os16: vsprintf(3) 3443

os16: vsscanf(3) 3443

abort() 3300 abs() 3301 access() 3298 asctime() 3311 atexit() 3302 atoi() 3303 atol() 3303 basename() 3305 bp() 3310 clearerr() 3307 closedir() 3308 creat() 3309 cs() 3310 ctime() 3311 dirname() 3305 div() 3314 ds() 3310 endpwent() 3360 errfn 3316 errln 3316 errno 3316 errset() 3316 es() 3310 execl() 3327 execle() 3327 execlp() 3327 execv() 3327 execvp() 3327 exit() 3302 fclose() 3330 feof() 3331 ferror() 3332 fflush() 3333 fgetc() 3334 fgetpos() 3336 fgets() 3337 fileno() 3339 fopen() 3340 fprintf() 3385 fputc() 3344 fputs() 3345 fread() 3347 free() 3372 freopen() 3340 fscanf() 3400 fseek() 3348 fseeko() 3348 fsetpos() 3336 ftell() 3350 ftello() 3350 fwrite() 3352 getc() 3334 getchar() 3334 getenv() 3353 getopt() 3354 getpwent() 3360 getpwnam() 3363 getpwuid() 3363 gets() 3337 gmtime() 3311 heap_clear() 3366 heap_min() 3366 input_line() 3368 isatty() 3370 labs() 3301 ldiv() 3314 localtime() 3311 major() 3371 makedev() 3371 malloc() 3372 memccpy() 3374 memchr() 3375 memcmp() 3376 memcpy() 3377 memmove() 3378 memset() 3378 minor() 3371 mktime() 3311 namep() 3379 offsetof() 3381 opendir() 3382 perror() 3384 printf() 3385 process_info() 3391 putc() 3344 putchar() 3344 putenv() 3392 puts() 3345 qsort() 3394 rand() 3395 readdir() 3396 realloc()

3372 rewind() 3398 rewinddir() 3399 scanf() 3400 seg_d() 3409 seg_i() 3409 setbuf() 3411 setenv() 3411 setpwent() 3360 setvbuf() 3411 snprintf() 3385 sp() 3310 sprintf() 3385 srand() 3395 ss() 3310 sscanf() 3400 stdio.h 3414 strcat() 3417 strchr() 3418 strcmp() 3419 strcoll() 3419 strcpy() 3421 strcspn() 3426 strdup() 3422 strerror() 3423 strlen() 3424 strncat() 3417 strncmp() 3419 strncpy() 3421 strpbrk() 3425 strrchr() 3418 strspn() 3426 strstr() 3427 strtok() 3428 strtol() 3432 strtoul() 3432 strxfrm() 3434 ttyname() 3435 unsetenv() 3411 vfprintf() 3437 vfscanf() 3439 vprintf() 3437 vscanf() 3439 vsnprintf() 3437 vsprintf() 3437 vsscanf() 3439

os16: 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 ***errno***.

ERRORI

Valore di ***errno***	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, senza usare una chiamata di sistema. Pertanto, si tratta di una valutazione presunta e non reale.

FILE SORGENTI

'`lib/unistd.h`' [u0.17]

‘lib/unistd/access.c’ [i161.17.2]

VEDERE ANCHE

stat(2) [u0.36].

os16: 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 os16 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' [u0.10]

'lib/stdlib/abort.c' [i161.10.2]

VEDERE ANCHE

signal(2) [u0.34].

os16: abs(3)

NOME

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

SINTASSI

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

DESCRIZIONE

Le funzioni '...**abs()**' restituiscono il valore assoluto del loro argomento. Si distinguono per tipo di intero e, nel caso di os16, non essendo disponibile il tipo '**long long int**', si limitano a ***abs()*** e ***labs()***.

VALORE RESTITUITO

Il valore assoluto del numero intero fornito come argomento.

FILE SORGENTI

'lib/stdlib.h' [u0.10]

'lib/stdlib/abs.c' [i161.10.3]

'lib/stdlib/labs.c' [i161.10.12]

VEDERE ANCHE

div(3) [u0.15], *ldiv(3)* [u0.15], *rand(3)* [u0.85].

os16: 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)* [u0.2], 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 os16 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`' [i161.1.8]

'`lib/stdlib.h`' [u0.10]

'`lib/stdlib/atexit.c`' [i161.10.5]

'`lib/stdlib/exit.c`' [i161.10.10]

VEDERE ANCHE

_exit(2) [u0.2], *_Exit(2)* [u0.2].

os16: 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`' [u0.10]

'`lib/stdlib/atoi.c`' [i161.10.6]

'`lib/stdlib/atol.c`' [i161.10.7]

VEDERE ANCHE

strtol(3) [u0.121], *strtoul(3)* [u0.121].

os16: atol(3)

«

Vedere *atoi(3)* [u0.5].

os16: 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 ***path***	Risultato prodotto da '`dirname(`*path*`)`'	Risultato prodotto da '`basename(`*path*`)`'
`"/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`' [u0.6]

'`lib/libgen/basename.c`' [i161.6.1]

'`lib/libgen/dirname.c`' [i161.6.2]

os16: bp(3)

«

Vedere *cs(3)* [u0.12].

os16: 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()*** azzera gli indicatori di errore e di fine file, del flusso di file indicato come argomento.

FILE SORGENTI

'lib/stdio.h' [u0.9]

'lib/stdio/clearerr.c' [i161.9.2]

VEDERE ANCHE

feof(3) [u0.28], *ferror(3)* [u0.29], *fileno(3)* [u0.34], *stdio(3)* [u0.103].

os16: 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 ***errno*** viene impostata di conseguenza.

ERRORI

Valore di ***errno***	Significato
`EBADF`	La directory rappresentata da ***dp***, non è valida.

FILE SORGENTI

'`lib/sys/types.h`' [u0.14]

'`lib/dirent.h`' [u0.2]

'`lib/dirent/DIR.c`' [i161.2.1]

'`lib/dirent/closedir.c`' [i161.2.2]

VEDERE ANCHE

close(2) [u0.7], *opendir(3)* [u0.76], *readdir(3)* [u0.86], *rewinddir(3)* [u0.89].

os16: 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)* [u0.28].

FILE SORGENTI

'`lib/sys/types.h`' [u0.14]

'`lib/sys/stat.h`' [u0.13]

'`lib/fcntl.h`' [u0.4]

'`lib/fcntl/creat.c`' [i161.4.1]

VEDERE ANCHE

chmod(2) [u0.4], *chown(2)* [u0.5], *close(2)* [u0.7], *dup(2)* [u0.8], *fcntl(2)* [u0.13], *link(2)* [u0.24], *mknod(2)* [u0.26], *mount(2)* [u0.27], *open(2)* [u0.28] *read(2)* [u0.29], *stat(2)* [u0.36], *umask(2)* [u0.40], *unlink(2)* [u0.42], *write(2)* [u0.44], *fopen(3)* [u0.35].

os16: cs(3)

«

NOME

'**bp**', '**cs**', '**ds**', '**es**', '**sp**', '**ss**' - stato dei registri della CPU

SINTASSI

```
#include <sys/os16.h>
unsigned int cs (void);
unsigned int ds (void);
unsigned int ss (void);
unsigned int es (void);
unsigned int sp (void);
unsigned int bp (void);
```

DESCRIZIONE

Le funzioni elencate nel quadro sintattico, sono in realtà delle macroistruzioni, chiamanti funzioni con nomi analoghi, ma preceduti da un trattino basso (***_bp()***, ***_cs()***, ***_ds()***, ***_es()***, ***_sp()***, ***_ss()***), per interrogare lo stato di alcuni registri della CPU a fini diagnostici. I registri interessati sono quelli con lo stesso nome della macroistruzione usata per interrogarli.

FILE SORGENTI

'lib/sys/os16.h' [u0.12]

'lib/sys/os16/_cs.s' [i161.12.2]

'lib/sys/os16/_ds.s' [i161.12.3]

'lib/sys/os16/_ss.s' [i161.12.8]

'lib/sys/os16/_es.s' [i161.12.4]

'lib/sys/os16/_sp.s' [i161.12.7]

'lib/sys/os16/_bp.s' [i161.12.1]

VEDERE ANCHE

seg_i(3) [u0.91].

seg_d(3) [u0.91].

os16: 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. os16 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 os16 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 os16) 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`' [u0.16]

'`lib/time/asctime.c`' [i161.16.1]

'`lib/time/gmtime.c`' [i161.16.3]

‘lib/time/mktime.c’ [i161.16.4]

VEDERE ANCHE

date(1) [u0.8], *clock(2)* [u0.6], *time(2)* [u0.39].

os16: dirname(3)

«

Vedere *basename(3)* [u0.7].

os16: div(3)

«

NOME

‘**div**’, ‘**ldiv**’ - 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);
```

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**’ e ‘**ldiv_t**’, sono dichiarati nel file ‘stdlib.h’ nel modo seguente:

```
typedef struct {
    int      quot;
    int      rem;
} div_t;
//
typedef struct {
    long int quot;
    long int rem;
} ldiv_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`' [u0.10]

'`lib/stdlib/div.c`' [i161.10.8]

'`lib/stdlib/ldiv.c`' [i161.10.13]

VEDERE ANCHE

abs(3) [u0.3].

os16: ds(3)

«

Vedere *cs(3)* [u0.12].

os16: endpwent(3)

«

Vedere *getpwent(3)* [u0.53].

os16: 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 os16 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 os16 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)* [u0.77] può visualizzare più dati.

Pertanto, nel codice di os16, si usa sempre la macroistruzione ***errset()***, invece di assegnare semplicemente un valore alla variabile ***errno***.

ERRORI

Gli errori previsti dalla libreria di os16 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 os16.

Valore di ***errno***	Definizione
`E2BIG`	Argument list too long.
`EACCES`	Permission denied.
`EADDRINUSE`	Address in use.
`EADDRNOTAVAIL`	Address not available.
`EAFNOSUPPORT`	Address family not supported.

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

Valore di ***errno***	Definizione
`ECANCELED`	Operation canceled.
`ECHILD`	No child processes.
`ECONNABORTED`	Connection aborted.
`ECONNREFUSED`	Connection refused.
`ECONNRESET`	Connection reset.

Valore di ***errno***	Definizione
`EDEADLK`	Resource deadlock would occur.
`EDESTADDRREQ`	Destination address required.
`EDOM`	Mathematics argument out of domain of function.
`EDQUOT`	Reserved.
`EEXIST`	File exists.

Valore di ***errno***	Definizione
`EFAULT`	Bad address.
`EFBIG`	File too large.
`EHOSTUNREACH`	Host is unreachable.
`EIDRM`	Identifier removed.
`EILSEQ`	Illegal byte sequence.

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

Valore di ***errno***	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 ***errno***	Definizione
`EMULTIHOP`	Reserved.
`ENAMETOOLONG`	Filename too long.
`ENETDOWN`	Network is down.
`ENETRESET`	Connection aborted by network.
`ENETUNREACH`	Network unreachable.

Valore di ***errno***	Definizione
`ENFILE`	Too many files open in system.
`ENOBUFS`	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 ***errno***	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 ***errno***	Definizione
`ENOPROTOOPT`	Protocol not available.
`ENOSPC`	No space left on device.
`ENOSR`	No stream resources.
`ENOSTR`	Not a stream.
`ENOSYS`	Function not supported.

Valore di ***errno***	Definizione
`ENOTCONN`	The socket is not connected.
`ENOTDIR`	Not a directory.
`ENOTEMPTY`	Directory not empty.
`ENOTSOCK`	Not a socket.
`ENOTSUP`	Not supported.

Valore di ***errno***	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 ***errno***	Definizione
`EPIPE`	Broken pipe.
`EPROTO`	Protocol error.
`EPROTONOSUPPORT`	Protocol not supported.
`EPROTOTYPE`	Protocol wrong type for socket.
`ERANGE`	Result too large.
`EROFS`	Read-only file system.
`ESPIPE`	Invalid seek.
`ESRCH`	No such process.
`ESTALE`	Reserved.
`ETIME`	Stream ioctl() timeout.

Valore di *errno*	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 os16.

Valore di *errno*	Definizione
EUNKNOWN	Unknown error.
E_FILE_TYPE	File type not compatible.
E_ROOT_INODE_NOT_CACHED	The root directory inode is not cached.
E_CANNOT_READ_SUPERBLOCK	Cannot read super block.
E_MAP_INODE_TOO_BIG	Map inode too big.

Valore di ***errno***	Definizione
`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.
`E_CANNOT_FIND_ROOT_INODE`	Cannot find root inode.
`E_FILE_TYPE_UNSUPPORTED`	File type unsupported.

Valore di ***errno***	Definizione
`E_ENV_TOO_BIG`	Environment too big.
`E_LIMIT`	Exceeded implementation limits.
`E_NOT_MOUNTED`	Not mounted.
`E_NOT_IMPLEMENTED`	Not implemented.

FILE SORGENTI

'`lib/errno.h`' [u0.3]

'`lib/errno/errno.c`' [i161.3.1]

VEDERE ANCHE

perror(3) [u0.77], *strerror(3)* [u0.111].

os16: es(3)

«

Vedere *cs(3)* [u0.12].

os16: 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)* [u0.10].

Il primo argomento delle funzioni descritte qui è il percorso, rappresentato 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 gli argomenti successivi sono gli argomenti da passare al programma stesso.

Rispetto al gruppo di funzioni '**`execl...()`**', 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 argomento 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, queste funzioni possono restituire soltanto un valore che rappresenta un errore, ovvero −1, aggiornando anche la variabile ***errno*** di conseguenza.

ERRORI

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

FILE SORGENTI

'`lib/unistd.h`' [u0.17]

'`lib/unistd/execl.c`' [i161.17.9]

'`lib/unistd/execle.c`' [i161.17.10]

'`lib/unistd/execlp.c`' [i161.17.11]

'`lib/unistd/execv.c`' [i161.17.12]

'`lib/unistd/execvp.c`' [i161.17.14]

'`lib/unistd/execve.c`' [i161.17.13]

VEDERE ANCHE

execve(2) [u0.10], *fork(2)* [u0.14], *environ(7)* [u0.1].

os16: execl(3)

«

Vedere *exec(3)* [u0.20].

os16: execle(3)

«

Vedere *exec(3)* [u0.20].

os16: execlp(3)

«

Vedere *exec(3)* [u0.20].

os16: execv(3)

«

Vedere *exec(3)* [u0.20].

os16: execvp(3)

«

Vedere *exec(3)* [u0.20].

os16: exit(3)

«

Vedere *atexit(3)* [u0.4].

os16: 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 os16, 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 ***errno*** viene impostata di conseguenza.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il descrittore di file associato al flusso da chiudere, non è valido.

FILE SORGENTI

'`lib/stdio.h`' [u0.9]

'`lib/stdio/fclose.c`' [i161.9.3]

VEDERE ANCHE

close(2) [u0.7], *fopen(3)* [u0.35], *stdio(3)* [u0.103].

os16: 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`' [u0.9]

'`lib/stdio/feof.c`' [i161.9.4]

VEDERE ANCHE

clearerr(3) [u0.9], *ferror(3)* [u0.29], *fileno(3)* [u0.34], *stdio(3)* [u0.103].

os16: 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`' [u0.9]

'`lib/stdio/ferror.c`' [i161.9.5]

VEDERE ANCHE

clearerr(3) [u0.9], *feof(3)* [u0.28], *fileno(3)* [u0.34], *stdio(3)* [u0.103].

os16: 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 os16, 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`' [u0.9]

'`lib/stdio/fflush.c`' [i161.9.6]

VEDERE ANCHE

fclose(3) [u0.27], *fopen(3)* [u0.35].

os16: 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 ***errno***	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`' [u0.9]

'`lib/stdio/fgetc.c`' [i161.9.7]

'`lib/stdio/getchar.c`' [i161.9.24]

VEDERE ANCHE

fgets(3) [u0.33], *gets(3)* [u0.33].

os16: 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 os16, 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 ***errno***.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il descrittore di file associato al flusso, non è valido.

FILE SORGENTI

'`lib/stdio.h`' [u0.9]

'`lib/stdio/fgetpos.c`' [i161.9.8]

'`lib/stdio/fsetpos.c`' [i161.9.20]

VEDERE ANCHE

fseek(3) [u0.43], *ftell(3)* [u0.46], *rewind(3)* [u0.88].

os16: 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 ***errno***	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`' [u0.9]

'`lib/stdio/fgets.c`' [i161.9.9]

'`lib/stdio/gets.c`' [i161.9.25]

VEDERE ANCHE

fgetc(3) [u0.31], *getc(3)* [u0.31].

os16: 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 ***errno***	Significato
`EBADF`	È stato richiesto di risolvere il puntatore nullo.

FILE SORGENTI

'`lib/stdio.h`' [u0.9]

'`lib/stdio/FILE.c`' [i161.9.1]

'`lib/stdio/fileno.c`' [i161.9.10]

VEDERE ANCHE

clearerr(3) [u0.9], *feof(3)* [u0.28], *ferror(3)* [u0.29], *stdio(3)* [u0.103].

os16: 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.

mode	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

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

ERRORI

Valore di ***errno***	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’ [u0.9]

‘lib/stdio/FILE.c’ [i161.9.1]

‘lib/stdio/fopen.c’ [i161.9.11]

‘lib/stdio/freopen.c’ [i161.9.16]

VEDERE ANCHE

open(2) [u0.28], *fclose(3)* [u0.27], *stdio(3)* [u0.103].

os16: fprintf(3)

«

Vedere *printf(3)* [u0.78].

os16: 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 *errno*	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`' [u0.9]

'`lib/stdio/fputc.c`' [i161.9.13]

VEDERE ANCHE

fputs(3) [u0.38], *puts(3)* [u0.38].

os16: 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
≥ 0	Rappresenta il successo dell'operazione.
'**`EOF`**'	Indica il verificarsi di un errore, il quale può essere interpretato eventualmente leggendo la variabile ***errno***.

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 ***errno***	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`' [u0.9]

'`lib/stdio/fputs.c`' [i161.9.14]

'`lib/stdio/puts.c`' [i161.9.28]

VEDERE ANCHE

fputc(3) [u0.37], *putc(3)* [u0.37], *putchar(3)* [u0.37].

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)* [u0.28] e di *ferror(3)* [u0.29].

FILE SORGENTI

'lib/stdio.h' [u0.9]

'lib/stdio/fread.c' [i161.9.15]

VEDERE ANCHE

read(2) [u0.29], *write(2)* [u0.44], *feof(3)* [u0.28], *ferror(3)* [u0.29], *fwrite(3)* [u0.48].

os16: free(3)

«

Vedere *malloc(3)* [u0.66].

os16: freopen(3)

«

Vedere *fopen(3)* [u0.35].

os16: fscanf(3)

«

Vedere *scanf(3)* [u0.90].

os16: 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 ***whence***	Significato
`SEEK_SET`	lo scostamento si riferisce all'inizio del file.
`SEEK_CUR`	lo scostamento si riferisce alla posizione che ha già l'indice interno al file.
`SEEK_END`	lo scostamento si riferisce alla fine del file.

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 ***errno***.

ERRORI

Valore di ***errno***	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`' [u0.9]

'`lib/stdio/FILE.c`' [i161.9.1]

'`lib/stdio/fseek.c`' [i161.9.18]

VEDERE ANCHE

lseek(2) [u0.24], *fgetpos(3)* [u0.32], *fsetpos(3)* [u0.32], *ftell(3)* [u0.46], *rewind(3)* [u0.88].

os16: fseeko(3)

«

Vedere *fseek(3)* [u0.43].

os16: fsetpos(3)

«

Vedere *fgetpos(3)* [u0.32].

os16: 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
≥ 0	Rappresenta l'indice interno al file.
-1	Indica il verificarsi di un errore, il quale può essere interpretato leggendo la variabile ***errno***.

ERRORI

Valore di ***errno***	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`' [u0.9]

'`lib/stdio/FILE.c`' [i161.9.1]

'`lib/stdio/ftell.c`' [i161.9.21]

'`lib/stdio/ftello.c`' [i161.9.22]

VEDERE ANCHE

lseek(2) [u0.24], *fgetpos(3)* [u0.32], *fsetpos(3)* [u0.32], *ftell(3)* [u0.43], *rewind(3)* [u0.88].

os16: ftello(3)

Vedere *ftell(3)* [u0.46].

os16: 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)* [u0.29].

FILE SORGENTI

‘`lib/stdio.h`’ [u0.9]

‘`lib/stdio/fwrite.c`’ [i161.9.23]

VEDERE ANCHE

read(2) [u0.29], *write(2)* [u0.44], *feof(3)* [u0.28], *ferror(3)* [u0.29], *fread(3)* [u0.39].

os16: getc(3)

«

Vedere *fgetc(3)* [u0.31].

os16: getchar(3)

«

Vedere *fgetc(3)* [u0.31].

os16: 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`' [u0.10]

'`applic/crt0.s`' [i162.1.9]

‘lib/stdlib/environment.c’ [i161.10.9]

‘lib/stdlib/getenv.c’ [i161.10.11]

VEDERE ANCHE

environ(7) [u0.1], *putenv(3)* [u0.82], *setenv(3)* [u0.94], *unsetenv(3)* [u0.94].

os16: 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)* [u0.10], dove si ammette solo l'invio dei segnali SIGTERM e SIGQUIT.

```
#include <sys/os16.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`' [u0.17]

'`lib/unistd/getopt.c`' [i161.17.20]

os16: 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 fun-

zione occorre azzerare il valore della variabile ***errno***, verificando successivamente se ha acquisito un valore differente.

ERRORI

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

FILE SORGENTI

'`lib/sys/types.h`' [u0.14]

'`lib/pwd.h`' [u0.7]

'`lib/pwd/pwent.c`' [i161.7.1]

VEDERE ANCHE

getpwnam(3) [u0.54], *getpwuid(3)* [u0.54], *passwd(5)* [u0.3].

os16: 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 ***errno***	Significato
`EINVAL`	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
`EPERM`	Operazione non consentita.
`EEXIST`	Il file da creare esiste già.
`EACCES`	Accesso non consentito.
`ENOENT`	Il file non esiste, oppure non esiste il percorso che porta al file da creare.
`EROFS`	Avendo richiesto un accesso in scrittura, si ottiene che il file system che lo contiene consente soltanto un accesso in lettura.
`ENOTDIR`	Il percorso che porta al file da aprire non è valido, in quanto ciò che dovrebbe essere una directory, non lo è.
`ENFILE`	Non si possono aprire altri file nell'ambito del sistema operativo (il sistema ha raggiunto il limite).
`EMFILE`	Non si possono aprire altri file nell'ambito del processo in corso.

FILE SORGENTI

'`lib/sys/types.h`' [u0.14]

'`lib/pwd.h`' [u0.7]

'`lib/pwd/pwent.c`' [i161.7.1]

VEDERE ANCHE

getpwent(3) [u0.53], *setpwent(3)* [u0.53], *endpwent(3)* [u0.53], *passwd(5)* [u0.3].

os16: getpwuid(3)

«

Vedere *getpwnam(3)* [u0.54].

os16: gets(3)

«

Vedere *fgets(3)* [u0.33].

os16: heap(3)

«

NOME

'**heap_clear**', '**heap_min**' - verifica dello spazio disponibile per la pila dei dati

SINTASSI

```
#include <sys/os16.h>
void heap_clear    (void);
int  heap_min      (void);
```

DESCRIZIONE

Le funzioni ***heap_clear()*** e ***heap_min()*** servono per poter conoscere, in un certo momento, lo spazio di memoria disponibile per la pila dei dati, durante il funzionamento del processo elaborativo.

La funzione ***heap_clear()*** sovrascrive la memoria tra la fine della memoria utilizzata per le variabili non inizializzate (BSS) e la parte superiore della pila dei dati. In altri termini, sovrascrive la parte di memoria disponibile per la pila dei dati, che in quel momento non è utilizzata. Vengono scritte sequenze di bit a uno.

La funzione ***heap_min()***, da usare successivamente a ***heap_clear()***, anche più avanti nell'esecuzione del processo, scandisce questa memoria e verifica, empiricamente, il livello minimo di memoria rimasto libero per la pila, in base all'utilizzo che se ne è fatto fino a quel punto. In pratica, serve a verificare se il programma da cui ha origine il processo ha uno spazio sufficiente per la pila dei dati o se ci sia il rischio di sovrapposizione con le altre aree dei dati.

VALORE RESTITUITO

La funzione ***heap_min()*** restituisce la quantità di byte di memoria continua, presumibilmente non ancora utilizzata dalla pila dei dati, che separa la pila stessa dalle altre aree di dati.

FILE SORGENTI

'`lib/sys/os16.h`' [u0.12]

'`lib/sys/os16/heap_clear.c`' [i161.12.9]

'`lib/sys/os16/heap_min.c`' [i161.12.10]

VEDERE ANCHE

cs(3) [u0.12], *ds(3)* [u0.12], *es(3)* [u0.12], *ss(3)* [u0.12], *bp(3)* [u0.12], *sp(3)* [u0.12].

os16: heap_clear(3)

«

Vedere *heap(3)* [u0.57].

os16: heap_min(3)

«

Vedere *heap(3)* [u0.57].

os16: input_line(3)

«

NOME

'**input_line**' - riga di comando

SINTASSI

```
#include <sys/os16.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 dell'unico mezzo corretto di inserimento di un dato da tastiera, per os16, il quale non dispone di una gestione completa dei terminali.

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/os16.h':

Macro-variabile	Descrizione
`INPUT_LINE_ECHO`	Produce un comportamento «normale», per cui ciò che viene digitato è rappresentato conformemente sullo schermo del terminale.
`INPUT_LINE_HIDDEN`	Fa sì che quanto digitato non appaia: si usa per esempio per l'inserimento di una parola d'ordine.
`INPUT_LINE_STARS`	Fa sì che per ogni carattere digitato appaia sullo schermo un asterisco: si usa per esempio per l'inserimento di una parola d'ordine, quando si vuole agevolare l'utente.

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/os16.h`' [u0.12]

'`lib/sys/os16/input_line.c`' [i161.12.11]

VEDERE ANCHE

shell(1) [u0.19].

login(1) [u0.12].

os16: 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 ***errno*** viene aggiornata.

ERRORI

Valore di ***errno***	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`’ [u0.17]

‘`lib/unistd/isatty.c`’ [i161.17.25]

VEDERE ANCHE

stat(2) [u0.36], *ttyname(3)* [u0.124].

os16: labs(3)

«

Vedere *abs(3)* [u0.3].

os16: ldiv(3)

«

Vedere *div(3)* [u0.15].

os16: major(3)

«

Vedere *makedev(3)* [u0.65].

os16: 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`' [u0.14]

'`lib/sys/types/makedev.c`' [i161.14.2]

'`lib/sys/types/major.c`' [i161.14.1]

'`lib/sys/types/minor.c`' [i161.14.3]

os16: 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 ***errno***	Significato
`ENOMEM`	Memoria insufficiente.

DIFETTI

L'allocazione dinamica di memoria, della libreria di os16, utilizza un metodo rudimentale, basato su un array statico che viene

allocato completamente se nella compilazione si utilizzano queste funzioni. Questo array, denominato ***_alloc_memory[]***, viene utilizzato come area per l'allocazione della memoria, con l'ausilio di altre due variabili allo scopo di tenere traccia della mappa di allocazione. In pratica, la memoria che si può gestire in questo modo è molto poca, ma soprattutto, i processi che ne fanno uso, in realtà, la allocano subito tutta.

FILE SORGENTI

'`lib/limits.h`' [i161.1.8]

'`lib/stdlib.h`' [u0.10]

'`lib/stdlib/alloc.c`' [i161.10.4]

os16: memccpy(3)

«

NOME

'**`memccpy`**' - copia di un'area di memoria

SINTASSI

```
#include <string.h>
void *memccpy (void *restrict dst,
               const void *restrict org,
               int c, 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`' [u0.11]

'`lib/string/memccpy.c`' [i161.11.1]

VEDERE ANCHE

memcpy(3) [u0.70], *memmove(3)* [u0.71], *strcpy(3)* [u0.108], *strncpy(3)* [u0.108].

os16: 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`' [u0.11]

'`lib/string/memchr.c`' [i161.11.2]

VEDERE ANCHE

strchr(3) [u0.105], *strrchr(3)* [u0.105], *strpbrk(3)* [u0.116].

os16: 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	***memory1 < memory2***
0	***memory1 == memory2***
+1	***memory1 > memory2***

FILE SORGENTI

'lib/string.h' [u0.11]

'lib/string/memcmp.c' [i161.11.3]

VEDERE ANCHE

strcmp(3) [u0.106], *strncmp(3)* [u0.106].

os16: 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' [u0.11]

'lib/string/memcpy.c' [i161.11.4]

VEDERE ANCHE

memccpy(3) [u0.67], *memmove(3)* [u0.71], *strcpy(3)* [u0.108], *strncpy(3)* [u0.108].

os16: 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`' [u0.11]

'`lib/string/memmove.c`' [i161.11.5]

VEDERE ANCHE

memccpy(3) [u0.67], *memcpy(3)* [u0.70], *strcpy(3)* [u0.108], *strncpy(3)* [u0.108].

os16: memset(3)

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`' [u0.11]

'`lib/string/memset.c`' [i161.11.6]

VEDERE ANCHE

memcpy(3) [u0.70].

os16: minor(3)

Vedere *makedev(3)* [u0.65].

os16: namep(3)

NOME

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

SINTASSI

```
#include <sys/os16.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 ***errno***.

ERRORI

Valore di *errno*	Significato
EINVAL	Uno degli argomenti non è valido.
ENOENT	La variabile di ambiente ***PATH*** 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/os16.h' [u0.12]

'lib/sys/os16/namep.c' [i161.12.13]

VEDERE ANCHE

shell(1) [u0.19], *execvp(3)* [u0.20], *execlp(3)* [u0.20].

os16: 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`' [i161.1.14]

os16: 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 ***errno***	Significato
`EINVAL`	Il valore di ***name*** non è valido.
`EMFILE`	Troppi file aperti dal processo.
`ENFILE`	Troppi file aperti complessivamente nel sistema.
`ENOTDIR`	Il percorso indicato in ***name*** 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`' [u0.14]

'`lib/dirent.h`' [u0.2]

'`lib/dirent/DIR.c`' [i161.2.1]

'`lib/dirent/opendir.c`' [i161.2.3]

VEDERE ANCHE

open(2) [u0.28], *closedir(3)* [u0.10], *readdir(3)* [u0.86], *rewinddir(3)* [u0.89].

os16: 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 os16, 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)* [u0.18], la quale non è prevista dagli standard.

FILE SORGENTI

'`lib/errno.h`' [u0.3]

'`lib/stdio.h`' [u0.9]

'`lib/stdio/perror.c`' [i161.9.26]

VEDERE ANCHE

errno(3) [u0.18], *strerror(3)* [u0.111].

«

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 os16:

```
%[simbolo][n_ampiezza][.n_precisione][hh|h|l|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 '**`unsigned char`**'.
`%…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 conversione, si vede che subito dopo il segno di percentuale può apparire un simbolo (*flag*).

Simbolo	Corrispondenza
`%+`... `%+0`***ampiezza***...	Il segno «+» fa sì che i numeri con segno lo mostrino anche se è positivo. Può combinarsi con lo zero.
`%0`***ampiezza***... `%+0`***ampiezza***...	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 «+».
`%`***ampiezza***... `% `***ampiezza***...	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.
`%-`***ampiezza***... `%-+`***ampiezza***...	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`	`unsigned char`
`%...hd` `%...hi`	`short int`	`%...hu` `%...ho` `%...hx` | `%...hX`	`unsigned short int`
`%...ld` `%...li`	`long int`	`%...lu` `%...lo` `%...lx` | `%...lX`	`unsigned long int`

Simbolo	Tipo	Simbolo	Tipo
%…jd %…ji	intmax_t	%…ju %…jo %…jx | %…jX	uintmax_t
%…zd %…zi	size_t	%…zu %…zo %…zx | %…zX	size_t
%…td %…ti	ptrdiff_t	%…tu %…to %…tx | %…tX	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 os16, 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`' [u0.9]

‘lib/stdio/FILE.c’ [i161.9.1]

‘lib/stdio/fprintf.c’ [i161.9.12]

‘lib/stdio/printf.c’ [i161.9.27]

‘lib/stdio/sprintf.c’ [i161.9.34]

‘lib/stdio/snprintf.c’ [i161.9.33]

VEDERE ANCHE

vfprintf(3) [u0.128], *vprintf(3)* [u0.128], *vsprintf(3)* [u0.128], *vsnprintf(3)* [u0.128], *scanf(3)* [u0.90].

os16: process_info(3)

NOME

‘**process_info**’ - funzione diagnostica

SINTASSI

```
#include <sys/os16.h>
void process_info (void);
```

DESCRIZIONE

Si tratta di una funzione diagnostica che non richiede argomenti e non restituisce alcunché, per visualizzare, attraverso lo standard output, lo stato dei registri della CPU, i riferimenti principali della collocazione in memoria del processo elaborativo e lo spazio ancora non utilizzato dalla pila dei dati.

Per poter dare un’informazione utile sullo spazio non ancora utilizzato dalla pila dei dati, occorre che prima di questa funzione sia stata chiamata ***heap_clear()***.

FILE SORGENTI

'lib/sys/os16.h' [u0.12]

'lib/sys/os16/process_info.c' [i161.12.14]

VEDERE ANCHE

cs(3) [u0.12].

ds(3) [u0.12].

es(3) [u0.12].

ss(3) [u0.12].

bp(3) [u0.12].

sp(3) [u0.12].

heap_clear(3) [u0.57].

heap_min(3) [u0.57].

os16: putc(3)

«

Vedere *fputc(3)* [u0.37].

os16: putchar(3)

«

Vedere *fputc(3)* [u0.37].

os16: 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 ***errno***.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Argomento non valido.
`ENOMEM`	Memoria insufficiente.

FILE SORGENTI

'`lib/stdlib.h`' [u0.10]

'`lib/stdlib/environment.c`' [i161.10.9]

'`lib/stdlib/putenv.c`' [i161.10.14]

VEDERE ANCHE

environ(7) [u0.1], *getenv(3)* [u0.51], *setenv(3)* [u0.94], *unsetenv(3)* [u0.94].

os16: puts(3)

«

Vedere *fputs(3)* [u0.38].

os16: 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`’ [u0.10]

‘`lib/stdlib/qsort.c`’ [i161.10.15]

os16: 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`' [u0.10]

'`lib/stdlib/rand.c`' [i161.10.16]

os16: 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 *errno*	Significato
EBADF	La directory rappresentata da ***dp*** non è valida.

FILE SORGENTI

'lib/sys/types.h' [u0.14]

'lib/dirent.h' [u0.2]

'lib/dirent/DIR.c' [i161.2.1]

'lib/dirent/readdir.c' [i161.2.4]

VEDERE ANCHE

read(2) [u0.29], *closedir(3)* [u0.10], *opendir(3)* [u0.76], *rewinddir(3)* [u0.89].

os16: realloc(3)

«

Vedere *malloc(3)* [u0.66].

os16: 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`' [u0.9]

'`lib/stdio/FILE.c`' [i161.9.1]

'`lib/stdio/rewind.c`' [i161.9.29]

VEDERE ANCHE

lseek(2) [u0.24], *fgetpos(3)* [u0.32], *fsetpos(3)* [u0.32], *ftell(3)* [u0.46], *fseek(3)* [u0.43], *rewind(3)* [u0.88].

os16: 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`' [u0.14]

'`lib/dirent.h`' [u0.2]

'`lib/dirent/DIR.c`' [i161.2.1]

'`lib/dirent/rewinddir.c`' [i161.2.5]

VEDERE ANCHE

rewind(3) [u0.88], *closedir(3)* [u0.10], *opendir(3)* [u0.76], *rewinddir(3)* [u0.86].

os16: 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 os16:

```
%[*][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
`%…d`	`int *`	Numero intero con segno rappresentato in base dieci.
`%…i`	`int *`	Numero intero con segno rappresentare in base dieci o in base otto, avendo come prefisso uno zero, oppure in base sedici, avendo come prefisso '**`0x`**' o '**`0X`**'.
`%…u`	`unsigned int *`	Numero intero senza segno rappresentato in base dieci.
`%…o`	`unsigned int *`	Numero intero senza segno rappresentato in ottale (con o senza lo zero iniziale).

Simbolo	Tipo di argomento	Conversione applicata
`%…x`	`unsigned int *`	Numero intero senza segno rappresentato in esadecimale (con o senza il prefisso '**`0x`**' o '**`0X`**').
`%…c`	`char *`	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.
`%…s`	`char *`	Interpreta una sequenza di caratteri che non siano spazi, aggiungendo alla fine il carattere nullo di terminazione.
`%p`	`void *`	Interpreta il valore di un puntatore che sia rappresentato nello stesso modo in cui farebbe la funzione '**`printf("%p",`** ***puntatore*****`)`**'.

Simbolo	Tipo di argomento	Conversione applicata
`%n`	`int *`	Questo specificatore non esegue alcuna conversione e si limita a memorizzare la quantità di caratteri ('**`char`**') letti fino a quel punto.
`%…[…]`	`char *`	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: '**`%…[]…]`**'.
`%…[^…]`	`char *`	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: '**`%…[^]…]`**'.
`%%`		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 in-

terpretare, 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 *errno*	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' [u0.9]

'lib/stdio/fscanf.c' [i161.9.17]

'lib/stdio/scanf.c' [i161.9.30]

'lib/stdio/sscanf.c' [i161.9.35]

VEDERE ANCHE

vfscanf(3) [u0.129], *vscanf(3)* [u0.129], *vsscanf(3)* [u0.129], *printf(3)* [u0.78].

os16: seg_d(3)

«

NOME

'**seg_d**', '**seg_i**' - collocazione del processo in memoria

SINTASSI

```
#include <sys/os16.h>
unsigned int seg_d (void);
unsigned int seg_i (void);
```

DESCRIZIONE

Le funzioni elencate nel quadro sintattico, sono in realtà delle macroistruzioni, chiamanti funzioni con nomi analoghi, ma preceduti da un trattino basso (***_seg_d()*** e ***_seg_i()***), per interrogare, rispettivamente, lo stato del registro ***DS*** e ***CS***. Questi due registri indicano, rispettivamente, la collocazione dell'area dati e dell'area codice del processo in corso. Eventualmente, per conoscere l'indirizzo efficace di memoria corrispondente, occorre moltiplicare questi valori per 16.

FILE SORGENTI

'`lib/sys/os16.h`' [u0.12]

'`lib/sys/os16/_seg_i.s`' [i161.12.6]

'`lib/sys/os16/_seg_d.s`' [i161.12.5]

VEDERE ANCHE

cs(3) [u0.12].

ds(3) [u0.12].

es(3) [u0.12].

ss(3) [u0.12].

bp(3) [u0.12].

sp(3) [u0.12].

os16: seg_i(3)

«

Vedere *seg_d(3)* [u0.91].

os16: 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 os16, non fanno alcunché, perché os16 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`' [u0.9]

'`lib/stdio/setbuf.c`' [i161.9.31]

'`lib/stdio/setvbuf.c`' [i161.9.32]

VEDERE ANCHE

fflush(3) [u0.30].

os16: 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 ***errno***.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Argomento non valido.
`ENOMEM`	Memoria insufficiente.

FILE SORGENTI

'`lib/stdlib.h`' [u0.10]

'`applic/crt0.s`' [i162.1.9]

'`lib/stdlib/environment.c`' [i161.10.9]

'`lib/stdlib/setenv.c`' [i161.10.17]

'`lib/stdlib/unsetenv.c`' [i161.10.20]

VEDERE ANCHE

environ(7) [u0.1], *getenv(3)* [u0.51], *putenv(3)* [u0.82].

os16: setpwent(3)

«

Vedere *getpwent(3)* [u0.53].

os16: setvbuf(3)

«

Vedere *setbuf(3)* [u0.93].

os16: snprintf(3)

«

Vedere *printf(3)* [u0.78].

os16: sp(3)

«

Vedere *cs(3)* [u0.12].

os16: sprintf(3)

«

Vedere *printf(3)* [u0.78].

os16: srand(3)

«

Vedere *rand(3)* [u0.85].

os16: ss(3)

«

Vedere *cs(3)* [u0.12].

os16: sscanf(3)

«

Vedere *scanf(3)* [u0.90].

os16: 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 os16 non fornisce.

Nella libreria di os16, 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, contenuta 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`' [u0.14]

'`lib/stdio.h`' [u0.9]

'`lib/stdio/FILE.c`' [i161.9.1]

'`applic/crt0.s`' [i162.1.9]

VEDERE ANCHE

close(2) [u0.7], *open(2)* [u0.28], *read(2)* [u0.29], *write(2)* [u0.44].

os16: 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`' [u0.11]

'`lib/string/strcat.c`' [i161.11.7]

'`lib/string/strncat.c`' [i161.11.16]

VEDERE ANCHE

memccpy(3) [u0.67], *memcpy(3)* [u0.70], *strcpy(3)* [u0.108], *strncpy(3)* [u0.108].

os16: strchr(3)

«

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' [u0.11]

'lib/string/strchr.c' [i161.11.8]

'lib/string/strrchr.c' [i161.11.20]

VEDERE ANCHE

memchr(3) [u0.68], *strlen(3)* [u0.112], *strpbrk(3)* [u0.116], *strspn(3)* [u0.118].

os16: 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, os16 non è in grado di gestire le configurazioni locali, pertanto questa funzione coincide esattamente con ***strcmp()***.

VALORE RESTITUITO

Valore	Esito del confronto.
−1	***string1*** < ***string2***
0	***string1*** == ***string2***
+1	***string1*** > ***string2***

FILE SORGENTI

'`lib/string.h`' [u0.11]

'`lib/string/strcmp.c`' [i161.11.9]

'`lib/string/strncmp.c`' [i161.11.17]

'`lib/string/strcoll.c`' [i161.11.10]

VEDERE ANCHE

memcmp(3) [u0.69].

os16: strcoll(3)

«

Vedere *strcmp(3)* [u0.106].

«

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' [u0.11]

'lib/string/strcpy.c' [i161.11.11]

'lib/string/strncpy.c' [i161.11.18]

VEDERE ANCHE

memccpy(3) [u0.67], *memcpy(3)* [u0.70], *memmove(3)* [u0.71].

os16: strcspn(3)

«

Vedere *strspn(3)* [u0.118].

os16: 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 ***errno***	Significato
ENOMEM	Memoria insufficiente.

FILE SORGENTI

'lib/string.h' [u0.11]

'lib/string/strdup.c' [i161.11.13]

VEDERE ANCHE

free(3) [u0.66], *malloc(3)* [u0.66], *realloc(3)* [u0.66].

os16: 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' [u0.3]

'lib/string.h' [u0.11]

'lib/string/strerror.c' [i161.11.14]

VEDERE ANCHE

errno(3) [u0.18], *perror(3)* [u0.77].

os16: 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' [u0.11]

'lib/string/strlen.c' [i161.11.15]

os16: strncat(3)

«

Vedere *strcat(3)* [u0.104].

os16: strncmp(3)

«

Vedere *strcmp(3)* [u0.106].

os16: strncpy(3)

«

Vedere *strcpy(3)* [u0.108].

os16: 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' [u0.11]

'lib/string/strpbrk.c' [i161.11.19]

VEDERE ANCHE

memchr(3) [u0.68], *strchr(3)* [u0.105], *strstr(3)* [u0.119], *strtok(3)* [u0.120].

os16: strrchr(3)

«

Vedere *strchr(3)* [u0.105].

os16: 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`' [u0.11]

'`lib/string/strspn.c`' [i161.11.21]

'`lib/string/strcspn.c`' [i161.11.12]

VEDERE ANCHE

memchr(3) [u0.68], *strchr(3)* [u0.105], *strpbrk(3)* [u0.116], *strstr(3)* [u0.119], *strtok(3)* [u0.120].

os16: 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`' [u0.11]

'`lib/string/strstr.c`' [i161.11.22]

VEDERE ANCHE

memchr(3) [u0.68], *strchr(3)* [u0.105], *strpbrk(3)* [u0.116], *strtok(3)* [u0.120].

os16: 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.

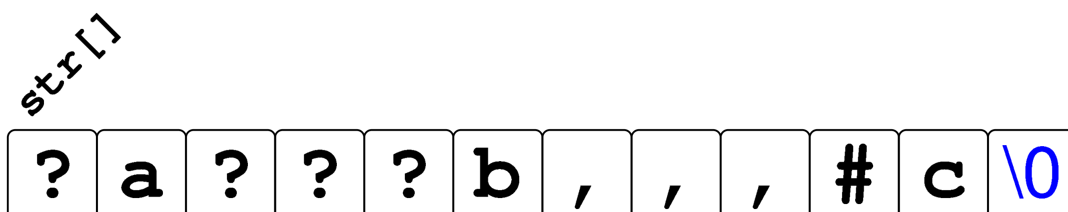
Per comprendere lo scopo della funzione viene utilizzato lo stesso esempio che appare nel documento *ISO/IEC 9899:TC2*, al paragrafo 7.21.5.7, con qualche piccola modifica per poterlo rendere un programma autonomo:

```
#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;
}
```

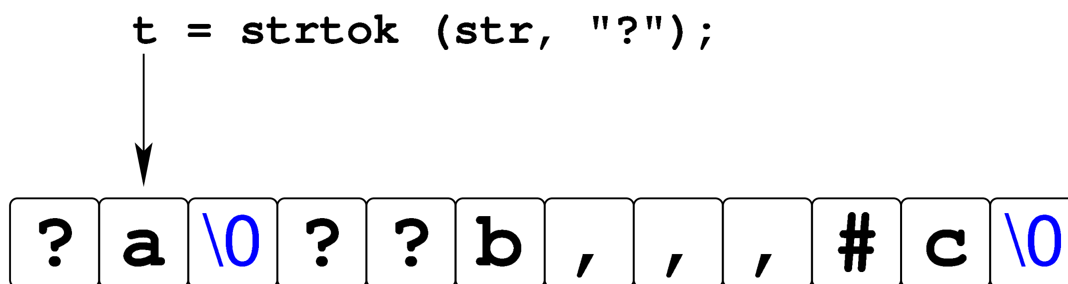
Avviando il programma si ottiene quanto già descritto dai commenti inseriti nel codice:

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

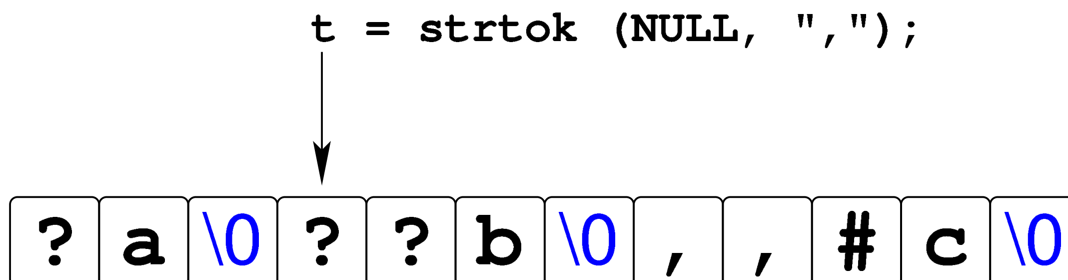
Ciò che avviene nell'esempio può essere schematizzato come segue. Inizialmente la stringa '**str**' ha in memoria l'aspetto seguente:



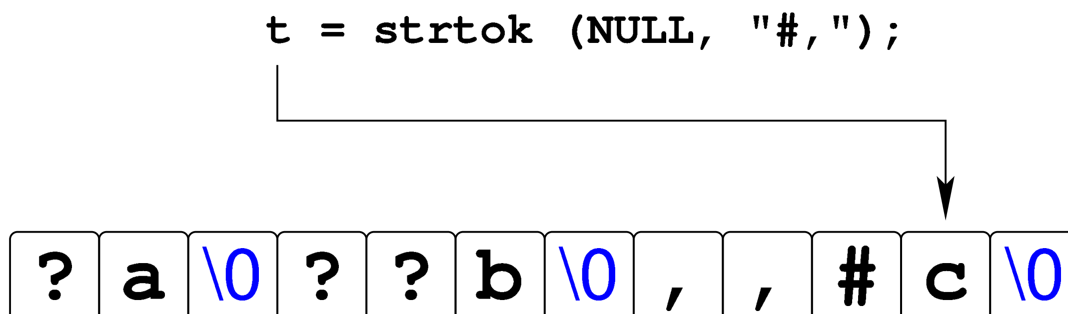
Dopo la prima chiamata della funzione ***strtok()*** la stringa risulta alterata e il puntatore ottenuto raggiunge la lettera '**`a`**':



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:



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' [u0.11]

'lib/string/strtok.c' [i161.11.23]

VEDERE ANCHE

memchr(3) [u0.68], *strchr(3)* [u0.105], *strpbrk(3)* [u0.116], *strspn(3)* [u0.118].

os16: 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 nu-

merazione 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 ***errno***	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`' [u0.10]

'`lib/stdlib/strtol.c`' [i161.10.18]

'`lib/stdlib/strtoul.c`' [i161.10.19]

os16: strtoul(3)

«

Vedere *strtol(3)* [u0.121].

os16: strxfrm(3)

«

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*** caratte-

ri nella destinazione, ma applicando una trasformazione relativa alla configurazione locale.

os16 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`' [u0.11]

'`lib/string/strxfrm.c`' [i161.11.24]

VEDERE ANCHE

memcmp(3) [u0.69], *strcmp(3)* [u0.106], *strcoll(3)* [u0.106].

os16: 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 ***errno***	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`' [u0.17]

'`lib/unistd/ttyname.c`' [i161.17.34]

VEDERE ANCHE

stat(2) [u0.36], *isatty(3)* [u0.61].

os16: unsetenv(3)

«

Vedere *setenv(3)* [u0.94].

os16: vfprintf(3)

«

Vedere *vprintf(3)* [u0.128].

os16: vfscanf(3)

«

Vedere *vfscanf(3)* [u0.129].

os16: 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 argomenti 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)* [u0.78], per la descrizione di come va predisposta la stringa ***format***. Nella realizzazione di os16, di tutte que-

ste 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`' [i161.1.12]

'`lib/stdio.h`' [u0.9]

'`lib/stdio/FILE.c`' [i161.9.1]

'`lib/stdio/vfprintf.c`' [i161.9.36]

'`lib/stdio/vprintf.c`' [i161.9.39]

'`lib/stdio/vsprintf.c`' [i161.9.42]

'`lib/stdio/vsnprintf.c`' [i161.9.41]

VEDERE ANCHE

fprintf(3) [u0.78], *printf(3)* [u0.78], *sprintf(3)* [u0.78], *snprintf(3)* [u0.78], *scanf(3)* [u0.90].

os16: 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)* [u0.90]. 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 ***vfsscanf()***, 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 *errno*	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' [u0.9]

'lib/stdio/vfscanf.c' [i161.9.37]

'lib/stdio/vscanf.c' [i161.9.40]

'lib/stdio/vsscanf.c' [i161.9.43]

'lib/stdio/vfsscanf.c' [i161.9.38]

VEDERE ANCHE

fscanf(3) [u0.90], *scanf(3)* [u0.90], *sscanf(3)* [u0.90], *printf(3)* [u0.78].

os16: vsnprintf(3)

«

Vedere *vprintf(3)* [u0.128].

os16: vsprintf(3)

Vedere *vprintf(3)* [u0.128].

os16: vsscanf(3)

Vedere *vsscanf(3)* [u0.129].