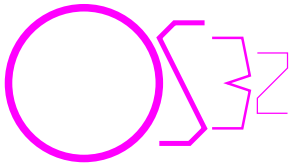
Parte viii

Manuale

«



os32, come già os16, ha due modalità di funzionamento: si può interagire direttamente con il kernel, oppure si può avviare il processo '**init**' e procedere con l'organizzazione consueta di un sistema Unix tradizionale. La modalità di colloquio diretto con il kernel è servita per consentire lo sviluppo di os32 e potrebbe essere utile per motivi di studio. Va osservato che durante l'interazione diretta con il kernel si dispone di una sola console, mentre quando si avvia '**init**' si possono avere delle console virtuali in base alla configurazione.

Video *http://www.youtube.com/watch?v=Bc33a2-NRzM*

85.1 Disco fisso in un file-immagine

os32 è distribuito in modo da funzionare con l'ausilio di Qemu o Bochs, installato in un disco fisso virtuale, rappresentato da un file-immagine. Tale file-immagine, denominato '`disk.hda`' è suddiviso in almeno due partizioni: la prima è necessaria per l'avvio e il suo formato dipende dal sistema di avvio installato (dovrebbe essere SYSLINUX, pertanto il file system dovrebbe essere di tipo Dos-FAT); la seconda è di tipo Minix-1 e ospita il sistema os32.

La creazione e l'accesso alle partizioni contenute nel file '`disk.hda`' è un po' complicato da un sistema GNU/Linux, perché occorre estrapolarle di volta in volta in file indipendenti, per aggregarle poi nuovamente in un file-immagine unico. Per queste operazioni si usano degli script già predisposti:

Script	Descrizione
`file_image_functions`	Contiene delle funzioni utilizzate dagli altri script [94.1.5].

Script	Descrizione
`fdisk` ***file***	Consente di utilizzare il programma '**`fdisk`**' sul file-immagine indicato, con l'ausilio delle funzioni contenute nel file '`file_image_functions`' [94.1.4].
`format` ***file n dos|minix***	Consente di inizializzare la partizione ***n***-esima del file-immagine indicato come primo argomento, con un file system di tipo Dos-FAT o di tipo Minix-1 [94.1.6].
`syslinux` ***file n***	Installa SYSLINUX nella partizione ***n*** del file-immagine indicato [94.1.10].

`mount`
 	
`umount`
	Innesta o stacca le partizioni contenute nel file-immagine '`disk.hda`' utilizzando le directory '`/mnt/disk.hda.`***n***' che però devono già essere disponibili.

Quando si compila il sistema con lo script ***makeit.mer*** o con ***makeit.sep***, al termine del processo questo tenta di copiare il kernel '`kimage`' nella directory '`/mnt/disk.hda1/`' e poi il sistema nella directory '`/mnt/disk.hda2/`'. Queste due directory dovrebbero innestare rispettivamente le prime due partizioni del file-immagine '`disk.hda`'; tuttavia, occorre disporre dei privilegi necessari, come dire che occorre fare questo in qualità di utente '**`root`**'.

La copia del sistema nella seconda partizione non provvede però a produrre i file di dispositivo necessari nella directory '`dev/`'. Eventualmente, all'interno di tale directory si ottiene lo script '`.makedev`', da avviare attraverso il sistema GNU/Linux.

Va comunque ricordato che prima di avviare os32 è necessario distaccare gli innesti delle partizioni del file-immagine, perché os32 funziona utilizzando il file-immagine nel suo complesso, mentre

l'innesto delle partizioni comporta l'estrapolazione delle partizioni e la successiva riaggregazione; inoltre, os32 non ha un alcun sistema di arresto del sistema, per cui è facile rendere incoerente il file system, quindi è bene fare spesso delle copie del file-immagine ed essere comunque pronti a ricostruirlo.

85.2 Configurazione dell'avvio e opzioni del kernel

Si utilizza SYSLINUX nella prima partizione per l'avvio di os32. Premesso che il kernel di os32 è contenuto nel file '`kimage`' e si colloca assieme ai file di SYSLINUX, nella configurazione contenuta nel file '`syslinux.cfg`' (relativo a SYSLINUX stesso) si leggono le istruzioni seguenti:

```
TIMEOUT 10
PROMPT 1
DEFAULT os32
#
LABEL os32
 KERNEL mboot.c32
 APPEND kimage net1=1,172.21.11.16,16 ↩
↪                route0=0.0.0.0,0,172.21.11.18,1 ↩
↪                route1=172.21.254.254,32,172.21.11.18,1
```

Con le opzioni passate al kernel di os32 si ottiene la configurazione dell'interfaccia di rete '**`net1`**' ('**`net0`**' corrisponde all'interfaccia *loopback*), con l'indirizzo 172.21.11.16, maschera di rete 255.255.0.0 e gli instradamenti necessari.

Opzione del kernel	Descrizione
`net`***i***`=`***i*****`,`*** ***ipv4*****`,`*** ***m***	Dichiara l'interfaccia di rete '**`net`*****i***', corrispondente all'indice ***i*** nella tabella delle interfacce, con l'indirizzo IPv4 specificato da ***ipv4*** e la maschera di rete lunga ***m*** bit.

Opzione del kernel	Descrizione
`route`***n*=*dest* , *m* , *router* , *i***	Dichiara la destinazione rappresentata dall'indirizzo ***dest*** e la maschera di rete composta da ***m*** bit, raggiungibile attraverso l'indirizzo ***router*** corrispondente all'interfaccia ***i*** nella tabella delle interfacce.

Volendo tradurre le opzioni che appaiono nell'esempio in comandi di un sistema GNU/Linux, sarebbe come se fosse stato scritto:

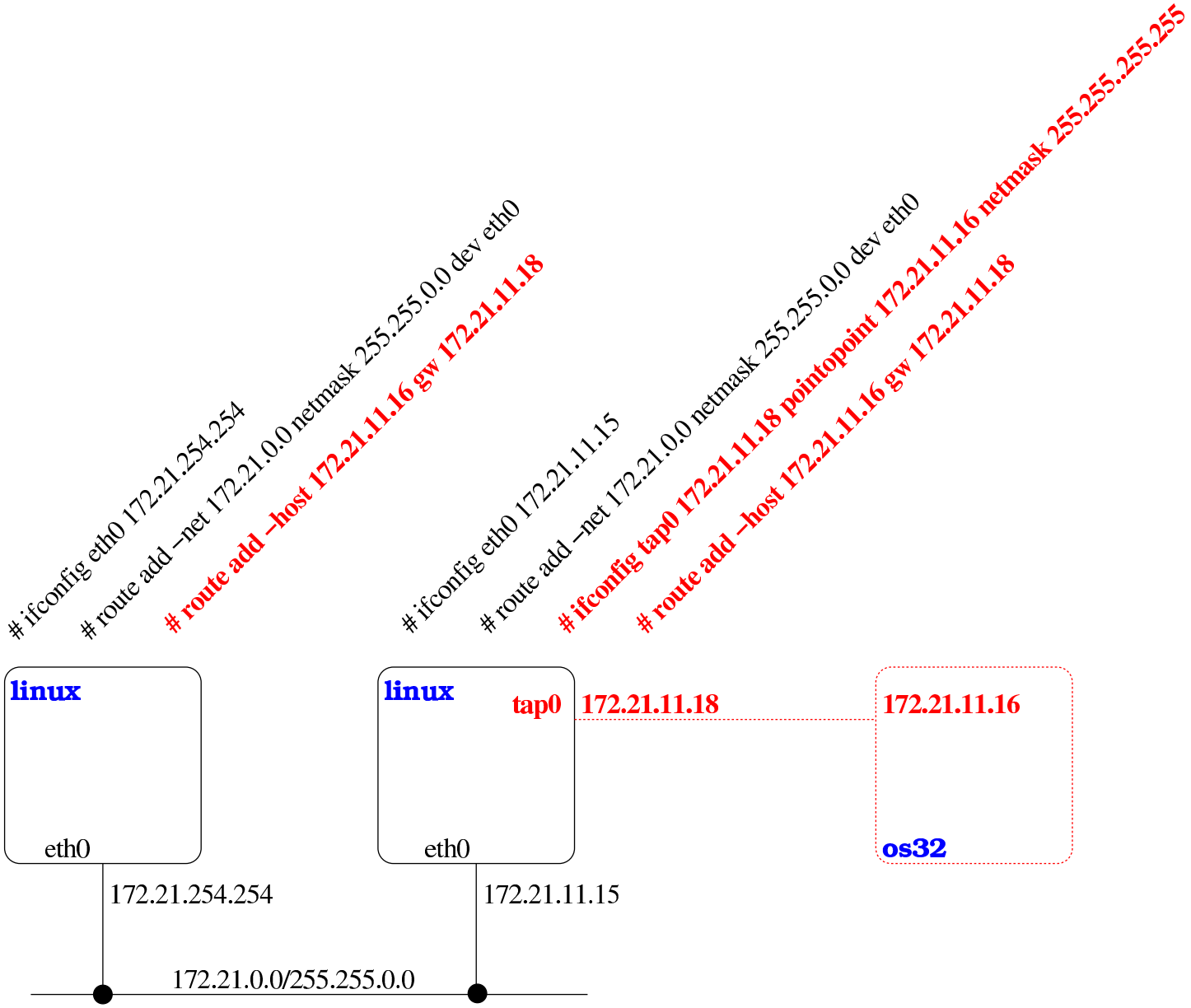
```
# ifconfig eth1 172.21.11.16 netmask 255.255.0.0 [Invio]

# route add -net 0.0.0.0 netmask 0.0.0.0 gw 172.21.11.18 ↩
↪       dev eth1 [Invio]

# route add -host 172.21.254.254 gw 172.21.11.18 ↩
↪       dev eth1 [Invio]
```

85.3 Configurazione per l'uso della rete

«

Per poter utilizzare le funzionalità di rete di os32, attraverso l'emulatore, è necessario predisporre nel sistema ospitante un'interfaccia di rete virtuale, connessa con quella di os32 attraverso una connessione punto-punto (sempre virtuale). Gli script che accompagnano os32 prevedono l'uso con Qemu o Bochs, in un sistema GNU/Linux, creando una connessione come quella schematizzata nella figura successiva.

Figura 85.4. Ipotesi di collegamento di os32 con il sistema ospitante e con il resto della rete locale: l'elaboratore che esegue os32 all'interno di un emulatore è quello corrispondente all'indirizzo 172.21.11.15. Le istruzioni che riguardano la configurazione necessaria a connettersi con os32 appaiono in rosso.



Il sistema os32, a sua volta, si configura esclusivamente attraverso le opzioni di avvio, come descritto nella sezione precedente.

85.4 Avvio di os32

«

L'avvio di os32 è un po' complicato, a causa della necessità di predisporre la rete nel modo appropriato, nel sistema ospitante, ma in pratica si usa lo script '`qemu`' o lo script '`bochs`', i quali si avvalgono a loro volta di '`tap0`', per creare l'interfaccia di rete virtuale '**`tap0`**' nel sistema ospitante. Tuttavia, l'avvio deve avvenire con i privilegi dell'utente '**`root`**' per poter predisporre le funzionalità di rete; pertanto va usato un programma come '**`gksu`**' per ottenere tali privilegi:

```
$ gksu ./qemu [Invio]
```

Oppure:

```
$ gksu ./bochs [Invio]
```

Negli esempi gli script vengono avviati indicando il percorso, per evitare che vengano confusi con i nomi dei file eseguibili di Qemu e di Bochs, i quali si trovano presumibilmente nella directory '`/usr/bin/`'.

Script	Descrizione
`tap0`	Predispone l'interfaccia '**`tap0`**' con l'instradamento necessario a raggiungere os32; questo script viene utilizzato da Bochs o da Qemu [94.1.11].

Script	Descrizione
`qemu`	Avvia Qemu con le opzioni appropriate per eseguire os32 nel file-immagine '`disk.hda`', configurando la rete in modo appropriato. Tra le opzioni dell'avvio di Qemu si trova la richiesta di utilizzare lo script '`tap0`' per la configurazione della rete nel sistema ospitante [94.1.9].
`bochs`	Avvia Bochs con le opzioni appropriate per eseguire os32 nel file-immagine '`disk.hda`', configurando la rete in modo appropriato. Tra le opzioni dell'avvio di Bochs si trova la richiesta di utilizzare lo script '`tap0`' per la configurazione della rete nel sistema ospitante [94.1.2].

85.5 Interazione con il kernel

L'avvio di os32 passa, in ogni caso, per una prima fase di colloquio con il kernel. Si ottiene un menù e si possono premere semplicemente dei tasti, seguiti però da [*Invio*], secondo l'elenco previsto, per ottenere delle azioni molto semplici. In questa fase il disco da cui risulta avviato il kernel è già innestato ed è prevista la possibilità di avviare tre programmi: '`/bin/aaa`', '`/bin/bbb`' e '`/bin/ccc`'. In tal modo, si ha la possibilità di avviare qualcosa, a titolo diagnostico, prima dello stesso '**`init`**' ('`/bin/init`').

Figura 85.6. Aspetto di os32 in funzione, con il menù in evidenza.

```
os32 build 20AAMMGGHHmm ram 130048 Kibyte
[ata_init] ATA drive 0 size 8064 Kib
[ata_drq] ERROR: drive 2 error
[dm_init] ATA drive=0 total sectors=16128
[dm_init] partition type=0c start sector=63 total sectors=2961
[dm_init] partition type=81 start sector=3024 total sectors=13104
.-------------------------------------------------------------------.
| h    show this menu                                 .--------------.|
| t    show internal timer values                     | all commands ||
| f    fork the kernel                                | followed by  ||
| m    memory map (HEX)                               | [Enter]      ||
| g|G  show GDT table first 21+21 items               `--------------'|
| i|I  show IDT table first 21+21 items                              |
| p    process status list                                         |
| s    super block list                                            |
| n    list of active inodes                                       |
| 1..9 kill process  1 to 9                                        |
| A..F kill process 10 to 15                                       |
| a..c run programs '/bin/aaa' to '/bin/ccc' in parallel           |
| x    exit interaction with kernel and start '/bin/init'          |
| q    quit kernel                                                 |
`-------------------------------------------------------------------'
```

Le funzioni principali disponibili in questa modalità diagnostica sono riassunte nella tabella successiva. È importante tenere presente che, a differenza di os16, anche in questa fase i comandi sono conclusi con la pressione del tasto [*Invio*].

Comando	Risultato
`h`	Mostra il menù di funzioni disponibili.
`t`	Mostra i valori gestiti internamente dell'orologio del kernel.

Comando	Risultato
`f`	Esegue una biforcazione del kernel, nella quale, il processo figlio si limita a mostrare ripetutamente il proprio numero di processo.
`m`	Mostra la mappa della memoria, elencando le aree continue utilizzate.
`g` `G`	Mostra le prime voci della tabella GDT, in binario.
`i` `I`	Mostra le prime voci della tabella IDT, in binario.
`p`	Mostra la situazione dei processi e altre informazioni.
`s`	Mostra delle informazioni sul super blocco.
`n`	Mostra l'elenco degli inode attivi.
`1`	Invia il segnale '**`SIGKILL`**' al processo numero uno.
`2|3|4|5|6|7|8|9|A|B|C|D|E|F`	Invia il segnale '**`SIGTERM`**' al processo con il numero corrispondente, da 2 a 15.
`a|b|c`	Avvia il programma '`/bin/aaa`', '`/bin/bbb`' o '`/bin/ccc`'.
`x`	Termina il ciclo e successivamente si passa all'avvio di '`/bin/init`'.
`q`	Ferma il sistema.

Figura 85.8. Aspetto di os32 in funzione mentre visualizza la tabella dei processi avviati e la mappa della memoria.

```
c
abaabaaba
p
pp  p pg                              T * 0x1000 D * 0x1000 stack
id id rp  tty  uid euid suid usage s addr  size addr  size pointer  name
 0  0  0 0000    0    0    0 00.03 R 00000 028e 00000 0000 028eb2c os32 kernel
 0  1  0 0000    0    0    0 00.09 r 0051e 000e 0052c 002d 002cf88 /bin/ccc
 1  2  0 0000   10   10   10 00.00 s 002bc 000e 002ca 002d 002cf34 /bin/aaa
 1  3  0 0000   11   11   11 00.00 s 002f7 000e 00305 002d 002cf34 /bin/bbb
ab
m
Hex mem map, blocks of 1000: 0-28f 2bc-332 51e-559
aabaab_
```

Con il comando '**x**' il ciclo termina e il kernel avvia '`/bin/init`', ma prima di farlo occorre che non ci siano altri processi in funzione, perché '**`init`**' deve assumere il ruolo di processo 1, ovvero il primo dopo il kernel.

Figura 85.9. Aspetto di os32 in funzione con il menù in evidenza, dopo aver dato il comando ‘**x**’ per avviare ‘**init**’.

```
os32 build 20AAMMGGHHmm ram 130048 Kibyte
[ata_init] ATA drive 0 size 16 Kib
[ata_init] ATA drive 1 size 16 Kib
.-------------------------------------------------------------------.
| h    show this menu                                  .--------------.|
| t    show internal timer values                      | all commands ||
| f    fork the kernel                                 | followed by  ||
| m    memory map (HEX)                                | [Enter]      ||
| p    process status list                             `--------------’|
| s    super block list                                               |
| n    list of active inodes                                          |
| 1..9 kill process  1 to 9                                           |
| A..F kill process 10 to 15                                          |
| a..c run programs ‘/bin/aaa’ to ‘/bin/ccc’ in parallel              |
| x    exit interaction with kernel and start ‘/bin/init’             |
| q    quit kernel                                                    |
`-------------------------------------------------------------------’
x
init
os32: a basic os. [Ctrl q], [Ctrl r], [Ctrl s], [Ctrl t] to change
console.
This is terminal /dev/console0
Log in as "root" or "user" with password "ciao" :-)
login:
```

85.6 Avvio e conclusione del sistema «normale»

Se non si intende operare direttamente con il kernel, come descritto nella sezione precedente, con il comando ‘**x**’ si avvia ‘**init**’.

Il programma ‘**init**’ legge il file ‘`/etc/inittab`’ e sulla base del suo contenuto, avvia uno o più processi ‘**getty**’, per la gestione dei vari terminali disponibili (si tratta comunque soltanto di console virtuali).

Il programma '**getty**' apre il terminale che gli viene indicato come opzione della chiamata (da '**init**' che lo determina in base al contenuto di '`/etc/inittab`'), facendo in modo che sia associato al descrittore zero (standard input). Quindi, dopo aver visualizzato il contenuto del file '`/etc/issue`', mostra un proprio messaggio e avvia il programma '**login**'.

Il programma '**login**' prende il posto di '**getty**' che così scompare dall'elenco dei processi. '**login**' procede chiedendo all'utente di identificarsi, utilizzando il file '`/etc/passwd`' per verificare le credenziali di accesso. Se l'identificazione ha successo, viene avviata la shell definita nel file '`/etc/passwd`' per l'utente, in modo da sostituirsi al programma '**login**', il quale scompare a sua volta dall'elenco dei processi.

Attraverso la shell è possibile interagire con il sistema operativo, secondo la modalità «normale», nei limiti delle possibilità di os32. Quando la shell termina di funzionare, '**init**' riavvia '**getty**'.

Per cambiare console virtuale si possono usare le combinazioni [*Ctrl q*], [*Ctrl r*], [*Ctrl s*] e [*Ctrl t*], ma bisogna considerare che dipende dalla configurazione del file '`/etc/inittab`' se effettivamente vengono attivate tutte queste console.

Per concludere l'attività del sistema, basta concludere il funzionamento delle varie sessioni di lavoro (la shell finisce di funzionare con il comando interno '**exit**') e non c'è bisogno di altro; pertanto, non è previsto l'uso di comandi come '**halt**' o '**shutdown**' e, d'altro canto, le operazioni di scrittura nel file system sono sincrone, in modo tale da non richiedere accorgimenti particolari per la chiusura delle attività.

Sezione 1: programmi eseguibili o comandi interni di shell

86.1 os32: aaa(1)

NOME

‘**aaa**’, ‘**bbb**’, ‘**ccc**’ - programmi elementari avviabili direttamente dal kernel

SINTASSI

```
aaa
```

```
bbb
```

```
ccc
```

DESCRIZIONE

‘**aaa**’ e ‘**bbb**’ si limitano a visualizzare una lettera, rispettivamente «a» e «b», attraverso lo standard output, a intervalli regolari. Precisamente, ‘**aaa**’ lo fa a intervalli di un secondo, mentre ‘**bbb**’ a intervalli di due secondi. Il lavoro di ‘**aaa**’ e di ‘**bbb**’ si conclude dopo l’emissione, rispettivamente, di 60 e 30 caratteri, pertanto nel giro di un minuto di tempo si esaurisce il loro compito.

Il programma ‘**ccc**’ è diverso, ma nasce per lo stesso scopo: controllare la gestione dei processi di os32. Questo programma si

limita ad avviare, ‘**aaa**’ e ‘**bbb**’, come propri processi-figli, rimanendo in funzione, senza fare nulla. Pertanto, se si usa ‘**ccc**’, il suo processo deve essere eliminato in modo esplicito, perché da solo non si concluderebbe mai.

Questi programmi sono indicati soprattutto per l’uso di os32 nella modalità interattiva che precede il funzionamento normale del sistema operativo, per la verifica della gestione dei processi.

FILE SORGENTI

‘`applic/crt0.mer.s`’ [96.1.12]

‘`applic/crt0.sep.s`’ [96.1.13]

‘`applic/aaa.c`’ [96.1.2]

‘`applic/bbb.c`’ [96.1.5]

‘`applic/ccc.c`’ [96.1.7]

86.2 os32: allocated(1)

«

NOME

‘**allocated**’ - mappa della memoria allocata

SINTASSI

```
allocated
```

DESCRIZIONE

Il programma ‘**allocated**’ si limita a leggere dal file di dispositivo ‘`/dev/kmem_map`’, producendo un elenco delle zone di memoria continue già in uso.

FILE SORGENTI

‘`applic/crt0.mer.s`’ [96.1.12]

‘`applic/crt0.sep.s`’ [96.1.13]

‘`applic/allocated.c`’ [96.1.3]

VEDERE ANCHE

mmcheck(1) [86.18].

86.3 os32: bbb(1)

Vedere *aaa(1)* [86.1].

86.4 os32: cat(1)

NOME

‘**`cat`**’ - emissione del contenuto di uno o più file attraverso lo standard output

SINTASSI

```
cat [file]...
```

DESCRIZIONE

‘**`cat`**’ legge il contenuto dei file indicati come argomento e li emette attraverso lo standard output, concatenati assieme in un unico flusso.

FILE SORGENTI

‘`applic/crt0.mer.s`’ [96.1.12]

‘`applic/crt0.sep.s`’ [96.1.13]

‘`applic/cat.c`’ [96.1.6]

VEDERE ANCHE

more(1) [86.19], *ed(1)* [86.11].

86.5 os32: ccc(1)

«

Vedere *aaa(1)* [86.1].

86.6 os32: chgrp(1)

«

NOME

‘**chgrp**’ - cambiamento del gruppo proprietario di un file

SINTASSI

```
chgrp nome_gruppo file...
```

```
chgrp gid file...
```

DESCRIZIONE

‘**chgrp**’ cambia l’utente proprietario dei file indicati. Il nuovo proprietario da attribuire può essere indicato per nome o per numero.

FILE SORGENTI

‘`applic/crt0.mer.s`’ [96.1.12]

‘`applic/crt0.sep.s`’ [96.1.13]

‘`applic/chgrp.c`’ [96.1.8]

VEDERE ANCHE

chgrp(1) [86.8].

chmod(1) [86.7].

86.7 os32: chmod(1)

«

NOME

‘**chmod**’ - cambiamento della modalità dei permessi dei file

SINTASSI

```
chmod mod_ottale file...
```

DESCRIZIONE

‘**chmod**’ cambia la modalità dei permessi associati ai file indicati, in base al numero ottale indicato come primo argomento.

NOTE

Questa versione di ‘**chmod**’ non permette di indicare la modalità dei permessi in forma simbolica.

FILE SORGENTI

‘`applic/crt0.mer.s`’ [96.1.12]

‘`applic/crt0.sep.s`’ [96.1.13]

‘`applic/chmod.c`’ [96.1.9]

VEDERE ANCHE

chown(1) [86.8].

86.8 os32: chown(1)

«

NOME

‘**chown**’ - cambiamento del proprietario di un file

SINTASSI

```
chown nome_utente file...
```

```
chown uid file...
```

DESCRIZIONE

'**chown**' cambia l'utente proprietario dei file indicati. Il nuovo proprietario da attribuire può essere indicato per nome o per numero.

FILE SORGENTI

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

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

'applic/chown.c' [96.1.10]

VEDERE ANCHE

chgrp(1) [86.6].

chmod(1) [86.7].

86.9 os32: cp(1)

«

NOME

'**cp**' - copia dei file

SINTASSI

```
cp file_orig file_nuovo...
```

```
cp file... directory_dest...
```

DESCRIZIONE

'**cp**' copia uno o più file. Se l'ultimo argomento è costituito da una directory esistente, la copia produce dei file con lo stesso nome degli originali, all'interno della directory; se l'ultimo argomento non è una directory già esistente, ci può essere un solo file da copiare, intendendo che si voglia creare una copia con quel nome specificato.

DIFETTI

Non è possibile copiare oggetti diversi dai file puri e semplici; quindi, la copia ricorsiva di una directory non è ammissibile.

FILE SORGENTI

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

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

'`applic/cp.c`' [96.1.11]

VEDERE ANCHE

touch(1) [86.26], *mkdir(1)* [86.17].

86.10 os32: date(1)

NOME

'**date**' - visualizzazione o impostazione della data e dell'ora di sistema

SINTASSI

```
date [MMGGhhmm [ [SS] AA ] ]
```

DESCRIZIONE

Se si utilizza il programma '**date**' senza argomenti, si ottiene la visualizzazione della data e dell'ora attuale del sistema operativo. Se si indica una sequenza numerica come argomento, si intende invece impostare la data e l'ora del sistema. In tal caso va indicato un numero preciso di cifre, che può essere di otto, dieci o dodici. Se si immettono otto cifre, si sta specificando il mese, il giorno, l'ora e i minuti dell'anno attuale; se si indicano dieci cifre, le ultime due rappresentano l'anno del secolo attuale; se si immettono dodici cifre, l'anno è indicato per esteso nelle ultime quattro cifre.

ESEMPI

```
# date 123123592012 [Invio]
```

Imposta la data di sistema al giorno 31 dicembre 2012, alle ore 23:59.

FILE SORGENTI

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

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

'applic/date.c' [96.1.14]

VEDERE ANCHE

time(2) [87.59], *stime(2)* [87.59].

86.11 os32: ed(1)

«

NOME

‘**ed**’ - creazione e modifica di file di testo

SINTASSI

```
ed
```

DESCRIZIONE

‘**ed**’ è un programma di creazione e modifica di file di testo che consente di operare su una riga alla volta.

‘**ed**’ opera in due modalità di funzionamento: comando e inserimento. All’avvio, ‘**ed**’ si trova in modalità di comando, durante la quale ciò che si inserisce attraverso lo standard input viene interpretato come un comando da eseguire. Per esempio, il comando ‘**1i**’ richiede di passare alla modalità di inserimento, immettendo delle righe a partire dalla prima posizione, spostando quelle presenti in basso. Quando ci si trova in modalità di inserimento, per poter passare alla modalità di comando si introduce un punto isolato, all’inizio di una nuova riga.

Per il momento, in questa pagina di manuale, si omette la descrizione completa dell’utilizzo di ‘**ed**’.

DIFETTI

Il file che si intende elaborare con ‘**ed**’ viene caricato o creato completamente nella memoria centrale. Dal momento che os32 consente a ogni processo di gestire una quantità limitata di memoria, si può lavorare soltanto con file di dimensioni ridotte.

AUTORI

Questa edizione di ‘**ed**’ è stata scritta originariamente da David I. Bell, per ‘**sash**’ (una shell che include varie funzionalità, da compilare in modo statico). Successivamente, il codice è stato estrapolato da ‘**sash**’ e reso indipendente, per gli scopi del sistema operativo ELKS (una versione a 16 bit di Linux). Dalla versione estratta per ELKS è stata ottenuta quella di os16, con delle modifiche apportate da Daniele Giacomini, tra cui risulta particolarmente evidente il cambiamento dello stile di impaginazione del codice. Dalla versione per os16 è stata ottenuta quella per os32, senza ulteriori modifiche.

FILE SORGENTI

‘`applic/crt0.mer.s`’ [96.1.12]

‘`applic/crt0.sep.s`’ [96.1.13]

‘`applic/ed.c`’ [96.1.15]

VEDERE ANCHE

shell(1) [86.24].

86.12 os32: kill(1)

«

NOME

‘**kill**’ - invio di un segnale ai processi

SINTASSI

```
kill -s nome_segnale pid...
```

```
kill -l
```

DESCRIZIONE

Il programma '**`kill`**' consente di inviare un segnale, indicato per nome, a uno o più processi, specificati per numero.

OPZIONI

Opzione	Descrizione
`-l`	Mostra l'elenco dei nomi dei segnali disponibili.
`-s` ***nome_segnale***	Specifica il nome del segnale da inviare ai processi.

NOTE

Non è possibile indicare il segnale per numero, perché lo standard definisce i nomi di un insieme di segnali necessari, ma non stabilisce il numero, il quale può essere attribuito liberamente in fase realizzativa.

DIFETTI

os32 non consente ai processi di attribuire azioni alternative ai segnali; pertanto, si possono ottenere solo quelle predefinite. Tutto quello che si può fare è, eventualmente, bloccare i segnali, esclusi però quelli che non sono mascherabili per loro natura.

FILE SORGENTI

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

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

'`applic/kill.c`' [96.1.20]

VEDERE ANCHE

kill(2) [87.29], *signal(2)* [87.52].

86.13 os32: ln(1)

«

NOME

‘**ln**’ - collegamento dei file

SINTASSI

```
ln file_orig file_nuovo...
```

```
ln file... directory_dest...
```

DESCRIZIONE

‘**ln**’ crea il collegamento fisico di uno o più file. Se l’ultimo argomento è costituito da una directory esistente, si producono collegamenti con gli stessi nomi degli originali, all’interno della directory; se l’ultimo argomento non è una directory già esistente, ci può essere un solo file da collegare, intendendo che si voglia creare un collegamento con quel nome specificato.

Essendo disponibile soltanto la creazione di collegamenti fisici, questi collegamenti possono essere collocati soltanto all’interno del file system di quelli originali, senza contare eventuali innesti ulteriori.

DIFETTI

Non è possibile creare dei collegamenti simbolici, perché os32 non sa gestirli.

FILE SORGENTI

‘`applic/crt0.mer.s`’ [96.1.12]

‘`applic/crt0.sep.s`’ [96.1.13]

‘`applic/ln.c`’ [96.1.21]

VEDERE ANCHE

cp(1) [86.9], *link(2)* [87.30].

86.14 os32: login(1)

NOME

‘**`login`**’ - inizio di una sessione presso un terminale

SINTASSI

```
login
```

DESCRIZIONE

‘**`login`**’ richiede l’inserimento di un nominativo-utente e di una parola d’ordine. Questa coppia viene verificata consultando il file ‘`/etc/passwd`’ e se coincide: vengono cambiati i permessi e la proprietà del file di dispositivo del terminale di controllo; viene cambiata la directory corrente in modo da farla coincidere con la directory personale dell’utente; viene avviata la shell, indicata sempre nel file ‘`/etc/passwd`’ per quel tale utente, con i privilegi di questo. La shell, avviata così, va a rimpiazzare il processo di ‘**`login`**’.

Il programma ‘**`login`**’ è fatto per essere avviato da ‘**`getty`**’, non avendo altri utilizzi pratici.

FILE

File	Descrizione
‘/etc/passwd’	Contiene l’elenco degli utenti, con l’associazione al numero UID, alla parola d’ordine necessaria per accedere, alla shell dell’utente. Le altre informazioni eventuali contenute nel file, non sono usate da ‘**login**’.

FILE SORGENTI

‘applic/crt0.mer.s’ [96.1.12]

‘applic/crt0.sep.s’ [96.1.13]

‘applic/login.c’ [96.1.22]

VEDERE ANCHE

getty(8) [92.2], *console(4)* [89.1].

86.15 os32: ls(1)

«

NOME

‘**ls**’ - elenco del contenuto delle directory

SINTASSI

```
ls [opzioni] [file]...
```

DESCRIZIONE

‘**ls**’ elenca i file e le directory indicati come argomenti della chiamata. Se non vengono indicati file o directory da visualizzare, si ottiene l’elenco del contenuto della directory corrente; inoltre,

questa realizzazione particolare di '**ls**', se si indica come argomento solo una directory, ne mostra il contenuto, altrimenti, se gli argomenti sono più di uno, mostra solo i nomi richiesti, eventualmente con le rispettive caratteristiche se è stata usata l'opzione '**-l**'.

OPZIONI

Opzione	Descrizione
`-a`	Quando si richiede di mostrare il contenuto di una directory (quella corrente o quella specificata espressamente come primo e unico argomento), con questa opzione si ottiene la visualizzazione anche dei nomi che iniziano con un punto, inclusi '.' e '..'.
`-l`	Con questa opzione si ottiene la visualizzazione di più informazioni sui file e sulle directory elencati.

NOTE

Dal momento che os32 non considera i gruppi, quando si usa l'opzione '**-l**', il nome del gruppo a cui appartiene un file o una directory, non viene visualizzato.

FILE SORGENTI

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

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

'`applic/ls.c`' [96.1.23]

86.16 os32: man(1)

«

NOME

‘**man**’ - visualizzazione delle pagine di manuale

SINTASSI

```
man [sezione] pagina
```

DESCRIZIONE

‘**man**’ visualizza la pagina di manuale indicata come argomento, consentendone lo scorrimento in avanti. La «pagina» viene cercata tra le sezioni, a partire dalla prima. In caso di omonimie tra sezioni differenti, si può specificare il numero della sezione prima del nome della pagina.

Le pagine di manuale di os32 sono semplicemente dei file di testo, collocati nella directory ‘`/usr/share/man/`’, con nomi del tipo ‘***pagina*** **.** ***n***’, dove ***n*** è il numero della sezione.

FILE SORGENTI

‘`applic/crt0.mer.s`’ [96.1.12]

‘`applic/crt0.sep.s`’ [96.1.13]

‘`applic/man.c`’ [96.1.24]

VEDERE ANCHE

cat(1) [86.4].

86.17 os32: mkdir(1)

NOME

‘**mkdir**’ - creazione di directory

SINTASSI

```
mkdir [-p] [-m mod_ottale] [directory] ...
```

DESCRIZIONE

‘**mkdir**’ crea una o più directory, corrispondenti ai nomi che costituiscono gli argomenti.

OPZIONI

Opzione	Descrizione
`-p`	Se la directory che si vuole creare, può richiedere prima la creazione di altre directory, con questa opzione (*parents*) si generano tutte le directory genitrici necessarie, purché quei nomi non siano già usati per dei file.
`-m` ***mod_ottale***	Quando si crea una directory, senza specificare questa opzione, si ottengono i permessi 0777_8 meno quanto contenuto nella maschera di creazione dei file e delle directory. Con l’opzione ‘**-m**’ si vanno invece a specificare i permessi desiderati in modo esplicito.

FILE SORGENTI

‘`applic/crt0.mer.s`’ [96.1.12]

‘`applic/crt0.sep.s`’ [96.1.13]

‘applic/mkdir.c’ [96.1.25]

VEDERE ANCHE

mkdir(2) [87.34], *rmdir(2)* [87.41].

86.18 os32: mmcheck(1)

«

NOME

‘**mmcheck**’ - verifica della coerenza tra processi e mappa della memoria allocata

SINTASSI

```
mmcheck
```

DESCRIZIONE

Il programma ‘**mmcheck**’ si limita a leggere dai file di dispositivo ‘/dev/kmem_map’ e ‘/dev/kmem_ps’, per verificare che le informazioni sull’allocazione della memoria relative ai processi in corso siano coerenti con la mappa effettiva.

FILE SORGENTI

‘applic/crt0.mer.s’ [96.1.12]

‘applic/crt0.sep.s’ [96.1.13]

‘applic/mmcheck.c’ [96.1.26]

VEDERE ANCHE

allocated(1) [86.2].

86.19 os32: more(1)

«

NOME

‘**more**’ - visualizzazione di file sullo schermo, permettendo il controllo dello scorrimento dei dati, ma in un solo verso

SINTASSI

```
more file...
```

DESCRIZIONE

‘**more**’ visualizza i file indicati come argomenti della chiamata, sospendendo lo scorrimento del testo dopo alcune righe, consentendo all’utente di decidere come proseguire.

COMANDI

Quando ‘**more**’ sospende lo scorrimento del testo, è possibile introdurre un comando, composto da un solo carattere seguito da [*Invio*], tenendo conto che ciò che non è previsto fa comunque proseguire lo scorrimento:

Comando	Descrizione
`n`	si richiede di saltare al file successivo, ammesso che ce ne sia un altro;
`q`	si richiede di interrompere lo scorrimento e di concludere il funzionamento del programma;
`Spazio`	si richiede di proseguire nella visualizzazione progressiva dei file.

FILE SORGENTI

‘`applic/crt0.mer.s`’ [96.1.12]

‘applic/crt0.sep.s’ [96.1.13]

‘applic/more.c’ [96.1.27]

VEDERE ANCHE

cat(1) [86.4].

86.20 os32: nc(1)

«

NOME

‘**nc**’ - gestione di connessioni arbitrarie di tipo UDP o TCP

SINTASSI

```
nc [-u] [-l] indirizzo porta
```

DESCRIZIONE

Il programma ‘**nc**’ (noto come «netcat») consente di gestire delle connessioni UDP o TCP, utilizzando il flusso di standard input in trasmissione ed emettendo il flusso ricevuto dalla controparte attraverso lo standard output.

OPZIONI

Opzione	Descrizione
`-u`	Utilizza il protocollo UDP invece di TCP.
`-l`	Si mette in ascolto e attende una richiesta di connessione.

Quando non si utilizza l’opzione ‘**-l**’, l’indirizzo e la porta indicati nella riga di comando, rappresentano la controparte che si intende contattare; se invece si usa l’opzione ‘**-l**’, si tratta di indirizzo e porta locali.

FILE SORGENTI

‘`applic/crt0.mer.s`’ [96.1.12]

‘`applic/crt0.sep.s`’ [96.1.13]

‘`applic/nc.c`’ [96.1.29]

VEDERE ANCHE

arp(8) [92.1], *ipconfig(8)* [92.5], *route(8)* [92.9], *http(8)* [92.3], *ping(8)* [92.8].

86.21 os32: ps(1)

NOME

‘**ps**’ - visualizzazione dello stato dei processi elaborativi

SINTASSI

```
ps [-u|-g]
```

DESCRIZIONE

‘**ps**’ visualizza l’elenco dei processi, con le informazioni disponibili sul loro stato. L’elenco è provvisto di un’intestazione, come si vede nell’esempio seguente:

```
pp  p pg                              T * 0x1000 D * 0x1000 stack
id id rp  tty  uid euid suid usage s addr  size addr  size usage    name
 0  0  0 0000    0    0    0 00.02 r 00000 028e 00000 0000   0% os32 kernel
 0  1  0 0000    0    0    0 00.00 s 0051e 003a 00000 0000   9% /bin/init
 1  2  2 0500 1001 1001 1001 00.00 s 0033d 003a 00000 0000  26% /bin/shell
 1  3  3 0501    0    0    0 00.00 s 0028f 003a 00000 0000  36% /bin/login
 1  4  4 0502    0    0    0 00.00 s 002c9 003a 00000 0000  36% /bin/login
 1  5  5 0503    0    0    0 00.00 s 00303 003a 00000 0000  36% /bin/login
 2  6  2 0500 1001 1001 1001 00.00 R 003b1 003a 00000 0000  20% /bin/ps
```

La prima colonna, con la sigla «ppid», ovvero *parent pid*, riporta il numero del processo genitore; la seconda, con la sigla «pid», *process id*, indica il numero del processo preso in considerazione; la terza, con la sigla «pgrp», *process group*, indica il gruppo a cui appartiene il processo; la quarta colonna, «tty», indica il terminale associato, ammesso che ci sia, come numero di dispositivo, ma in base sedici. Le colonne «uid», «euid» e «suid», riguardano l'identità dell'utente, per conto del quale sono in funzione i processi, rappresentando, nell'ordine, l'identità reale (*real user id*), quella efficace (*effective user id*) e quella salvata precedentemente (*saved user id*).

La colonna «usage» indica il tempo di utilizzo della CPU; la colonna «s» indica lo stato del processo, il cui significato può essere interpretato con l'aiuto della tabella successiva:

Lettera	Significato	Descrizione
R	*running*	in corso di esecuzione
r	*ready*	pronto per essere messo in esecuzione
s	*sleeping*	in attesa
z	*zombie*	terminato e non più in memoria, per il quale si attende di passare lo stato di uscita al processo genitore.

Le prime due colonne «addr» e «size» indicano l'indirizzo iniziale e l'estensione dell'area codice del processo, in memoria (*text*); le altre due successive indicano l'indirizzo iniziale e l'estensione dell'area dati del processo, in memoria (*data*). Gli indirizzi e le dimensioni delle aree di memoria utilizzate appaiono divisi per 1000_{16}, ovvero sono multipli di 4096 byte. La colonna «stack

usage» indica la percentuale di utilizzo dello spazio attribuito alla pila dei dati (*stack pointer*).

L'ultima colonna indica il nome del programma, assieme al suo percorso, con il quale il processo è stato avviato.

OPZIONI

Opzione	Descrizione
`-u`	Nell'elenco mostra i numeri UID relativi al processo. Questa opzione è predefinita.
`-g`	Nell'elenco, al posto dei numeri UID, mostra i numeri GID.

NOTE

L'elenco dei processi include anche il kernel, il quale occupa correttamente la prima posizione (processo zero).

FILE

'**`ps`**' trae le informazioni sullo stato dei processi da un file di dispositivo speciale: '`/dev/kmem_ps`'.

FILE SORGENTI

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

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

'`applic/ps.c`' [96.1.31]

86.22 os32: rm(1)

«

NOME

'**`rm`**' - cancellazione di file

SINTASSI

```
rm file...
```

DESCRIZIONE

‘**rm**’ consente di cancellare i file indicati come argomento.

DIFETTI

Non è possibile eseguire la cancellazione ricorsiva di una directory.

FILE SORGENTI

‘applic/crt0.mer.s’ [96.1.12]

‘applic/crt0.sep.s’ [96.1.13]

‘applic/rm.c’ [96.1.32]

VEDERE ANCHE

unlink(2) [87.62].

86.23 os32: rmdir(1)

«

NOME

‘**rmdir**’ - cancellazione di directory vuote

SINTASSI

```
rmdir directory...
```

DESCRIZIONE

Il programma ‘**rmdir**’ consente di cancellare le directory indicate come argomento, purché queste siano vuote.

FILE SORGENTI

‘`applic/crt0.mer.s`’ [96.1.12]

‘`applic/crt0.sep.s`’ [96.1.13]

‘`applic/rmdir.c`’ [96.1.33]

VEDERE ANCHE

unlink(2) [87.62].

86.24 os32: shell(1)

NOME

‘**`shell`**’ - interprete dei comandi

SINTASSI

```
shell
```

DESCRIZIONE

‘**`shell`**’ è l’interprete dei comandi di os32. Di norma viene avviato da ‘**`login`**’, in base alla configurazione contenuta nel file ‘`/etc/passwd`’.

‘**`shell`**’ interpreta i comandi inseriti; se si tratta di un comando interno lo esegue direttamente, altrimenti cerca e avvia un programma con il nome corrispondente, rimanendo in attesa fino alla conclusione del processo relativo, per riprendere poi il controllo.

DIFETTI

L’interpretazione della riga di comando è letterale, pertanto non c’è alcuna espansione di caratteri speciali, variabili di ambiente o altro; inoltre, non è possibile eseguire script.

A volte, quando un processo avviato da '**shell**' termina di funzionare, il processo di '**shell**' non viene risvegliato correttamente, rendendo inutilizzabile il terminale. Per ovviare all'inconveniente, si può premere la combinazione [*Ctrl c*], con la quale viene inviato il segnale '**SIGINT**' a tutti i processi del gruppo associato al terminale.

Anche il fatto che un segnale generato con una combinazione di tasti si trasmetta a tutti i processi del gruppo associato al terminale è un'anomalia, tuttavia fa parte delle particolarità dovute alla semplificazione di os32.

FILE SORGENTI

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

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

'`applic/shell.c`' [96.1.35]

VEDERE ANCHE

input_line(3) [88.68].

86.25 os32: t_(1)

«

NOME

'**t_**...' - programmi di prova

DESCRIZIONE

I programmi con prefisso '**t_**...' sono delle prove per verificare alcune funzionalità di os32.

86.26 os32: touch(1)

«

NOME

‘**touch**’ - creazione di un file vuoto oppure aggiornamento della data di modifica

SINTASSI

```
touch file...
```

DESCRIZIONE

‘**touch**’ crea dei file vuoti, se quelli indicati come argomento non sono esistenti; altrimenti, aggiorna le date di accesso e di modifica, sulla base dello stato dell’orologio di sistema.

DIFETTI

Non è possibile attribuire una data arbitraria; inoltre, a causa della limitazione del tipo di file system utilizzato, non è possibile distinguere tra date di accesso e modifica dei file.

FILE SORGENTI

‘`applic/crt0.mer.s`’ [96.1.12]

‘`applic/crt0.sep.s`’ [96.1.13]

‘`applic/touch.c`’ [96.1.51]

86.27 os32: tty(1)

«

NOME

‘**tty**’ - nome del file di dispositivo del terminale associato allo standard input

SINTASSI

```
tty
```

DESCRIZIONE

Il programma '**tty**' individua il dispositivo del terminale associato allo standard input e lo traduce in un percorso che descrive il file di dispositivo corrispondente (ovvero il file di dispositivo che dovrebbe corrispondergli)

FILE SORGENTI

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

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

'applic/tty.c' [96.1.52]

86.28 os32: yes(1)

«

NOME

'**yes**' - visualizzazione ripetuta di un testo

SINTASSI

```
yes [testo]
```

DESCRIZIONE

Il programma '**yes**' emette sullo schermo, all'infinito, il contenuto del testo fornito come argomento, oppure la lettera '**y**'. In tutti i casi, il testo o la lettera sono seguiti dal codice di interruzione di riga (*new line*).

FILE SORGENTI

‘applic/crt0.mer.s’ [96.1.12]

‘applic/crt0.sep.s’ [96.1.13]

‘applic/yes.c’ [96.1.54]

87.1 os32: _Exit(2)

Vedere *_exit(2)* [87.2].

87.2 os32: _exit(2)

NOME

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

SINTASSI

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

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

DESCRIZIONE

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

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

VALORE RESTITUITO

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

FILE SORGENTI

‘`lib/unistd.h`’ [95.30]

‘`lib/unistd/_exit.c`’ [95.30.1]

‘`lib/stdlib.h`’ [95.19]

‘`lib/stdlib/_Exit.c`’ [95.19.1]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

‘`kernel/lib_s/s__exit.c`’ [94.8.1]

VEDERE ANCHE

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

87.3 os32: accept(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

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

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

‘kernel/lib_s/s_accept.c’ [94.8.2]

VEDERE ANCHE

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

87.4 os32: bind(2)

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

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

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

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

VEDERE ANCHE

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

87.5 os32: brk(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

VALORE RESTITUITO

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

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

ERRORI

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

NOTE

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

FILE SORGENTI

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

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

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

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

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

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

‘kernel/lib_s/s_brk.c’ [94.8.4]

‘kernel/lib_s/s_sbrk.c’ [94.8.33]

VEDERE ANCHE

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

87.6 os32: chdir(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘`lib/unistd.h`’ [95.30]

‘`lib/unistd/chdir.c`’ [95.30.4]

‘`lib/unistd/fchdir.c`’ [95.30.16]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

‘`kernel/lib_s/s_chdir.c`’ [94.8.5]

VEDERE ANCHE

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

87.7 os32: chmod(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘`lib/sys/stat.h`’ [95.25]

‘`lib/sys/stat/chmod.c`’ [95.25.1]

‘`lib/sys/stat/fchmod.c`’ [95.25.2]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

‘`kernel/lib_s/s_chmod.c`’ [94.8.6]

‘`kernel/lib_s/s_fchmod.c`’ [94.8.13]

VEDERE ANCHE

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

87.8 os32: chown(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

ERRORI

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

DIFETTI

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

FILE SORGENTI

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

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

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

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

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

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

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

VEDERE ANCHE

chmod(2) [87.7].

87.9 os32: clock(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

FILE SORGENTI

‘`lib/time.h`’ [95.29]

‘`lib/time/clock.c`’ [95.29.2]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

‘`kernel/lib_s/s_clock.c`’ [94.8.8]

VEDERE ANCHE

time(2) [87.59].

87.10 os32: close(2)

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

'lib/unistd.h' [95.30]

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

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

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

‘kernel/proc/sysroutine.c’ [94.14.28]

‘kernel/lib_s/s_close.c’ [94.8.9]

VEDERE ANCHE

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

87.11 os32: connect(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

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

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

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

VEDERE ANCHE

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

87.12 os32: dup(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘lib/unistd.h’ [95.30]

‘lib/unistd/dup.c’ [95.30.7]

‘lib/unistd/dup2.c’ [95.30.8]

‘lib/sys/os32/sys.s’ [95.21.7]

‘kernel/ibm_i386/isr.s’ [94.6.21]

‘kernel/proc/sysroutine.c’ [94.14.28]

‘kernel/fs/fd_dup.c’ [94.5.1]

‘kernel/lib_s/s_dup.c’ [94.8.11]

‘kernel/lib_s/s_dup2.c’ [94.8.12]

VEDERE ANCHE

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

87.13 os32: dup2(2)

Vedere *dup(2)* [87.12].

87.14 os32: execve(2)

NOME

‘**execve**’ - esecuzione di un file

SINTASSI

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

DESCRIZIONE

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

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

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

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

VALORE RESTITUITO

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

ERRORI

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

DIFETTI

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

FILE SORGENTI

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

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

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

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

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

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

VEDERE ANCHE

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

87.15 os32: fchdir(2)

«

Vedere *chdir(2)* [87.6].

87.16 os32: fchmod(2)

«

Vedere *chmod(2)* [87.7].

87.17 os32: fchown(2)

«

Vedere *chown(2)* [87.8].

87.18 os32: fcntl(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

VALORE RESTITUITO

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

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

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

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

ERRORI

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

FILE SORGENTI

‘`lib/fcntl.h`’ [95.6]

‘`lib/fcntl/fcntl.c`’ [95.6.2]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘kernel/ibm_i386/isr.s’ [94.6.21]

‘kernel/proc/sysroutine.c’ [94.14.28]

‘kernel/lib_s/s_fcntl.c’ [94.8.15]

VEDERE ANCHE

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

87.19 os32: fork(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘`lib/unistd.h`’ [95.30]

‘`lib/unistd/fork.c`’ [95.30.18]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

‘`kernel/lib_s/s_fork.c`’ [94.8.16]

VEDERE ANCHE

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

87.20 os32: fstat(2)

Vedere *stat(2)* [87.55].

87.21 os32: getcwd(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

ERRORI

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

DIFETTI

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

FILE SORGENTI

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

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

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

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

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

VEDERE ANCHE

chdir(2) [87.6].

87.22 os32: getgid(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

DIFETTI

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

FILE SORGENTI

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

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

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

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

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

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

VEDERE ANCHE

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

87.23 os32: geteuid(2)

«

Vedere *getuid(2)* [87.27].

87.24 os32: getpgrp(2)

«

Vedere *getpid(2)* [87.25].

87.25 os32: getpid(2)

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

DIFETTI

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

FILE SORGENTI

‘`lib/unistd.h`’ [95.30]

‘lib/unistd/getpid.c’ [95.30.25]

‘lib/unistd/getppid.c’ [95.30.26]

‘lib/unistd/getpgrp.c’ [95.30.24]

‘lib/sys/os32/sys.s’ [95.21.7]

‘kernel/ibm_i386/isr.s’ [94.6.21]

‘kernel/proc/sysroutine.c’ [94.14.28]

VEDERE ANCHE

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

87.26 os32: getppid(2)

«

Vedere *getpid(2)* [87.25].

87.27 os32: getuid(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

DIFETTI

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

FILE SORGENTI

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

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

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

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

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

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

VEDERE ANCHE

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

87.28 os32: ipconfig(2)

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

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

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

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

VEDERE ANCHE

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

87.29 os32: kill(2)

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘`lib/sys/types.h`’ [95.26]

‘`lib/signal.h`’ [95.17]

‘`lib/signal/kill.c`’ [95.17.2]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

‘`kernel/lib_s/s_kill.c`’ [94.8.19]

VEDERE ANCHE

signal(2) [87.52].

87.30 os32: link(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘`lib/unistd.h`’ [95.30]

‘`lib/unistd/link.c`’ [95.30.29]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

‘`kernel/lib_s/s_link.c`’ [94.8.20]

VEDERE ANCHE

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

87.31 os32: listen(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘lib/sys/socket.h’ [95.23]

‘lib/sys/socket/listen.c’ [95.23.4]

‘kernel/lib_s/s_listen.c’ [94.8.21]

VEDERE ANCHE

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

87.32 os32: longjmp(2)

«

Vedere *setjmp(2)* [87.49].

87.33 os32: lseek(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

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

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

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

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

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

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

VEDERE ANCHE

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

87.34 os32: mkdir(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘`lib/sys/stat.h`’ [95.25]

‘`lib/sys/stat/mkdir.c`’ [95.25.4]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

‘`kernel/lib_s/s_mkdir.c`’ [94.8.24]

VEDERE ANCHE

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

87.35 os32: mknod(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘`lib/sys/stat.h`’ [95.25]

‘`lib/sys/stat/mknod.c`’ [95.25.5]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

‘`kernel/lib_s/s_mknod.c`’ [94.8.25]

VEDERE ANCHE

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

87.36 os32: mount(2)

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘`lib/sys/os32.h`’ [95.21]

‘`lib/sys/os32/mount.c`’ [95.21.3]

‘`lib/sys/os32/umount.c`’ [95.21.8]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

‘kernel/lib_s/s_mount.c’ [94.8.26]

‘kernel/lib_s/s_umount.c’ [94.8.47]

VEDERE ANCHE

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

87.37 os32: open(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

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

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

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

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

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

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

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

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

VEDERE ANCHE

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

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

87.38 os32: pipe(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

'lib/unistd.h' [95.30]

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

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

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

VEDERE ANCHE

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

87.39 os32: read(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

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

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

‘lib/sys/os32/sys.s’ [95.21.7]

‘kernel/lib_s/s_read.c’ [94.8.29]

VEDERE ANCHE

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

87.40 os32: recvfrom(2)

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘lib/sys/socket.h’ [95.23]

‘lib/sys/socket/recvfrom.c’ [95.23.5]

‘kernel/lib_s/s_recvfrom.c’ [94.8.30]

VEDERE ANCHE

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

87.41 os32: rmdir(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

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

‘lib/unistd/rmdir.c’ [95.30.33]

‘lib/sys/os32/sys.s’ [95.21.7]

‘kernel/lib_s/s_unlink.c’ [94.8.48]

VEDERE ANCHE

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

87.42 os32: routeadd(2)

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

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

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

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

VEDERE ANCHE

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

87.43 os32: routedel(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

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

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

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

VEDERE ANCHE

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

87.44 os32: sbrk(2)

«

Vedere *brk(2)* [87.5].

87.45 os32: send(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘`lib/sys/socket.h`’ [95.23]

‘`lib/sys/socket/send.c`’ [95.23.6]

‘`kernel/lib_s/s_send.c`’ [94.8.34]

VEDERE ANCHE

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

87.46 os32: setegid(2)

«

Vedere *setgid(2)* [87.48].

87.47 os32: seteuid(2)

«

Vedere *setuid(2)* [87.51].

87.48 os32: setgid(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

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

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

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

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

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

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

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

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

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

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

VEDERE ANCHE

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

87.49 os32: setjmp(2)

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

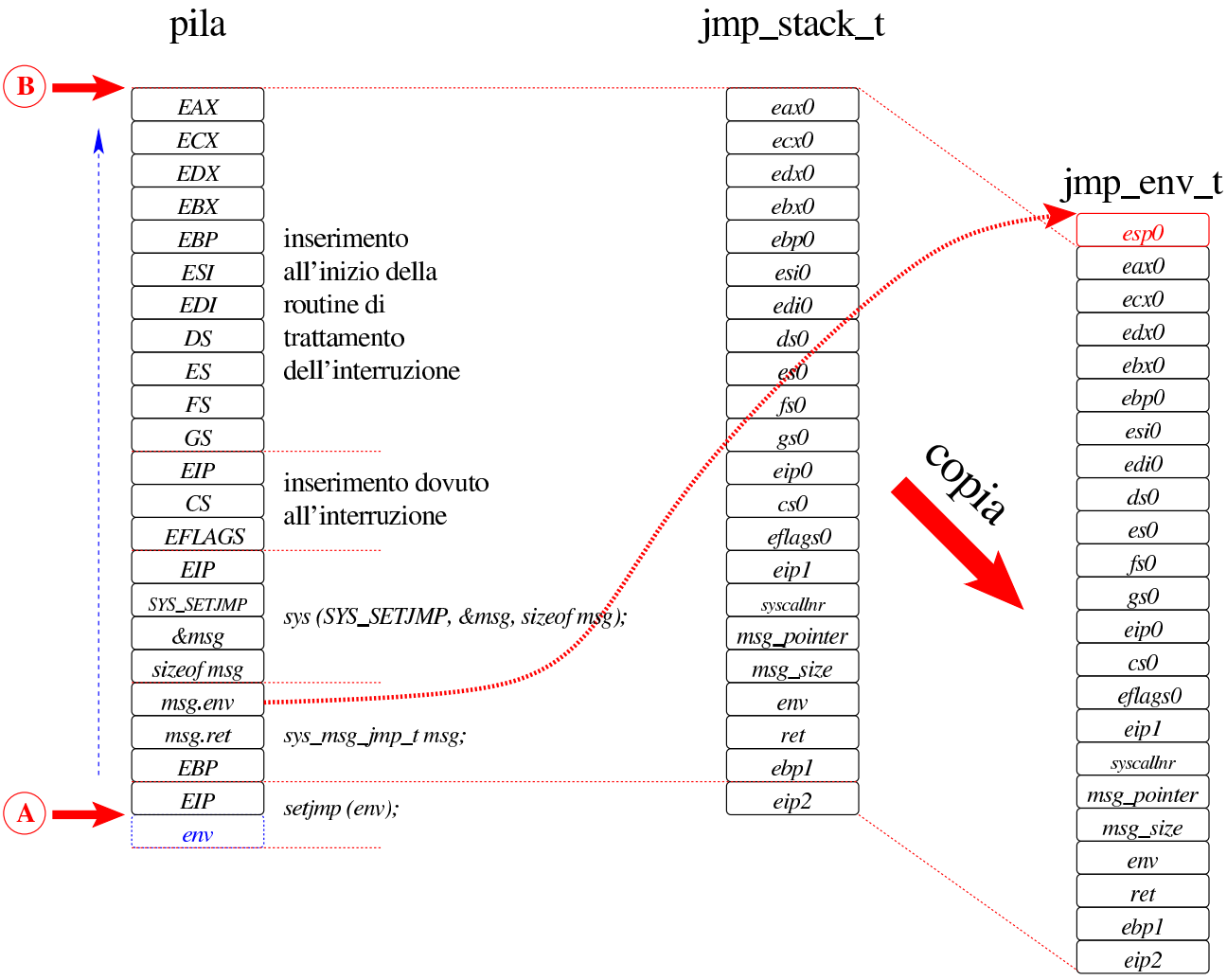
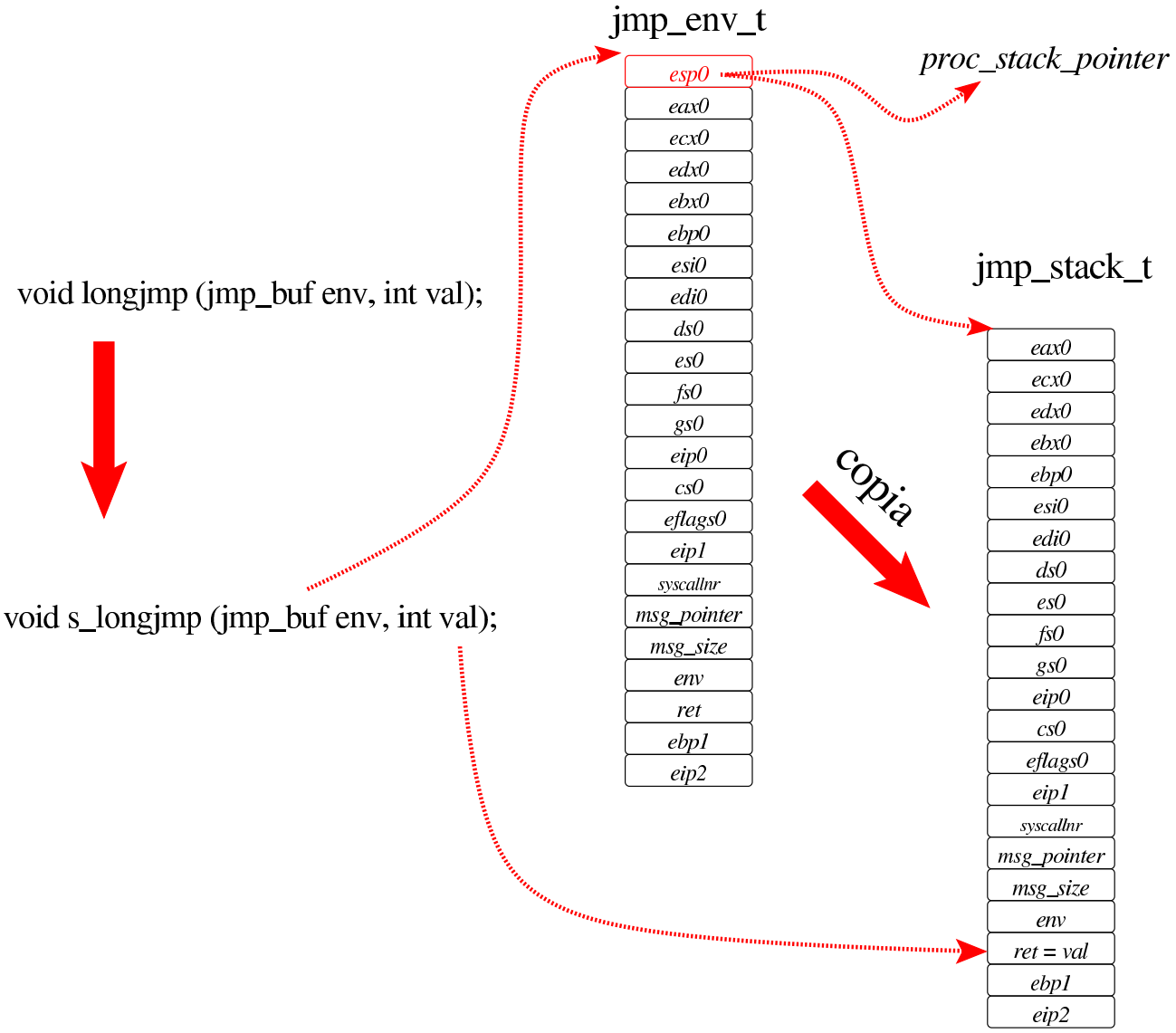


Figura 87.53. La chiamata di ***longjmp()*** ricostruisce la vecchia pila di ***setjmp()***, nella posizione in cui si trovava, ricollocando l'indice della pila e modificando il valore che poi ***setjmp()*** rediviva va a restituire.



VALORE RESTITUITO

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

NOTE

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

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

FILE SORGENTI

‘`lib/setjmp.h`’ [95.16]

‘`lib/setjmp/setjmp.s`’ [95.16.2]

‘`lib/setjmp/longjmp.c`’ [95.16.1]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

‘`kernel/lib_s/s_setjmp.c`’ [94.8.38]

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

VEDERE ANCHE

signal(2) [87.52].

87.50 os32: setpgrp(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

La funzione termina sempre con successo e restituisce sempre zero.

FILE SORGENTI

‘`lib/unistd.h`’ [95.30]

‘`lib/unistd/setpgrp.c`’ [95.30.38]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

VEDERE ANCHE

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

87.51 os32: setuid(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

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

‘lib/unistd/setuid.c’ [95.30.39]

‘lib/unistd/seteuid.c’ [95.30.36]

‘lib/sys/os32/sys.s’ [95.21.7]

‘kernel/ibm_i386/isr.s’ [94.6.21]

‘kernel/proc/sysroutine.c’ [94.14.28]

‘kernel/lib_s/s_setuid.c’ [94.8.39]

‘kernel/lib_s/s_seteuid.c’ [94.8.36]

VEDERE ANCHE

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

VEDERE ANCHE

getuid(2) [87.51].

87.52 os32: signal(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

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

VALORE RESTITUITO

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

ERRORI

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

NOTE

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

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

FILE SORGENTI

‘`lib/signal.h`’ [95.17]

‘`lib/signal/signal.c`’ [95.17.3]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

‘`kernel/lib_s/s_signal.c`’ [94.8.40]

VEDERE ANCHE

kill(2) [87.29].

87.53 os32: sleep(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

FILE SORGENTI

‘`lib/unistd.h`’ [95.30]

‘`lib/unistd/sleep.c`’ [95.30.40]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

VEDERE ANCHE

signal(2) [87.52].

87.54 os32: socket(2)

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

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

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

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

VEDERE ANCHE

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

87.55 os32: stat(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘`lib/sys/stat.h`’ [95.25]

‘`lib/sys/stat/stat.c`’ [95.25.6]

‘`lib/sys/stat/fstat.c`’ [95.25.3]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

‘`kernel/lib_s/s_stat.c`’ [94.8.42]

‘`kernel/lib_s/s_fstat.c`’ [94.8.17]

VEDERE ANCHE

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

87.56 os32: sys(2)

«

NOME

‘**sys**’ - chiamata di sistema

SINTASSI

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

DESCRIZIONE

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

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

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

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

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

FILE SORGENTI

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

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

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

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

87.57 os32: stime(2)

«

Vedere *time(2)* [87.59].

87.58 os32: tcgetattr(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

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

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

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

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

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

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

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

VALORE RESTITUITO

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

ERRORI

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

DIFETTI

È disponibile soltanto una modalità canonica di funzionamento del terminale.

FILE SORGENTI

‘`lib/termios.h`’ [95.28]

‘`lib/termios/tcgetattr.c`’ [95.28.1]

‘`lib/termios/tcsetattr.c`’ [95.28.2]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

‘`kernel/lib_s.h`’ [94.8]

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

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

VEDERE ANCHE

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

87.59 os32: time(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

VALORE RESTITUITO

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

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

DIFETTI

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

FILE SORGENTI

‘`lib/time.h`’ [95.29]

‘`lib/time/time.c`’ [95.29.6]

‘`lib/time/stime.c`’ [95.29.5]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

‘`kernel/lib_k.h`’ [94.7]

‘`kernel/lib_s/s_time.c`’ [94.8.46]

‘`kernel/lib_s/s_stime.c`’ [94.8.43]

VEDERE ANCHE

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

87.60 os32: umask(2)

«

NOME

‘**`umask`**’ - maschera dei permessi

SINTASSI

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

DESCRIZIONE

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

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

VALORE RESTITUITO

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

FILE SORGENTI

‘`lib/sys/stat.h`’ [95.25]

‘`lib/sys/stat/umask.c`’ [95.25.7]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

VEDERE ANCHE

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

87.61 os32: umount(2)

«

Vedere *mount(2)* [87.36].

87.62 os32: unlink(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘lib/unistd.h’ [95.30]

‘lib/unistd/unlink.c’ [95.30.42]

‘lib/sys/os32/sys.s’ [95.21.7]

‘kernel/ibm_i386/isr.s’ [94.6.21]

‘kernel/proc/sysroutine.c’ [94.14.28]

‘kernel/lib_s/s_unlink.c’ [94.8.48]

VEDERE ANCHE

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

87.63 os32: wait(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

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

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

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

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

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

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

VEDERE ANCHE

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

87.64 os32: write(2)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘`lib/unistd.h`’ [95.30]

‘`lib/unistd/write.c`’ [95.30.43]

‘`lib/sys/os32/sys.s`’ [95.21.7]

‘`kernel/ibm_i386/isr.s`’ [94.6.21]

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

‘`kernel/lib_s/s_write.c`’ [94.8.50]

VEDERE ANCHE

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

87.65 os32: z(2)

«

NOME

‘**z**_...’ - funzioni provvisorie

SINTASSI

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

DESCRIZIONE

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

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

FILE SORGENTI

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

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

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

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

VEDERE ANCHE

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

87.66 os32: z_perror(2)

«

Vedere *z(2)* [87.65].

87.67 os32: z_printf(2)

«

Vedere *z(2)* [87.65].

87.68 os32: z_vprintf(2)

«

Vedere *z(2)* [87.65].

88.1 os32: _gcc(3)

NOME

'**_gcc**' - libreria per il compilatore

DESCRIZIONE

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

FILE SORGENTI

'`lib/_gcc.h`' [95.2].

88.2 os32: abort(3)

NOME

'**abort**' - conclusione anormale del processo

SINTASSI

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

DESCRIZIONE

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

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

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

FILE SORGENTI

'`lib/stdlib.h`' [95.19]

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

VEDERE ANCHE

signal(2) [87.52].

88.3 os32: abs(3)

«

NOME

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

SINTASSI

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

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

DESCRIZIONE

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

VALORE RESTITUITO

Il valore assoluto del numero intero fornito come argomento.

FILE SORGENTI

'`lib/stdlib.h`' [95.19]

'`lib/stdlib/abs.c`' [95.19.3]

'`lib/stdlib/labs.c`' [95.19.11]

'`lib/stdlib/llabs.c`' [95.19.13]

'`lib/inttypes.h`' [95.8]

'`lib/inttypes/imaxabs.c`' [95.8.1]

VEDERE ANCHE

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

88.4 os32: access(3)

«

NOME

'**access**' - verifica dei permessi di accesso dell'utente

SINTASSI

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

DESCRIZIONE

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

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

VALORE RESTITUITO

Valore	Significato
0	I permessi di accesso richiesti sono tutti disponibili.
−1	I permessi non sono tutti disponibili, oppure si è verificato un errore di altro genere, da chiarire analizzando la variabile ***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; pertanto, si tratta di una valutazione presunta e non reale.

FILE SORGENTI

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

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

VEDERE ANCHE

stat(2) [87.55].

88.5 os32: asctime(3)

Vedere *ctime(3)* [88.15].

88.6 os32: assert(3)

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

FILE SORGENTI

‘`lib/assert.h`’ [95.1.3]

88.7 os32: atexit(3)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

VALORE RESTITUITO

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

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

FILE SORGENTI

'`lib/limits.h`' [95.1.6]

'`lib/stdlib.h`' [95.19]

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

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

VEDERE ANCHE

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

88.8 os32: atoi(3)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

VALORE RESTITUITO

Il valore numerico ottenuto dall'interpretazione della stringa.

FILE SORGENTI

'`lib/stdlib.h`' [95.19]

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

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

VEDERE ANCHE

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

88.9 os32: atol(3)

Vedere *atoi(3)* [88.8].

88.10 os32: basename(3)

NOME

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

SINTASSI

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

DESCRIZIONE

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

Contenuto originale di ***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`’ [95.9]

‘`lib/libgen/basename.c`’ [95.9.1]

‘`lib/libgen/dirname.c`’ [95.9.2]

88.11 os32: byteorder(3)

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

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

FILE SORGENTI

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

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

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

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

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

VEDERE ANCHE

inet_ntop(3) [88.66], *inet_pton(3)* [88.67].

88.12 os32: clearerr(3)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

FILE SORGENTI

‘`lib/stdio.h`’ [95.18]

‘`lib/stdio/clearerr.c`’ [95.18.2]

VEDERE ANCHE

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

88.13 os32: closedir(3)

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
−1	Errore: la variabile ***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’ [95.26]

‘lib/dirent.h’ [95.4]

‘lib/dirent/DIR.c’ [95.4.1]

‘lib/dirent/closedir.c’ [95.4.2]

VEDERE ANCHE

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

88.14 os32: creat(3)

«

NOME

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

SINTASSI

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

DESCRIZIONE

La funzione ***creat()*** equivale esattamente all’uso della funzione ***open()***, con le opzioni ‘**O_WRONLY|O_CREAT|O_TRUNC**’:

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

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

FILE SORGENTI

‘lib/sys/types.h’ [95.26]

‘lib/sys/stat.h’ [95.25]

‘lib/fcntl.h’ [95.6]

‘lib/fcntl/creat.c’ [95.6.1]

VEDERE ANCHE

chmod(2) [87.7], *chown(2)* [87.8], *close(2)* [87.10], *dup(2)* [87.12], *fcntl(2)* [87.18], *link(2)* [87.33], *mknod(2)* [87.35], *mount(2)* [87.36], *open(2)* [87.37] *read(2)* [87.39], *stat(2)* [87.55], *umask(2)* [87.60], *unlink(2)* [87.62], *write(2)* [87.64], *fopen(3)* [88.36].

88.15 os32: ctime(3)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

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

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

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

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

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

VALORE RESTITUITO

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

FILE SORGENTI

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

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

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

‘lib/time/mktime.c’ [95.29.4]

VEDERE ANCHE

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

88.16 os32: dirname(3)

«

Vedere *basename(3)* [88.10].

88.17 os32: div(3)

«

NOME

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

SINTASSI

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

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

DESCRIZIONE

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

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

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

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

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

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

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

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

VALORE RESTITUITO

Il risultato della divisione, strutturato in quoziente e resto.

FILE SORGENTI

'`lib/stdlib.h`' [95.19]

'`lib/stdlib/div.c`' [95.19.7]

'`lib/stdlib/ldiv.c`' [95.19.12]

‘lib/stdlib/lldiv.c’ [95.19.14]

‘lib/inttypes.h’ [95.8]

‘lib/inttypes/imaxdiv.c’ [95.8.2]

VEDERE ANCHE

abs(3) [88.3].

88.18 os32: endgrent(3)

«

Vedere *getgrent(3)* [88.53].

88.19 os32: endpwent(3)

«

Vedere *getpwent(3)* [88.57].

88.20 os32: errno(3)

«

NOME

‘**errno**’ - numero dell’ultimo errore riportato

SINTASSI

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

DESCRIZIONE

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

```
extern int errno;
```

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

```
errno = ENOMEM;
```

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

```
errset (ENOMEM);
```

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

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

ERRORI

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

Valore di *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.

Valore di *errno*	Definizione
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 os32.

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

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

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

Valore di *errno*	Definizione
`E_NOT_IMPLEMENTED`	Not implemented.
`E_HARDWARE_FAULT`	Hardware fault.
`E_DRIVER_FAULT`	Driver fault.
`E_PIPE_FULL`	Pipe full.
`E_PIPE_EMPTY`	Pipe empty.

FILE SORGENTI

‘`lib/errno.h`’ [95.5]

‘`lib/errno/errno.c`’ [95.5.1]

VEDERE ANCHE

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

88.21 os32: exec(3)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

Rispetto al gruppo di funzioni '**`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 parametro il puntatore a un array di stringhe, dove ***argv[0]*** deve essere il nome del file da avviare, mentre da ***argv[1]*** in poi, si tratta degli argomenti da passare al programma. Anche in questo caso, per riconoscere l'ultimo elemento di questo array, gli si deve assegnare il puntatore nullo.

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

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

VALORE RESTITUITO

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

stituire soltanto un valore che rappresenta un errore, ovvero −1, aggiornando anche la variabile ***errno*** di conseguenza.

ERRORI

Valore di ***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`’ [95.30]

‘`lib/unistd/execl.c`’ [95.30.10]

‘`lib/unistd/execle.c`’ [95.30.11]

‘`lib/unistd/execlp.c`’ [95.30.12]

‘`lib/unistd/execv.c`’ [95.30.13]

‘`lib/unistd/execvp.c`’ [95.30.15]

‘`lib/unistd/execve.c`’ [95.30.14]

VEDERE ANCHE

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

88.22 os32: execl(3)

Vedere *exec(3)* [88.21].

88.23 os32: execle(3)

Vedere *exec(3)* [88.21].

88.24 os32: execlp(3)

Vedere *exec(3)* [88.21].

88.25 os32: execv(3)

Vedere *exec(3)* [88.21].

88.26 os32: execvp(3)

Vedere *exec(3)* [88.21].

88.27 os32: exit(3)

Vedere *atexit(3)* [88.7].

88.28 os32: fclose(3)

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
'**`EOF`**'	Errore: la variabile ***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`' [95.18]

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

VEDERE ANCHE

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

88.29 os32: feof(3)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

FILE SORGENTI

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

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

VEDERE ANCHE

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

88.30 os32: ferror(3)

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

FILE SORGENTI

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

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

VEDERE ANCHE

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

88.31 os32: fflush(3)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

Valore	Significato
0	Rappresenta il successo dell'operazione.

FILE SORGENTI

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

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

VEDERE ANCHE

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

88.32 os32: fgetc(3)

NOME

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

SINTASSI

```
#include <stdio.h>
int fgetc   (FILE *fp);
int getc    (FILE *fp);
int getchar (void);
```

DESCRIZIONE

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

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

VALORE RESTITUITO

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

ERRORI

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

Valore di ***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`' [95.18]

'`lib/stdio/fgetc.c`' [95.18.7]

'`lib/stdio/getchar.c`' [95.18.24]

VEDERE ANCHE

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

88.33 os32: fgetpos(3)

NOME

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

SINTASSI

```
#include <stdio.h>
int fgetpos (FILE *restrict fp, fpos_t *restrict pos);
int fsetpos (FILE *restrict fp, fpos_t *restrict pos);
```

DESCRIZIONE

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

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

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

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

VALORE RESTITUITO

Valore	Significato
0	Rappresenta il successo dell'operazione.
−1	Indica il verificarsi di un errore, il quale può essere interpretato leggendo la variabile ***errno***.

ERRORI

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

FILE SORGENTI

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

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

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

VEDERE ANCHE

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

88.34 os32: fgets(3)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

VALORE RESTITUITO

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

ERRORI

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

Valore di ***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`’ [95.18]

‘`lib/stdio/fgets.c`’ [95.18.9]

‘`lib/stdio/gets.c`’ [95.18.25]

VEDERE ANCHE

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

88.35 os32: fileno(3)

«

NOME

‘**`fileno`**’ - traduzione di un flusso di file nel numero di descrittore corrispondente

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

ERRORI

Valore di *errno*	Significato
EBADF	È stato richiesto di risolvere il puntatore nullo.

FILE SORGENTI

‘lib/stdio.h’ [95.18]

‘lib/stdio/FILE.c’ [95.18.1]

‘lib/stdio/fileno.c’ [95.18.10]

VEDERE ANCHE

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

88.36 os32: fopen(3)

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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 ot-

tiene 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`’ [95.18]

‘`lib/stdio/FILE.c`’ [95.18.1]

‘`lib/stdio/fopen.c`’ [95.18.11]

‘`lib/stdio/freopen.c`’ [95.18.16]

VEDERE ANCHE

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

88.37 os32: fprintf(3)

«

Vedere *printf(3)* [88.91].

88.38 os32: fputc(3)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

ERRORI

Valore di *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`’ [95.18]

‘`lib/stdio/fputc.c`’ [95.18.13]

VEDERE ANCHE

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

88.39 os32: fputs(3)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

VALORE RESTITUITO

Valore	Significato
≥ 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`' [95.18]

'`lib/stdio/fputs.c`' [95.18.14]

'`lib/stdio/puts.c`' [95.18.29]

VEDERE ANCHE

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

88.40 os32: fread(3)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

FILE SORGENTI

‘`lib/stdio.h`’ [95.18]

‘`lib/stdio/fread.c`’ [95.18.15]

VEDERE ANCHE

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

88.41 os32: free(3)

Vedere *malloc(3)* [88.76].

88.42 os32: freopen(3)

Vedere *fopen(3)* [88.36].

88.43 os32: fscanf(3)

Vedere *scanf(3)* [88.102].

88.44 os32: fseek(3)

NOME

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

SINTASSI

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

DESCRIZIONE

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

Valore di ***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`' [95.18]

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

'`lib/stdio/fseek.c`' [95.18.18]

VEDERE ANCHE

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

88.45 os32: fseeko(3)

Vedere *fseek(3)* [88.44].

88.46 os32: fsetpos(3)

Vedere *fgetpos(3)* [88.33].

88.47 os32: ftell(3)

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

Valore	Significato
≥ 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`' [95.18]

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

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

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

VEDERE ANCHE

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

88.48 os32: ftello(3)

«

Vedere *ftell(3)* [88.47].

88.49 os32: fwrite(3)

«

NOME

'**fwrite**' - scrittura attraverso un flusso di file

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

FILE SORGENTI

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

'`lib/stdio/fwrite.c`' [95.18.23]

VEDERE ANCHE

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

88.50 os32: getc(3)

«

Vedere *fgetc(3)* [88.32].

88.51 os32: getchar(3)

«

Vedere *fgetc(3)* [88.32].

88.52 os32: getenv(3)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

VALORE RESTITUITO

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

FILE SORGENTI

‘`lib/stdlib.h`’ [95.19]

‘`applic/crt0.mer.s`’ [96.1.12]

‘applic/crt0.sep.s’ [96.1.13]

‘lib/stdlib/environment.c’ [95.19.8]

‘lib/stdlib/getenv.c’ [95.19.10]

VEDERE ANCHE

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

88.53 os32: getgrent(3)

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

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

VALORE RESTITUITO

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

ERRORI

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

DIFETTI

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

FILE SORGENTI

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

'`lib/pwd.h`' [95.15]

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

VEDERE ANCHE

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

88.54 os32: getgrnam(3)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

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

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

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

'`lib/grp.h`' [95.7]

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

VEDERE ANCHE

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

88.55 os32: getpwuid(3)

«

Vedere *getgrnam(3)* [88.54].

88.56 os32: getopt(3)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

programma [−***x*** [***argomento***]] ... [***argomento***] ...

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

FILE SORGENTI

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

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

88.57 os32: getpwent(3)

«

NOME

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

SINTASSI

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

DESCRIZIONE

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

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

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

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

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

VALORE RESTITUITO

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

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

ERRORI

Valore di *errno*	Significato
`EINVAL`	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
`EPERM`	Operazione non consentita.
`EACCES`	Accesso non consentito.
`ENOENT`	Il file non esiste, oppure non esiste il percorso che porta al file da creare.
`ENOTDIR`	Il percorso che porta al file da aprire non è valido, in quanto ciò che dovrebbe essere una directory, non lo è.
`ENFILE`	Non si possono aprire altri file nell'ambito del sistema operativo (il sistema ha raggiunto il limite).
`EMFILE`	Non si possono aprire altri file nell'ambito del processo in corso.

DIFETTI

Nel file '`/etc/passwd`' non è possibile lasciare campi vuoti, anche se questi non sono usati, perché la scansione avviene con la funzione ***strtok()***.

FILE SORGENTI

'`lib/sys/types.h`' [95.26]

'`lib/pwd.h`' [95.15]

'lib/pwd/pwent.c' [95.15.1]

VEDERE ANCHE

getpwnam(3) [88.58], *getpwuid(3)* [88.58], *passwd(5)* [90.4].

88.58 os32: getpwnam(3)

NOME

'**getpwnam**', '**getpwuid**' - selezione di una voce dal file '/etc/passwd'

SINTASSI

```
#include <sys/types.h>
#include <pwd.h>
struct passwd *getpwnam (const char *name);
struct passwd *getpwuid (uid_t uid);
```

DESCRIZIONE

La funzione ***getpwnam()*** restituisce il puntatore a una variabile strutturata, di tipo '**struct passwd**', come definito nel file 'pwd.h', contenente le informazioni sull'utenza specificata per nome, dal '/etc/passwd'. La funzione ***getpwuid()*** si comporta in modo analogo, individuando però l'utenza da selezionare in base al numero UID.

Il tipo '**struct passwd**' è definito nel file 'pwd.h' nel modo seguente:

```
struct passwd {
    char  *pw_name;
    char  *pw_passwd;
    uid_t  pw_uid;
    gid_t  pw_gid;
    char  *pw_gecos;
    char  *pw_dir;
    char  *pw_shell;
};
```

La sequenza dei campi della struttura corrisponde a quella contenuta nel file '`/etc/passwd`'.

VALORE RESTITUITO

Le funzioni ***getpwnam()*** e ***getpwuid()*** restituiscono il puntatore a una variabile strutturata di tipo '**`struct passwd`**', se l'operazione ha avuto successo. Se il nome o il numero dell'utente non si trovano nel file '`/etc/passwd`', oppure se si presenta un errore, il valore restituito è '**`NULL`**'. Per poter distinguere tra una voce non trovata o il verificarsi di un errore di accesso al file '`/etc/passwd`', prima della chiamata della funzione occorre azzerare il valore della variabile ***errno***, verificando successivamente se ha acquisito un valore differente.

ERRORI

Valore di *errno*	Significato
`EINVAL`	Gli argomenti forniti alla chiamata non sono validi per qualche ragione.
`EPERM`	Operazione non consentita.
`EEXIST`	Il file da creare esiste già.
`EACCES`	Accesso non consentito.
`ENOENT`	Il file non esiste, oppure non esiste il percorso che porta al file da creare.
`EROFS`	Avendo richiesto un accesso in scrittura, si ottiene che il file system che lo contiene consente soltanto un accesso in lettura.
`ENOTDIR`	Il percorso che porta al file da aprire non è valido, in quanto ciò che dovrebbe essere una directory, non lo è.
`ENFILE`	Non si possono aprire altri file nell'ambito del sistema operativo (il sistema ha raggiunto il limite).
`EMFILE`	Non si possono aprire altri file nell'ambito del processo in corso.

FILE SORGENTI

'`lib/sys/types.h`' [95.26]

'`lib/pwd.h`' [95.15]

'`lib/pwd/pwent.c`' [95.15.1]

VEDERE ANCHE

getpwent(3) [88.57], *setpwent(3)* [88.57], *endpwent(3)* [88.57], *passwd(5)* [90.4].

88.59 os32: getpwuid(3)

«

Vedere *getpwnam(3)* [88.58].

88.60 os32: gets(3)

«

Vedere *fgets(3)* [88.34].

88.61 os32: gmtime(3)

«

Vedere *ctime(3)* [88.15].

88.62 os32: htonl(3)

«

Vedere *byteorder(3)* [88.11].

88.63 os32: htons(3)

«

Vedere *byteorder(3)* [88.11].

88.64 os32: imaxabs(3)

«

Vedere *abs(3)* [88.3].

88.65 os32: imaxdiv(3)

«

Vedere *div(3)* [88.17].

88.66 os32: inet_ntop(3)

«

NOME

‘**inet_ntop**’ - conversione di un indirizzo IP dalla sua forma numerica a quella testuale

SINTASSI

```
#include <arpa/inet.h>
const char *inet_ntop (int family, const void *src,
                       char *dst, socklen_t size);
```

DESCRIZIONE

La funzione ***inet_ntop()*** converte un indirizzo IPv4, contenuto in una variabile strutturata di tipo '**`struct in_addr`**', a cui punta ***src***, in una stringa, annotandola nella memoria tampone a cui punta ***dst***, la cui dimensione massima è di ***size*** byte.

Il parametro ***family*** indica il tipo di indirizzo da convertire, ma per os32 è ammesso soltanto il tipo ***AF_INET***, corrispondente a IPv4.

VALORE RESTITUITO

Se l'operazione si conclude con successo, la funzione restituisce il puntatore a ***dst***; altrimenti, si ottiene il valore ***NULL*** e va verificato il contenuto della variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EAFNOSUPPORT`	È stata richiesta la conversione di un tipo di indirizzo diverso da ***AF_INET***.
`EINVAL`	In corrispondenza del parametro ***src*** o di ***dst*** si ha un puntatore nullo.

FILE SORGENTI

'`lib/arpa/inet.h`' [95.3]

'`lib/arpa/inet/inet_ntop.c`' [95.3.3]

VEDERE ANCHE

inet_pton(3) [88.67].

88.67 os32: inet_pton(3)

«

NOME

‘**inet_pton**’ - conversione di un indirizzo IP dalla sua forma testuale a quella numerica

SINTASSI

```
#include <arpa/inet.h>
int inet_pton (int family, const char *src, void *dst);
```

DESCRIZIONE

La funzione ***inet_pton()*** converte un indirizzo IPv4, contenuto nella stringa ***src***, in una variabile strutturata di tipo ‘**struct in_addr**’, a cui punta ***dst***.

Il parametro ***family*** indica il tipo di indirizzo da convertire, ma per os32 è ammesso soltanto il tipo ***AF_INET***, corrispondente a IPv4.

VALORE RESTITUITO

Se l’operazione si conclude con successo, la funzione restituisce il valore numerico 1; se invece la stringa ***src*** non contiene un indirizzo valido, la funzione restituisce 0; se il valore di ***family*** non è valido o non è gestito, restituisce −1.

ERRORI

La variabile ***errno*** non viene modificata, perché tutti i casi che possono presentarsi sono espressi attraverso il valore restituito dalla funzione.

FILE SORGENTI

‘`lib/arpa/inet.h`’ [95.3]

‘`lib/arpa/inet/inet_pton.c`’ [95.3.4]

VEDERE ANCHE

inet_ntop(3) [88.66].

88.68 os32: input_line(3)

NOME

‘**`input_line`**’ - riga di comando

SINTASSI

```
#include <sys/os32.h>
void input_line (char *line, char *prompt, size_t size,
                 int type);
```

DESCRIZIONE

La funzione ***input_line()*** consente di inserire un’informazione da tastiera, interpretando in modo adeguato i codici usati per cancellare. Si tratta del mezzo principale di inserimento di un dato da tastiera per os32.

Il parametro ***line*** è il puntatore a un’area di memoria, da modificare con l’inserimento che si intende fare; questa area di memoria

deve essere in grado di contenere tanti byte quanto indicato con il parametro ***size***. Il parametro ***prompt*** indica una stringa da usare come invito, a sinistra della riga da inserire. Il parametro ***type*** serve a specificare il tipo di visualizzazione sullo schermo di ciò che si inserisce. Si utilizzano della macro-variabili dichiarate nel file ‘`sys/os32.h`’:

Macro-variabile	Descrizione
`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.

La funzione conclude il suo funzionamento quando si preme [*Invio*].

VALORE RESTITUITO

La funzione non restituisce alcunché, ma ciò che viene digitato è disponibile nella memoria tampone rappresentata dal puntatore ***line***, da intendere come stringa terminata correttamente.

FILE SORGENTI

‘`lib/sys/os32.h`’ [95.21]

‘`lib/sys/os32/input_line.c`’ [95.21.1]

VEDERE ANCHE

shell(1) [86.24].

login(1) [86.14].

88.69 os32: isatty(3)

NOME

'**isatty**' - verifica che un certo descrittore di file si riferisca a un terminale

SINTASSI

```
#include <unistd.h>
int isatty (int fdn);
```

DESCRIZIONE

La funzione ***isatty()*** verifica se il descrittore di file specificato con il parametro ***fdn*** si riferisce a un dispositivo di terminale.

VALORE RESTITUITO

Valore	Significato
1	Si tratta effettivamente del file di dispositivo di un terminale.
0	Il descrittore di file non riguarda un terminale, oppure si è verificato un errore; in ogni caso la variabile ***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' [95.30]

'lib/unistd/isatty.c' [95.30.28]

VEDERE ANCHE

stat(2) [87.55], *ttyname(3)* [88.133].

88.70 os32: labs(3)

«

Vedere *abs(3)* [88.3].

88.71 os32: ldiv(3)

«

Vedere *div(3)* [88.17].

88.72 os32: llabs(3)

«

Vedere *abs(3)* [88.3].

88.73 os32: lldiv(3)

«

Vedere *div(3)* [88.17].

88.74 os32: major(3)

«

Vedere *makedev(3)* [88.75].

88.75 os32: makedev(3)

«

NOME

‘**makedev**’, ‘**major**’, ‘**minor**’ - gestione dei numeri di dispositivo

SINTASSI

```
#include <sys/types.h>
dev_t makedev (int major, int minor);
int   major   (dev_t device);
int   minor   (dev_t device);
```

DESCRIZIONE

La funzione ***makedev()*** restituisce il numero di dispositivo complessivo, partendo dal numero primario (***major***) e dal numero secondario (***minor***), presi separatamente.

Le funzioni ***major()*** e ***minor()***, rispettivamente, restituiscono il numero primario o il numero secondario, partendo da un numero di dispositivo completo.

Si tratta di funzioni non previste dallo standard, ma ugualmente diffuse.

FILE SORGENTI

‘`lib/sys/types.h`’ [95.26]

‘`lib/sys/types/makedev.c`’ [95.26.2]

‘`lib/sys/types/major.c`’ [95.26.1]

‘`lib/sys/types/minor.c`’ [95.26.3]

88.76 os32: malloc(3)

«

NOME

‘**`malloc`**’, ‘**`free`**’, ‘**`realloc`**’ - allocazione e rilascio dinamico di memoria

SINTASSI

```
#include <stdlib.h>
void *malloc (size_t size);
void free (void *address);
void *realloc (void *address, size_t size);
```

DESCRIZIONE

Le funzioni '...**`alloc()`**' e ***free()*** consentono di allocare, riallocare e liberare delle aree di memoria, in modo dinamico.

La funzione ***malloc()*** (*memory allocation*) si usa per richiedere l'allocazione di una dimensione di almeno ***size*** byte di memoria. Se l'allocazione avviene con successo, la funzione restituisce il puntatore generico di tale area allocata.

Quando un'area di memoria allocata precedentemente non serve più, va liberata espressamente con l'ausilio della funzione ***free()***, la quale richiede come argomento il puntatore generico all'inizio di tale area. Naturalmente, si può liberare la memoria una volta sola e un'area di memoria liberata non può più essere raggiunta.

Quando un'area di memoria già allocata richiede una modifica nella sua estensione, si può usare la funzione ***realloc()***, la quale necessita di conoscere il puntatore precedente e la nuova estensione. La funzione restituisce un nuovo puntatore, il quale potrebbe eventualmente, ma non necessariamente, coincidere con quello dell'area originale.

Se le funzioni ***malloc()*** e ***realloc()*** falliscono nel loro intento, restituiscono un puntatore nullo.

VALORE RESTITUITO

Le funzioni ***malloc()*** e ***realloc()*** restituiscono il puntatore generico all'area di memoria allocata; se falliscono, restituiscono invece un puntatore nullo.

ERRORI

Valore di *errno*	Significato
ENOMEM	Memoria insufficiente.

FILE SORGENTI

‘lib/limits.h’ [95.1.6]

‘lib/stdlib.h’ [95.19]

‘lib/stdlib_alloc/malloc.c’ [95.19.24]

‘lib/stdlib_alloc/realloc.c’ [95.19.25]

‘lib/stdlib_alloc/free.c’ [95.19.23]

‘lib/stdlib_alloc/_alloc_list.c’ [95.19.22]

88.77 os32: memccpy(3)

«

NOME

‘**memccpy**’ - copia di un’area di memoria

SINTASSI

```
#include <string.h>
void *memccpy (void *restrict dst, const void *restrict org, int
               size_t n);
```

DESCRIZIONE

La funzione ***memccpy()*** copia al massimo ***n*** byte a partire dall’area di memoria a cui punta ***org***, verso l’area che inizia da ***dst***, fermandosi se si incontra il carattere ***c***, il quale viene copiato regolarmente, fermo restando il limite massimo di ***n*** byte.

Le due aree di memoria, origine e destinazione, non devono sovrapporsi.

VALORE RESTITUITO

Nel caso in cui la copia sia avvenuta con successo, fino a incontrare il carattere ***c***, la funzione restituisce il puntatore al carattere successivo a ***c***, nell'area di memoria di destinazione. Se invece tale carattere non viene trovato nei primi ***n*** byte, restituisce il puntatore nullo '**`NULL`**'. La variabile ***errno*** non viene modificata.

FILE SORGENTI

'`lib/string.h`' [95.20]

'`lib/string/memccpy.c`' [95.20.1]

VEDERE ANCHE

memcpy(3) [88.80], *memmove(3)* [88.81], *strcpy(3)* [88.117], *strncpy(3)* [88.117].

88.78 os32: memchr(3)

«

NOME

'**`memchr`**' - scansione della memoria alla ricerca di un carattere

SINTASSI

```
#include <string.h>
void *memchr (const void *memory, int c, size_t n);
```

DESCRIZIONE

La funzione ***memchr()*** scandisce l'area di memoria a cui punta ***memory***, fino a un massimo di ***n*** byte, alla ricerca del carattere ***c***.

VALORE RESTITUITO

Se la funzione trova il carattere, restituisce il puntatore al carattere trovato, altrimenti restituisce il puntatore nullo '**NULL**'.

FILE SORGENTI

'lib/string.h' [95.20]

'lib/string/memchr.c' [95.20.2]

VEDERE ANCHE

strchr(3) [88.114], *strrchr(3)* [88.114], *strpbrk(3)* [88.125].

88.79 os32: memcmp(3)

NOME

'**memcmp**' - confronto di due aree di memoria

SINTASSI

```
#include <string.h>
int memcmp (const void *memory1, const void *memory2,
            size_t n);
```

DESCRIZIONE

La funzione ***memcmp()*** confronta i primi ***n*** byte di memoria delle aree che partono, rispettivamente, da ***memory1*** e da ***memory2***.

VALORE RESTITUITO

Valore	Esito del confronto.
−1	***memory1 < memory2***
0	***memory1 == memory2***
+1	***memory1 > memory2***

FILE SORGENTI

‘lib/string.h’ [95.20]

‘lib/string/memcmp.c’ [95.20.3]

VEDERE ANCHE

strcmp(3) [88.115], *strncmp(3)* [88.115].

88.80 os32: memcpy(3)

«

NOME

‘**memcpy**’ - copia di un’area di memoria

SINTASSI

```
#include <string.h>
void *memcpy (void *restrict dst, const void *restrict org,
              size_t n);
```

DESCRIZIONE

La funzione ***memcpy()*** copia al massimo ***n*** byte a partire dall’area di memoria a cui punta ***org***, verso l’area che inizia da ***dst***.

Le due aree di memoria, origine e destinazione, non devono sovrapporsi.

VALORE RESTITUITO

La funzione restituisce ***dst***.

FILE SORGENTI

‘lib/string.h’ [95.20]

‘lib/string/memcpy.c’ [95.20.4]

VEDERE ANCHE

memccpy(3) [88.77], *memmove(3)* [88.81], *strcpy(3)* [88.117], *strncpy(3)* [88.117].

88.81 os32: memmove(3)

NOME

‘**memmove**’ - copia di un’area di memoria

SINTASSI

```
#include <string.h>
void *memmove (void *dst, const void *org, size_t n);
```

DESCRIZIONE

La funzione ***memmove()*** copia al massimo ***n*** byte a partire dall’area di memoria a cui punta ***org***, verso l’area che inizia da ***dst***. A differenza di quanto fa ***memcpy()***, la funzione ***memmove()*** esegue la copia correttamente anche se le due aree di memoria sono sovrapposte.

VALORE RESTITUITO

La funzione restituisce ***dst***.

FILE SORGENTI

‘`lib/string.h`’ [95.20]

‘`lib/string/memmove.c`’ [95.20.5]

VEDERE ANCHE

memccpy(3) [88.77], *memcpy(3)* [88.80], *strcpy(3)* [88.117], *strncpy(3)* [88.117].

88.82 os32: memset(3)

«

NOME

‘**memset**’ - scrittura della memoria con un byte sempre uguale

SINTASSI

```
#include <string.h>
void *memset (void *memory, int c, size_t n);
```

DESCRIZIONE

La funzione ***memset()*** scrive ***n*** byte, contenenti il valore di ***c***, ridotto a un carattere, a partire dal ciò a cui punta ***memory***.

FILE SORGENTI

‘`lib/string.h`’ [95.20]

‘`lib/string/memset.c`’ [95.20.6]

VEDERE ANCHE

memcpy(3) [88.80].

88.83 os32: minor(3)

«

Vedere *makedev(3)* [88.75].

88.84 os32: mktime(3)

«

Vedere *ctime(3)* [88.15].

88.85 os32: namep(3)

«

NOME

'**namep**' - ricerca del percorso di un programma utilizzando la variabile di ambiente ***PATH***

SINTASSI

```
#include <sys/os32.h>
int namep (const char *name, char *path, size_t size);
```

DESCRIZIONE

La funzione ***namep()*** trova il percorso di un programma, tenendo conto delle informazioni contenute nella variabile di ambiente ***PATH***.

Il parametro ***name*** rappresenta una stringa con il nome del comando da cercare nel file system; il parametro ***path*** deve essere il puntatore di un'area di memoria, da sovrascrivere con il percorso assoluto del programma da avviare, una volta trovato, con l'accortezza di far sì che risulti una stringa terminata correttamente; il parametro ***size*** specifica la dimensione massima che può avere la stringa ***path***.

Questa funzione viene utilizzata in particolare da ***execvp()***.

VALORE RESTITUITO

Valore	Significato
0	Operazione riuscita.
−1	Operazione fallita. Va verificato l'errore indicato dalla variabile ***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/os32.h`' [95.21]

'`lib/sys/os32/namep.c`' [95.21.4]

VEDERE ANCHE

shell(1) [86.24], *execvp(3)* [88.21], *execlp(3)* [88.21].

88.86 os32: ntohl(3)

«

Vedere *byteorder(3)* [88.11].

88.87 os32: ntohs(3)

«

Vedere *byteorder(3)* [88.11].

88.88 os32: offsetof(3)

«

NOME

'**`offsetof`**' - posizione di un membro di una struttura, dall'inizio della stessa

SINTASSI

```
#include <stddef.h>
size_t offsetof (type, member);
```

DESCRIZIONE

La macroistruzione ***offsetof()*** consente di determinare la collocazione relativa di un membro di una variabile strutturata, restituendo la quantità di byte che la struttura occupa prima dell'inizio del membro richiesto. Per ottenere questo risultato, il primo argomento deve essere il nome del tipo del membro cercato, mentre il secondo argomento deve essere il nome del membro stesso.

VALORE RESTITUITO

La macroistruzione restituisce lo scostamento del membro specificato, rispetto all'inizio della struttura a cui appartiene, espresso in byte.

FILE SORGENTI

'`lib/stddef.h`' [95.1.12]

88.89 os32: opendir(3)

«

NOME

'**`opendir`**' - apertura di una directory

SINTASSI

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir (const char *name);
```

DESCRIZIONE

La funzione ***opendir()*** apre la directory rappresentata da ***name***, posizionando l'indice interno per le operazioni di accesso alla prima voce della directory stessa.

VALORE RESTITUITO

La funzione restituisce il puntatore al flusso aperto; in caso di errore, restituisce '**`NULL`**' e aggiorna la variabile ***errno***.

ERRORI

Valore di ***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`' [95.26]

'`lib/dirent.h`' [95.4]

'`lib/dirent/DIR.c`' [95.4.1]

'`lib/dirent/opendir.c`' [95.4.3]

VEDERE ANCHE

open(2) [87.37], *closedir(3)* [88.13], *readdir(3)* [88.98], *rewinddir(3)* [88.101].

88.90 os32: perror(3)

«

NOME

‘**perror**’ - emissione di un messaggio di errore di sistema

SINTASSI

```
#include <stdio.h>
void perror (const char *string);
```

DESCRIZIONE

La funzione ***perror()*** legge il valore della variabile ***errno*** e, se questo è diverso da zero, emette attraverso lo standard output la stringa fornita come argomento, ammesso che non si tratti del puntatore nullo, quindi continua con l’emissione della descrizione dell’errore.

La funzione ***perror()*** di os32, nell’emettere il testo dell’errore, mostra anche il nome del file sorgente e il numero della riga in cui si è verificato. Ma questi dati sono validi soltanto se l’annotazione dell’errore è avvenuta, a suo tempo, con l’ausilio della funzione *errset(3)* [88.20], la quale non è prevista dagli standard.

FILE SORGENTI

‘`lib/errno.h`’ [95.5]

‘`lib/stdio.h`’ [95.18]

‘`lib/stdio/perror.c`’ [95.18.26]

VEDERE ANCHE

errno(3) [88.20], *strerror(3)* [88.120].

88.91 os32: printf(3)

«

NOME

‘**printf**’, ‘**fprintf**’, ‘**sprintf**’, ‘**snprintf**’ - composizione dei dati per la visualizzazione

SINTASSI

```
#include <stdio.h>
int printf   (char *restrict format, ...);
int fprintf  (FILE *fp, char *restrict format, ...);
int sprintf  (char *restrict string,
              const char *restrict format, ...);
int snprintf (char *restrict string, size_t size,
              const char *restrict format, ...);
```

DESCRIZIONE

Le funzioni del gruppo ‘...**printf()**’ hanno in comune lo scopo di comporre dei dati in forma di stringa, generalmente per la visualizzazione, o comunque per la fruizione a livello umano.

I dati in ingresso possono essere vari e si collocano come argomenti finali, di tipo e quantità non noti nel prototipo delle funzioni. Per quantificare e qualificare questi argomenti aggiuntivi, la stringa a cui punta il parametro ***format***, deve contenere degli ***specificatori di conversione***, oltre eventualmente ad altri caratteri. Pertanto, queste funzioni, prendono la stringa a cui punta

format, la interpretano e determinano quali argomenti variabili sono presenti, quindi producono un'altra stringa, composta dalla stringa precedente, sostituendo gli specificatori di conversione con i dati a cui questi si riferiscono, secondo una forma di conversione definita dagli specificatori stessi. Si osservi l'esempio seguente:

```
printf ("Valore: %x %i %o\n", 123, 124, 125);
```

In questo modo si ottiene la visualizzazione, attraverso lo standard output, della stringa '**`Valore: 7b 124 175`**'. Infatti: '**`%x`**' è uno specificatore di conversione che richiede di interpretare il proprio parametro (in questo caso il primo) come intero normale e di rappresentarlo in esadecimale; '**`%i`**' legge un numero intero normale e lo rappresenta nella forma decimale consueta; '**`%o`**' legge un intero e lo mostra in ottale.

La funzione ***printf()*** emette il risultato della composizione attraverso lo standard output; la funzione ***fprintf()*** lo fa attraverso il flusso di file ***fp***; le funzioni ***sprintf()*** e ***snprintf()*** si limitano a scrivere il risultato a partire da ciò a cui punta ***string***, con la particolarità di ***snprintf()*** che si dà comunque un limite da non superare, per evitare che la scrittura vada a sovrascrivere altri dati in memoria.

Gli specificatori di conversione devono rispettare la sintassi seguente per la libreria di os32:

```
% [simbolo] [n_ampiezza] [.n_precisione] [hh|h|l|ll|j|z|t] tipo
```

La prima cosa da individuare in uno specificatore di conversione è il tipo di argomento che viene interpretato e, di conseguenza, il

genere di rappresentazione che se ne vuole produrre. Il tipo viene espresso da una lettera alfabetica, alla fine dello specificatore di conversione.

Simbolo	Tipo di argomento	Conversione applicata
`%…d` `%…i`	`int`	Numero intero con segno da rappresentare in base dieci.
`%…u`	`unsigned int`	Numero intero senza segno da rappresentare in base dieci.
`%…o`	`unsigned int`	Numero intero senza segno da rappresentare in ottale (senza lo zero iniziale che viene usato spesso per caratterizzare un tale tipo di rappresentazione).
`%…x` `%…X`	`unsigned int`	Numero intero senza segno da rappresentare in esadecimale (senza il prefisso '**`0x`**' o '**`0X`**' che viene usato spesso per caratterizzare un tale tipo di rappresentazione).
`%…c`	`int`	Un carattere singolo, dopo la conversione in '**`unsigned char`**'.
`%…b`	`unsigned int`	Numero intero senza segno da rappresentare in binario (si tratta di un'estensione non prevista nello standard).
`%…s`	`char *`	Una stringa.
`%%`		Questo specificatore si limita a produrre un carattere di percentuale ('`%`') che altrimenti non sarebbe rappresentabile.

Nel modello sintattico che descrive lo specificatore di conversio-

ne, si vede che subito dopo il segno di percentuale può apparire un simbolo (*flag*).

Simbolo	Corrispondenza
%+... %+0*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 %...hhb	unsigned char
%...hd %...hi	short int	%...hu %...ho %...hx %...hX %...hb	unsigned short int

Simbolo	Tipo	Simbolo	Tipo
%…ld %…li	long int	%…lu %…lo %…lx| %…lX %…lb	unsigned long int
%…lld %…lli	long long int	%…llu %…llo %…llx| %…llX %…llb	unsigned long long int

Simbolo	Tipo	Simbolo	Tipo
%…jd %…ji	intmax_t	%…ju %…jo %…jx| %…jX %…jb	uintmax_t

Simbolo	Tipo	Simbolo	Tipo
%…zd %…zi	size_t	%…zu %…zo %…zx %…zX %…zb	size_t

Simbolo	Tipo	Simbolo	Tipo
%…td %…ti	ptrdiff_t	%…tu %…to %…tx %…tX %…tb	ptrdiff_t

Tra il simbolo (*flag*) e il modificatore di lunghezza può apparire un numero che rappresenta l'ampiezza da usare nella trasformazione ed eventualmente la precisione: '***ampiezza*** **[** **.*precisione*** **]** '. Per os32, la precisione si applica esclusivamente alle stringhe, la quale specifica la quantità di caratteri da considerare, troncando il resto.

VALORE RESTITUITO

Le funzioni restituiscono la quantità di caratteri utilizzati nella composizione della nuova stringa, escluso il carattere nullo di terminazione.

FILE SORGENTI

‘lib/stdio.h’ [95.18]

‘lib/stdio/FILE.c’ [95.18.1]

‘lib/stdio/fprintf.c’ [95.18.12]

‘lib/stdio/printf.c’ [95.18.27]

‘lib/stdio/sprintf.c’ [95.18.35]

‘lib/stdio/snprintf.c’ [95.18.34]

VEDERE ANCHE

vfprintf(3) [88.137], *vprintf(3)* [88.137], *vsprintf(3)* [88.137], *vsnprintf(3)* [88.137], *scanf(3)* [88.102].

88.92 os32: putc(3)

Vedere *fputc(3)* [88.38].

88.93 os32: putchar(3)

Vedere *fputc(3)* [88.38].

88.94 os32: putenv(3)

NOME

‘**putenv**’ - assegnamento di una variabile di ambiente

SINTASSI

```
#include <stdlib.h>
int putenv (const char *string);
```

DESCRIZIONE

La funzione ***putenv()*** assegna una variabile di ambiente. Se questa esiste già, va a rimpiazzare il valore assegnatole in precedenza, altrimenti la crea contestualmente.

La funzione richiede un solo parametro, costituito da una stringa in cui va specificato il nome della variabile e il contenuto da assegnargli, usando la forma '***nome=valore***'. Per esempio, '**`PATH=/bin:/usr/bin`**'.

VALORE RESTITUITO

Valore	Significato
0	Operazione riuscita.
−1	Operazione fallita. Va verificato l'errore indicato dalla variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Argomento non valido.
`ENOMEM`	Memoria insufficiente.

FILE SORGENTI

'`lib/stdlib.h`' [95.19]

'`applic/crt0.mer.s`' [96.1.12]

'`applic/crt0.sep.s`' [96.1.13]

'`lib/stdlib/environment.c`' [95.19.8]

'`lib/stdlib/putenv.c`' [95.19.15]

VEDERE ANCHE

environ(7) [91.1], *getenv(3)* [88.52], *setenv(3)* [88.104], *unsetenv(3)* [88.104].

88.95 os32: puts(3)

Vedere *fputs(3)* [88.39].

88.96 os32: qsort(3)

NOME

'**qsort**' - riordino di un array

SINTASSI

```
#include <stdlib.h>
void qsort (void *base, size_t nmemb, size_t size,
            int (*compare)(const void *, const void *));
```

DESCRIZIONE

La funzione ***qsort()*** riordina un array composto da ***nmemb*** elementi da ***size*** byte ognuno. Il primo argomento, ovvero il parametro ***base***, è il puntatore all'indirizzo iniziale di questo array in memoria.

Il riordino avviene comparando i vari elementi con l'ausilio di una funzione, passata tramite il suo puntatore, la quale deve ricevere due argomenti, costituiti dai puntatori agli elementi dell'array da confrontare. Tale funzione deve restituire un valore minore di zero per un confronto in cui il suo primo argomento deve essere collocato prima del secondo; un valore pari a zero se

gli argomenti sono uguali ai fini del riordino; un valore maggiore di zero se il suo primo argomento va collocato dopo il secondo nel riordino.

Segue un esempio di utilizzo della funzione ***qsort()***:

```
#include <stdio.h>
#include <stdlib.h>

int confronta (const void *a, const void *b)
{
    int x = *((int *) a);
    int y = *((int *) b);
    return x - y;
}

int main (void)
{
    int a[] = {3, 1, 5, 2};

    qsort (&a[0], 4, sizeof (int), confronta);
    printf ("%d %d %d %d\n", a[0], a[1], a[2], a[3]);

    return 0;
}
```

FILE SORGENTI

‘lib/stdlib.h’ [95.19]

‘lib/stdlib/qsort.c’ [95.19.16]

88.97 os32: rand(3)

«

NOME

‘**rand**’ - generazione di numeri pseudo-casuali

SINTASSI

```
#include <stdlib.h>
int  rand  (void);
void srand (unsigned int seed);
```

DESCRIZIONE

La funzione ***rand()*** produce un numero intero casuale, sulla base di un seme, il quale può essere cambiato in ogni momento, con l’ausilio di ***srand()***. A ogni chiamata della funzione ***rand()***, il risultato ottenuto, viene utilizzato anche come seme per la chiamata successiva. Se inizialmente non viene assegnato alcun seme, il primo valore predefinito è pari a 1.

VALORE RESTITUITO

La funzione ***rand()*** restituisce un numero intero casuale, determinato sulla base del seme accumulato in precedenza.

FILE SORGENTI

‘`lib/stdlib.h`’ [95.19]

‘`lib/stdlib/rand.c`’ [95.19.17]

88.98 os32: readdir(3)

«

NOME

‘**readdir**’ - lettura di una directory

SINTASSI

```
#include <sys/types.h>
#include <dirent.h>
struct dirent *readdir (DIR *dp);
```

DESCRIZIONE

La funzione ***readdir()*** legge una voce dalla directory rappresentata da ***dp*** e restituisce il puntatore a una variabile strutturata di tipo ‘**struct dirent**’, contenente le informazioni tratte dalla voce letta. La variabile strutturata in questione si trova in memoria statica e viene sovrascritta con le chiamate successive della funzione ***readdir()***.

Il tipo ‘**struct dirent**’ è definito nel file di intestazione ‘`dirent.h`’, nel modo seguente:

```
struct dirent {
    ino_t       d_ino;
    char        d_name[NAME_MAX+1];
};
```

Il membro ***d_ino*** è il numero di inode del file il cui nome appare nel membro ***d_name***. La macro-variabile ***NAME_MAX*** è dichiarata a sua volta nel file di intestazione ‘`limits.h`’. La dimensione del membro ***d_name*** è tale da permettere di includere anche il valore zero di terminazione delle stringhe.

VALORE RESTITUITO

La funzione restituisce il puntatore a una variabile strutturata di tipo '**`struct dirent`**'; se la lettura ha già raggiunto la fine della directory, oppure per qualunque altro tipo di errore, la funzione restituisce '**`NULL`**' e aggiorna eventualmente la variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EBADF`	La directory rappresentata da ***dp*** non è valida.

FILE SORGENTI

'`lib/sys/types.h`' [95.26]

'`lib/dirent.h`' [95.4]

'`lib/dirent/DIR.c`' [95.4.1]

'`lib/dirent/readdir.c`' [95.4.4]

VEDERE ANCHE

read(2) [87.39], *closedir(3)* [88.13], *opendir(3)* [88.89], *rewinddir(3)* [88.101].

88.99 os32: realloc(3)

«

Vedere *malloc(3)* [88.76].

88.100 os32: rewind(3)

«

NOME

'**`rewind`**' - riposizionamento all'inizio dell'indice di accesso a un flusso di file

SINTASSI

```
#include <stdio.h>
void rewind (FILE *fp);
```

DESCRIZIONE

La funzione ***rewind()*** azzera l'indice della posizione interna del flusso di file specificato con il parametro ***fp***; inoltre azzera anche l'indicatore di errore dello stesso flusso. In pratica si ottiene la stessa cosa di:

```
(void) fseek (fp, 0L, SEEK_SET);
clearerr (fp);
```

FILE SORGENTI

'`lib/stdio.h`' [95.18]

'`lib/stdio/FILE.c`' [95.18.1]

'`lib/stdio/rewind.c`' [95.18.30]

VEDERE ANCHE

lseek(2) [87.33], *fgetpos(3)* [88.33], *fsetpos(3)* [88.33], *ftell(3)* [88.47], *fseek(3)* [88.44], *rewind(3)* [88.100].

88.101 os32: rewinddir(3)

«

NOME

'**`rewinddir`**' - riposizionamento all'inizio del riferimento per l'accesso a una directory

SINTASSI

```
#include <sys/types.h>
#include <dirent.h>
void rewinddir (DIR *dp);
```

DESCRIZIONE

La funzione ***rewinddir()*** riposiziona i riferimenti per l'accesso alla directory indicata, in modo che la prossima lettura o scrittura avvenga dalla prima posizione.

VALORE RESTITUITO

La funzione non restituisce alcunché e non si presenta nemmeno la possibilità di segnalare errori attraverso la variabile ***errno***.

FILE SORGENTI

'`lib/sys/types.h`' [95.26]

'`lib/dirent.h`' [95.4]

'`lib/dirent/DIR.c`' [95.4.1]

'`lib/dirent/rewinddir.c`' [95.4.5]

VEDERE ANCHE

rewind(3) [88.100], *closedir(3)* [88.13], *opendir(3)* [88.89], *rewinddir(3)* [88.98].

88.102 os32: scanf(3)

«

NOME

'**`scanf`**', '**`fscanf`**', '**`sscanf`**' - interpretazione dell'input e conversione

SINTASSI

```
#include <stdio.h>
int scanf  (const char *restrict format, ...);
int fscanf (FILE *restrict fp,
            const char *restrict format, ...);
int sscanf (char *restrict string,
            const char *restrict format, ...);
```

DESCRIZIONE

Le funzioni del gruppo '...**scanf()**' hanno in comune lo scopo di interpretare dei dati, forniti in forma di stringa, convertendoli opportunamente.

I dati in ingresso sono costituiti da una sequenza di caratteri, la quale viene fornita tramite lo standard input per ***scanf()***, tramite il flusso di file ***fp*** per ***fscanf()***, oppure tramite la stringa ***string*** per ***sscanf()***. L'interpretazione dei dati in ingresso viene guidata da una stringa di formato, costituita dal parametro ***format***, per le tre funzioni. La stringa di formato contiene degli ***specificatori di conversione***, con cui si determina il tipo degli argomenti variabili che non sono esplicitati nel prototipo delle funzioni.

Per ogni specificatore di conversione contenuto nella stringa di formato, deve esistere un argomento, successivo al parametro ***format***, costituito dal puntatore a una variabile di tipo conforme a quanto indicato dallo specificatore relativo. La conversione per quello specificatore, comporta la memorizzazione del risultato in memoria, in corrispondenza del puntatore relativo. Si osservi l'esempio seguente:

```
int valore;
…
scanf ("%i", &valore);
```

In questo modo si attende l'inserimento, attraverso lo standard input, di un numero intero, da convertire e assegnare così alla variabile ***valore***; Infatti, lo specificatore di conversione '**`%i`**', consente di interpretare un numero intero.

Gli specificatori di conversione devono rispettare la sintassi seguente per la libreria di os32:

```
%[*] [n_ampiezza] [hh|h|l|j|z|t] tipo
```

Come si può vedere, all'inizio può apparire un asterisco, il cui scopo è quello di annullare l'assegnamento del valore a una variabile. In pratica, con l'asterisco il dato corrispondente allo specificatore viene interpretato, ma poi non viene salvato.

Successivamente può apparire un numero che rappresenta l'ampiezza del dato da interpretare, in byte, il cui scopo è quello di limitare la lettura fino a un certo carattere.

Dopo può apparire una sigla, composta da una o più lettere, il cui scopo è quello di modificare la dimensione predefinita della variabile di destinazione. In altri termini, senza questo modificatore si intende che la variabile ricevente debba essere di una certa grandezza, ma con l'aggiunta del «modificatore di lunghezza» si precisa invece qualcosa di diverso. In pratica, il modificatore di lunghezza usato da queste funzioni è equivalente a quello delle funzioni di composizione dell'output.

Al termine dello specificatore di conversione appare una lette-

ra che dichiara come deve essere interpretato il dato in ingresso e, in mancanza del modificatore di lunghezza, indica anche la dimensione della variabile ricevente.

Tipi di conversione.

Simbolo	Tipo di argomento	Conversione applicata
`%…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

interpretare, vengono eliminati automaticamente gli spazi iniziali e finali (tutto ciò che si può considerare spazio, anche il codice di interruzione di riga), quando all'inizio o alla fine non ci sono corrispondenze con specificatori di conversione che possono interpretarli.

Quando la direttiva di interpretazione inizia con uno o più spazi orizzontali, significa che si vogliono ignorare gli spazi a partire dalla posizione corrente nella lettura dei dati in ingresso; inoltre, la presenza di un carattere che non fa parte di uno specificatore di conversione indica che quello stesso carattere deve essere incontrato nell'interpretazione dei dati in ingresso, altrimenti il procedimento di lettura e valutazione si deve interrompere. Se due specificatori di conversione appaiono adiacenti, i dati in ingresso corrispondenti possono essere separati da spazi orizzontali o da spazi verticali (il codice di interruzione di riga).

VALORE RESTITUITO

Le funzioni restituiscono la quantità di elementi in ingresso interpretati e assegnati correttamente: una quantità inferiore al previsto indica pertanto un errore. Se le funzioni restituiscono il valore '**`EOF`**', si tratta di un errore, dovuto eventualmente a un problema di interpretazione del formato o a un problema di accesso al flusso di file da cui deve provenire l'input.

ERRORI

Valore di *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`’ [95.18]

‘`lib/stdio/fscanf.c`’ [95.18.17]

‘`lib/stdio/scanf.c`’ [95.18.31]

‘`lib/stdio/sscanf.c`’ [95.18.36]

VEDERE ANCHE

vfscanf(3) [88.138], *vscanf(3)* [88.138], *vsscanf(3)* [88.138], *printf(3)* [88.91].

88.103 os32: setbuf(3)

«

NOME

‘**`setbuf`**’, ‘**`setvbuf`**’ - modifica della memoria tampone per i flussi di file

SINTASSI

```
#include <stdio.h>
void setbuf (FILE *restrict fp, char *restrict buffer);
int setvbuf (FILE *restrict fp, char *restrict buffer,
             int buf_mode, size_t size);
```

DESCRIZIONE

Le funzioni ***setbuf()*** e ***setvbuf()*** della libreria di os32, non fanno alcunché, perché os32 non gestisce una memoria tampone per i flussi di file.

VALORE RESTITUITO

La funzione ***setvbuf()*** restituisce, in tutti i casi, il valore zero.

FILE SORGENTI

‘`lib/stdio.h`’ [95.18]

‘`lib/stdio/setbuf.c`’ [95.18.32]

‘`lib/stdio/setvbuf.c`’ [95.18.33]

VEDERE ANCHE

fflush(3) [88.31].

88.104 os32: setenv(3)

«

NOME

‘**`setenv`**’, ‘**`unsetenv`**’ - assegnamento o cancellazione di una variabile di ambiente

SINTASSI

```
#include <stdlib.h>
int setenv   (const char *name, const char *value,
              int overwrite);
int unsetenv (const char *name);
```

DESCRIZIONE

La funzione ***setenv()*** crea o assegna un valore a una variabile di ambiente. Se questa variabile esiste già, la modifica del valore assegnatole può avvenire soltanto se l'argomento corrispondente al parametro ***overwrite*** risulta essere diverso da zero; in caso contrario, la modifica non ha luogo.

La funzione ***unsetenv()*** si limita a cancellare la variabile di ambiente specificata come argomento.

VALORE RESTITUITO

Valore	Significato
0	Operazione riuscita.
−1	Operazione fallita. Va verificato l'errore indicato dalla variabile ***errno***.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Argomento non valido.
`ENOMEM`	Memoria insufficiente.

FILE SORGENTI

'`lib/stdlib.h`' [95.19]

‘applic/crt0.mer.s’ [96.1.12]

‘applic/crt0.sep.s’ [96.1.13]

‘lib/stdlib/environment.c’ [95.19.8]

‘lib/stdlib/setenv.c’ [95.19.18]

‘lib/stdlib/unsetenv.c’ [95.19.21]

VEDERE ANCHE

environ(7) [91.1], *getenv(3)* [88.52], *putenv(3)* [88.94].

88.105 os32: setgrent(3)

Vedere *getgrent(3)* [88.53].

88.106 os32: setpwent(3)

Vedere *getpwent(3)* [88.57].

88.107 os32: setvbuf(3)

Vedere *setbuf(3)* [88.103].

88.108 os32: snprintf(3)

Vedere *printf(3)* [88.91].

88.109 os32: sprintf(3)

Vedere *printf(3)* [88.91].

88.110 os32: srand(3)

«

Vedere *rand(3)* [88.97].

88.111 os32: sscanf(3)

«

Vedere *scanf(3)* [88.102].

88.112 os32: stdio(3)

«

NOME

‘**stdio**’ - libreria per la gestione dei file in forma di flussi di file (*stream*)

SINTASSI

```
#include <stdio.h>
```

DESCRIZIONE

Le funzioni di libreria che fanno capo al file di intestazione ‘stdio.h’, consentono di gestire i file in forma di «flussi», rappresentati da puntatori al tipo ‘**FILE**’. Questa gestione si sovrappone a quella dei file in forma di «descrittori», la quale avviene tramite chiamate di sistema. Lo scopo della sovrapposizione dovrebbe essere quello di gestire i file con l’ausilio di una memoria tampone, cosa che però la libreria di os32 non fornisce.

Nella libreria di os32, il tipo ‘**FILE ***’ è un puntatore a una variabile strutturata che contiene solo tre informazioni: il numero del descrittore del file a cui il flusso si associa; lo stato di errore; lo stato di raggiungimento della fine del file.

```
typedef struct {
    int             fdn;        // File descriptor number.
    char            error;      // Error indicator.
    char            eof;        // End of file indicator.
} FILE;
```

Le variabili strutturate necessarie per questa gestione, sono raccolte in un array, dichiarato nel file '`lib/stdio/FILE.c`', con il nome ***_stream[]***, dove per il descrittore di file ***n***, si associano sempre i dati di ***_stream[n]***.

```
FILE _stream[FOPEN_MAX];
```

Così come sono previsti tre descrittori (zero, uno e due) per la gestione di standard input, standard output e standard error, tutti i processi inizializzano l'array ***_stream[]*** con l'abbinamento a tali descrittori, per i primi tre flussi.

```
void
_stdio_stream_setup (void)
{
    _stream[0].fdn   = 0;
    _stream[0].error = 0;
    _stream[0].eof   = 0;

    _stream[1].fdn   = 1;
    _stream[1].error = 0;
    _stream[1].eof   = 0;

    _stream[2].fdn   = 2;
    _stream[2].error = 0;
    _stream[2].eof   = 0;
}
```

Ciò avviene attraverso il codice contenuto nel file '`crt0.s`', dove si chiama la funzione che provvede a tale inizializzazione, conte-

nuta nel file ‘`lib/stdio/FILE.c`’. Per fare riferimento ai flussi predefiniti, si usano i nomi ‘**`stdin`**’, ‘**`stdout`**’ e ‘**`stderr`**’, i quali sono dichiarati nel file ‘`stdio.h`’, come puntatori ai primi tre elementi dell’array ***_stream[]***:

```
#define stdin   (&_stream[0])
#define stdout  (&_stream[1])
#define stderr  (&_stream[2])
```

FILE SORGENTI

‘`lib/sys/types.h`’ [95.26]

‘`lib/stdio.h`’ [95.18]

‘`lib/stdio/FILE.c`’ [95.18.1]

‘`applic/crt0.mer.s`’ [96.1.12]

‘`applic/crt0.sep.s`’ [96.1.13]

VEDERE ANCHE

close(2) [87.10], *open(2)* [87.37], *read(2)* [87.39], *write(2)* [87.64].

88.113 os32: strcat(3)

«

NOME

‘**`strcat`**’, ‘**`strncat`**’ - concatenamento di una stringa a un’altra già esistente

SINTASSI

```
#include <string.h>
char *strcat  (char *restrict dst,
               const char *restrict org);
char *strncat (char *restrict dst,
               const char *restrict org,
               size_t n);
```

DESCRIZIONE

Le funzioni ***strcat()*** e ***strncat()*** copiano la stringa di origine ***org***, aggiungendola alla stringa di destinazione ***dst***, nel senso che la scrittura avviene a partire dal codice di terminazione '`\0`' che viene così sovrascritto. Al termine della copia, viene aggiunto nuovamente il codice di terminazione di stringa '`\0`', nella nuova posizione conclusiva.

Nel caso particolare di ***strncat()***, la copia si arresta al massimo dopo il trasferimento di ***n*** caratteri. Pertanto, la stringa di origine per ***strncat()*** potrebbe anche non essere terminata correttamente, se raggiunge o supera la dimensione di ***n*** caratteri. In ogni caso, nella destinazione viene aggiunto il codice nullo di terminazione di stringa, dopo la copia del carattere ***n***-esimo.

VALORE RESTITUITO

Le due funzioni restituiscono ***dst***.

FILE SORGENTI

'`lib/string.h`' [95.20]

'`lib/string/strcat.c`' [95.20.7]

'`lib/string/strncat.c`' [95.20.16]

VEDERE ANCHE

memccpy(3) [88.77], *memcpy(3)* [88.80], *strcpy(3)* [88.117], *strncpy(3)* [88.117].

88.114 os32: strchr(3)

«

NOME

‘**strchr**’, ‘**strrchr**’ - ricerca di un carattere all’interno di una stringa

SINTASSI

```
#include <string.h>
char *strchr  (const char *string, int c);
char *strrchr (const char *string, int c);
```

DESCRIZIONE

Le funzioni ***strchr()*** e ***strrchr()*** scandiscono la stringa ***string*** alla ricerca di un carattere uguale al valore di ***c***. La funzione ***strchr()*** scandisce a partire da «sinistra», ovvero ricerca la prima corrispondenza con il carattere ***c***, mentre la funzione ***strrchr()*** cerca l’ultima corrispondenza con il carattere ***c***, pertanto è come se scandisse da «destra».

VALORE RESTITUITO

Se le due funzioni trovano il carattere che cercano, ne restituiscono il puntatore, altrimenti restituiscono ‘**NULL**’.

FILE SORGENTI

‘`lib/string.h`’ [95.20]

‘lib/string/strchr.c’ [95.20.8]

‘lib/string/strrchr.c’ [95.20.20]

VEDERE ANCHE

memchr(3) [88.78], *strlen(3)* [88.121], *strpbrk(3)* [88.125], *strspn(3)* [88.127].

88.115 os32: strcmp(3)

NOME

‘**strcmp**’, ‘**strncmp**’ - confronto di due stringhe

SINTASSI

```
#include <string.h>
int strcmp  (const char *string1, const char *string2);
int strncmp (const char *string1, const char *string2,
             size_t n);
int strcoll (const char *string1, const char *string2);
```

DESCRIZIONE

Le funzioni ***strcmp()*** e ***strncmp()*** confrontano due stringhe, nel secondo caso, il confronto avviene al massimo fino al ***n***-esimo carattere.

La funzione ***strcoll()*** dovrebbe eseguire il confronto delle due stringhe tenendo in considerazione la configurazione locale. Tuttavia, os32 non è in grado di gestire le configurazioni locali, pertanto questa funzione coincide esattamente con ***strcmp()***.

VALORE RESTITUITO

Valore	Esito del confronto.
−1	***string1 < string2***
0	***string1 == string2***
+1	***string1 > string2***

FILE SORGENTI

‘`lib/string.h`’ [95.20]

‘`lib/string/strcmp.c`’ [95.20.9]

‘`lib/string/strncmp.c`’ [95.20.17]

‘`lib/string/strcoll.c`’ [95.20.10]

VEDERE ANCHE

memcmp(3) [88.79].

88.116 os32: strcoll(3)

«

Vedere *strcmp(3)* [88.115].

88.117 os32: strcpy(3)

«

NOME

‘**`strcpy`**’, ‘**`strncpy`**’ - copia di una stringa

SINTASSI

```
#include <string.h>
char *strcpy  (char *restrict dst,
               const char *restrict org);
char *strncpy (char *restrict dst,
               const char *restrict org,
               size_t n);
```

DESCRIZIONE

Le funzioni ***strcpy()*** e ***strncpy()***, copiano la stringa ***org***, completa di codice nullo di terminazione, nella destinazione ***dst***. Eventualmente, nel caso di ***strncpy()***, la copia non supera i primi ***n*** caratteri, con l'aggravante che in tal caso, se nei primi ***n*** caratteri non c'è il codice nullo di terminazione delle stringhe, nella destinazione ***dst*** si ottiene una stringa non terminata.

VALORE RESTITUITO

Le funzioni restituiscono ***dst***.

FILE SORGENTI

'`lib/string.h`' [95.20]

'`lib/string/strcpy.c`' [95.20.11]

'`lib/string/strncpy.c`' [95.20.18]

VEDERE ANCHE

memccpy(3) [88.77], *memcpy(3)* [88.80], *memmove(3)* [88.81].

88.118 os32: strcspn(3)

Vedere *strspn(3)* [88.127].

88.119 os32: strdup(3)

NOME

'**`strdup`**' - duplicazione di una stringa

SINTASSI

```
#include <string.h>
char *strdup  (const char *string);
```

DESCRIZIONE

La funzione ***strdup()***, alloca dinamicamente una quantità di memoria, necessaria a copiare la stringa ***string***, quindi esegue tale copia e restituisce il puntatore alla nuova stringa allocata. Tale puntatore può essere usato successivamente per liberare la memoria, con l'ausilio della funzione ***free()***.

VALORE RESTITUITO

La funzione restituisce il puntatore alla nuova stringa ottenuta dalla copia, oppure '**`NULL`**' nel caso non fosse possibile allocare la memoria necessaria.

ERRORI

Valore di ***errno***	Significato
`ENOMEM`	Memoria insufficiente.

FILE SORGENTI

'`lib/string.h`' [95.20]

'`lib/string/strdup.c`' [95.20.13]

VEDERE ANCHE

free(3) [88.76], *malloc(3)* [88.76], *realloc(3)* [88.76].

88.120 os32: strerror(3)

«

NOME

'**`strerror`**' - descrizione di un errore in forma di stringa

SINTASSI

```
#include <string.h>
char *strerror (int errnum);
```

DESCRIZIONE

La funzione ***strerror()*** interpreta il valore ***errnum*** come un errore, di quelli che può rappresentare la variabile ***errno*** del file ‘`errno.h`’.

VALORE RESTITUITO

La funzione restituisce il puntatore a una stringa contenente la descrizione dell’errore, oppure soltanto ‘**`Unknown error`**’, se l’argomento ricevuto non è traducibile.

FILE SORGENTI

‘`lib/errno.h`’ [95.5]

‘`lib/string.h`’ [95.20]

‘`lib/string/strerror.c`’ [95.20.14]

VEDERE ANCHE

errno(3) [88.20], *perror(3)* [88.90].

88.121 os32: strlen(3)

«

NOME

‘**`strlen`**’ - lunghezza di una stringa

SINTASSI

```
#include <string.h>
size_t strlen (const char *string);
```

DESCRIZIONE

La funzione ***strlen()*** calcola la lunghezza della stringa, ovvero la quantità di caratteri che la compone, escludendo il codice nullo di conclusione.

VALORE RESTITUITO

La funzione restituisce la quantità di caratteri che compone la stringa, escludendo il codice '`\0`' finale.

FILE SORGENTI

'`lib/string.h`' [95.20]

'`lib/string/strlen.c`' [95.20.15]

88.122 os32: strncat(3)

«

Vedere *strcat(3)* [88.113].

88.123 os32: strncmp(3)

«

Vedere *strcmp(3)* [88.115].

88.124 os32: strncpy(3)

«

Vedere *strcpy(3)* [88.117].

88.125 os32: strpbrk(3)

«

NOME

'**`strpbrk`**' - scansione di una stringa alla ricerca di un carattere

SINTASSI

```
#include <string.h>
char *strpbrk (const char *string, const char *accept);
```

DESCRIZIONE

La funzione ***strpbrk()*** cerca il primo carattere, nella stringa ***string***, che corrisponda a uno di quelli contenuti nella stringa ***accept***.

VALORE RESTITUITO

Restituisce il puntatore al primo carattere che, nella stringa ***string*** corrisponde a uno di quelli contenuti nella stringa ***accept***. In mancanza di alcuna corrispondenza, restituisce '**`NULL`**'.

FILE SORGENTI

'`lib/string.h`' [95.20]

'`lib/string/strpbrk.c`' [95.20.19]

VEDERE ANCHE

memchr(3) [88.78], *strchr(3)* [88.114], *strstr(3)* [88.128], *strtok(3)* [88.129].

88.126 os32: strrchr(3)

Vedere *strchr(3)* [88.114].

88.127 os32: strspn(3)

«

NOME

‘**strspn**’, ‘**strcspn**’ - scansione di una stringa, limitatamente a un certo insieme di caratteri

SINTASSI

```
#include <string.h>
size_t strspn  (const char *string, const char *accept);
size_t strcspn (const char *string, const char *reject);
```

DESCRIZIONE

La funzione ***strspn()*** scandisce la stringa ***string***, calcolando la lunghezza di questa che contiene, a partire dall’inizio, soltanto caratteri che si trovano nella stringa ***accept***.

La funzione ***strcspn()*** scandisce la stringa ***string***, calcolando la lunghezza di questa che contiene, a partire dall’inizio, soltanto caratteri che non si trovano nella stringa ***reject***.

VALORE RESTITUITO

La funzione ***strspn()*** restituisce la lunghezza della stringa che contiene soltanto caratteri contenuti in ***accept***.

La funzione ***strcspn()*** restituisce la lunghezza della stringa che contiene soltanto caratteri che non sono contenuti in ***reject***.

FILE SORGENTI

‘`lib/string.h`’ [95.20]

‘`lib/string/strspn.c`’ [95.20.21]

‘`lib/string/strcspn.c`’ [95.20.12]

VEDERE ANCHE

memchr(3) [88.78], *strchr(3)* [88.114], *strpbrk(3)* [88.125], *strstr(3)* [88.128], *strtok(3)* [88.129].

88.128 os32: strstr(3)

NOME

‘**strstr**’ - ricerca di una sottostringa

SINTASSI

```
#include <string.h>
char *strstr (const char *string, const char *substring);
```

DESCRIZIONE

La funzione ***strstr()*** scandisce la stringa ***string***, alla ricerca della prima corrispondenza con la stringa ***substring***, restituendo eventualmente il puntatore all’inizio di tale corrispondenza.

VALORE RESTITUITO

Se la ricerca termina con successo, viene restituito il puntatore all’inizio della sottostringa contenuta in ***string***; diversamente viene restituito il puntatore nullo ‘**NULL**’.

FILE SORGENTI

‘lib/string.h’ [95.20]

‘lib/string/strstr.c’ [95.20.22]

VEDERE ANCHE

memchr(3) [88.78], *strchr(3)* [88.114], *strpbrk(3)* [88.125], *strtok(3)* [88.129].

88.129 os32: strtok(3)

«

NOME

'**strtok**' - *string token*, ovvero estrazione di pezzi da una stringa

SINTASSI

```
#include <string.h>
char *strtok (char *restrict string,
              const char *restrict delim);
```

DESCRIZIONE

La funzione ***strtok()*** serve a suddividere una stringa in unità, definite *token*, specificando un elenco di caratteri da intendere come delimitatori, in una seconda stringa. La funzione va usata in fasi successive, fornendo solo inizialmente la stringa da suddividere che continua poi a essere utilizzata se al suo posto viene fornito il puntatore nullo. La funzione restituisce, di volta in volta, il puntatore alla sottostringa contenente l'unità individuata, oppure il puntatore nullo, se non può trovarla.

La funzione deve tenere memoria di un puntatore in modo persistente e deve isolare le unità modificando la stringa originale, inserendo il carattere nullo di terminazione alla fine delle unità individuate.

Quando la funzione viene chiamata indicando al posto della stringa da scandire il puntatore nullo, l'insieme dei delimitatori può essere diverso da quello usato nelle fasi precedenti.

Per comprendere lo scopo della funzione viene utilizzato lo stesso esempio che appare nel documento *ISO/IEC 9899:TC2*, al para-

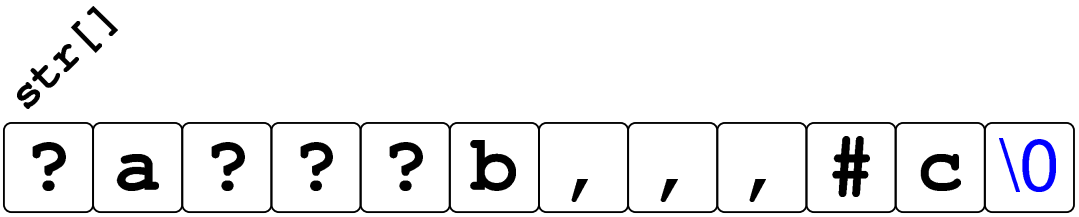
grafo 7.21.5.7, con qualche piccola modifica per poterlo rendere un programma autonomo:

```
#include <stdio.h>
#include <string.h>
int
main (void)
{
  char str[] = "?a???b,,,#c";
  char *t;

  t = strtok (str, "?");          // t punta all'unità "a"
  printf ("strtok: \"%s\"\n", t);
  t = strtok (NULL, ",");         // t punta all'unità "??b"
  printf ("strtok: \"%s\"\n", t);
  t = strtok (NULL, "#,");        // t punta all'unità "c"
  printf ("strtok: \"%s\"\n", t);
  t = strtok (NULL, "?");         // t è un puntatore nullo
  printf ("strtok: \"%s\"\n", t);

  return 0;
}
```

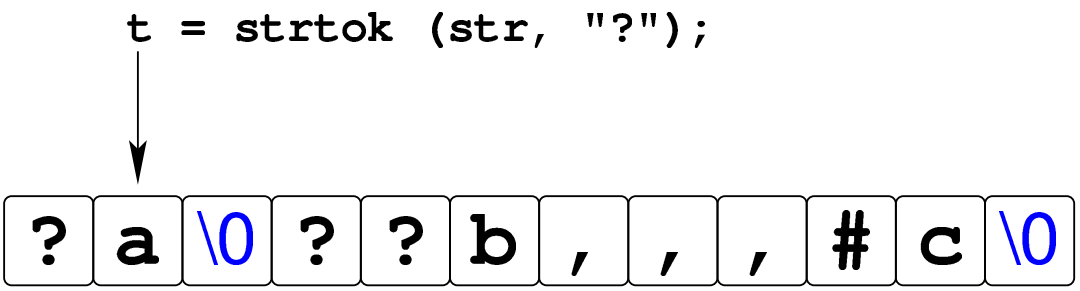
Avviando il programma si ottiene quanto già descritto dai commenti inseriti nel codice:

```
strtok: "a"
strtok: "??b"
strtok: "c"
strtok: "(null)"
```

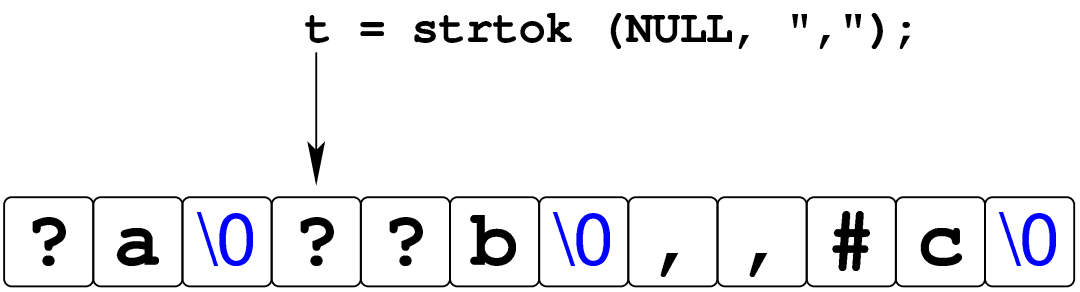
Ciò che avviene nell'esempio può essere schematizzato come segue. Inizialmente la stringa '**str**' ha in memoria l'aspetto seguente:



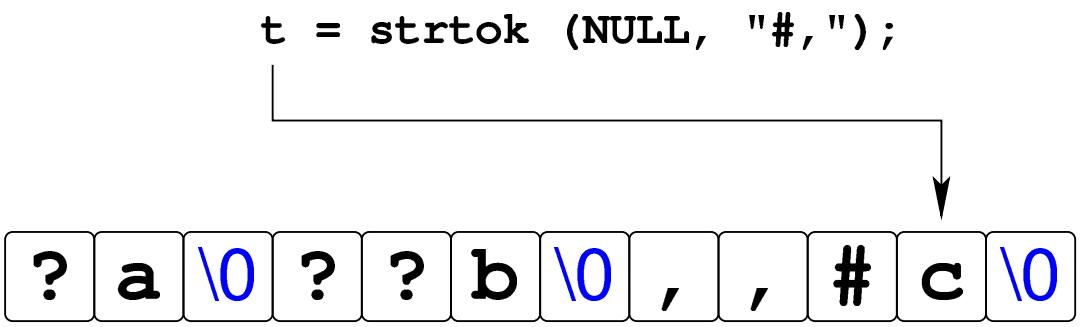
Dopo la prima chiamata della funzione ***strtok()*** la stringa risulta alterata e il puntatore ottenuto raggiunge la lettera '**a**':



Dopo la seconda chiamata della funzione, in cui si usa il puntatore nullo per richiedere una scansione ulteriore della stringa originale, si ottiene un nuovo puntatore che, questa volta, inizia a partire dal quarto carattere, rispetto alla stringa originale, dal momento che il terzo è già stato sovrascritto da un carattere nullo:



La penultima chiamata della funzione ***strtok()*** raggiunge la lettera '**c**' che è anche alla fine della stringa originale:



L'ultimo tentativo di chiamata della funzione non può dare alcun esito, perché la stringa originale si è già conclusa.

VALORE RESTITUITO

La funzione restituisce il puntatore al prossimo «pezzo«, oppure '**NULL**' se non ce ne sono più.

FILE SORGENTI

'lib/string.h' [95.20]

'lib/string/strtok.c' [95.20.23]

VEDERE ANCHE

memchr(3) [88.78], *strchr(3)* [88.114], *strpbrk(3)* [88.125], *strspn(3)* [88.127].

88.130 os32: strtol(3)

«

NOME

'**strtol**', '**strtoul**' - conversione di una stringa in un numero

SINTASSI

```
#include <stdlib.h>
long int strtol (const char *restrict string,
                 char **restrict endptr,
                 int base);

unsigned long int strtoul (const char *restrict string,
                           char **restrict endptr,
                           int base);
```

DESCRIZIONE

Le funzioni ***strtol()*** e ***strtoul()***, convertono la stringa ***string*** in un numero, intendendo la sequenza di caratteri nella base di numerazione indicata come ultimo argomento (***base***). Tuttavia, la base di numerazione potrebbe essere omessa (valore zero) e in tal caso la stringa deve essere interpretata ugualmente in qualche modo: se (dopo un segno eventuale) inizia con zero seguito da un'altra cifra numerica, deve trattarsi di una sequenza ottale; se inizia con zero, quindi appare una lettera «x» deve trattarsi di un numero esadecimale; se inizia con una cifra numerica diversa da zero, deve trattarsi di un numero in base dieci.

La traduzione della stringa ha luogo progressivamente, arrestandosi quando si incontra un carattere incompatibile con la base di numerazione selezionata o stabilita automaticamente. Il valore convertito viene restituito; inoltre, se il puntatore ***endptr*** è valido (diverso da '**`NULL`**'), si assegna a ****endptr*** la posizione raggiunta nella stringa, corrispondente al primo carattere che non può essere convertito. Pertanto, nello stesso modo, se la stringa non può

essere convertita affatto e si può assegnare qualcosa a ***endptr***, alla fine, ***endptr*** corrisponde esattamente a ***string***.

VALORE RESTITUITO

Le funzioni restituiscono il valore tratto dall'interpretazione della stringa, ammesso che sia rappresentabile, altrimenti si ottiene '**`LONG_MIN`**' o '**`LONG_MAX`**', a seconda dei casi, sapendo che occorre consultare la variabile ***errno*** per maggiori dettagli.

ERRORI

Valore di ***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`' [95.19]

'`lib/stdlib/strtol.c`' [95.19.19]

'`lib/stdlib/strtoul.c`' [95.19.20]

88.131 os32: strtoul(3)

«

Vedere *strtol(3)* [88.130].

88.132 os32: strxfrm(3)

«

NOME

‘**strxfrm**’ - *string transform*, ovvero trasformazione di una stringa

SINTASSI

```
#include <string.h>
size_t strxfrm (char *restrict dst,
                const char *restrict org,
                size_t n);
```

DESCRIZIONE

Lo scopo della funzione ***strxfrm()*** sarebbe quello di copiare la stringa ***org***, sovrascrivendo ***dst***, fino a un massimo di ***n*** caratteri nella destinazione, ma applicando una trasformazione relativa alla configurazione locale.

os32 non gestisce la configurazione locale, pertanto questa funzione si comporta in modo simile a ***strncpy()***, con una differenza in ciò che viene restituito.

VALORE RESTITUITO

La funzione restituisce la quantità di byte utilizzati per contenere la trasformazione in ***dst***, senza però contare il carattere nullo di terminazione.

FILE SORGENTI

‘lib/string.h’ [95.20]

‘lib/string/strxfrm.c’ [95.20.24]

VEDERE ANCHE

memcmp(3) [88.79], *strcmp(3)* [88.115], *strcoll(3)* [88.115].

88.133 os32: ttyname(3)

«

NOME

'**ttyname**' - determinazione del percorso del file di dispositivo di un terminale aperto

SINTASSI

```
#include <unistd.h>
char *ttyname (int fdn);
```

DESCRIZIONE

La funzione ***ttyname()*** richiede come unico argomento il numero che identifica il descrittore di un file. Ammesso che tale descrittore si riferisca a un terminale, la funzione restituisce il puntatore a una stringa che rappresenta il percorso del file di dispositivo corrispondente.

La stringa in questione viene modificata se si usa la funzione in altre occasioni.

VALORE RESTITUITO

La funzione restituisce il puntatore a una stringa che descrive il percorso del file di dispositivo, presunto, del terminale aperto con il numero ***fdn***. Se non si tratta di un terminale, si ottiene un errore. In ogni caso, se la funzione non può restituire un'informazione corretta, produce semplicemente il puntatore nullo e aggiorna la variabile ***errno***.

ERRORI

Valore di *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`’ [95.30]

‘`lib/unistd/ttyname.c`’ [95.30.41]

VEDERE ANCHE

stat(2) [87.55], *isatty(3)* [88.69].

88.134 os32: unsetenv(3)

«

Vedere *setenv(3)* [88.104].

88.135 os32: vfprintf(3)

«

Vedere *vprintf(3)* [88.137].

88.136 os32: vfscanf(3)

«

Vedere *vfscanf(3)* [88.138].

88.137 os32: vprintf(3)

«

NOME

‘**`vprintf`**’, ‘**`vfprintf`**’, ‘**`vsprintf`**’, ‘**`vsnprintf`**’ - composizione dei dati per la visualizzazione

SINTASSI

```
#include <stdarg.h>
#include <stdio.h>
int vprintf   (char *restrict format, va_list arg);
int vfprintf  (FILE *fp, char *restrict format,
               va_list arg);
int vsnprintf (char *restrict string, size_t size,
               const char *restrict format, va_list ap);
int vsprintf  (char *string, char *restrict format,
               va_list arg);
```

DESCRIZIONE

Le funzioni del gruppo '**v...printf()**' hanno in comune lo scopo di comporre dei dati in forma di stringa, generalmente per la visualizzazione, o comunque per la fruizione a livello umano.

I dati in ingresso possono essere vari e vengono comunicati attraverso un puntatore di tipo '**va_list**'. Per quantificare e qualificare questi dati in ingresso, la stringa a cui punta il parametro ***format***, deve contenere degli ***specificatori di conversione***, oltre eventualmente ad altri caratteri. Pertanto, queste funzioni, prendono la stringa a cui punta ***format***, la interpretano e determinano come scandire gli argomenti a cui fa riferimento il puntatore ***arg***, quindi producono un'altra stringa, composta dalla stringa precedente, sostituendo gli specificatori di conversione con i dati a cui questi si riferiscono, secondo una forma di conversione definita dagli specificatori stessi.

In generale, le funzioni '**v...printf()**' servono per realizzare le altre funzioni '...**printf()**', le quali invece ricevono gli argo-

menti variabili direttamente. Per esempio, la funzione ***printf()*** può essere realizzata utilizzando in pratica ***vprintf()***:

```
#include <stdio.h>
#include <stdarg.h>
int
printf (char *restrict format, ...)
{
    va_list ap;
    va_start (ap, format);
    return (vprintf (format, ap));
}
```

Si veda *printf(3)* [88.91], per la descrizione di come va predisposta la stringa ***format***. Nella realizzazione di os32, di tutte queste funzioni, quella che compie effettivamente il lavoro di interpretazione della stringa di formato e che in qualche modo viene chiamata da tutte le altre, è soltanto ***vsnprintf()***.

VALORE RESTITUITO

Le funzioni restituiscono la quantità di caratteri utilizzati nella composizione della nuova stringa, escluso il carattere nullo di terminazione.

FILE SORGENTI

‘`lib/stdarg.h`’ [95.1.10]

‘`lib/stdio.h`’ [95.18]

‘`lib/stdio/FILE.c`’ [95.18.1]

‘`lib/stdio/vfprintf.c`’ [95.18.37]

‘`lib/stdio/vprintf.c`’ [95.18.40]

‘`lib/stdio/vsprintf.c`’ [95.18.43]

‘`lib/stdio/vsnprintf.c`’ [95.18.42]

VEDERE ANCHE

fprintf(3) [88.91], *printf(3)* [88.91], *sprintf(3)* [88.91], *snprintf(3)* [88.91], *scanf(3)* [88.102].

88.138 os32: vscanf(3)

«

NOME

‘**vscanf**’, ‘**vfscanf**’, ‘**vsscanf**’ - interpretazione dell’input e conversione

SINTASSI

```
#include <stdarg.h>
#include <stdio.h>
int vscanf  (const char *restrict format, va_list ap);
int vfscanf (FILE *restrict fp, const char *restrict format,
             va_list ap);
int vsscanf (const char *string, const char *restrict format,
             va_list ap);
```

DESCRIZIONE

Le funzioni del gruppo ‘**v...scanf()**’ hanno in comune lo scopo di interpretare dei dati, forniti in forma di stringa, convertendoli opportunamente.

I dati in ingresso sono costituiti da una sequenza di caratteri, la quale viene fornita tramite lo standard input per ***vscanf()***, tramite il flusso di file ***fp*** per ***vfscanf()***, oppure tramite la stringa ***string*** per ***vsscanf()***. L’interpretazione dei dati in ingresso viene guidata da una stringa di formato, costituita dal parametro ***format***, per le

tre funzioni. La stringa di formato contiene degli ***specificatori di conversione***, con cui si determina il tipo degli argomenti variabili a cui punta inizialmente ***ap***.

Queste funzioni servono per realizzare in pratica quelle corrispondenti che hanno nomi privi della lettera «v» iniziale. Per esempio, per ottenere ***scanf()*** si può utilizzare ***vscanf()***:

```
#include <stdio.h>
#include <stdarg.h>
int
scanf (const char *restrict format, ...)
{
    va_list ap;
    va_start (ap, format);
    return vscanf (format, ap);
}
```

Il modo in cui va predisposta la stringa di formato (***format***) è descritto in *scanf(3)* [88.102]. La funzione più importante di questo gruppo, in quanto svolge effettivamente il lavoro di interpretazione e viene chiamata, più o meno indirettamente, da tutte le altre, è ***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`’ [95.18]

‘`lib/stdio/vfscanf.c`’ [95.18.38]

‘`lib/stdio/vscanf.c`’ [95.18.41]

‘`lib/stdio/vsscanf.c`’ [95.18.44]

‘`lib/stdio/vfsscanf.c`’ [95.18.39]

VEDERE ANCHE

fscanf(3) [88.102], *scanf(3)* [88.102], *sscanf(3)* [88.102], *printf(3)* [88.91].

88.139 os32: vsnprintf(3)

Vedere *vprintf(3)* [88.137].

88.140 os32: vsprintf(3)

«

Vedere *vprintf(3)* [88.137].

88.141 os32: vsscanf(3)

«

Vedere *vsscanf(3)* [88.138].

Sezione 4: file speciali

89.1 os32: console(4)

NOME

‘/dev/console’ - file di dispositivo che rappresenta la console e le console virtuali

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘/dev/console’	file di dispositivo a caratteri	5	255	0644_8
‘/dev/console0’	file di dispositivo a caratteri	5	0	0644_8
‘/dev/console1’	file di dispositivo a caratteri	5	1	0644_8
‘/dev/console2’	file di dispositivo a caratteri	5	2	0644_8
‘/dev/console3’	file di dispositivo a caratteri	5	3	0644_8

DESCRIZIONE

Il file di dispositivo ‘/dev/console’ rappresenta la console virtuale attiva in un certo momento; i file ‘/etc/console***n***’ rappresentano la console virtuale ***n***, dove ***n*** va da zero a quattro.

I permessi di accesso a questi file di dispositivo sono limitati in modo da consentire solo al proprietario di accedere in scrittura. Tuttavia, per i file di dispositivo usati effettivamente come terminali di controllo, i permessi e la proprietà sono gestiti automaticamente dai programmi ‘**getty**’ e ‘**login**’.

VEDERE ANCHE

MAKEDEV(8) [92.6], *tty(4)* [89.14].

89.2 os32: ata(4)

«

NOME

‘`/dev/ata`***n***’ - file di dispositivo per le unità di memorizzazione a disco ATA.

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘`/dev/ata0`’	file di dispositivo a blocchi	6	0	0644_8
‘`/dev/ata1`’	file di dispositivo a blocchi	6	1	0644_8
‘`/dev/ata2`’	file di dispositivo a blocchi	6	2	0644_8
‘`/dev/ata3`’	file di dispositivo a blocchi	6	3	0644_8
‘`/dev/ata4`’	file di dispositivo a blocchi	6	4	0644_8
‘`/dev/ata5`’	file di dispositivo a blocchi	6	5	0644_8
‘`/dev/ata6`’	file di dispositivo a blocchi	6	6	0644_8
‘`/dev/ata7`’	file di dispositivo a blocchi	7	6	0644_8

DESCRIZIONE

I file di dispositivo ‘`/dev/ata`***n***’ rappresentano, ognuno, un’unità di memorizzazione a disco ATA. La prima unità è ‘`/dev/ata0`’, quelle successive procedono con la numerazione.

os32 gestisce solo unità a disco ATA; inoltre, non è ammissibile la suddivisione in partizioni.

VEDERE ANCHE

MAKEDEV(8) [92.6].

89.3 os32: kmem_arp(4)

NOME

‘`/dev/kmem_arp`’ - accesso alla memoria del kernel contenente la tabella ARP

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘`/dev/kmem_arp`’	file di dispositivo a caratteri	4	6	0444_8

DESCRIZIONE

Il file di dispositivo ‘`/dev/kmem_arp`’ consente di accedere in lettura all’area di memoria che, nel kernel, rappresenta la tabella ARP. La tabella ARP è un array di ‘**`ARP_MAX_ITEMS`**’ elementi, di tipo ‘**`arp_t`**’, secondo le definizioni contenute nel file ‘`kernel/net/arp.h`’.

VEDERE ANCHE

MAKEDEV(8) [92.6], *kmem_ps(4)* [89.8], *kmem_mmp(4)* [89.6], *kmem_sb(4)* [89.10], *kmem_inode(4)* [89.5], *kmem_file(4)* [89.4].

89.4 os32: kmem_file(4)

«

NOME

‘`/dev/kmem_file`’ - accesso alla memoria del kernel contenente la tabella dei file

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘`/dev/kmem_file`’	file di dispositivo a caratteri	4	5	0444_8

DESCRIZIONE

Il file di dispositivo ‘`/dev/kmem_file`’ consente di accedere in lettura all’area di memoria che, nel kernel, rappresenta la tabella dei file. La tabella dei file è un array di ‘**`FILE_MAX_SLOTS`**’ elementi, di tipo ‘**`file_t`**’, secondo le definizioni contenute nel file ‘`kernel/fs.h`’.

VEDERE ANCHE

MAKEDEV(8) [92.6], *kmem_ps(4)* [89.8], *kmem_mmp(4)* [89.6], *kmem_sb(4)* [89.10], *kmem_inode(4)* [89.5].

89.5 os32: kmem_inode(4)

«

NOME

‘/dev/kmem_inode’ - accesso alla memoria del kernel contenente la tabella degli inode

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘/dev/kmem_inode’	file di dispositivo a caratteri	4	4	0444_8

DESCRIZIONE

Il file di dispositivo ‘/dev/kmem_inode’ consente di accedere in lettura all’area di memoria che, nel kernel, rappresenta la tabella degli inode. La tabella degli inode è un array di ‘**INODE_MAX_SLOTS**’ elementi, di tipo ‘**inode_t**’, secondo le definizioni contenute nel file ‘kernel/fs.h’.

VEDERE ANCHE

MAKEDEV(8) [92.6], *kmem_ps(4)* [89.8], *kmem_mmp(4)* [89.6], *kmem_sb(4)* [89.10], *kmem_file(4)* [89.4].

89.6 os32: kmem_mmp(4)

«

NOME

‘/dev/kmem_mmp’ - accesso alla memoria del kernel contenente la mappa di utilizzo della memoria

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘/dev/kmem_mmp’	file di dispositivo a caratteri	4	2	0444_8

DESCRIZIONE

Il file di dispositivo ‘/dev/kmem_mmp’ consente di accedere in lettura all’area di memoria che, nel kernel, rappresenta la mappa di utilizzo della memoria.

VEDERE ANCHE

MAKEDEV(8) [92.6], *kmem_ps(4)* [89.8], *kmem_sb(4)* [89.10], *kmem_inode(4)* [89.5], *kmem_file(4)* [89.4].

89.7 os32: kmem_net(4)

«

NOME

‘/dev/kmem_net’ - accesso alla memoria del kernel contenente la tabella delle interfacce

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘/dev/kmem_net’	file di dispositivo a caratteri	4	7	0444_8

DESCRIZIONE

Il file di dispositivo '`/dev/kmem_net`' consente di accedere in lettura all'area di memoria che, nel kernel, rappresenta la tabella delle interfacce di rete. La tabella delle interfacce è un array di '**`NET_MAX_DEVICES`**' elementi, di tipo '**`net_t`**', secondo le definizioni contenute nel file '`kernel/net.h`'.

VEDERE ANCHE

MAKEDEV(8) [92.6], *kmem_ps(4)* [89.8], *kmem_mmp(4)* [89.6], *kmem_sb(4)* [89.10], *kmem_inode(4)* [89.5], *kmem_file(4)* [89.4], *kmem_arp(4)* [89.3].

89.8 os32: kmem_ps(4)

NOME

'`/dev/kmem_ps`' - accesso alla memoria del kernel contenente lo stato dei processi

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
'`/dev/kmem_ps`'	file di dispositivo a caratteri	4	1	0444_8

DESCRIZIONE

Il file di dispositivo '`/dev/kmem_ps`' consente di accedere in lettura all'area di memoria che, nel kernel, rappresenta la tabella dei processi. La tabella dei processi è un array di '**`PROCESS_MAX`**' elementi, di tipo '**`proc_t`**', secondo le definizioni contenute

nel file ‘`kernel/proc.h`’. Questo meccanismo viene usato dal programma ‘**`ps`**’ per leggere e visualizzare lo stato dei processi.

VEDERE ANCHE

MAKEDEV(8) [92.6], *kmem_mmp(4)* [89.6], *kmem_sb(4)* [89.10], *kmem_inode(4)* [89.5], *kmem_file(4)* [89.4].

89.9 os32: kmem_route(4)

«

NOME

‘`/dev/kmem_route`’ - accesso alla memoria del kernel contenente la tabella degli instradamenti

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘`/dev/kmem_route`’	file di dispositivo a caratteri	4	8	0444_8

DESCRIZIONE

Il file di dispositivo ‘`/dev/kmem_route`’ consente di accedere in lettura all’area di memoria che, nel kernel, rappresenta la tabella degli instradamenti. La tabella degli instradamenti è un array di ‘**`ROUTE_MAX_ROUTES`**’ elementi, di tipo ‘**`route_t`**’, secondo le definizioni contenute nel file ‘`kernel/net/route.h`’.

VEDERE ANCHE

MAKEDEV(8) [92.6], *kmem_ps(4)* [89.8], *kmem_mmp(4)* [89.6], *kmem_sb(4)* [89.10], *kmem_inode(4)* [89.5], *kmem_file(4)* [89.4], *kmem_arp(4)* [89.3], *kmem_net(4)* [89.7].

89.10 os32: kmem_sb(4)

NOME

'/dev/kmem_sb' - accesso alla memoria del kernel contenente la tabella dei super blocchi

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
'/dev/kmem_sb'	file di dispositivo a caratteri	4	3	0444_8

DESCRIZIONE

Il file di dispositivo '/dev/kmem_sb' consente di accedere in lettura all'area di memoria che, nel kernel, rappresenta la tabella dei super blocchi. La tabella dei super blocchi è un array di '**SB_MAX_SLOTS**' elementi, di tipo '**sb_t**', secondo le definizioni contenute nel file 'kernel/fs.h'.

VEDERE ANCHE

MAKEDEV(8) [92.6], *kmem_ps(4)* [89.8], *kmem_mmp(4)* [89.6], *kmem_inode(4)* [89.5], *kmem_file(4)* [89.4].

89.11 os32: mem(4)

NOME

'/dev/mem' - file di dispositivo per l'acceso alla memoria del processo

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘/dev/mem’	file di dispositivo a caratteri	1	1	0444_8

DESCRIZIONE

Il file di dispositivo ‘/dev/mem’ consente di leggere la memoria del processo.

VEDERE ANCHE

MAKEDEV(8) [92.6].

89.12 os32: null(4)

«

NOME

‘/dev/null’ - file di dispositivo per la distruzione dei dati

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘/dev/null’	file di dispositivo a caratteri	1	2	0666_8

DESCRIZIONE

Il file di dispositivo ‘/dev/null’ appare in lettura come un file completamente vuoto, mentre in scrittura è un file in cui si può scrivere indefinitivamente, perdendo però i dati che vi si immettono.

VEDERE ANCHE

MAKEDEV(8) [92.6], *zero(4)* [89.15].

89.13 os32: port(4)

«

NOME

‘/dev/port’ - file di dispositivo per accedere alle porte di I/O

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘/dev/port’	file di dispositivo a caratteri	1	3	0644_8

DESCRIZIONE

Il file di dispositivo ‘/dev/port’ consente di accedere alle porte di I/O. Tali porte consentono di leggere uno o al massimo due byte, pertanto la dimensione della lettura può essere ‘**(size_t) 1**’ oppure ‘**(size_t) 2**’. Per selezionare l’indirizzo della porta occorre posizionare il riferimento interno al file a un indirizzo pari a quello della porta, prima di eseguire la lettura o la scrittura.

VEDERE ANCHE

MAKEDEV(8) [92.6], *mem(4)* [89.11].

89.14 os32: tty(4)

«

NOME

‘/dev/tty’ - file di dispositivo che rappresenta il terminale di controllo del processo

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘/dev/tty’	file di dispositivo a caratteri	2	0	0666_8

DESCRIZIONE

Il file di dispositivo ‘/dev/tty’ rappresenta il terminale di controllo del processo; in altri termini, il processo che accede al file ‘/dev/tty’, raggiunge il proprio terminale di controllo.

VEDERE ANCHE

MAKEDEV(8) [92.6], *console(4)* [89.1].

89.15 os32: zero(4)

«

NOME

‘/dev/zero’ - file di dispositivo per la produzione del valore zero

CONFIGURAZIONE

File	Tipo	Numero primario	Numero secondario	Permessi
‘/dev/zero’	file di dispositivo a caratteri	1	4	0666_8

DESCRIZIONE

Il file di dispositivo ‘/dev/zero’ appare in lettura come un file di lunghezza indefinita, contenente esclusivamente il valore zero (lo zero binario), mentre in scrittura è un file in cui si può scrivere indefinitivamente, perdendo però i dati che vi si immettono.

VEDERE ANCHE

MAKEDEV(8) [92.6], *null(4)* [89.12].

90.1 os32: group(5)

NOME

'/etc/group' - elenco dei gruppi

DESCRIZIONE

Il file '/etc/group' contiene l'elenco dei gruppi di utenti del sistema, uno per ogni riga. Le righe sono divise in quattro campi, delimitati con il carattere due punti (:), come nell'esempio seguente, che rappresenta l'impostazione predefinita di os32:

```
root:x:0:
user:y:233:
```

I campi hanno il significato descritto nell'elenco seguente:

1. nominativo utente;
2. parola d'ordine, ma non usato, anche se **è comunque necessario scriverci qualcosa**;
3. numero GID, ovvero il numero del gruppo;
4. elenco di utenti aggregati, separati da una virgola, ma questa informazione non viene usata da os32.

Il file deve essere accessibile in lettura a tutti gli utenti.

VEDERE ANCHE

login(1) [86.14].

90.2 os32: inittab(5)

«

NOME

‘/etc/inittab’ - configurazione di ‘**init**’

DESCRIZIONE

Il file ‘/etc/inittab’ contiene la configurazione di ‘**init**’, per la definizione dei processi da avviare per la messa in funzione del sistema operativo. Il file può contenere dei commenti, preceduti dal carattere «#» e «voci» costituite da righe suddivise in quattro campi, separati da due punti (:), come nell’esempio seguente:

```
c0:1:respawn:/bin/getty /dev/console0
c1:1:respawn:/bin/getty /dev/console1
c2:1:respawn:/bin/getty /dev/console2
c3:1:respawn:/bin/getty /dev/console3
```

I campi hanno il significato descritto nell’elenco seguente:

1. codice che identifica univocamente la voce;
2. i livelli di esecuzione per cui la voce è valida;
3. l’azione da compiere sulla voce;
4. il programma da avviare, con tutte le opzioni e gli argomenti necessari.

Il programma ‘**init**’ di os32 non distingue i livelli di esecuzione e considera soltanto l’azione ‘**respawn**’, con la quale si intende che ‘**init**’ debba riavviare il processo, quando questo muore, o comunque quando muore quel processo che ha preso il suo posto.

VEDERE ANCHE

init(8) [92.4], *getty(8)* [92.2], *login(1)* [86.14].

90.3 os32: issue(5)

NOME

‘/etc/issue’ - messaggio che precede ‘**login**’

DESCRIZIONE

Il file ‘/etc/issue’ viene visualizzato da ‘**getty**’, prima dell’avvio di ‘**login**’. Il contenuto predefinito di questo file, per os32, è il seguente:

```
os32: a basic os. [Ctrl q], [Ctrl r], [Ctrl s], [Ctrl t] to change console.
```

Il programma ‘**getty**’ di os32 non è in grado di interpretare il contenuto del file, pertanto lo visualizza letteralmente; tuttavia, ‘**getty**’ mostra, indipendentemente dalla presenza e dal contenuto del file ‘/etc/issue’, delle informazioni sul terminale per il quale è in funzione.

VEDERE ANCHE

getty(8) [92.2].

90.4 os32: passwd(5)

NOME

‘/etc/passwd’ - elenco delle utenze

DESCRIZIONE

Il file ‘/etc/passwd’ contiene l’elenco degli utenti del sistema, uno per ogni riga. Le righe sono divise in sette campi, delimitati con il carattere due punti (:), come nell’esempio seguente, che rappresenta l’impostazione predefinita di os32:

```
root:ciao:0:0:root:/root:/bin/shell
user:ciao:1001:1001:test user:/home/user:/bin/shell
```

I campi hanno il significato descritto nell'elenco seguente:

1. nominativo utente;
2. parola d'ordine, in chiaro, per l'identificazione con il programma '**`login`**';
3. numero UID, ovvero il numero dell'utente;
4. numero GID, ovvero il numero del gruppo;
5. descrizione dell'utenza;
6. shell.

Trattandosi di un sistema operativo elementare, la parola d'ordine appare in chiaro nel secondo campo, senza altri accorgimenti. Inoltre, il file deve essere accessibile in lettura a tutti gli utenti.

VEDERE ANCHE

login(1) [86.14].

Sezione 7: varie

91.1 os32: environ(7)

NOME

‘**environ**’ - ambiente del processo elaborativo

SINTASSI

```
extern char **environ;
```

DESCRIZIONE

La variabile ***environ***, dichiarata nel file ‘`unistd.h`’, punta a un array di stringhe, ognuna delle quali rappresenta una variabile di ambiente, con il valore a lei assegnato. Pertanto, il contenuto di queste stringhe ha una forma del tipo ‘***nome=valore***’. Per esempio ‘**HOME=/home/user**’.

In generale, l’accesso diretto ai contenuti di questo array non è conveniente, in quanto sono disponibili delle funzioni che facilitano la gestione di questi dati in forma di variabili di ambiente.

Dal momento che le funzioni di accesso alle informazioni sulle variabili di ambiente sono definite nel file ‘`stdlib.h`’, la gestione effettiva dell’array di stringhe a cui punta ***environ*** è inserita nei file contenuti nella directory ‘`lib/stdlib/`’ di os32. Per la precisione, nel file ‘`lib/stdlib/environment.c`’ si dichiara l’array di caratteri ***_environment_table[][]*** e array di puntatori a caratteri ***_environment[]***:

```
char  _environment_table[ARG_MAX/32][ARG_MAX/16];
char *_environment[ARG_MAX/32+1];
```

L'array ***_environment_table[][]*** viene inizializzato con lo stato delle variabili di ambiente ereditate con l'avvio del processo; inoltre, all'array ***_environment[]*** vengono assegnati i puntatori alle varie stringhe che si possono estrapolare da ***_environment_table[]***. Questo lavoro iniziale avviene per opera della funzione ***_environment_setup()***, la quale viene avviata a sua volta dal file '`crt0.s`'. Successivamente, nello stesso file '`crt0.s`', viene copiato l'indirizzo dell'***_environment[]*** nella variabile ***environ***, di cui sopra.

FILE SORGENTI

'`lib/unistd.h`' [95.30]

'`lib/stdlib.h`' [95.19]

'`lib/unistd/environ.c`' [95.30.9]

'`applic/crt0.mer.s`' [96.1.12]

'`applic/crt0.sep.s`' [96.1.13]

'`lib/stdlib/environment.c`' [95.19.8]

'`lib/stdlib/getenv.c`' [95.19.10]

'`lib/stdlib/putenv.c`' [95.19.15]

'`lib/stdlib/setenv.c`' [95.19.18]

'`lib/stdlib/unsetenv.c`' [95.19.21]

VEDERE ANCHE

getenv(3) [88.52], *putenv(3)* [88.94], *setenv(3)* [88.104], *unsetenv(3)* [88.104].

91.2 os32: socket(7)

«

NOME

‘**socket**’ - gestione dei socket

DESCRIZIONE

Il sistema os32 gestisce esclusivamente socket di dominio Internet, con indirizzi IPv4. Per creare e gestire una connessione si usano le funzioni seguenti:

Funzione	Scopo
`socket()`	Crea la terminazione locale di una connessione, ma senza attribuire indirizzi o porte [87.54].
`connect()`	Connette un socket locale a un socket remoto [87.11].
`bind()`	Attribuisce a un socket locale un indirizzo [87.4].
`listen()`	Mette un socket locale in ascolto, attendendo una richiesta di connessione da un socket remoto [87.31].
`accept()`	Recepisce una richiesta di connessione ricevuta da un socket locale, generando con questa un nuovo socket locale connesso a quello da cui ha avuto origine la richiesta [87.3].
`send()`	Invia dati attraverso una connessione [87.45].
`recvfrom()`	Riceve dati attraverso una connessione [87.40].
`close()`	Chiude un socket locale [87.10].

Quando si crea un socket per poi contattare un socket remoto in attesa di connessioni, è possibile utilizzare le funzioni ***socket()*** e ***connect()***, senza avvalersi di ***bind()***, lasciando che le informazioni mancanti nel socket locale siano determinate o fissate automaticamente.

Quando si crea un socket da mettere in ascolto, dopo la funzione ***socket()*** è necessario utilizzare ***bind()*** per determinare i parametri locali; poi, se si tratta del protocollo TCP si usa la funzione ***listen()*** per attendere una richiesta di connessione da un socket remoto. Quindi, per recepire una richiesta di connessione si usa la funzione ***accept()***, con cui si genera un nuovo socket locale connesso a quello remoto che ha emesso la richiesta.

Quando il socket deve essere in ascolto, ma in attesa di pacchetti UDP, dopo l'utilizzo della funzione ***bind()*** si può passare subito all'uso di ***recvfrom()***.

VEDERE ANCHE

socket(2) [87.54], *accept(2)* [87.3], *bind(2)* [87.4], *connect(2)* [87.11], *listen(2)* [87.31].

91.3 os32: undocumented(7)

«

Questa sezione ha il solo scopo di raccogliere i riferimenti ipertestuali dei listati che, per qualche ragione, sono privi di una documentazione specifica.

Sezione 8: comandi per l'amministrazione del sistema

92.1 os32: arp(8)

NOME

'**arp**' - mostra la tabella ARP

SINTASSI

```
arp
```

DESCRIZIONE

Il programma '**arp**' consente di visualizzare la tabella ARP (*Address resolution protocol*, ma senza la possibilità di poter-vi intervenire per modificarla. Pertanto, l'uso del programma è ammissibile per qualunque utente.

os32 fornisce l'accesso alla tabella ARP attraverso il file di dispositivo '`/dev/kmem_arp`', da cui attinge il programma '**arp**'.

FILE

'`/dev/kmem_arp`'
: È il file di dispositivo attraverso il quale è possibile leggere la tabella ARP del kernel di os32.

FILE SORGENTI

'`applic/crt0.mer.s`' [96.1.12]

'`applic/crt0.sep.s`' [96.1.13]

'`applic/arp.c`' [96.1.4]

VEDERE ANCHE

kmem_arp(4) [89.3], *ipconfig(8)* [92.5], *route(8)* [92.9].

92.2 os32: getty(8)

«

NOME

‘**getty**’ - predisposizione di un terminale e avvio di ‘**login**’

SINTASSI

```
getty terminale
```

DESCRIZIONE

Il programma ‘**getty**’ viene avviato da ‘**init**’ per predisporre il terminale, ripristinando anche i permessi predefiniti, e per avviare successivamente il programma ‘**login**’. Prima di avviare ‘**login**’, ‘**getty**’ visualizza il contenuto del file ‘`/etc/issue`’, se disponibile, inoltre mostra almeno l’indicazione del terminale attuale. Va osservato che questa realizzazione di ‘**getty**’ lascia a ‘**login**’ il compito di chiedere l’inserimento del nominativo utente.

FILE

‘`/etc/issue`’
: ‘**getty**’ visualizza il contenuto di questo file prima di avviare ‘**login**’.

FILE SORGENTI

‘`applic/crt0.mer.s`’ [96.1.12]

‘`applic/crt0.sep.s`’ [96.1.13]

'applic/getty.c' [96.1.16]

VEDERE ANCHE

login(1) [86.14], *issue(5)* [90.3].

92.3 os32: http(8)

NOME

'**http**' - servente HTTP elementare

SINTASSI

```
http porta directory_radice
```

DESCRIZIONE

Il programma '**http**' si comporta come servente HTTP, in ascolto nella porta indicata come primo argomento, per pubblicare i file che si trovano a partire dalla directory specificata come secondo argomento della riga di comando.

Dal momento che os32 non dispone di una shell in grado di interpretare script, il modo migliore per avviare il servizio è quello di inserire la richiesta di avvio del programma '**http**' nel file '/etc/inittab', con una direttiva simile a quella seguente:

```
s0:1:respawn:/bin/http 80 /var/www
```

FILE SORGENTI

'applic/crt0.mer.s' [96.1.12]

'applic/crt0.sep.s' [96.1.13]

'applic/http.c' [96.1.17]

VEDERE ANCHE

inittab(5) [90.2], *arp(8)* [92.1], *ipconfig(8)* [92.9], *route(8)* [92.9], *ping(8)* [92.8], *nc(8)* [86.20].

92.4 os32: init(8)

«

NOME

‘**init**’ - progenitore di tutti gli altri processi

SINTASSI

```
init
```

DESCRIZIONE

Il programma ‘**init**’ viene avviato dal kernel (deve trattarsi precisamente del file ‘`/bin/init`’) come primo e unico processo figlio del kernel stesso. Pertanto, ‘**init**’ deve assumere il numero PID uno.

Questa realizzazione di ‘**init**’ si limita a leggere il file ‘`/etc/inittab`’ per determinare quali programmi figli avviare, senza poter distinguere da diversi livelli di esecuzione. In pratica, all’interno di questo file si indica l’uso di ‘**getty**’, per la gestione dei terminali disponibili.

FILE

‘`/etc/inittab`’

Contiene l’indicazione dei processi che ‘**init**’ deve avviare.

DIFETTI

Con os32 non è possibile associare ai segnali un’azione diversa da quella predefinita; quindi ‘**init**’ non può essere informato

dell'intenzione di arrestare il sistema. Pertanto, tale funzionalità non è stata realizzata nella versione di '**init**' di os32.

FILE SORGENTI

'applic/crt0.mer.s' [96.1.12]

'applic/crt0.sep.s' [96.1.13]

'applic/init.c' [96.1.18]

VEDERE ANCHE

inittab(5) [90.2].

92.5 os32: ipconfig(8)

NOME

'**ipconfig**' - mostra la configurazione delle interfacce di rete

SINTASSI

```
ipconfig
```

DESCRIZIONE

Il programma '**ipconfig**' consente di visualizzare la tabella delle interfacce di rete, con la loro configurazione, ma senza la possibilità di potervi intervenire per modificarla (la configurazione delle interfacce di rete avviene esclusivamente attraverso le opzioni di avvio del kernel). Pertanto, l'uso del programma è ammissibile per qualunque utente.

Va osservato che l'interfaccia '**net0**' è costituita sempre dal dispositivo interno associato all'indirizzo 127.0.0.1 (*loopback*).

os32 fornisce l'accesso alla tabella delle interfacce di rete attraverso il file di dispositivo '`/dev/kmem_net`', da cui attinge il programma '**`ipconfig`**'.

FILE

'`/dev/kmem_net`'
È il file di dispositivo attraverso il quale è possibile leggere la tabella delle interfacce del kernel di os32.

FILE SORGENTI

'`applic/crt0.mer.s`' [96.1.12]

'`applic/crt0.sep.s`' [96.1.13]

'`applic/ipconfig.c`' [96.1.19]

VEDERE ANCHE

kmem_net(4) [89.7], *arp(8)* [92.1], *route(8)* [92.9].

92.6 os32: MAKEDEV(8)

«

NOME

'**`MAKEDEV`**' - creazione dei file di dispositivo

SINTASSI

```
MAKEDEV
```

DESCRIZIONE

'**`MAKEDEV`**' è un programma che crea, nella directory corrente, tutti i file di dispositivo previsti per os32. Tali file devono trovarsi normalmente nella directory '`/dev/`', pertanto, prima di usa-

re '**MAKEDEV**' è necessario che la directory corrente corrisponda precisamente a tale posizione.

OPZIONI

Non sono previste opzioni per l'uso di '**MAKEDEV**', dal momento che vengono creati tutti i file di dispositivo, considerato il loro numero limitato.

NOTE

Tradizionalmente '**MAKEDEV**' viene realizzato in forma di script, ma os32 non dispone di una shell adeguata e non è possibile utilizzare script.

FILE SORGENTI

'`applic/crt0.mer.s`' [96.1.12]

'`applic/crt0.sep.s`' [96.1.13]

'`lib/sys/os32.h`' [95.21]

'`applic/MAKEDEV.c`' [96.1.1]

92.7 os32: mount(8)

NOME

'**mount**', '**umount**' - innesto e distacco di un file system

SINTASSI

```
mount dispositivo dir_innesto [opzioni]
```

```
umount directory
```

DESCRIZIONE

‘**mount**’ innesta il file system contenuto nell’unità di memorizzazione rappresentata dal file di dispositivo che va indicato come primo argomento, nella directory che appare come secondo argomento. Eventualmente si possono specificare delle opzioni di innesto, come terzo argomento.

‘**umount**’ stacca il file system innestato precedentemente nella directory indicata come unico argomento del comando.

OPZIONI DI INNESTO

Opzione	Descrizione
`ro`	Innesta il file system in sola lettura.
`rw`	Innesta il file system in lettura e scrittura. Si tratta comunque del comportamento predefinito, in mancanza di un’opzione contraria.

DIFETTI

Non viene preso in considerazione un eventuale file ‘`/etc/fstab`’; inoltre, l’utente non può conoscere lo stato degli innesti già in essere e, a questo proposito, l’uso di ‘**mount**’ senza argomenti produce semplicemente un errore.

FILE SORGENTI

‘`applic/crt0.mer.s`’ [96.1.12]

‘`applic/crt0.sep.s`’ [96.1.13]

‘`applic/mount.c`’ [96.1.28]

‘`applic/umount.c`’ [96.1.53]

92.8 os32: ping(8)

«

NOME

'**ping**' - invio di richieste di eco: ICMP ECHO_REQUEST.

SINTASSI

```
ping indirizzo_ipv4
```

DESCRIZIONE

Il programma '**ping**' invia una richiesta di eco (ICMP ECHO_REQUEST) alla destinazione indicata attraverso un indirizzo IPv4. Se si ottiene risposta, il programma termina con successo, altrimenti si ottiene un errore; in ogni caso, viene eseguito un solo tentativo.

Per utilizzare '**ping**' è necessario operare in qualità di utente '**root**'.

FILE SORGENTI

'applic/crt0.mer.s' [96.1.12]

'applic/crt0.sep.s' [96.1.13]

'applic/ping.c' [96.1.30]

VEDERE ANCHE

arp(8) [92.1], *ipconfig(8)* [92.9], *route(8)* [92.9], *http(8)* [92.3], *nc(8)* [86.20].

92.9 os32: route(8)

«

NOME

‘**route**’ - mostra la tabella degli instradamenti

SINTASSI

```
route
```

DESCRIZIONE

Il programma ‘**route**’ consente di visualizzare la tabella degli instradamenti, ma senza la possibilità di potervi intervenire per modificarla (la configurazione delle interfacce di rete e degli instradamenti avviene esclusivamente attraverso le opzioni di avvio del kernel). Pertanto, l’uso del programma è ammissibile per qualunque utente.

os32 fornisce l’accesso alla tabella degli instradamenti attraverso il file di dispositivo ‘`/dev/kmem_route`’, da cui attinge il programma ‘**route**’.

FILE

‘`/dev/kmem_route`’
: È il file di dispositivo attraverso il quale è possibile leggere la tabella degli instradamenti del kernel di os32.

FILE SORGENTI

‘`applic/crt0.mer.s`’ [96.1.12]

‘`applic/crt0.sep.s`’ [96.1.13]

‘`applic/route.c`’ [96.1.34]

VEDERE ANCHE

kmem_route(4) [89.9], *arp(8)* [92.1], *ipconfig(8)* [92.5].

92.10 os32: umount(8)

Vedere *mount(8)* [92.7].

Sezione 9: kernel

93.1 os32: arp(9)

Il file ‘kernel/net/arp.h’ [94.12.1] descrive le funzioni per la gestione della tabella ARP, per la trasformazione degli indirizzi IPv4 in indirizzi Ethernet.

Per la descrizione sulla gestione della tabella ARP da parte di os32, si rimanda alla sezione 84.9.3. La tabella successiva che sintetizza l’uso delle funzioni di questo gruppo, è tratta da lì.

Tabella 84.123. Funzioni per la gestione della tabella ARP, contenute nella directory '`kernel/net/arp/`'.

Funzione	Descrizione
`void arp_init (void);`	Azzera completamente la tabella ARP: si usa una volta sola all'avvio della gestione della rete [94.12.4].
`void arp_clean (void);`	Azzera le voci della tabella ARP che risultano troppo vecchie e che devono essere rinnovate [94.12.2].
`int arp_index (unsigned char` ***mac[6]***`, h_addr_t` ***ip***`);`	Restituisce l'indice della tabella ARP, corrispondente all'indirizzo Ethernet o all'indirizzo IPv4 fornito [94.12.3].
`arp_t *arp_reference (void);`	Restituisce il puntatore a un elemento della tabella ARP contenente la voce più vecchia, allo scopo presumibile di riutilizzarla per un indirizzo nuovo [94.12.7].
`void arp_request (h_addr_t` ***ip***`);`	Invia una richiesta ARP, preparando il pacchetto relativo e inviandolo attraverso la funzione ***ethernet_tx()*** [94.12.8].

Funzione	Descrizione
`int arp_rx (int n, int f);`	Legge dalla tabella delle interfacce il pacchetto individuato dall'indice ***n*** per l'interfaccia e dall'indice ***f*** per la trama relativa. Il pacchetto in questione deve essere relativo al protocollo ARP: se si tratta di una richiesta, provvede a inviare una risposta, se invece si tratta di una risposta, allora aggiorna la tabella ARP [94.12.9].

93.2 os32: ata(9)

Il file '`kernel/driver/ata.h`' [94.4.3] descrive le funzioni per la gestione delle unità a disco PATA.

Per la descrizione dell'organizzazione della gestione delle unità PATA di os32, si rimanda alla sezione 84.7.7. La tabella successiva che sintetizza l'uso delle funzioni di questo gruppo, è tratta da lì.

Tabella 84.94. Funzioni per la gestione delle unità PATA, dichiarate nel file di intestazione '`kernel/driver/ata.h`' e descritte nei file contenuti nella directory '`kernel/driver/ata/`'. Le funzioni sono raggruppate in insiemi logici.

Funzione	Descrizione
`void ata_init (void);`	Inizializza la gestione delle unità PATA, predisponendo i contenuti della tabella ***ata_table[]***, verificando la presenza delle unità. Questa funzione viene usata una volta sola, nella funzione ***proc_init()***.
`void ata_reset (int` ***drive*`);`	Azzera lo stato di funzionamento dell'unità PATA specificata.
`int ata_valid (int` ***drive*`);`	Verifica se l'unità richiesta è presente effettivamente. In caso di successo restituisce il valore zero, altrimenti si ottiene −1.

Funzione	Descrizione
`int ata_cmd_identify_device (int` ***drive*****,** `void *`***buffer*****);**	Richiede all'unità specificata le informazioni sulla sua identificazione. Se l'unità è presente, in corrispondenza del puntatore fornito si ottengono le informazioni nello spazio di un settore (***ATA_SECTOR_SIZE***); l'analisi successiva di questi dati può dare maggiori informazioni sull'unità.
`int ata_cmd_read_sectors (int` ***drive*****,** `unsigned int` ***sector*****,** `unsigned char` ***count*****,** `void *`***buffer*****);**	Legge dall'unità ***drive***, a partire dal settore ***sector***, una quantità pari a ***count*** settori, mettendo il risultato a partire dall'indirizzo di memoria ***buffer***. Se ***count*** fosse pari a zero, si intenderebbero 256 settori. Se l'operazione fallisce, restituisce un valore negativo.

Funzione	Descrizione
`int ata_cmd_write_sectors (int drive, unsigned int sector, unsigned char count, void *buffer);`	Scrive nell'unità ***drive***, a partire dal settore ***sector***, una quantità pari a ***count*** settori, leggendoli a partire dall'indirizzo di memoria ***buffer***. Se ***count*** fosse pari a zero, si intenderebbero 256 settori. Se l'operazione fallisce, restituisce un valore negativo.
`int ata_device (int drive, unsigned int sector);`	Imposta il registro ***device*** dell'unità PATA specificata, con l'indicazione di un numero di settore.

Funzione	Descrizione
`int ata_rdy (int drive, clock_t timeout);`	Attende che l'unità ***drive*** sia pronta, purché ciò avvenga entro il tempo ***timeout***. Se l'operazione ha successo, la funzione restituisce zero, altrimenti dà un valore negativo.
`int ata_drq (int drive, clock_t timeout);`	Attende che l'unità ***drive*** sia pronta a ricevere dati, purché ciò avvenga entro il tempo ***timeout***. Se l'operazione ha successo, la funzione restituisce zero, altrimenti dà un valore negativo.

Funzione	Descrizione
`int ata_lba28 (int` ***drive*****`,`** `unsigned int` ***sector*****`,`** `unsigned char` ***count*****`);`**	Invia all'unità ***drive*** la prima parte di un comando, in cui sono contenute le coordinate LBA28.

Funzione	Descrizione

`int`
`ata_read_sector (`
`int` ***drive*****`,`**
`unsigned int` ***sector*****`,`**
`void *`***buffer***`);`
	È una macroistruzione che legge dall'unità ***drive***, il settore ***sector***, mettendo il risultato a partire dall'indirizzo di memoria ***buffer***. La macroistruzione si avvale praticamente della funzione ***ata_cmd_read_sectors()***, per leggere un solo settore.

`int`
`ata_write_sector (`
`int` ***drive*****`,`**
`unsigned int` ***sector*****`,`**
`void *`***buffer***`);`
	È una macroistruzione che scrive nell'unità ***drive***, il settore ***sector***, traendo i dati dall'indirizzo di memoria ***buffer***. La macroistruzione si avvale praticamente della funzione ***ata_cmd_write_sectors()***, per scrivere un solo settore.

93.3 os32: blk(9)

«

Il file ‘`kernel/blk.h`’ [94.2] descrive ciò che serve per la gestione dei blocchi di dati, in relazione ai dispositivi di memorizzazione a blocchi.

93.3.1 os32: blk_ata(9)

«

NOME

‘**`blk_ata`**’ - interfaccia di accesso ai dispositivi di memorizzazione PATA

SINTASSI

```
<kernel/blk.h>
void *blk_ata (dev_t device, int rw, unsigned int n,
               void *buffer);
```

ARGOMENTI

Argomento	Descrizione
`dev_t` ***device***	Dispositivo, in forma numerica.
`int` ***rw***	Può assumere i valori ‘**`DEV_READ`**’ o ‘**`DEV_WRITE`**’, per richiedere rispettivamente un accesso in lettura oppure in scrittura.
`unsigned int` ***n***	Numero del blocco da leggere o scrivere, riferito all’unità complessiva.
`void *`***buffer***	Origine dei dati da trasferire in caso di scrittura (per la lettura questa informazione non viene usata).

DESCRIZIONE

La funzione ***blk_ata()*** è un'interfaccia per l'accesso alle unità PATA, un blocco alla volta, che si avvale di una memoria *cache* per ridurre gli accessi fisici ripetuti agli stessi blocchi.

VALORE RESTITUITO

La funzione restituisce il puntatore a un'area di memoria contenente il blocco di dati letto o scritto. In caso di errore restituisce il puntatore nullo e aggiorna la variabile ***errno*** del kernel. Il blocco in questione si riferisce all'unità complessiva, pertanto, se originariamente si faceva riferimento a un dispositivo di una partizione, prima di arrivare a questa funzione il blocco relativo della partizione deve essere stato convertito in un blocco assoluto dell'unità complessiva.

ERRORI

Valore di ***errno***	Significato
`E_HARDWARE_FAULT`	Errore nell'hardware PATA.
`E_DRIVER_FAULT`	Errore nella gestione software dell'unità PATA.
`ETIME`	Tempo scaduto.

FILE SORGENTI

'`kernel/blk.h`' [94.2]

'`kernel/blk/blk_ata.c`' [94.2.1]

VEDERE ANCHE

dev_dm(9) [93.4.2], *dev_ata(9)* [93.4.3], *blk_cache_read(9)* [93.3.4], *blk_cache_save(9)* [93.3.4].

93.3.2 os32: blk_cache_check(9)

«

NOME

‘**blk_cache_check**’ - verifica della validità del contenuto della tabella dei blocchi conservati in memoria

SINTASSI

```
<kernel/blk.h>
void blk_cache_check (void);
```

DESCRIZIONE

La funzione ***blk_cache_check()*** serve a verificare che la tabella dei blocchi conservati in memoria (***blk_table[]***) contenga dati validi, per ciò che riguarda le età degli stessi. In pratica, il valore dell’età dei blocchi deve essere sequenziale, iniziando da zero e terminando con il valore massimo consentito: non ci devono essere valori mancanti e non ci devono essere valori doppi. Questa funzione serve solo a titolo diagnostico e in pratica non viene usata.

VALORE RESTITUITO

La funzione non restituisce alcunché; se viene usata e se individua errori, questi vengono visualizzati attraverso la funzione ***k_printf()***.

FILE SORGENTI

‘`kernel/blk.h`’ [94.2]

‘`kernel/blk/blk_cache_check.c`’ [94.2.2]

VEDERE ANCHE

blk_cache_init(9) [93.3.3], *blk_cache_read(9)* [93.3.4], *blk_cache_save(9)* [93.3.4].

93.3.3 os32: blk_cache_init(9)

NOME

'**blk_cache_init**' - inizializzazione della tabella dei blocchi conservati in memoria

SINTASSI

```
<kernel/blk.h>
void blk_cache_init (void);
```

DESCRIZIONE

La funzione ***blk_cache_init()*** serve a inizializzare la tabella dei blocchi conservati in memoria (***blk_table[]***), assegnando anche il valore dell'età in modo progressivo, da zero fino al valore massimo consentito. Questa funzione viene usata una volta sola, prima che i dispositivi di memorizzazione a blocchi possano avvalersi della tabella stessa.

VALORE RESTITUITO

La funzione non restituisce alcunché.

FILE SORGENTI

'`kernel/blk.h`' [94.2]

'`kernel/blk/blk_cache_init.c`' [94.2.3]

VEDERE ANCHE

blk_cache_check(9) [93.3.2], *blk_cache_read(9)* [93.3.4], *blk_cache_save(9)* [93.3.4].

93.3.4 os32: blk_cache_read(9)

«

NOME

'**blk_cache_read**', '**blk_cache_save**', - lettura e scrittura nella tabella dei blocchi

SINTASSI

```
<kernel/blk.h>
void *blk_cache_read (dev_t device, unsigned int n);
void *blk_cache_save (dev_t device, unsigned int n,
                      void *block);
```

ARGOMENTI

Argomento	Descrizione
`dev_t` ***device***	Dispositivo, in forma numerica.
`unsigned int` ***n***	Numero del blocco da leggere o scrivere, riferito all'unità complessiva.
`void *`***block***	Origine dei dati da salvare nella tabella.

DESCRIZIONE

Le funzioni ***blk_cache_read()*** e ***blk_cache_save()*** si occupano di trovare un blocco già salvato in precedenza nella tabella dei blocchi, o di salvarne uno nuovo o di aggiornarne il contenuto. Le funzioni restituiscono il puntatore al blocco trovato o memorizzato, oppure il puntatore nullo in caso di fallimento dell'operazione.

Quando si legge un blocco, lo si ottiene solo se viene trovata la corrispondenza con il numero di dispositivo (riferito all'unità complessiva) e al numero del blocco stesso. Quando si salva un blocco, viene prima cercato lo stesso blocco nella tabella, per aggiornarlo, ma se non c'è, viene riutilizzata una cella corrispondente a un vecchio blocco che non risulta usato da più tempo.

Una lettura con successo o il salvataggio di un blocco, implica l'attribuzione allo stesso di un'età pari a zero, modificando di conseguenza quella degli altri blocchi in modo coerente.

VALORE RESTITUITO

La funzione restituisce il puntatore all'area di memoria contenente il blocco di dati letto o scritto. Mentre il salvataggio avviene sempre con successo, la lettura può fallire e in tal caso si ottiene il puntatore nullo.

FILE SORGENTI

'`kernel/blk.h`' [94.2]

'`kernel/blk/blk_cache_read.c`' [94.2.4]

'`kernel/blk/blk_cache_save.c`' [94.2.5]

VEDERE ANCHE

dev_dm(9) [93.4.2], *blk_ata(9)* [93.3.1].

93.3.5 os32: blk_cache_save(9)

Vedere *blk_cache_read(9)* [93.3.4].

93.4 os32: dev(9)

«

Il file ‘`kernel/dev.h`’ [94.3] descrive ciò che serve per la gestione dei dispositivi. Tuttavia, la definizione dei numeri di dispositivo è contenuta nel file ‘`lib/sys/os32.h`’ [95.21], il quale viene incluso da ‘`dev.h`’.

Tabella 84.82. Classificazione dei dispositivi di os32.

Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`DEV_MEM`	r/w	diretto	Permette l’accesso alla memoria, in modo indiscriminato; tuttavia, solo al kernel è permessa la scrittura.
`DEV_NULL`	r/w	nessuno	Consente la lettura e la scrittura, ma non si legge e non si scrive alcunché.

Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`DEV_PORT`	r/w	sequenziale	Consente di leggere e scrivere da o verso una porta di I/O, individuata attraverso l'indirizzo di accesso (l'indirizzo, o meglio lo scostamento, viene trattato come la porta a cui si vuole accedere). Tuttavia, la dimensione dell'informazione da trasferire è valida solo se si tratta di uno o di due byte: per la dimensione di un byte si usano le funzioni ***in_8()*** e ***out_8()***; per due byte si usano le funzioni ***in_16()*** e ***out_16()***. Per dimensioni differenti la lettura o la scrittura non ha effetto.
`DEV_ZERO`	r	sequenziale	Consente solo la lettura di valori a zero (zero inteso in senso binario).
`DEV_TTY`	r/w	sequenziale	Rappresenta il terminale virtuale del processo attivo.

Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`DEV_DM`***mn***	r/w	diretto	Rappresenta la partizione ***n*** dell'unità di memorizzazione ***m***. La prima unità PATA disponibile ottiene il dispositivo ***DEV_DM00***, la seconda il numero ***DEV_DM10***, ecc.
`DEV_KMEM_PS`	r	diretto	Rappresenta la tabella contenente le informazioni sui processi. L'indirizzo di accesso indica il numero del processo di partenza; la dimensione da leggere dovrebbe essere abbastanza grande da contenere un processo, ma anche richiedendo una dimensione maggiore, se ne legge uno solo.
`DEV_KMEM_MMP`	r	sequenziale	Rappresenta la mappa della memoria, alla quale si può accedere solo dal suo principio. In pratica, l'indirizzo di accesso viene ignorato, mentre conta solo la quantità di byte richiesta.

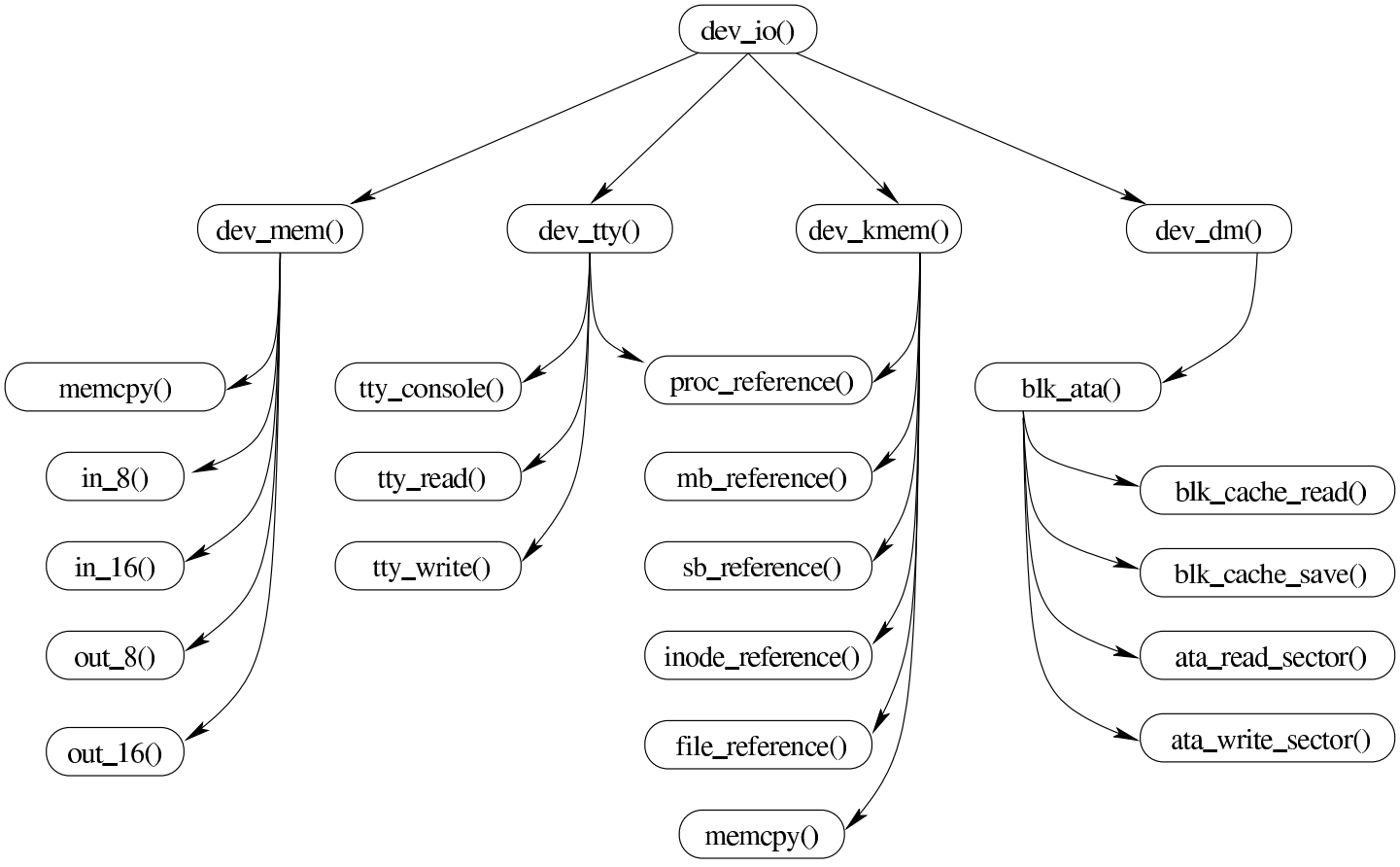
Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`DEV_KMEM_SB`	r	diretto	Rappresenta la tabella dei super blocchi (per la gestione delle unità di memorizzazione). L'indirizzo di accesso serve a individuare il super blocco; la dimensione richiesta dovrebbe essere abbastanza grande da contenere un super blocco, ma anche richiedendo una dimensione maggiore, se ne legge uno solo.
`DEV_KMEM_INODE`	r	diretto	Rappresenta la tabella degli inode (per la gestione delle unità di memorizzazione). L'indirizzo di accesso serve a individuare l'inode; la dimensione richiesta dovrebbe essere abbastanza grande da contenere un inode, ma anche richiedendo una dimensione maggiore, se ne legge uno solo.

Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`DEV_KMEM_FILE`	r	diretto	Rappresenta la tabella dei file (per la gestione delle unità di memorizzazione). L'indirizzo di accesso serve a individuare il file; la dimensione richiesta dovrebbe essere abbastanza grande da contenere le informazioni di un file, ma anche richiedendo una dimensione maggiore, se ne legge uno solo.
`DEV_KMEM_ARP`	r	diretto	Rappresenta la tabella ARP (per la trasformazione degli indirizzi IPv4 in indirizzi Ethernet). L'indirizzo di accesso serve a individuare la voce; la dimensione richiesta dovrebbe essere abbastanza grande da contenere le informazioni di una voce, ma anche richiedendo una dimensione maggiore, se ne legge una sola.

Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`DEV_KMEM_NET`	r	diretto	Rappresenta la tabella delle interfacce di rete. L'indirizzo di accesso serve a individuare la voce della tabella; la dimensione richiesta dovrebbe essere abbastanza grande da contenere le informazioni di una voce, ma anche richiedendo una dimensione maggiore, se ne legge una sola.
`DEV_KMEM_ROUTE`	r	diretto	Rappresenta la tabella degli instradamenti IPv4. L'indirizzo di accesso serve a individuare la voce della tabella; la dimensione richiesta dovrebbe essere abbastanza grande da contenere le informazioni di una voce, ma anche richiedendo una dimensione maggiore, se ne legge una sola.
`DEV_CONSOLE`	r/w	sequenziale	Legge o scrive relativamente alla console attiva la quantità di byte richiesta, ignorando l'indirizzo di accesso.

Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`DEV_CONSOLE`***n***	r/w	sequenziale	Legge o scrive relativamente alla console ***n*** la quantità di byte richiesta, ignorando l'indirizzo di accesso.

Figura 84.80. Interdipendenza tra la funzione ***dev_io()*** e le altre. I collegamenti con le funzioni ***major()*** e ***minor()*** sono omesse.



93.4.1 os32: dev_io(9)

NOME

‘**dev_io**’ - interfaccia di accesso ai dispositivi

SINTASSI

```
<kernel/dev.h>
ssize_t dev_io (pid_t pid, dev_t device, int rw, off_t offset,
                void *buffer, size_t size, int *eof);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`dev_t` ***device***	Dispositivo, in forma numerica.
`int` ***rw***	Può assumere i valori ‘**DEV_READ**’ o ‘**DEV_WRITE**’, per richiedere rispettivamente un accesso in lettura oppure in scrittura.
`off_t` ***offset***	Posizione per l’accesso al dispositivo.
`void *`***buffer***	Memoria tampone, per la lettura o la scrittura.
`size_t` ***size***	Quantità di byte da leggere o da scrivere.
`int *`***eof***	Puntatore a una variabile in cui annotare, eventualmente, il raggiungimento della fine del file.

DESCRIZIONE

La funzione ***dev_io()*** è un’interfaccia generale per l’accesso ai dispositivi gestiti da os32.

VALORE RESTITUITO

La funzione restituisce la quantità di byte letti o scritti effettivamente. In caso di errore restituisce il valore −1 e aggiorna la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`ENODEV`	Il numero del dispositivo non è valido.
`EIO`	Errore di input-output.

FILE SORGENTI

‘`kernel/dev.h`’ [94.3]

‘`kernel/dev/dev_io.c`’ [94.3.3]

VEDERE ANCHE

dev_dm(9) [93.4.2], *dev_kmem(9)* [93.4.4], *dev_mem(9)* [93.4.5], *dev_tty(9)* [93.4.6].

93.4.2 os32: dev_dm(9)

«

NOME

‘**`dev_dm`**’ - interfaccia di accesso alle unità di memorizzazione di massa

SINTASSI

```
<kernel/dev.h>
ssize_t dev_dm (pid_t pid, dev_t device, int rw, off_t offset,
                void *buffer, size_t size, int *eof);
```

DESCRIZIONE

La funzione ***dev_dm()*** consente di accedere alle unità di memorizzazione di massa, che per os32 si riducono alle sole unità PATA.

Per il significato degli argomenti, il valore restituito e gli eventuali errori, si veda *dev_io(9)* [93.4.1].

FILE SORGENTI

‘`kernel/dev.h`’ [94.3]

‘`kernel/dev/dev_io.c`’ [94.3.3]

‘`kernel/dev/dev_dm.c`’ [94.3.2]

93.4.3 os32: dev_ata(9)

NOME

‘**`dev_ata`**’ - interfaccia di accesso alle unità di memorizzazione di massa PATA

SINTASSI

```
<kernel/dev.h>
ssize_t dev_ata (pid_t pid, dev_t device, int rw, off_t offset,
                 void *buffer, size_t size, int *eof);
```

DESCRIZIONE

La funzione ***dev_ata()*** consente di accedere alle unità di memorizzazione di massa PATA, traducendo le operazioni da flusso di byte a blocchi. Questa funzione viene usata da ***dev_dm()*** e si avvale a sua volta di ***blk_ata()***.

Per il significato degli argomenti, il valore restituito e gli eventuali errori, si veda *dev_io(9)* [93.4.1].

FILE SORGENTI

‘`kernel/dev.h`’ [94.3]

‘`kernel/dev/dev_io.c`’ [94.3.3]

‘`kernel/dev/dev_dm.c`’ [94.3.2]

‘`kernel/dev/dev_ata.c`’ [94.3.1]

93.4.4 os32: dev_kmem(9)

«

NOME

‘**`dev_kmem`**’ - interfaccia di accesso alle tabelle di dati del kernel, rappresentate in memoria

SINTASSI

```
<kernel/dev.h>
ssize_t dev_kmem (pid_t pid, dev_t device, int rw,
                  off_t offset, void *buffer,
                  size_t size, int *eof);
```

DESCRIZIONE

La funzione ***dev_kmem()*** consente di accedere, solo in lettura, alle porzioni di memoria che il kernel utilizza per rappresentare alcune tabelle importanti. Per poter interpretare ciò che si ottiene occorre riprodurre la struttura di un elemento della tabella a cui si è interessati, pertanto occorre incorporare il file di intestazione del kernel che la descrive.

Dispositivo	Tabella a cui si riferisce
`DEV_KMEM_PS`	Si accede alla tabella dei processi, all'elemento rappresentato da ***offset***: ***proc_table[offset]***.
`DEV_KMEM_MMP`	Si accede alla mappa della memoria, ovvero la tabella ***mb_table[]***, senza considerare il valore di ***offset***.
`DEV_KMEM_SB`	Si accede alla tabella dei super blocchi, all'elemento rappresentato da ***offset***: ***sb_table[offset]***.
`DEV_KMEM_INODE`	Si accede alla tabella degli inode, all'elemento rappresentato da ***offset***: ***inode_table[offset]***.
`DEV_KMEM_FILE`	Si accede alla tabella dei file di sistema, all'elemento rappresentato da ***offset***: ***file_table[offset]***.

Per il significato degli argomenti della chiamata, per interpretare il valore restituito e gli eventuali errori, si veda *dev_io(9)* [93.4.1].

FILE SORGENTI

'`kernel/dev.h`' [94.3]

'`kernel/dev/dev_io.c`' [94.3.3]

'`kernel/dev/dev_kmem.c`' [94.3.4]

93.4.5 os32: dev_mem(9)

«

NOME

'**`dev_mem`**' - interfaccia di accesso alla memoria, in modo indiscriminato

SINTASSI

```
<kernel/fs.h>
ssize_t dev_mem (pid_t pid, dev_t device, int rw,
                 off_t offset, void *buffer,
                 size_t size, int *eof);
```

DESCRIZIONE

La funzione ***dev_mem()*** consente di accedere, in lettura e in scrittura alla memoria e alle porte di input-output.

Dispositivo	Descrizione
`DEV_MEM`	Si tratta della memoria centrale, complessiva, dove il valore di ***offset*** rappresenta l'indirizzo efficace (complessivo) a partire da zero.
`DEV_NULL`	Si tratta di ciò che realizza il file di dispositivo tradizionale '`/dev/null`': la scrittura si perde semplicemente e la lettura non dà alcunché.
`DEV_ZERO`	Si tratta di ciò che realizza il file di dispositivo tradizionale '`/dev/zero`': la scrittura si perde semplicemente e la lettura produce byte a zero (zero binario).
`DEV_PORT`	Consente l'accesso alle porte di input-output. La dimensione rappresentata da ***size*** può essere solo pari a uno o due: una dimensione pari a uno richiede di comunicare un solo byte con una certa porta; una dimensione pari a due richiede la comunicazione di un valore a 16 bit. Il valore di ***offset*** serve a individuare la porta di input-output con cui si intende comunicare (leggere o scrivere un valore).

Per quanto non viene descritto qui, si veda *dev_io(9)* [93.4.1].

FILE SORGENTI

'`kernel/dev.h`' [94.3]

'`kernel/dev/dev_io.c`' [94.3.3]

'`kernel/dev/dev_mem.c`' [94.3.5]

93.4.6 os32: dev_tty(9)

«

NOME

‘**dev_tty**’ - interfaccia di accesso alla console

SINTASSI

```
<kernel/dev.h>
ssize_t dev_tty (pid_t pid, dev_t device, int rw, off_t offset,
                 void *buffer, size_t size, int *eof);
```

DESCRIZIONE

La funzione ***dev_tty()*** consente di accedere, in lettura e in scrittura, a una console virtuale, scelta in base al numero del dispositivo.

Quando la lettura richiede l’attesa per l’inserimento da tastiera, se il processo elaborativo ***pid*** non è il kernel, allora viene messo in pausa, in attesa di un evento legato al terminale.

Il sistema di gestione del terminale è molto povero con os32, in quanto il terminale può funzionare soltanto in modalità canonica; in altri termini, si può interagire soltanto come se si trattasse della telescrivente tradizionale.

Per quanto non viene descritto qui, si veda *dev_io(9)* [93.4.1].

FILE SORGENTI

‘kernel/dev.h’ [94.3]

‘kernel/dev/dev_io.c’ [94.3.3]

‘kernel/dev/dev_tty.c’ [94.3.6]

93.5 os32: dm(9)

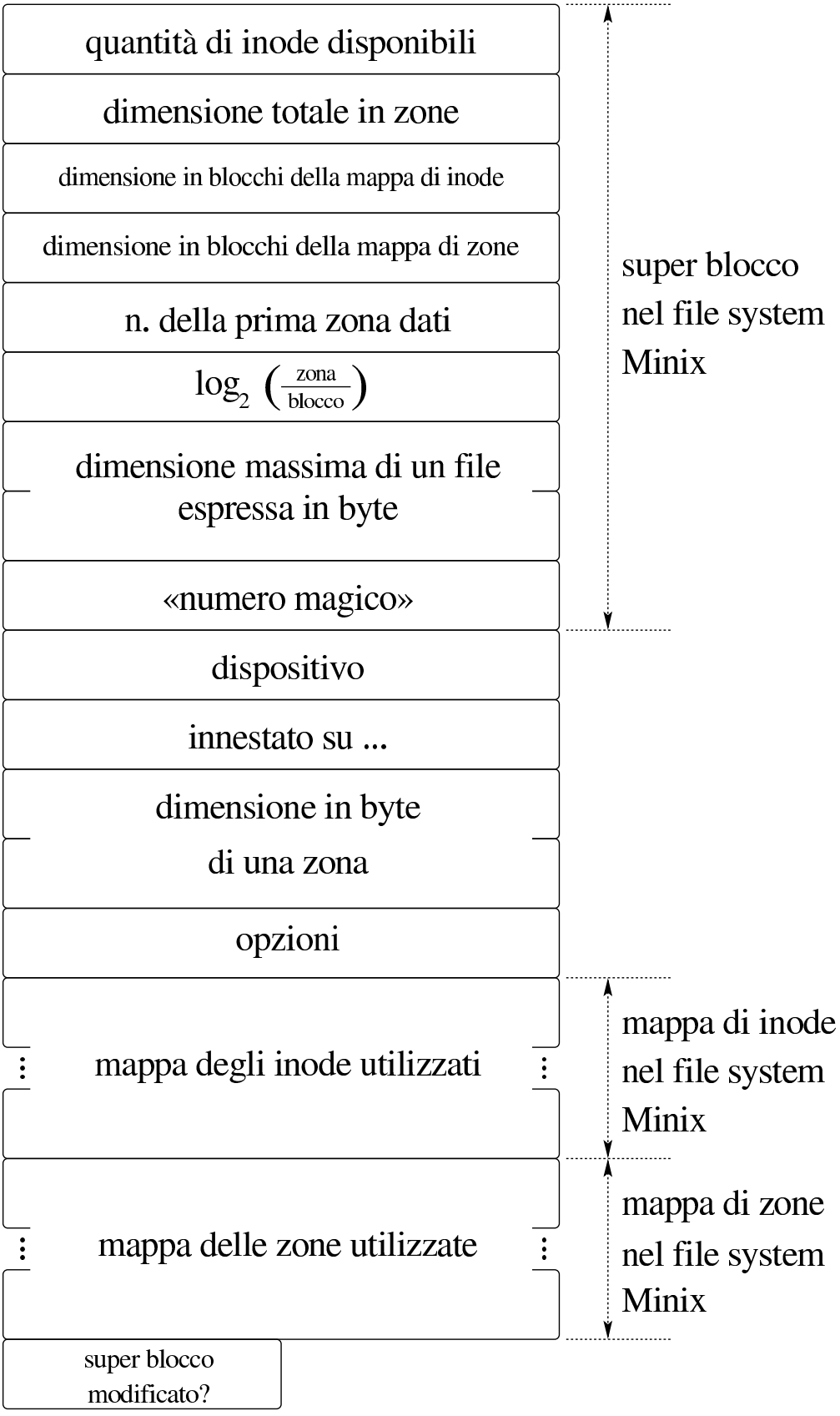
Il file ‘`kernel/dm.h`’ [94.4] descrive ciò che serve per la gestione delle unità di memorizzazione, in generale. È disponibile soltanto la funzione ***dm_init()*** per la predisposizione iniziale della tabella ***dm_table[]***.

93.6 os32: fs(9)

Il file ‘`kernel/fs.h`’ [94.5] descrive ciò che serve per la gestione del file system, che per os32 corrisponde al tipo Minix 1, assieme ai socket, essendo questi assimilati ai descrittori di file.

La gestione del file system e dei socket, a livello complessivo di sistema, è suddivisa in quattro aspetti principali: super blocco, inode, file e socket. Per ognuno di questi è prevista una tabella (di super blocchi, di inode, di file e di socket). Seguono delle figure e i listati che descrivono l’organizzazione di queste tabelle.

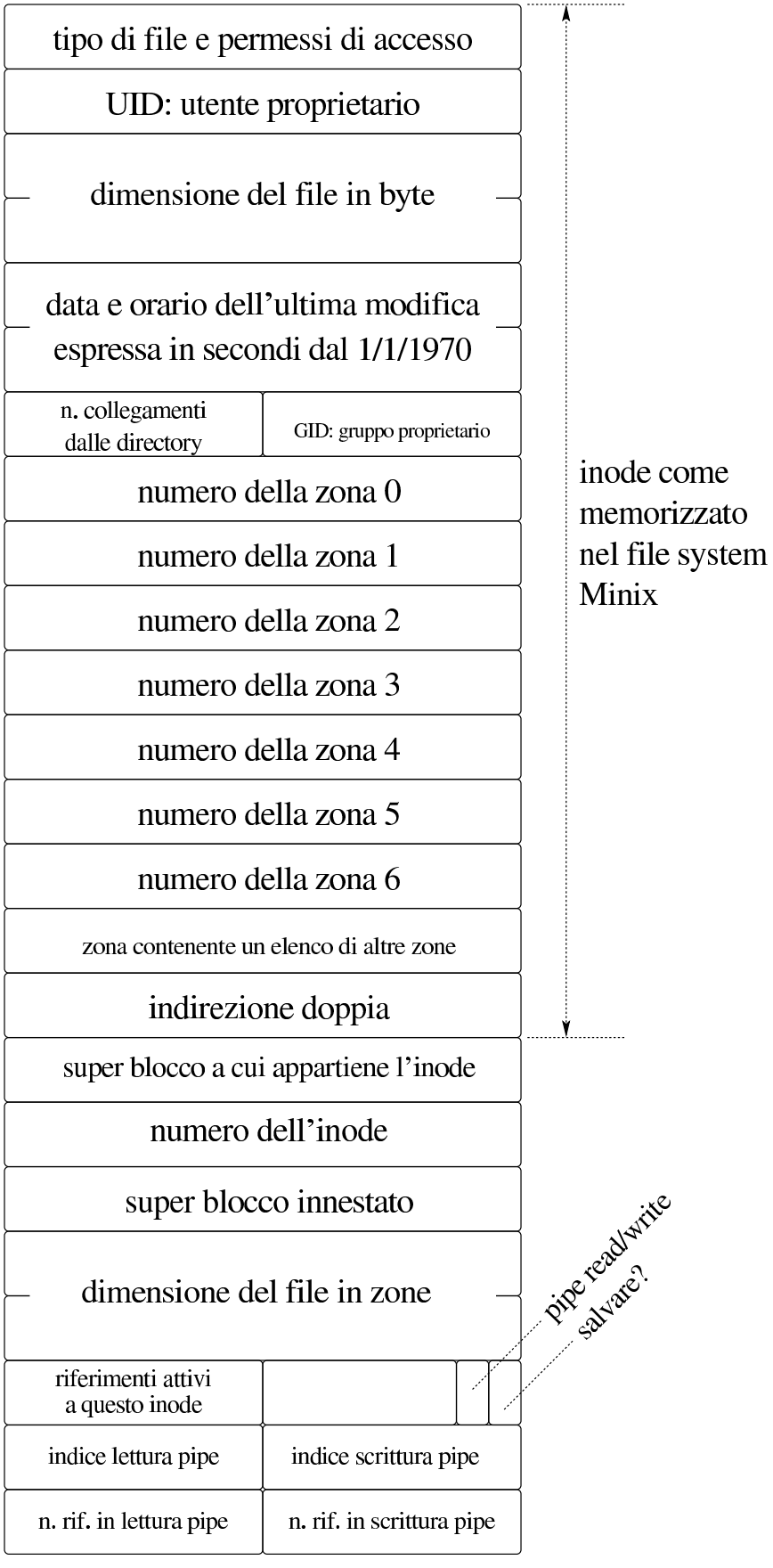
Figura 84.95. Struttura del tipo '**sb_t**', corrispondente agli elementi dell'array ***sb_table[]***.



Listato 84.96. Struttura del tipo '**sb_t**', corrispondente agli elementi dell'array ***sb_table[]***.

```
typedef struct sb        sb_t;

struct sb {
  uint16_t  inodes;
  uint16_t  zones;
  uint16_t  map_inode_blocks;
  uint16_t  map_zone_blocks;
  uint16_t  first_data_zone;
  uint16_t  log2_size_zone;
  uint32_t  max_file_size;
  uint16_t  magic_number;
  //------------------------
  dev_t     device;
  inode_t  *inode_mounted_on;
  blksize_t blksize;
  int       options;
  uint16_t  map_inode[SB_MAP_INODE_SIZE];
  uint16_t  map_zone[SB_MAP_ZONE_SIZE];
  char      changed;
};
```

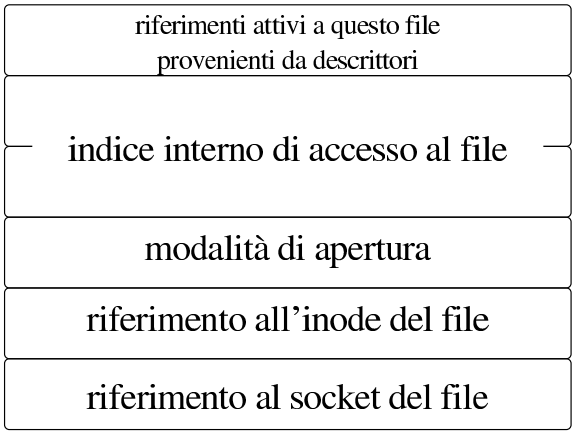
Figura 84.100. Struttura del tipo '**`inode_t`**', corrispondente agli elementi dell'array ***inode_table[]***.



Listato 84.101. Struttura del tipo '**`inode_t`**', corrispondente agli elementi dell'array ***inode_table[]***.

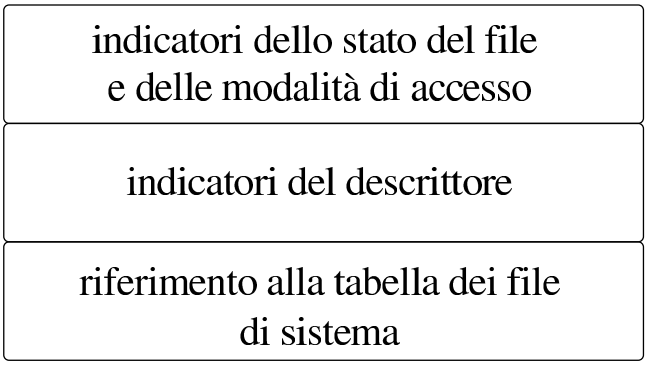
```
typedef struct inode     inode_t;

struct inode {
    mode_t         mode;
    uid_t          uid;
    ssize_t        size;
    time_t         time;
    uint8_t        gid;
    uint8_t        links;
    zno_t          direct[7];
    zno_t          indirect1;
    zno_t          indirect2;
    //-----------------------
    sb_t          *sb;
    ino_t          ino;
    sb_t          *sb_attached;
    blkcnt_t       blkcnt;
    unsigned char  references;
    char           changed  : 1,
                   pipe_dir : 1;
    unsigned char pipe_off_read;
    unsigned char pipe_off_write;
    unsigned char pipe_ref_read;
    unsigned char pipe_ref_write;
};
```

Figura 84.108. Struttura del tipo '**file_t**', corrispondente agli elementi dell'array ***file_table[]***.




```
typedef struct file file_t;

struct file {
    int        references;
    off_t      offset;
    int        oflags;
    inode_t   *inode;
    sock_t    *sock;
};
```

Figura 84.111. Struttura del tipo '**fd_t**', con cui si costituiscono gli elementi delle tabelle dei descrittori di file, una per ogni processo.




```
typedef struct fd fd_t;
struct fd {
    int      fl_flags;
    int      fd_flags;
    file_t *file;
};
```

Listato 84.131. Struttura di ogni elemento della tabella dei socket.

```
struct sock
{
  int          family;
  int          type;
  int          protocol;
  h_addr_t     laddr;          // indirizzo locale
  h_port_t     lport;          // porta locale
  h_addr_t     raddr;          // indirizzo remoto
  h_port_t     rport;          // porta remota
```

```
  struct {
    clock_t   clock[IP_MAX_PACKETS];
  } read;
  uint8_t     active      : 1, // socket utilizzato?
              unreach_net : 1, // rete irraggiungibile?
              unreach_host: 1, // destinazione
                               //  irraggiungibile?
              unreach_prot: 1, // protocollo
                               //  irraggiungibile?
              unreach_port: 1; // porta irraggiungibile?
  struct
  {
    uint16_t  conn        : 4, // stato della connessione
              can_write   : 1, // si può scrivere in
                               //  'send_data[]'?
              can_read    : 1, // si può leggere a
                               //  partire da
                               //   '*recv_index'?
              can_send    : 1, // si possono inviare
                               //  dati?
              can_recv    : 1, // si possono ricevere
                               //  dati?
              send_closed : 1, // il canale di
                               //  trasmissione
                               //   è chiuso?
              recv_closed : 1; // il canale di ricezione
                               //  è chiuso?
    //
    uint32_t  lsq[16];         // array della sequenza locale
    uint32_t  lsq_ack;         // numero di sequenza in attesa
                               //  di conferma
    uint32_t  rsq[16];         // array della sequenza remota
    uint8_t   lsqi       : 4, // indice dell'array della
                               //  sequenza locale
```

```
                rsqi        : 4; // indice dell'array della
                                 //  sequenza remota
    //
    clock_t     clock;           // istante dell'ultima
                                 //  trasmissione
    //
    uint8_t     send_data[TCP_MSS - sizeof (struct tcphdr)];
                                 // dati da trasmettere
    size_t      send_size;       // dimensione dei dati da
                                 //  trasmettere
    int         send_flags;
    uint8_t     recv_data[TCP_MAX_DATA_SIZE];
                                 // dati ricevuti
    size_t      recv_size;       // dimensione dei dati
                                 //  ricevuti
    uint8_t   *recv_index;       // indice per la lettura dei
                                 //  dati ricevuti
    pid_t       listen_pid;      // processo in ascolto della
                                 //  porta locale, in attesa di
                                 //  connessioni
    int         listen_max;      // numero massimo di richieste
                                 //  di connessione accettabili
    int         listen_queue[SOCK_MAX_QUEUE];
                                 // descrittori di connessioni
                                 //  realizzate
  } tcp;
};
```

93.6.1 os32: fd_dup(9)

«

NOME

'**fd_dup**' - duplicazione di un descrittore di file

SINTASSI

```
<kernel/fs.h>
int fd_dup  (pid_t pid, int fdn_old, int fdn_min);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`int` ***fdn_old***	Numero del descrittore di file da duplicare.
`int` ***fdn_min***	Primo numero di descrittore da usare per la copia.

DESCRIZIONE

La funzione ***fd_dup()*** duplica un descrittore, nel senso che sdoppia l'accesso a un file in due descrittori, cercando un descrittore libero a cominciare da ***fdn_min*** e continuando progressivamente, fino al primo disponibile. Il descrittore ottenuto dalla copia, viene privato dell'indicatore '**`FD_CLOEXEC`**', ammesso che nel descrittore originale ci fosse.

Questa funzione viene usata da *s_dup(9)* [93.12] e da *s_fcntl(9)* [93.12], per la duplicazione di un descrittore.

VALORE RESTITUITO

La funzione restituisce il numero del descrittore prodotto dalla duplicazione. In caso di errore, invece, restituisce il valore −1, aggiornando la variabile ***errno*** del kernel.

ERRORI

Valore di *errno*	Significato
`EINVAL`	Il valore di ***fdn_min*** è impossibile.
`EBADF`	Il valore di ***fdn_old*** non è valido.
`EMFILE`	Non è possibile allocare un nuovo descrittore.

FILE SORGENTI

‘`kernel/fs.h`’ [94.5]

‘`kernel/fs/fd_dup.c`’ [94.5.1]

VEDERE ANCHE

dup(2) [87.12], *dup2(2)* [87.12], *sysroutine(9)* [93.20.28], *proc_reference(9)* [93.20.5].

93.6.2 os32: fd_reference(9)

«

NOME

‘**`fd_reference`**’ - riferimento a un elemento della tabella dei descrittori

SINTASSI

```
<kernel/fs.h>
fd_t *fd_reference (pid_t pid, int *fdn);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`int` ***fdn***	Il numero del descrittore di file.

DESCRIZIONE

La funzione ***fd_reference()*** restituisce il puntatore all'elemento della tabella dei descrittori, corrispondente al processo e al numero di descrittore specificati. Se però viene fornito un numero di descrittore negativo, si ottiene il puntatore al primo elemento che risulta libero nella tabella.

VALORE RESTITUITO

La funzione restituisce il puntatore a un elemento della tabella dei descrittori, oppure il puntatore nullo in caso di errore, ma **senza aggiornare** la variabile ***errno*** del kernel. Infatti, l'unico errore che può verificarsi consiste nel non poter trovare il descrittore richiesto.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/fd_reference.c`' [94.5.2]

VEDERE ANCHE

file_reference(9) [93.6.5], *inode_reference(9)* [93.6.21], *sb_reference(9)* [93.6.29], *proc_reference(9)* [93.20.5].

93.6.3 os32: fs_init(9)

«

NOME

‘**fs_init**’ - inizializzazione delle tabelle relative alla gestione del file system

SINTASSI

```
<kernel/fs.h>
void fs_init  (void);
```

DESCRIZIONE

La funzione ***fs_init()*** azzera il contenuto delle tabelle dei super blocchi, degli inode e dei file. Questa funzione viene usata esclusivamente da *kmain(9)* [93.13], per predisporre le tabelle di gestione del file system.

FILE SORGENTI

‘kernel/fs.h’ [94.5]

‘kernel/fs/fs_init.c’ [94.5.6]

VEDERE ANCHE

fd_reference(9) [93.6.2], *inode_reference(9)* [93.6.21], *sb_reference(9)* [93.6.29].

93.6.4 os32: file_pipe_make(9)

«

NOME

‘**file_pipe_make**’ - creazione di una voce relativa a un condotto, nella tabella dei file di sistema

SINTASSI

```
<kernel/fs.h>
file_t *file_pipe_make (void);
```

DESCRIZIONE

La funzione ***file_pipe_make()*** produce una voce nella tabella dei file di sistema, relativa a un condotto (*pipe*).

Per ottenere questo risultato occorre coinvolgere anche la funzione *inode_pipe_make(9)* [93.6.16], la quale si occupa di predisporre un inode, privo però di un collegamento a un file vero e proprio.

Questa funzione viene usata esclusivamente da *s_pipe(9)* [93.12], per la realizzazione della chiamata di sistema *pipe(2)* [87.38].

VALORE RESTITUITO

La funzione restituisce il puntatore a un elemento della tabella dei file di sistema, oppure il puntatore nullo in caso di errore, aggiornando la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`ENFILE`	Non è possibile allocare un altro file di sistema.

FILE SORGENTI

‘`kernel/fs.h`’ [94.5]

‘`kernel/fs/file_pipe_make.c`’ [94.5.3]

VEDERE ANCHE

inode_pipe_make(9) [93.6.16], *file_reference(9)* [93.6.5], *inode_put(9)* [93.6.20].

93.6.5 os32: file_reference(9)

«

NOME

'**file_reference**' - riferimento a un elemento della tabella dei file di sistema

SINTASSI

```
<kernel/fs.h>
file_t *file_reference (int fno);
```

ARGOMENTI

Argomento	Descrizione
`int` ***fno***	Il numero della voce della tabella dei file, a partire da zero.

DESCRIZIONE

La funzione ***file_reference()*** restituisce il puntatore all'elemento della tabella dei file di sistema, corrispondente al numero indicato come argomento. Se però tale numero fosse negativo, viene restituito il puntatore al primo elemento libero.

VALORE RESTITUITO

La funzione restituisce il puntatore a un elemento della tabella dei file di sistema, oppure il puntatore nullo in caso di errore, ma **senza aggiornare** la variabile ***errno*** del kernel. Infatti, l'unico errore che può verificarsi consiste nel non poter trovare la voce richiesta.

FILE SORGENTI

‘kernel/fs.h’ [94.5]

‘kernel/fs/fs_public.c’ [94.5.7]

‘kernel/fs/file_reference.c’ [94.5.4]

VEDERE ANCHE

fd_reference(9) [93.6.2], *inode_reference(9)* [93.6.21], *sb_reference(9)* [93.6.29], *proc_reference(9)* [93.20.5].

93.6.6 os32: file_stdio_dev_make(9)

NOME

‘**file_stdio_dev_make**’ - creazione di una voce relativa a un dispositivo di input-output standard, nella tabella dei file di sistema

SINTASSI

```
<kernel/fs.h>
file_t *file_stdio_dev_make (dev_t device, mode_t mode,
                             int oflags);
```

ARGOMENTI

Argomento	Descrizione
dev_t *device*	Il numero del dispositivo da usare per l’input o l’output.
mode_t *mode*	Tipo di file di dispositivo: è praticamente obbligatorio l’uso di ‘**S_IFCHR**’.
int *oflags*	Modalità di accesso: ‘**O_RDONLY**’ oppure ‘**O_WRONLY**’.

DESCRIZIONE

La funzione ***file_stdio_dev_make()*** produce una voce nella tabella dei file di sistema, relativa a un dispositivo di input-output, da usare come flusso standard. In altri termini, serve per creare le voci della tabella dei file, relative a standard input, standard output e standard error.

Per ottenere questo risultato occorre coinvolgere anche la funzione *inode_stdio_dev_make(9)* [93.6.23], la quale si occupa di predisporre un inode, privo però di un collegamento a un file vero e proprio.

Questa funzione viene usata esclusivamente da *proc_sys_exec(9)* [93.20.21], per attribuire standard input, standard output e standard error, che non fossero già disponibili.

VALORE RESTITUITO

La funzione restituisce il puntatore a un elemento della tabella dei file di sistema, oppure il puntatore nullo in caso di errore, aggiornando la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`ENFILE`	Non è possibile allocare un altro file di sistema.

FILE SORGENTI

‘`kernel/fs.h`’ [94.5]

‘`kernel/fs/file_stdio_dev_make.c`’ [94.5.5]

VEDERE ANCHE

proc_sys_exec(9) [93.20.21], *inode_stdio_dev_make(9)* [93.6.23], *file_reference(9)* [93.6.5], *inode_put(9)* [93.6.20].

93.6.7 os32: inode_alloc(9)

NOME

'**inode_alloc**' - allocazione di un inode

SINTASSI

```
<kernel/fs.h>
inode_t *inode_alloc (dev_t device, mode_t mode, uid_t uid,
                      gid_t gid);
```

ARGOMENTI

Argomento	Descrizione
`dev_t` ***device***	Il numero del dispositivo in cui si trova il file system dove allocare l'inode.
`mode_t` ***mode***	Tipo di file e modalità dei permessi da associare all'inode.
`uid_t` ***uid***	Utente proprietario dell'inode.
`gid_t` ***gid***	Gruppo proprietario dell'inode.

DESCRIZIONE

La funzione ***inode_alloc()*** cerca un inode libero nel file system del dispositivo indicato, quindi lo alloca (lo segna come utilizzato) e lo modifica aggiornando il tipo e la modalità dei permessi, oltre al proprietario del file. Se la funzione riesce nel suo intento,

restituisce il puntatore all'inode in memoria, il quale rimane così aperto e disponibile per ulteriori elaborazioni.

Questa funzione viene usata esclusivamente da *path_inode_link(9)* [93.6.42], per la creazione di un nuovo file.

VALORE RESTITUITO

La funzione restituisce il puntatore a un elemento della tabella degli inode di sistema, oppure il puntatore nullo in caso di errore, aggiornando la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il valore fornito per il parametro ***mode*** non è ammissibile.
`ENODEV`	Il dispositivo non corrisponde ad alcuna voce della tabella dei super blocchi; per esempio, il file system cercato potrebbe non essere ancora stato innestato.
`ENOSPC`	Non è possibile allocare l'inode, per mancanza di spazio.
`ENFILE`	Non c'è spazio nella tabella degli inode.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/inode_alloc.c`' [94.5.8]

VEDERE ANCHE

path_inode_link(9) [93.6.42], *sb_reference(9)* [93.6.29], *inode_get(9)* [93.6.15], *inode_put(9)* [93.6.20], *inode_truncate(9)* [93.6.24], *inode_save(9)* [93.6.22].

93.6.8 os32: inode_check(9)

NOME

‘**inode_check**’ - verifica delle caratteristiche di un inode

SINTASSI

```
<kernel/fs.h>
int inode_check (inode_t *inode, mode_t type, int perm,
                 uid_t uid, gid_t gid);
```

ARGOMENTI

Argomento	Descrizione
`inode_t` ***inode***	Puntatore a un elemento della tabella degli inode.
`mode_t` ***type***	Tipo di file desiderato. Può trattarsi di ‘**S_IFBLK**’, ‘**S_IFCHR**’, ‘**S_IFIFO**’, ‘**S_IFREG**’, ‘**S_IFDIR**’, ‘**S_IFLNK**’, ‘**S_IFSOCK**’, come dichiarato nel file ‘`lib/sys/stat.h`’, tuttavia os32 gestisce solo file di dispositivo, file normali e directory.
`int` ***perm***	Permessi richiesti dall’utente ***uid***, rappresentati nei tre bit meno significativi.
`uid_t` ***uid***	Utente nei confronti del quale vanno verificati i permessi di accesso.
`gid_t` ***gid***	Gruppo di utenti nei confronti del quale vanno verificati i permessi di accesso.

DESCRIZIONE

La funzione ***inode_check()*** verifica che l’inode indicato sia di un certo tipo e abbia i permessi di accesso necessari a un certo utente e gruppo. Tali permessi vanno rappresentati utilizzando

solo gli ultimi tre bit (4 = lettura, 2 = scrittura, 1 = esecuzione o attraversamento) e si riferiscono alla richiesta di accesso all'inode, da parte dell'utente ***uid*** e del gruppo ***gid***, tenendo conto del complesso dei permessi che li riguardano.

Nel parametro ***type*** è ammessa la sovrapposizione di più tipi validi.

Questa funzione viene usata in varie situazioni, internamente al kernel, per verificare il tipo o l'accessibilità di un file.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Le caratteristiche dell'inode sono compatibili con quanto richiesto.
−1	Le caratteristiche dell'inode non sono compatibili, oppure si è verificato un errore. In ogni caso si ottiene l'aggiornamento della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il valore di ***inode*** corrisponde a un puntatore nullo.
`E_FILE_TYPE`	Il tipo di file dell'inode non corrisponde a quanto richiesto.
`EACCES`	I permessi di accesso non sono compatibili con la richiesta.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/inode_check.c`' [94.5.9]

93.6.9 os32: inode_dir_empty(9)

NOME

'**inode_dir_empty**' - verifica della presenza di contenuti in una directory

SINTASSI

```
<kernel/fs.h>
int inode_dir_empty (inode_t *inode);
```

ARGOMENTI

Argomento	Descrizione
`inode_t *`***inode***	Puntatore a un elemento della tabella degli inode.

DESCRIZIONE

La funzione ***inode_dir_empty()*** verifica che la directory, a cui si riferisce l'inode a cui punta ***inode***, sia vuota.

VALORE RESTITUITO

Valore	Significato del valore restituito
1	*Vero*: si tratta di una directory vuota.
0	*Falso*: se è effettivamente una directory, questa non è vuota, altrimenti non è nemmeno una directory.

ERRORI

Dal momento che un risultato *Falso* non rappresenta necessariamente un errore, per verificare il contenuto della variabile ***errno***, prima dell'uso della funzione occorre azzerarla.

Valore di *errno*	Significato
`EINVAL`	L'inode non riguarda una directory.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/inode_dir_empty.c`' [94.5.10]

VEDERE ANCHE

inode_file_read(9) [93.6.10].

93.6.10 os32: inode_file_read(9)

«

NOME

'**`inode_file_read`**' - lettura di un file rappresentato da un inode

SINTASSI

```
<kernel/fs.h>
ssize_t inode_file_read (inode_t *inode, off_t offset,
                         void *buffer, size_t count,
                         int *eof);
```

ARGOMENTI

Argomento	Descrizione
`inode_t *`***inode***	Puntatore a un elemento della tabella degli inode, che rappresenta il file da leggere.
`off_t` ***offset***	Posizione, riferita all'inizio del file, a partire dalla quale eseguire la lettura.
`void *`***buffer***	Puntatore all'area di memoria in cui scrivere ciò che si ottiene dalla lettura del file.
`size_t` ***count***	Quantità massima di byte da leggere.
`int *`***eof***	Puntatore a un indicatore di fine file, da aggiornare (purché sia un puntatore valido) in base all'esito della lettura.

DESCRIZIONE

La funzione ***inode_file_read()*** legge il contenuto del file a cui si riferisce l'inode ***inode*** e se il puntatore ***eof*** è valido, aggiorna anche la variabile ****eof***.

Questa funzione si avvale a sua volta di *inode_fzones_read(9)* [93.6.13], per accedere ai contenuti del file, suddivisi in zone, secondo l'organizzazione del file system Minix 1.

VALORE RESTITUITO

La funzione restituisce la quantità di byte letti e resi effettivamente disponibili a partire da ciò a cui punta ***buffer***. Se la variabile ***var*** è un puntatore valido, aggiorna anche il suo valore, azzerandolo se la lettura avviene in una posizione interna al file, oppure impostandolo a uno se la lettura richiesta è oltre la fine del file. Se invece si tenta una lettura con un valore di ***offset*** negativo,

o specificando il puntatore nullo al posto dell'inode, la funzione restituisce −1 e aggiorna la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	L'inode non è valido, oppure il valore di ***offset*** è negativo.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/inode_file_read.c`' [94.5.11]

VEDERE ANCHE

inode_fzones_read(9) [93.6.13].

93.6.11 os32: inode_file_write(9)

«

NOME

'**`inode_file_write`**' - scrittura di un file rappresentato da un inode

SINTASSI

```
<kernel/fs.h>
ssize_t inode_file_write (inode_t *inode, off_t offset,
                          void *buffer, size_t count);
```

ARGOMENTI

Argomento	Descrizione
`inode_t *`***inode***	Puntatore a un elemento della tabella degli inode, che rappresenta il file da scrivere.
`off_t` ***offset***	Posizione, riferita all'inizio del file, a partire dalla quale eseguire la scrittura.
`void *`***buffer***	Puntatore all'area di memoria da cui trarre i dati da scrivere nel file.
`size_t` ***count***	Quantità massima di byte da scrivere.

DESCRIZIONE

La funzione ***inode_file_write()*** scrive nel file rappresentato da ***inode***, a partire dalla posizione ***offset*** (purché non sia un valore negativo), la quantità massima di byte indicati con ***count***, ciò che si trova in memoria a partire da ***buffer***.

Questa funzione si avvale a sua volta di *inode_fzones_read(9)* [93.6.13], per accedere ai contenuti del file, suddivisi in zone, secondo l'organizzazione del file system Minix 1, e di *zone_write(9)* [93.6.37], per la riscrittura delle zone relative.

Per os32, le operazioni di scrittura nel file system sono sincrone, senza alcun trattenimento in memoria (ovvero senza *cache*).

VALORE RESTITUITO

La funzione restituisce la quantità di byte scritti. La scrittura può avvenire oltre la fine del file, anche in modo discontinuo; tuttavia, non è ammissibile un valore di ***offset*** negativo.

ERRORI

Valore di *errno*	Significato
`EINVAL`	L'inode non è valido, oppure il valore di ***offset*** è negativo.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/inode_file_write.c`' [94.5.12]

VEDERE ANCHE

inode_fzones_read(9) [93.6.13], *zone_write(9)* [93.6.37].

93.6.12 os32: inode_free(9)

«

NOME

'**`inode_free`**' - deallocazione di un inode

SINTASSI

```
<kernel/fs.h>
int inode_free (inode_t *inode);
```

ARGOMENTI

Argomento	Descrizione
`inode_t *`***inode***	Puntatore a un elemento della tabella degli inode.

DESCRIZIONE

La funzione ***inode_free()*** libera l'inode specificato attraverso il puntatore ***inode***, rispetto al proprio super blocco. L'operazione

comporta semplicemente il fatto di indicare questo inode come libero, senza controlli per verificare se effettivamente non esistono più collegamenti nel file system che lo riguardano.

Questa funzione viene usata esclusivamente da *inode_put(9)* [93.6.20], per completare la cancellazione di un inode che non ha più collegamenti nel file system, quando non vi si fa più riferimento nel sistema in funzione.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dalla variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	L'inode non è valido.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/inode_free.c`' [94.5.13]

VEDERE ANCHE

inode_save(9) [93.6.22], *inode_alloc(9)* [93.6.7].

93.6.13 os32: inode_fzones_read(9)

«

NOME

'**`inode_fzones_read`**', '**`inode_fzones_write`**' - lettura e scrittura di zone relative al contenuto di un file

SINTASSI

```
<kernel/fs.h>
blkcnt_t inode_fzones_read  (inode_t *inode, zno_t zone_start,
                             void *buffer, blkcnt_t blkcnt);
blkcnt_t inode_fzones_write (inode_t *inode, zno_t zone_start,
                             void *buffer, blkcnt_t blkcnt);
```

ARGOMENTI

Argomento	Descrizione
`inode_t *inode`	Puntatore a un elemento della tabella degli inode, con cui si individua il file da cui leggere o in cui scrivere.
`zno_t zone_start`	Il numero di zona, relativo al file, a partire dalla quale iniziare la lettura o la scrittura.
`void *buffer`	Il puntatore a un'area di memoria tampone, da usare per depositare i dati letti o per trarre i dati da scrivere.
`blkcnt_t blkcnt`	La quantità di zone da leggere o scrivere.

DESCRIZIONE

Le funzioni ***inode_fzones_read()*** e ***inode_fzones_write()***, consentono di leggere e di scrivere un file, a zone intere (la zona è un multiplo del blocco, secondo la filosofia del file system Minix 1).

Questa funzione vengono usate soltanto da *inode_file_read(9)* [93.6.10] e *inode_file_write(9)* [93.6.11], con le quali l'accesso ai file si semplifica a livello di byte.

VALORE RESTITUITO

Le due funzioni restituiscono la quantità di zone lette o scritte

effettivamente. Una quantità pari a zero potrebbe eventualmente rappresentare un errore, ma solo in alcuni casi. Per poterlo verificare, occorre azzerare la variabile ***errno*** prima di chiamare le funzioni, riservandosi di verificarne successivamente il valore.

ERRORI

Valore di ***errno***	Significato
`EIO`	L'accesso alla zona richiesta non è potuto avvenire.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/inode_fzones_read.c`' [94.5.14]

'`kernel/fs/inode_fzones_write.c`' [94.5.15]

VEDERE ANCHE

inode_file_read(9) [93.6.10], *inode_file_write(9)* [93.6.11], *zone_read(9)* [93.6.37], *zone_write(9)* [93.6.37].

93.6.14 os32: inode_fzones_write(9)

«

Vedere *inode_fzones_read(9)* [93.6.13].

93.6.15 os32: inode_get(9)

«

NOME

'**`inode_get`**' - caricamento di un inode

SINTASSI

```
<kernel/fs.h>
inode_t *inode_get (dev_t device, ino_t ino);
```

ARGOMENTI

Argomento	Descrizione
`dev_t` ***device***	Dispositivo riferito a un'unità di memorizzazione, dove cercare l'inode numero ***ino***.
`ino_t` ***ino***	Numero di inode, relativo al file system contenuto nell'unità ***device***.

DESCRIZIONE

La funzione ***inode_get()*** consente di «aprire» un inode, fornendo il numero del dispositivo corrispondente all'unità di memorizzazione e il numero dell'inode del file system in essa contenuto. L'inode in questione potrebbe essere già stato aperto e quindi già disponibile in memoria nella tabella degli inode; in tal caso, la funzione si limita a incrementare il contatore dei riferimenti a tale inode, da parte del sistema in funzione, restituendo il puntatore all'elemento della tabella che lo contiene gia. Se invece l'inode non è ancora presente nella tabella rispettiva, la funzione deve provvedere a caricarlo.

Se si richiede un inode non ancora disponibile, contenuto in un'unità di cui non è ancora stato caricato il super blocco nella tabella rispettiva, la funzione deve provvedere anche a questo procedimento.

VALORE RESTITUITO

La funzione restituisce il puntatore all'elemento della tabella degli inode che rappresenta l'inode aperto. Se però si presenta un problema, restituisce il puntatore nullo e aggiorna la variabile ***errno*** del kernel.

ERRORI

Valore di *errno*	Significato
`EUNKNOWN`	Si tratta di un problema imprevisto e non meglio identificabile.
`ENFILE`	La tabella degli inode è già occupata completamente e non è possibile aprirne altri.
`ENODEV`	Il dispositivo richiesto non è valido.
`ENOENT`	Il numero di inode richiesto non esiste.
`EIO`	Errore nella lettura del file system.

FILE SORGENTI

‘`kernel/fs.h`’ [94.5]

‘`kernel/fs/inode_get.c`’ [94.5.16]

VEDERE ANCHE

offsetof(3) [88.88], *inode_put(9)* [93.6.20], *inode_reference(9)* [93.6.21], *sb_reference(9)* [93.6.29], *sb_inode_status(9)* [93.6.26], *dev_io(9)* [93.4.1].

93.6.16 os32: inode_pipe_make(9)

NOME

‘**`inode_pipe_make`**’ - creazione di una voce relativa a un condotto, nella tabella degli inode

SINTASSI

```
<kernel/fs.h>
inode_t *inode_pipe_make (void);
```

DESCRIZIONE

La funzione ***inode_pipe_make()*** produce una voce nella tabella degli inode, relativa a un condotto (*pipe*).

Questa funzione viene usata esclusivamente da *file_pipe_make(9)* [93.6.4], per creare una voce da usare condotto, nella tabella dei file.

VALORE RESTITUITO

La funzione restituisce il puntatore a un elemento della tabella degli inode, oppure il puntatore nullo in caso di errore, aggiornando la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Gli argomenti della chiamata non sono validi.
`ENFILE`	Non è possibile allocare un altro inode.

FILE SORGENTI

‘`kernel/fs.h`’ [94.5]

‘`kernel/fs/inode_pipe_make.c`’ [94.5.17]

VEDERE ANCHE

file_pipe_make(9) [93.6.4], *inode_reference(9)* [93.6.21].

93.6.17 os32: inode_pipe_read(9)

«

NOME

‘**`inode_pipe_read`**’ - lettura di un condotto rappresentato da un inode

SINTASSI

```
<kernel/fs.h>
ssize_t inode_file_read (inode_t *inode, void *buffer,
                         size_t count, int *eof);
```

ARGOMENTI

Argomento	Descrizione
`inode_t *inode`	Puntatore a un elemento della tabella degli inode, che rappresenta il condotto da leggere.
`void *buffer`	Puntatore all'area di memoria in cui scrivere ciò che si ottiene dalla lettura del condotto.
`size_t count`	Quantità massima di byte da leggere.
`int *eof`	Puntatore a un indicatore di fine file, da aggiornare (purché sia un puntatore valido) in base all'esito della lettura.

DESCRIZIONE

La funzione ***inode_pipe_read()*** legge il contenuto del file a cui si riferisce l'inode ***inode*** e se il puntatore ***eof*** è valido, aggiorna anche la variabile ****eof***.

VALORE RESTITUITO

La funzione restituisce la quantità di byte letti e resi effettivamente disponibili a partire da ciò a cui punta ***buffer***. Se la variabile ***var*** è un puntatore valido, aggiorna anche il suo valore, impostandolo a uno quando la lettura avviene mentre non ci sono più riferimenti in scrittura per il condotto.

ERRORI

Valore di *errno*	Significato
`EINVAL`	L'inode non è valido.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/inode_pipe_read.c`' [94.5.18]

VEDERE ANCHE

inode_file_read(9) [93.6.10].

93.6.18 os32: inode_pipe_write(9)

«

NOME

'**`inode_pipe_write`**' - scrittura di un condotto rappresentato da un inode

SINTASSI

```
<kernel/fs.h>
ssize_t inode_pipe_write (inode_t *inode, void *buffer,
                          size_t count);
```

ARGOMENTI

Argomento	Descrizione
`inode_t *`***inode***	Puntatore a un elemento della tabella degli inode che rappresenta il condotto in cui scrivere.
`void *`***buffer***	Puntatore all'area di memoria da cui trarre i dati da scrivere nel condotto.
`size_t` ***count***	Quantità massima di byte da scrivere.

DESCRIZIONE

La funzione ***inode_pipe_write()*** scrive nel condotto rappresentato da ***inode***, la quantità massima di byte indicati con ***count***, ciò che si trova in memoria a partire da ***buffer***.

VALORE RESTITUITO

La funzione restituisce la quantità di byte scritti.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	L'inode non è valido.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/inode_pipe_write.c`' [94.5.19]

VEDERE ANCHE

inode_file_write(9) [93.6.11].

93.6.19 os32: inode_print(9)

«

NOME

‘**inode_print**’ - visualizzazione diagnostica della tabella degli inode

SINTASSI

```
<kernel/fs.h>
void inode_print (void);
```

DESCRIZIONE

La funzione ***inode_print()*** visualizza sinteticamente i contenuti della tabella degli inode, per fini diagnostici.

FILE SORGENTI

‘kernel/fs.h’ [94.5]

‘kernel/fs/inode_print.c’ [94.5.20]

VEDERE ANCHE

sb_print(9) [93.6.28], *zone_print(9)* [93.6.36].

93.6.20 os32: inode_put(9)

«

NOME

‘**inode_put**’ - rilascio di un inode

SINTASSI

```
<kernel/fs.h>
int inode_put (inode_t *inode);
```

ARGOMENTI

Argomento	Descrizione
`inode_t *`***inode***	Puntatore a un elemento della tabella di inode.

DESCRIZIONE

La funzione ***inode_put()*** «chiude» un inode, riducendo il contatore degli accessi allo stesso. Tuttavia, se questo contatore, dopo il decremento, raggiunge lo zero, è necessario verificare se nel frattempo anche i collegamenti del file system si sono azzerati, perché in tal caso occorre anche rimuovere l'inode, nel senso di segnalarlo come libero per la creazione di un nuovo file. In ogni caso, le informazioni aggiornate dell'inode, ancora allocato o liberato, vengono memorizzate nel file system.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dalla variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il puntatore ***inode*** non è valido.
`EUNKNOWN`	Si tratta di un problema imprevisto e non meglio identificabile.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/inode_put.c`' [94.5.21]

VEDERE ANCHE

inode_truncate(9) [93.6.24], *inode_free(9)* [93.6.12], *inode_save(9)* [93.6.22].

93.6.21 os32: inode_reference(9)

«

NOME

'**inode_reference**' - riferimento a un elemento della tabella di inode

SINTASSI

```
<kernel/fs.h>
inode_t *inode_reference (dev_t device, ino_t ino);
```

ARGOMENTI

Argomento	Descrizione
`dev_t` ***device***	Dispositivo riferito a un'unità di memorizzazione, dove cercare l'inode numero ***ino***.
`ino_t` ***ino***	Numero di inode, relativo al file system contenuto nell'unità ***device***.

DESCRIZIONE

La funzione ***inode_reference()*** cerca nella tabella degli inode la voce corrispondente ai dati forniti come argomenti, ovvero quella dell'inode numero ***ino*** del file system contenuto nel dispositivo ***device***, restituendo il puntatore alla voce corrispondente. Tuttavia ci sono dei casi particolari:

- se il numero del dispositivo e quello dell'inode sono entrambi zero, viene restituito il puntatore all'inizio della tabella, ovvero al primo elemento della stessa;

- se il numero del dispositivo e quello dell'inode sono pari a un numero negativo (rispettivamente '`(dev_t) -1`' e '`(ino_t) -1`'), viene restituito il puntatore alla prima voce libera;
- se il numero del dispositivo è pari a zero e il numero dell'inode è pari a uno, si intende ricercare la voce dell'inode della directory radice del file system principale.

VALORE RESTITUITO

La funzione restituisce il puntatore a un elemento della tabella degli inode, se la ricerca si compie con successo. In caso di problemi, invece, la funzione restituisce il puntatore nullo e aggiorna la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`E_CANNOT_FIND_ROOT_DEVICE`	Nella tabella dei super blocchi non è possibile trovare il file system principale.
`E_CANNOT_FIND_ROOT_INODE`	Nella tabella degli inode non è possibile trovare la directory radice del file system principale.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/inode_reference.c`' [94.5.22]

VEDERE ANCHE

sb_reference(9) [93.6.29], *file_reference(9)* [93.6.5], *proc_reference(9)* [93.20.5].

93.6.22 os32: inode_save(9)

«

NOME

‘**inode_save**’ - memorizzazione dei dati di un inode

SINTASSI

```
<kernel/fs.h>
int inode_save (inode_t *inode);
```

ARGOMENTI

Argomento	Descrizione
`inode_t *`***inode***	Puntatore a una voce della tabella degli inode.

DESCRIZIONE

La funzione ***inode_save()*** memorizza l’inode a cui si riferisce la voce ****inode***, nel file system, ammesso che si tratti effettivamente di un inode relativo a un file system e che sia stato modificato dopo l’ultima memorizzazione precedente. In questo caso, la funzione, a sua volta, richiede la memorizzazione del super blocco.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dalla variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il puntatore ***inode*** è nullo.

FILE SORGENTI

‘`kernel/fs.h`’ [94.5]

‘`kernel/fs/inode_save.c`’ [94.5.23]

VEDERE ANCHE

sb_save(9) [93.6.30], *dev_io(9)* [93.4.1].

93.6.23 os32: inode_stdio_dev_make(9)

«

NOME

‘**`inode_stdio_dev_make`**’ - creazione di una voce relativa a un dispositivo di input-output standard, nella tabella degli inode

SINTASSI

```
<kernel/fs.h>
inode_t *inode_stdio_dev_make (dev_t device, mode_t mode);
```

ARGOMENTI

Argomento	Descrizione
`dev_t` ***device***	Il numero del dispositivo da usare per l’input o l’output.
`mode_t` ***mode***	Tipo di file di dispositivo: è praticamente obbligatorio l’uso di ‘**`S_IFCHR`**’.

DESCRIZIONE

La funzione ***inode_stdio_dev_make()*** produce una voce nella tabella degli inode, relativa a un dispositivo di input-output, da usare come flusso standard. In altri termini, serve per creare le voci della tabella degli inode, relative a standard input, standard output e standard error.

Questa funzione viene usata esclusivamente da *file_stdio_dev_make(9)* [93.6.6], per creare una voce da usare come flusso standard di input o di output, nella tabella dei file.

VALORE RESTITUITO

La funzione restituisce il puntatore a un elemento della tabella degli inode, oppure il puntatore nullo in caso di errore, aggiornando la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Gli argomenti della chiamata non sono validi.
`ENFILE`	Non è possibile allocare un altro inode.

FILE SORGENTI

‘`kernel/fs.h`’ [94.5]

‘`kernel/fs/inode_stdio_dev_make.c`’ [94.5.24]

VEDERE ANCHE

file_stdio_dev_make(9) [93.6.6], *inode_reference(9)* [93.6.21].

93.6.24 os32: inode_truncate(9)

«

NOME

‘**`inode_truncate`**’ - troncamento del file a cui si riferisce un inode

SINTASSI

```
<kernel/fs.h>
int inode_truncate (inode_t *inode);
```

ARGOMENTI

Argomento	Descrizione
`inode_t *`***inode***	Puntatore a una voce della tabella di inode.

DESCRIZIONE

La funzione ***inode_truncate()*** richiede che il puntatore ***inode*** si riferisca a una voce della tabella degli inode, relativa a un file contenuto in un file system. Lo scopo della funzione è annullare il contenuto di tale file, trasformandolo in un file vuoto.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dalla variabile ***errno*** del kernel.

ERRORI

Allo stato attuale dello sviluppo della funzione, non ci sono controlli e non sono previsti errori.

FILE SORGENTI

‘`kernel/fs.h`’ [94.5]

‘`kernel/fs/inode_truncate.c`’ [94.5.25]

VEDERE ANCHE

zone_free(9) [93.6.34], *sb_save(9)* [93.6.30], *inode_save(9)* [93.6.22].

93.6.25 os32: inode_zone(9)

«

NOME

‘**inode_zone**’ - traduzione del numero di zona relativo in un numero di zona assoluto

SINTASSI

```
<kernel/fs.h>
zno_t inode_zone (inode_t *inode, zno_t fzone, int write);
```

ARGOMENTI

Argomento	Descrizione
`inode_t` ***inode***	Puntatore a una voce della tabella di inode.
`zno_t` ***fzone***	Numero di zona relativo al file dell’inode preso in considerazione.
`int` ***write***	Valore da intendersi come *Vero* o *Falso*, con cui consentire o meno la creazione al volo di una zona mancante.

DESCRIZIONE

La funzione ***inode_zone()*** serve a tradurre il numero di una zona, inteso relativamente a un file, nel numero assoluto relativamente al file system in cui si trova. Tuttavia, un file può essere memorizzato effettivamente in modo discontinuo, ovvero con zone inesistenti nella sua parte centrale. Il contenuto di un file che non dispone effettivamente di zone allocate, corrisponde a un contenuto nullo dal punto di vista binario (zero binario), ma per la funzione, una zona assente comporta la restituzione di un valore nullo, perché nel file system non c’è. Pertanto, se l’argomento corrispondente al parametro ***write*** contiene un valore diverso da

zero, la funzione che non trova una zona, la alloca e quindi ne restituisce il numero.

VALORE RESTITUITO

La funzione restituisce il numero della zona che nel file system corrisponde a quella relativa richiesta per un certo file. Nel caso la zona non esista, perché non allocata, restituisce zero. Tuttavia, la zona zero di un file system Minix 1 esiste, ma contiene sostanzialmente le informazioni amministrative del super blocco, pertanto non può essere una traduzione valida di una zona di un file.

ERRORI

La funzione non prevede il verificarsi di errori.

FILE SORGENTI

‘`kernel/fs.h`’ [94.5]

‘`kernel/fs/inode_zone.c`’ [94.5.26]

VEDERE ANCHE

memset(3) [88.82], *zone_alloc(9)* [93.6.34], *zone_read(9)* [93.6.37], *zone_write(9)* [93.6.37].

93.6.26 os32: sb_inode_status(9)

«

NOME

‘**`sb_inode_status`**’, ‘**`sb_zone_status`**’ - verifica di utilizzazione attraverso il controllo delle mappe di inode e di zone

SINTASSI

```
<kernel/fs.h>
int sb_inode_status (sb_t *sb, ino_t ino);
int sb_zone_status  (sb_t *sb, zno_t zone);
```

ARGOMENTI

Argomento	Descrizione
`sb_t` ***sb***	Puntatore a una voce della tabella dei super blocchi.
`ino_t` ***ino***	Numero di inode.
`zno_t` ***ino***	Numero di zona.

DESCRIZIONE

La funzione ***sb_inode_status()*** verifica che un certo inode, individuato per numero, risulti utilizzato nel file system a cui si riferisce il super blocco a cui punta il primo argomento.

La funzione ***sb_zone_status()*** verifica che una certa zona, individuato per numero, risulti utilizzata nel file system a cui si riferisce il super blocco a cui punta il primo argomento.

La funzione ***sb_inode_status()*** viene usata soltanto da *inode_get(9)* [93.6.15]; la funzione ***sb_zone_status()*** non viene usata affatto.

VALORE RESTITUITO

Valore	Significato
1	L'inode o la zona risultano utilizzati.
0	L'inode o la zona risultano liberi (allocabili).
−1	Errore: è stato richiesto un numero di inode o di zona pari a zero, oppure ***sb*** è un puntatore nullo.

ERRORI

Valore di *errno*	Significato
`EINVAL`	È stato richiesto un numero di inode o di zona pari a zero, oppure ***sb*** è un puntatore nullo.

FILE SORGENTI

‘`kernel/fs.h`’ [94.5]

‘`kernel/fs/sb_inode_status.c`’ [94.5.32]

‘`kernel/fs/sb_zone_status.c`’ [94.5.37]

VEDERE ANCHE

inode_alloc(9) [93.6.7], *zone_alloc(9)* [93.6.34].

93.6.27 os32: sb_mount(9)

«

NOME

‘**`sb_mount`**’ - innesto di un dispositivo di memorizzazione

SINTASSI

```
<kernel/fs.h>
sb_t *sb_mount (dev_t device, inode_t **inode_mnt,
                int options);
```

ARGOMENTI

Argomento	Descrizione
`dev_t` ***device***	Dispositivo da innestare.
`inode_t **`***inode_mnt***	Puntatore di puntatore a una voce della tabella di inode. Il valore di ****inode_mnt*** potrebbe essere un puntatore nullo.
`int` ***options***	Opzioni per l'innesto.

DESCRIZIONE

La funzione ***sb_mount()*** innesta il dispositivo rappresentato numericamente dal primo parametro, sulla directory corrispondente all'inode a cui punta, indirettamente, il secondo parametro, con le opzioni del terzo parametro.

Il secondo parametro è un puntatore di puntatore al tipo '**`inode_t`**', in quanto il valore rappresentato da ****inode_mnt*** deve poter essere modificato dalla funzione. Infatti, quando si vuole innestare il file system principale, si crea una situazione particolare, perché la directory di innesto è la radice dello stesso file system da innestare; pertanto, ****inode_mnt*** deve essere un puntatore nullo ed è compito della funzione far sì che diventi il puntatore alla voce corretta nella tabella degli inode.

Questa funzione viene usata da *proc_init(9)* [93.20.3] per innestare il file system principale, e da *s_mount(9)* [93.12] per innestare un file system in condizioni diverse.

VALORE RESTITUITO

La funzione restituisce il puntatore all'elemento della tabella dei super blocchi che rappresenta il dispositivo innestato. In caso

si insuccesso, restituisce invece il puntatore nullo e aggiorna la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EBUSY`	Il dispositivo richiesto risulta già innestato; la directory di innesto è già utilizzata; la tabella dei super blocchi è già occupata del tutto.
`EIO`	Errore di input-output.
`ENODEV`	Il file system del dispositivo richiesto non può essere gestito.
`E_MAP_INODE_TOO_BIG`	La mappa che rappresenta lo stato di utilizzo degli inode del file system, è troppo grande e non può essere caricata in memoria.
`E_MAP_ZONE_TOO_BIG`	La mappa che rappresenta lo stato di utilizzo delle zone (i blocchi di dati del file system Minix 1) è troppo grande e non può essere caricata in memoria.
`E_DATA_ZONE_TOO_BIG`	Nel file system che si vorrebbe innestare, la dimensione della zona di dati è troppo grande rispetto alle possibilità di os32.
`EUNKNOWN`	Errore imprevisto e sconosciuto.

FILE SORGENTI

‘`kernel/fs.h`’ [94.5]

‘`kernel/fs/sb_mount.c`’ [94.5.33]

VEDERE ANCHE

sb_reference(9) [93.6.29], *dev_io(9)* [93.4.1], *inode_get(9)* [93.6.15].

93.6.28 os32: sb_print(9)

«

NOME

‘**sb_print**’ - visualizzazione diagnostica della tabella dei super blocchi

SINTASSI

```
<kernel/fs.h>
void sb_print (void);
```

DESCRIZIONE

La funzione ***sb_print()*** visualizza sinteticamente i contenuti della tabella dei super blocchi, per fini diagnostici.

FILE SORGENTI

‘kernel/fs.h’ [94.5]

‘kernel/fs/sb_print.c’ [94.5.34]

VEDERE ANCHE

inode_print(9) [93.6.19], *zone_print(9)* [93.6.36].

93.6.29 os32: sb_reference(9)

«

NOME

‘**sb_reference**’ - riferimento a un elemento della tabella dei super blocchi

SINTASSI

```
<kernel/fs.h>
sb_t *sb_reference (dev_t device);
```

ARGOMENTI

Argomento	Descrizione
`dev_t` ***device***	Dispositivo di un'unità di memorizzazione di massa.

DESCRIZIONE

La funzione ***sb_reference()*** serve a produrre il puntatore a una voce della tabella dei super blocchi. Se si fornisce il numero di un dispositivo già innestato nella tabella, si intende ottenere il puntatore alla voce relativa; se si fornisce il valore zero, si intende semplicemente avere un puntatore alla prima voce (ovvero all'inizio della tabella); se invece si fornisce il valore −1, si vuole ottenere il riferimento alla prima voce libera.

VALORE RESTITUITO

La funzione restituisce il puntatore all'elemento della tabella dei super blocchi che soddisfa la richiesta. In caso di errore, restituisce invece un puntatore nullo, ma senza dare informazioni aggiuntive con la variabile ***errno***, perché il motivo è implicito nel tipo di richiesta.

ERRORI

In caso di errore la variabile ***errno*** non viene aggiornata. Tuttavia, se l'errore deriva dalla richiesta di un dispositivo di memorizzazione, significa che non è presente nella tabella; se è stato richiesta una voce libera, significa che la tabella dei super blocchi è occupata completamente.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/fs_public.c`' [94.5.7]

‘kernel/fs/sb_reference.c’ [94.5.35]

VEDERE ANCHE

inode_reference(9) [93.6.21], *file_reference(9)* [93.6.5].

93.6.30 os32: sb_save(9)

«

NOME

‘**sb_save**’ - memorizzazione di un super blocco nel proprio file system

SINTASSI

```
<kernel/fs.h>
int sb_save (sb_t *sb);
```

ARGOMENTI

Argomento	Descrizione
`sb_t *sb`	Puntatore a una voce della tabella dei super blocchi in memoria.

DESCRIZIONE

La funzione ***sb_save()*** verifica se il super blocco conservato in memoria e rappresentato dal puntatore ***sb*** risulta modificato; in tal caso provvede ad aggiornarlo nell’unità di memorizzazione di origine, assieme alle mappe di utilizzo degli inode e delle zone di dati.

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
−1	Errore: la variabile ***errno*** viene impostata di conseguenza.

ERRORI

Valore di *errno*	Significato
`EINVAL`	Il riferimento al super blocco è un puntatore nullo.
`EIO`	Errore di input-output.

FILE SORGENTI

‘`kernel/fs.h`’ [94.5]

‘`kernel/fs/sb_save.c`’ [94.5.36]

VEDERE ANCHE

inode_save(9) [93.6.22], *dev_io(9)* [93.4.1].

93.6.31 os32: sb_zone_status(9)

Vedere *sb_inode_status(9)* [93.6.26].

93.6.32 os32: sock_free_port(9)

NOME

‘**`sock_free_port`**’ - scansione della tabella dei socket alla ricerca di una porta libera, da 1024 in su

SINTASSI

```
<kernel/fs.h>
h_port_t sock_free_port (void);
```

DESCRIZIONE

La funzione ***sock_free_port()*** restituisce il numero di una porta libera, dopo avere scandito la tabella dei socket. La ricerca riguarda esclusivamente le porte da 1024 in su, ovvero di quelle

che possono essere utilizzate da processi non privilegiati. Nel caso le porte siano tutte impegnate, la funzione restituisce il valore zero.

Il tipo '**h_port_t**' a cui si riferisce il valore restituito dalla funzione è un valore scalare, adatto a contenere il numero di una porta, rappresentato secondo l'ordinamento dei byte del sistema (*host byte order*).

VALORE RESTITUITO

La funzione restituisce il numero della porta libera trovata, oppure zero in mancanza di questo.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/sock_free_port.c`' [94.5.38]

VEDERE ANCHE

sock_reference(9) [93.6.33].

93.6.33 os32: sock_reference(9)

«

NOME

'**sock_reference**' - riferimento a un elemento della tabella dei socket

SINTASSI

```
<kernel/fs.h>
sock_t *sock_reference (int skn);
```

ARGOMENTI

Argomento	Descrizione
`int` ***skn***	Indice all'interno della tabella dei socket.

DESCRIZIONE

La funzione ***sock_reference()*** serve a produrre il puntatore a una voce della tabella dei super blocchi. Se si fornisce un indice positivo (maggiore o uguale a zero), si ottiene ***&sock_table[skn]***; altrimenti, con un valore negativo qualunque, viene scandita la tabella alla ricerca della prima voce libera, di cui si restituisce il puntatore allo stesso modo.

VALORE RESTITUITO

La funzione restituisce il puntatore all'elemento della tabella dei super blocchi che soddisfa la richiesta. Non si manifestano errori e se viene richiesto un indice positivo troppo grande, si ottiene un puntatore al di fuori della tabella.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/fs_public.c`' [94.5.7]

'`kernel/fs/sock_reference.c`' [94.5.39]

VEDERE ANCHE

sock_free_port(9) [93.6.32].

93.6.34 os32: zone_alloc(9)

«

NOME

‘**zone_alloc**’, ‘**zone_free**’ - allocazione di zone di dati

SINTASSI

```
<kernel/fs.h>
zno_t zone_alloc (sb_t *sb);
int   zone_free  (sb_t *sb, zno_t zone);
```

ARGOMENTI

Argomento	Descrizione
`sb_t *`***sb***	Puntatore a una voce della tabella dei super blocchi in memoria.
`zno_t` ***zone***	Numero di zona da liberare.

DESCRIZIONE

La funzione ***zone_alloc()*** occupa una zona nella mappa associata al super blocco a cui si riferisce ***sb***, restituendone il numero. La funzione ***zone_free()*** libera una zona che precedentemente risultava occupata nella mappa relativa.

VALORE RESTITUITO

La funzione ***zone_alloc()*** restituisce il numero della zona allocata. Se questo numero è zero, si tratta di un errore, e va considerato il contenuto della variabile ***errno***.

La funzione ***zone_free()*** restituisce zero in caso di successo, oppure -1 in caso di errore, aggiornando di conseguenza la variabile ***errno***.

ERRORI

Valore di *errno*	Significato
`EROFS`	Il file system è innestato in sola lettura, pertanto non è possibile apportare cambiamenti alla mappa di utilizzo delle zone.
`ENOSPC`	Non è possibile allocare una zona, perché non ce ne sono di libere.
`EINVAL`	L'argomento corrispondente a ***sb*** è un puntatore nullo; la zona di cui si richiede la liberazione è precedente alla prima zona dei dati (pertanto non può essere liberata, in quanto riguarda i dati amministrativi del super blocco); la zona da liberare è successiva allo spazio gestito dal file system.
`EUNKNOWN`	Errore imprevisto e sconosciuto.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/zone_alloc.c`' [94.5.40]

'`kernel/fs/zone_free.c`' [94.5.41]

VEDERE ANCHE

zone_write(9) [93.6.37], *sb_save(9)* [93.6.30].

93.6.35 os32: zone_free(9)

Vedere *zone_alloc(9)* [93.6.34].

93.6.36 os32: zone_print(9)

«

NOME

‘**zone_print**’ - visualizzazione diagnostica del contenuto della prima parte di una zona dati

SINTASSI

```
<kernel/fs.h>
void zone_print (sb_t *sb, zno_t zone);
```

DESCRIZIONE

La funzione ***zone_print()*** visualizza la prima parte del contenuto di una zona dati, riferita a un certo super blocco, in esadecimale, per fini diagnostici.

FILE SORGENTI

‘kernel/fs.h’ [94.5]

‘kernel/fs/zone_print.c’ [94.5.42]

VEDERE ANCHE

inode_print(9) [93.6.19], *sb_print(9)* [93.6.28].

93.6.37 os32: zone_read(9)

«

NOME

‘**zone_read**’, ‘**zone_write**’ - lettura o scrittura di una zona di dati

SINTASSI

```
<kernel/fs.h>
int zone_read  (sb_t *sb, zno_t zone, void *buffer);
int zone_write (sb_t *sb, zno_t zone, void *buffer);
```

ARGOMENTI

Argomento	Descrizione
`sb_t *sb`	Puntatore a una voce della tabella dei super blocchi in memoria.
`zno_t zone`	Numero di zona da leggere o da scrivere
`void *buffer`	Puntatore alla posizione iniziale in memoria dove depositare la zona letta o da dove trarre i dati per la scrittura della zona.

DESCRIZIONE

La funzione ***zone_read()*** legge una zona e ne trascrive il contenuto a partire da ***buffer***. La funzione ***zone_write()*** scrive una zona copiandovi al suo interno quanto si trova in memoria a partire da ***buffer***. La zona è individuata dal numero ***zone*** e riguarda il file system a cui si riferisce il super blocco ***sb***.

La lettura o la scrittura riguarda una zona soltanto, ma nella sua interezza.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dal contenuto della variabile ***errno*** del kernel.

ERRORI

Valore di *errno*	Significato
`EINVAL`	Gli argomenti non sono validi.
`EROFS`	Il file system è innestato in sola lettura.
`EIO`	Errore di input-output.

FILE SORGENTI

‘`kernel/fs.h`’ [94.5]

‘`kernel/fs/zone_read.c`’ [94.5.43]

‘`kernel/fs/zone_write.c`’ [94.5.44]

VEDERE ANCHE

zone_alloc(9) [93.6.34], *zone_free(9)* [93.6.34].

93.6.38 os32: path_device(9)

«

NOME

‘**`path_device`**’ - conversione di un file di dispositivo nel numero corrispondente

SINTASSI

```
<kernel/fs.h>
dev_t path_device (pid_t pid, const char *path);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Processo elaborativo per conto del quale si agisce.
`const char *`***path***	Il percorso del file di dispositivo.

DESCRIZIONE

La funzione ***path_device()*** consente di trarre il numero complessivo di un dispositivo, a partire da un file di dispositivo.

Questa funzione viene usata soltanto da *s_mount(9)* [93.12].

VALORE RESTITUITO

La funzione restituisce il numero del dispositivo corrispondente al file indicato, oppure il valore −1, in caso di errore, aggiornando la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`ENODEV`	Il file richiesto non è un file di dispositivo.
`ENOENT`	Il file richiesto non esiste.
`EACCES`	Il file richiesto non è accessibile secondo i privilegi del processo ***pid***.

FILE SORGENTI

‘`kernel/fs.h`’ [94.5]

‘`kernel/fs/path_device.c`’ [94.5.27]

VEDERE ANCHE

proc_reference(9) [93.20.5], *path_inode(9)* [93.6.41], *inode_put(9)* [93.6.20].

93.6.39 os32: path_fix(9)

«

NOME

‘**path_fix**’ - semplificazione di un percorso

SINTASSI

```
<kernel/fs.h>
int path_fix (char *path);
```

ARGOMENTI

Argomento	Descrizione
`char *`***path***	Il percorso da semplificare.

DESCRIZIONE

La funzione ***path_fix()*** legge la stringa del percorso ***path*** e la rielabora, semplificandolo. La semplificazione riguarda l’eliminazione di riferimenti inutili alla directory corrente e di indietreggiamenti. Il percorso può essere assoluto o relativo: la funzione non ne cambia l’origine.

VALORE RESTITUITO

La funzione restituisce sempre zero e non è prevista la manifestazione di errori.

FILE SORGENTI

‘`kernel/fs.h`’ [94.5]

'kernel/fs/path_fix.c' [94.5.28]

VEDERE ANCHE

strtok(3) [88.129], *strcmp(3)* [88.115], *strcat(3)* [88.113], *strncat(3)* [88.113], *strncpy(3)* [88.117].

93.6.40 os32: path_full(9)

NOME

'**path_full**' - traduzione di un percorso relativo in un percorso assoluto

SINTASSI

```
<kernel/fs.h>
int path_full (const char *path, const char *path_cwd,
               char *full_path);
```

ARGOMENTI

Argomento	Descrizione
`const char *`***path***	Il percorso relativo alla posizione ***path_cwd***.
`const char *`***path_cwd***	La directory corrente.
`char *`***full_path***	Il luogo in cui scrivere il percorso assoluto.

DESCRIZIONE

La funzione ***path_full()*** ricostruisce un percorso assoluto, mettendolo in memoria a partire da ciò a cui punta ***full_path***.

VALORE RESTITUITO

Valore	Significato del valore restituito
0	Operazione conclusa con successo.
−1	Errore, descritto dal contenuto della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	L'insieme degli argomenti non è valido.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/path_full.c`' [94.5.29]

VEDERE ANCHE

strtok(3) [88.129], *strcmp(3)* [88.115], *strcat(3)* [88.113], *strncat(3)* [88.113], *strncpy(3)* [88.117], *path_fix(9)* [93.6.39].

93.6.41 os32: path_inode(9)

«

NOME

'**`path_inode`**' - caricamento di un inode, partendo dal percorso del file

SINTASSI

```
<kernel/fs.h>
inode_t *path_inode (pid_t pid, const char *path);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`const char *`***path***	Il percorso del file del quale si vuole ottenere l'inode.

DESCRIZIONE

La funzione ***path_inode()*** carica un inode nella tabella degli inode, oppure lo localizza se questo è già caricato, partendo dal percorso di un file. L'operazione è subordinata all'accessibilità del percorso che conduce al file, nel senso che il processo ***pid*** deve avere il permesso di accesso («x») in tutti gli stadi dello stesso.

VALORE RESTITUITO

La funzione restituisce il puntatore all'elemento della tabella degli inode che contiene le informazioni caricate in memoria sull'inode. Se qualcosa non va, restituisce invece il puntatore nullo, aggiornando di conseguenza il contenuto della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`ENOENT`	Uno dei componenti del percorso non esiste.
`ENFILE`	Non è possibile allocare un inode ulteriore, perché la tabella è già occupata completamente.
`EIO`	Error di input-output.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

‘kernel/fs/path_inode.c’ [94.5.30]

VEDERE ANCHE

proc_reference(9) [93.20.5], *path_full(9)* [93.6.40], *inode_get(9)* [93.6.15], *inode_put(9)* [93.6.20], *inode_check(9)* [93.6.8], *inode_file_read(9)* [93.6.10].

93.6.42 os32: path_inode_link(9)

«

NOME

‘**path_inode_link**’ - creazione di un collegamento fisico o di un nuovo file

SINTASSI

```
<kernel/fs.h>
inode_t *path_inode_link (pid_t pid, const char *path,
                          inode_t *inode, mode_t mode);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`const char *`***path***	Il percorso del file per il quale si vuole creare il collegamento fisico.
`inode_t *`***inode***	Puntatore a una voce della tabella degli inode, alla quale si vuole collegare il nuovo file.
`mode_t` ***mode***	Nel caso l’inode non sia stato fornito, dovendo creare un nuovo file, questo parametro richiede il tipo e i permessi del file da creare.

DESCRIZIONE

La funzione ***path_inode_link()*** crea un collegamento fisico con il nome fornito in ***path***, riferito all'inode a cui punta ***inode***. Tuttavia, l'argomento corrispondente al parametro ***inode*** può essere un puntatore nullo, e in tal caso viene creato un file vuoto, allocando contestualmente un nuovo inode, usando l'argomento corrispondente al parametro ***mode*** per il tipo e la modalità dei permessi del nuovo file.

Il processo ***pid*** deve avere i permessi di accesso per tutte le directory che portano al file da collegare o da creare; inoltre, nell'ultima directory ci deve essere anche il permesso di scrittura, dovendo intervenire sulla stessa modificandola.

VALORE RESTITUITO

La funzione restituisce il puntatore all'elemento della tabella degli inode che descrive l'inode collegato o creato. In caso di problemi, restituisce invece il puntatore nullo, aggiornando di conseguenza il contenuto della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	L'insieme degli argomenti non è valido: se l'inode è stato indicato, il parametro ***mode*** deve essere nullo; al contrario, se l'inode non è specificato, il parametro ***mode*** deve contenere informazioni valide.
`EPERM`	Non è possibile creare il collegamento di un inode corrispondente a una directory.
`EMLINK`	Non è possibile creare altri collegamenti all'inode, il quale ha già raggiunto la quantità massima.
`EEXIST`	Il file ***path*** esiste già.
`EACCES`	Impossibile accedere al percorso che dovrebbe contenere il file da collegare.
`EROFS`	Il file system è innestato in sola lettura e non si può creare il collegamento.

FILE SORGENTI

'`kernel/fs.h`' [94.5]

'`kernel/fs/path_inode_link.c`' [94.5.31]

VEDERE ANCHE

proc_reference(9) [93.20.5], *path_inode(9)* [93.6.41], *inode_get(9)* [93.6.15], *inode_put(9)* [93.6.20], *inode_save(9)* [93.6.22], *inode_check(9)* [93.6.8], *inode_alloc(9)* [93.6.7], *inode_file_read(9)* [93.6.10], *inode_file_write(9)* [93.6.11].

93.7 os32: ibm_i386(9)

Il file ‘kernel/ibm_i386.h’ [94.6] descrive le funzioni e le macroistruzioni per la gestione dell’hardware e delle interruzioni, secondo l’architettura IBM i386.

La sezione 84.3 descrive complessivamente queste funzioni e le tabelle successive sono tratte da lì.

Tabella 84.28. Funzioni e macroistruzioni per l’input e l’output con le porte interne x86.

Funzione o macroistruzione	Descrizione
`uint32_t _in_8 (uint32_t port);` `unsigned int in_8 (port);`	Legge un valore a 8 bit da una porta. Listati 94.6 e 94.6.3.
`uint32_t _in_16 (uint32_t port);` `unsigned int in_16 (port);`	Legge un valore a 16 bit da una porta. Listati 94.6 e 94.6.1.

`void _out_8 (uint32_t port,`
`uint32_t value);`
`void out_8 (port, value);`
	Scrive un valore a 8 bit in una porta.
	Listati 94.6 e 94.6.6.

`void _out_16 (uint32_t port,`
`uint32_t value);`
`void out_16 (port, value);`
	Scrive un valore a 16 bit in una porta.
	Listati 94.6 e 94.6.4.

Tabella 84.29. Funzioni accessorie per il controllo delle interruzioni hardware.

Funzione o macroistruzione	Descrizione
`void cli (void);`	Disabilita le interruzioni hardware attraverso l'azzeramento dell'indicatore relativo nel registro EFLAGS. Listati 94.6 e 94.6.7.
`void sti (void);`	Abilita le interruzioni hardware attraverso l'attivazione dell'indicatore relativo nel registro EFLAGS. Listati 94.6 e 94.6.27.
`void irq_on (unsigned int irq);`	Abilita selettivamente l'interruzione hardware indicata per numero (da zero a 16). Listati 94.6 e 94.6.20.
`void irq_off (unsigned int irq);`	Disabilita selettivamente l'interruzione hardware indicata per numero (da zero a 16). Listati 94.6 e 94.6.19.

Tabella 84.32. Funzioni per la gestione della tabella GDT.

Funzione	Descrizione
`void gdt_segment (int segment, uint32_t base, uint32_t limit, bool present, bool code, unsigned char dpl);`	Scrive una voce della tabella GDT, sezionando e ricomponendo i dati come richiesto dal microprocessore. Listato 94.6.12.
`void gdt (void);`	Predispone e attiva la tabella GDT; per questo si avvale in modo particolare delle funzioni ***gdt_segment()*** e di ***gdt_load()***. Listato 94.6.8.
`void gdt_load (void *gdtr);`	Fa sì che il microprocessore carichi la tabella GDT, a partire dal puntatore al registro GDTR; registro che contiene l'informazione della collocazione in memoria della tabella GDT e della sua estensione. Listato 94.6.9.
`void gdt_print (void *gdtr, unsigned int first unsigned int last);`	Funzione diagnostica, usata per visualizzare il contenuto della tabella GDT in binario. Listato 94.6.10.

Tabella 84.35. Funzioni per la gestione della tabella IDT.

Funzione	Descrizione
`void idt_descriptor (int` ***desc*****`,`** `void *`***isr*****`,`** `uint16_t` ***selector*****`,`** `bool` ***present*****`,`** `char` ***type*****`,`** `char` ***dpl*****`);`	Scrive una voce della tabella IDT, sezionando e ricomponendo i dati come richiesto dal microprocessore. Listato 94.6.14.
`void idt (void);`	Predispone e attiva la tabella IDT; per questo si avvale in modo particolare delle funzioni ***idt_descriptor()*** e di ***idt_load()***. Listato 94.6.13.
`void idt_load (void *`***idtr***`);`	Fa sì che il microprocessore carichi la tabella IDT, a partire dal puntatore al registro IDTR; registro che contiene l'informazione della collocazione in memoria della tabella IDT e della sua estensione. Listato 94.6.16.

Funzione	Descrizione
`void idt_irq_remap (unsigned int` ***offset_1***`, unsigned int` ***offset_2***`);`	Modifica la mappatura delle interruzioni hardware (IRQ) spostando il primo gruppo a partire dal valore di ***offset_1*** e il secondo gruppo a partire da ***offset_2***. Listato 94.6.15.
`void idt_print (void *`***idtr***`, unsigned int` ***first*** `unsigned int` ***last***`);`	Funzione diagnostica, usata per visualizzare il contenuto della tabella IDT in binario. Listato 94.6.17.

Tabella 84.37. Funzioni per la gestione delle interruzioni.

Funzione	Descrizione
`void isr_`***n*** `(void);`	Si tratta di procedure attivate dalle interruzioni, dove per esempio ***isr_33()*** viene eseguita a seguito del verificarsi dell'interruzione numero 33, la quale ha origine da IRQ 1, ovvero dalla tastiera. L'indicazione di quale procedura attivare per ogni interruzione dipende dalla configurazione della tabella IDT. Listato 94.6.21.

Funzione	Descrizione
`void isr_exception_unrecoverable (uint32_t` ***eax***`, uint32_t` ***ecx***`, uint32_t` ***edx***`, uint32_t` ***ebx***`, uint32_t` ***ebp***`, uint32_t` ***esi***`, uint32_t` ***edi***`, uint32_t` ***ds***`, uint32_t` ***es***`, uint32_t` ***fs***`, uint32_t` ***gs***`, uint32_t` ***interrupt***`, uint32_t` ***error***`, uint32_t` ***eip***`, uint32_t` ***cs***`, uint32_t` ***eflags***`);`	Questa funzione viene usata all'interno del file '`isr.s`' per segnalare il verificarsi di un'eccezione non risolvibile, come nel caso di una divisione per zero. Pertanto, la funzione ha soprattutto un significato diagnostico. Listato 94.6.23.
`char *isr_exception_name (int` ***exception***`);`	Restituisce il puntatore alla stringa contenente il nome dell'eccezione corrispondente al numero di interruzione fornito. Viene usata da ***isr_exception_unrecoverable()*** per dare delle indicazioni comprensibili sull'eccezione che si è verificata. Listato 94.6.22.

Funzione	Descrizione
`void isr_irq_clear (uint32_t idtn);`	Avvisa il PIC (*programmable interrupt controller*) che l'interruzione hardware emessa è stata recepita e se ne possono ricevere altre. Tuttavia, essendoci due PIC, la funzione stabilisce quale dei due è coinvolto direttamente e di conseguenza come procedere. Listato 94.6.24.
`void isr_irq_clear_pic1 (void);`	Avvisa il PIC1 che l'interruzione hardware emessa è stata recepita e se ne possono ricevere altre. Listato 94.6.25.
`void isr_irq_clear_pic2 (void);`	Avvisa il PIC2 che l'interruzione hardware emessa è stata recepita e se ne possono ricevere altre. Listato 94.6.26.

93.8 os32: icmp(9)

Il file '`kernel/net/icmp.h`' [94.12.10] descrive le funzioni per la gestione del protocollo ICMP.

Per la descrizione sulla gestione del protocollo ICMP da parte di os32, si rimanda alla sezione 84.11. La tabella successiva che sintetizza l'uso delle funzioni di questo gruppo, è tratta da lì.

93.9 os32: ip(9)

«

Il file ‘`kernel/net/ip.h`’ [94.12.15] descrive le funzioni per la gestione del protocollo IPv4, esclusa però la questione degli instradamenti.

Per la descrizione sulla gestione del protocollo IPv4 eseguita da os32, si rimanda alla sezione 84.10. La tabella successiva che sintetizza l'uso delle funzioni di questo gruppo, è tratta da lì.

Tabella 84.124. Funzioni per la gestione dei pacchetti a livello IP.

Funzione	Descrizione
`uint16_t ip_checksum (uint16_t *data1, size_t size1, uint16_t *data2, size_t size2);`	Produce il codice di controllo usato nel protocollo IPv4, partendo da due blocchi di dati [94.12.16].
`int ip_rx (int n, int f);`	Si occupa di acquisire un pacchetto IPv4, dalla tabella ***net_table[]***, per copiarlo nella tabella ***ip_table[]*** e trattarlo per le questioni urgenti [94.12.21].
`ip_t *ip_reference (void);`	Restituisce il puntatore a un elemento della tabella ***ip_table[]*** che possa essere riutilizzato, perché mai usato prima oppure perché contenente il pacchetto IPv4 ricevuto che risulta essere più vecchio di tutti gli altri [94.12.20].

Funzione	Descrizione
`ssize_t ip_header (h_addr_t` ***src*** `, h_addr_t` ***dst*** `, uint16_t` ***id*** `, uint8_t` ***ttl*** `, uint8_t` ***protocol*** `, void *`***buffer*** `, size_t` ***length*** `);`	Scrive, in corrispondenza di ***buffer***, un'intestazione IPv4, sulla base dei dati contenuti negli altri parametri [94.12.17].
`int ip_tx (h_addr_t` ***src*** `, h_addr_t` ***dst*** `, int` ***protocol*** `, const void *`***buffer*** `, size_t` ***size*** `);`	Produce e trasmette un pacchetto IPv4, partendo dal contenuto che deve avere e dai dati necessari a costruire l'intestazione IPv4 [94.12.22].
`h_addr_t ip_mask (int` ***m*** `);`	A partire da un numero che rappresenta la dimensione di una maschera di rete, si ottiene il valore a 32 bit della maschera stessa. Per esempio, dal valore 16, si ottiene 255.255.0.0 [94.12.18].

93.10 os32: kbd(9)

Il file '`kernel/driver/kbd.h`' [94.4.15] descrive le funzioni per la gestione della tastiera, in relazione alla gestione complessiva dei terminali virtuali.

Per la descrizione dell'organizzazione della gestione della tastiera di os32, si rimanda alla sezione 84.7.5.1. La tabella successiva che

sintetizza l'uso delle funzioni di questo gruppo, è tratta da lì.

Tabella 84.87. Funzioni per la gestione della tastiera, dichiarate nel file di intestazione '`kernel/driver/kbd.h`' e descritte nei file contenuti nella directory '`kernel/driver/kbd/`'.

Funzione	Descrizione
`void kbd_isr (void);`	Questa funzione è chiamata dalla routine di gestione delle interruzioni da tastiera, contenuta nel file '`kernel/ibm_i386/isr.s`'. La funzione legge un carattere dalla porta di I/O 60_{16}, quindi lo interpreta e aggiorna il contenuto della variabile strutturata ***kbd*** di conseguenza.
`void kbd_load (void);`	Questa funzione è chiamata una sola volta da ***kmain()***, per associare la mappa della tastiera ai codici prodotti dalla stessa. Attualmente questa funzione produce esclusivamente l'associazione necessaria per una tastiera italiana.

93.11 os32: lib_k(9)

«

Il file '`kernel/lib_k.h`' [94.7] descrive alcune funzioni con nomi che iniziano per '**`k`**`_...`' (dove la lettera «k» sta per kernel) e riproducono il comportamento di funzioni standard, della libreria C. Per esempio, ***k_printf()*** è l'equivalente di ***printf()***, ma per la gestione interna del kernel.

Teoricamente, quando una funzione interna al kernel può ricondursi allo standard, dovrebbe avere il nome previsto. Tuttavia, per evitare di dover qualificare ogni volta l'ambito di una funzione, sono stati usati nomi differenti, ciò anche al fine di non creare complicazioni in fase di compilazione di tutto il sistema.

93.12 os32: lib_s(9)

Il file '`kernel/lib_s.h`' [94.8] descrive alcune funzioni con nomi che iniziano per '**s**_...' (dove la lettera «s» sta per sistema) e servono a realizzare le chiamate di sistema, all'interno del kernel. Per esempio, la funzione ***s__exit()*** realizza la funzione ***_exit()***. Il prototipo delle funzioni ***s_...()*** è lo stesso di quelle a cui si riferiscono, con l'aggiunta iniziale del numero del processo da cui ha origine la chiamata di sistema.

VEDERE ANCHE

sysroutine(9) [93.20.28], *brk(2)* [87.5], *chdir(2)* [87.6], *chmod(2)* [87.7], *chown(2)* [87.8], *clock(2)* [87.9], *close(2)* [87.10], *dup2(2)* [87.12], *dup(2)* [87.12], *_exit(2)* [87.2], *fchmod(2)* [87.7], *fchown(2)* [87.8], *fcntl(2)* [87.18], *fork(2)* [87.19], *fstat(2)* [87.55], *kill(2)* [87.29], *link(2)* [87.30], *longjmp(2)* [87.49], *lseek(2)* [87.33], *mkdir(2)* [87.34], *mknod(2)* [87.35], *mount(2)* [87.36], *open(2)* [87.37], *pipe(2)* [87.38], *read(2)* [87.39], *sbrk(2)* [87.5], *setegid(2)* [87.48], *seteuid(2)* [87.51], *setgid(2)* [87.48], *setjmp(2)* [87.49], *setuid(2)* [87.51], *signal(2)* [87.52], *stat(2)* [87.55], *stime(2)* [87.59], *tcgetattr(2)* [87.58], *tcsetattr(2)*

[87.58], *time(2)* [87.59], *umount(2)* [87.36], *unlink(2)* [87.62], *wait(2)* [87.63], *write(2)* [87.64].

93.13 os32: main(9)

«

Il file ‘`kernel/main.h`’ [94.9] descrive la funzione ***kmain()*** del kernel e altre funzioni accessorie, assieme al codice iniziale necessario per mettere in funzione il kernel stesso.

Si rimanda alla sezione 84.2 che descrive dettagliatamente il codice iniziale del kernel.

93.14 os32: memory(9)

«

Il file ‘`kernel/memory.h`’ [94.10] descrive le funzioni per la gestione della memoria, a livello di sistema.

Per la descrizione dell’organizzazione della gestione della memoria si rimanda alla sezione 84.6. Le tabelle successive che sintetizzano l’uso delle funzioni di questo gruppo, sono tratte da quel capitolo.

Tabella 84.77. Funzioni per la gestione della mappa della memoria, dichiarate nel file di intestazione ‘`kernel/memory.h`’ e realizzate nella directory ‘`kernel/memory/`’.

Funzione	Descrizione
`uint32_t *mb_reference (void);`	Restituisce il puntatore alla tabella dei blocchi di memoria, per uniformare l'accesso alla tabella dalle funzioni che non fanno parte del gruppo contenuto nella directory '`kernel/memory/`'.
`ssize_t mb_alloc (addr_t` ***address*****,** `size_t` ***size*****);**	Alloca la memoria a partire dall'indirizzo indicato, per la quantità di byte richiesta. L'allocazione ha termine anticipatamente se si incontra un blocco già utilizzato. La funzione restituisce la dimensione allocata effettivamente.
`void mb_free (addr_t` ***address*****,** `size_t` ***size*****);**	Libera la memoria a partire dall'indirizzo indicato, per la quantità di byte richiesta. Lo spazio viene liberato in ogni caso, anche se risulta già libero; tuttavia viene prodotto un avvertimento a video se si verifica tale ipotesi.

Funzione	Descrizione
`int mb_reduce (addr_t` ***address*****`,`** `size_t` ***new*****`,`** `size_t` ***previous*****`);`**	Riduce un'area di memoria già utilizzata. Restituisce zero se l'operazione si conclude con successo, oppure −1 in caso contrario, aggiornando la variabile ***errno*** di conseguenza.
`void mb_clean (addr_t` ***address*****`,`** `size_t` ***size*****`);`**	Azzera l'area di memoria specificata.
`addr_t mb_alloc_size (size_t` ***size*****`);`**	Alloca un'area di memoria della dimensione richiesta, restituendone l'indirizzo. La funzione conclude con successo il proprio lavoro se il valore restituito è diverso da zero; se invece l'indirizzo ottenuto è pari a zero si è verificato un errore che può essere verificato analizzando il contenuto della variabile ***errno***.

Funzione	Descrizione
`void mb_size (size_t` ***size*** `);`	Questa funzione, usata una sola volta all'interno di ***kmain()***, serve a definire la dimensione massima della memoria disponibile in blocchi. In pratica, le si fornisce la dimensione effettiva della memoria che viene così divisa per la dimensione del blocco, ignorando il resto. Questa informazione viene conservata nella variabile ***mb_max***.
`void mb_print (void);`	Funzione diagnostica che visualizza gli intervalli di memoria utilizzati, esprimendoli però in blocchi.

93.15 os32: multiboot(9)

Il file '`kernel/multiboot.h`' [94.11] descrive delle funzioni e un tipo derivato per il trattamento delle informazioni multiboot.

Per la descrizione della gestione delle informazioni multiboot da parte di os32, si rimanda alla sezione 84.2.1; la tabella successiva descrive brevemente le due funzioni disponibili per questa gestione.

Tabella 84.14. Funzioni per la gestione delle specifiche multiboot all'interno di os32.

Funzione	Descrizione
`void mboot_save (multiboot_t *mboot_data);`	Salva le informazioni multiboot all'interno della variabile strutturata pubblica ***multiboot***. Listati 94.11, 94.11.2 e 94.11.3.
`char ** mboot_cmdline_opt (const char *opt, const char *delim);`	Scandisce la stringa delle opzioni salvata all'interno di ***multiboot.cmdline***, alla ricerca di un'opzione il cui nome corrisponda alla stringa. Dopo il nome dell'opzione deve apparire il segno '=' e dopo devono trovarsi i valori associati all'opzione, separati da ***delim***. Questi valori vengono restituiti in forma di array di stringhe, dove l'ultima stringa si riconosce perché vuota. Listati 94.11, 94.11.2 e 94.11.1.

93.16 os32: ne2k(9)

Il file ‘kernel/driver/nic/ne2k.h’ [94.4.19] descrive le funzioni per la gestione delle interfacce di rete NE2K.

Per la descrizione della gestione dei dispositivi NE2K si rimanda alla sezione 84.9.1. La tabella successiva che sintetizza l’uso delle funzioni di questo gruppo, sono tratte da quella sezione.

Tabella 84.118. Funzioni per la gestione dell’interfaccia di rete NE2000.

Funzione	Descrizione
`int ne2k_check (uintptr_t io);`	Verifica se l’interfaccia corrispondente alla porta di I/O specificata è veramente di tipo NE2000 [94.4.20].
`int ne2k_isr (uintptr_t io);`	Verifica lo stato dell’interfaccia e ne acquisisce i dati, se disponibili. In caso di ricezione di una trama, viene chiamata la funzione ***ne2k_rx()*** per trasferirla nella memoria tampone della tabella delle interfacce [94.4.21].

Funzione	Descrizione
`int ne2k_isr_expect (uintptr_t io, unsigned int isr_expect);`	Rimane in attesa fino a che il registro ***ISR*** dell'interfaccia si attiva almeno un indicatore corrispondente a quanto richiesto con il parametro ***isr_expect***. Questa funzione viene usata, in particolare, nella trasmissione dei pacchetti, per i quali occorre verificare quando l'interfaccia ha completato il procedimento [94.4.22].
`int ne2k_reset (uintptr_t io, void *address);`	Azzera l'interfaccia e ne estrae l'indirizzo fisico, collocandolo in corrispondenza di ****address*** [94.4.23].
`int ne2k_rx (uintptr_t io);`	Copia tutte le trame accumulate nella memoria tampone interna dell'interfaccia, in quella della tabella delle interfacce [94.4.24].
`int ne2k_rx_reset (uintptr_t io);`	Reinizializza il processo di ricezione [94.4.25].

Funzione	Descrizione
`int ne2k_tx (uintptr_t` ***io***`,` `void *`***buffer***`, size_t` ***size***`);`	Trasmette una trama Ethernet, contenuta all'interno di ****buffer***, della lunghezza specificata da ***size***. La funzione attende il completamento dell'operazione, prima di concludere il proprio funzionamento [94.4.26].

93.17 os32: net(9)

Il file '`kernel/net.h`' [94.12] descrive le funzioni per la gestione della tabella delle interfacce, assieme ad altre funzioni accessorie relative alla trasmissione di trame Ethernet.

Per la descrizione sulla gestione della tabella delle interfacce, si rimanda alla sezione 84.9.2. La tabella successiva che sintetizza l'uso delle funzioni di questo gruppo, è tratta da lì.

Tabella 84.121. Funzioni per la gestione della tabella delle interfacce, contenute nella directory '`kernel/net/`', e altre accessorie relative alla gestione Ethernet.

Funzione	Descrizione
`net_buffer_eth_t *` `net_buffer_eth (int` ***n*****`);`**	Restituisce il puntatore a un elemento libero, o utilizzabile, della memoria tampone dell'interfaccia '**`net`*****n***', purché questa sia di tipo Ethernet [94.12.23].
`net_buffer_lo_t *` `net_buffer_lo (int` ***n*****`);`**	Restituisce il puntatore a un elemento libero, o utilizzabile, della memoria tampone dell'interfaccia '**`net`*****n***', purché questa sia di tipo *loopback*, ossia l'interfaccia locale virtuale [94.12.24].
`int net_index (h_addr_t` ***ip*****`);`**	Restituisce l'indice della tabella delle interfacce, corrispondente all'indirizzo IPv4 fornito come argomento [94.12.27].
`int net_index_eth (h_addr_t` ***ip*****`,`** `uint8_t` ***mac[6]*****`,`** `uintptr_t` ***io*****`);`**	Restituisce l'indice della tabella delle interfacce, corrispondente a uno dei dati forniti come argomento (i valori nulli vengono ignorati), purché si tratti di un'interfaccia Ethernet [94.12.28].

Funzione	Descrizione
`void net_init (void);`	Inizializza la gestione della rete, utilizzando le informazioni attraverso le opzioni di avvio per configurare anche le interfacce Ethernet [94.12.29].
`void net_rx (void);`	Scandisce i pacchetti memorizzati nella tabella delle interfacce, passandoli al gestore appropriato e rimuovendoli poi dalla tabella [94.12.32].

`int net_eth_ip_tx (h_addr_t` ***src*`,`
`h_addr_t` ***dst*`,`
`const void *`***packet*`,`
`size_t` ***size*`);`
	A partire da un pacchetto IPv4 completo e dagli indirizzi IPv4 di origine e di destinazione, viene assemblata e spedita una trama Ethernet. La funzione richiede separatamente l'indicazione degli indirizzi IPv4 di origine e destinazione, per semplificare il codice, evitando di estrapolarli dal pacchetto IPv4 stesso [94.12.25].

Funzione	Descrizione
`int net_eth_tx (int` ***n*****`,`** `void *`***buffer*****`,`** `size_t` ***size*****`);`**	Provvede a trasmettere una trama Ethernet attraverso l'interfaccia ***n*** (ovvero ***net_table[n]***), la quale deve essere di tipo Ethernet [94.12.26].

93.18 os32: part(9)

«

Il file '`kernel/part.h`' [94.13] descrive la struttura '**`part_t`**' e delle macro-variabili per la gestione delle partizioni. Solo la funzione ***dm_init()*** si avvale di questo file di intestazione.

93.19 os32: pci(9)

«

Il file '`kernel/driver/pci.h`' [94.4.27] descrive le funzioni per la gestione del bus PCI; ma in pratica è disponibile solo ***pci_init()*** che ha lo scopo di scandire il bus per trovare i dispositivi presenti e annotarli nella tabella PCI, funzione che viene usata una volta sola, all'avvio del sistema. A tale proposito va osservato che os32 è in grado di leggere solo il bus principale e non può aggiornare la tabella dei dispositivi.

Per una descrizione ulteriore del funzionamento del bus PCI, nell'ottica semplificata di os32, si veda la sezione 83.10.

93.20 os32: proc(9)

Il file ‘`kernel/proc.h`’ [94.14] descrive ciò che serve per la gestione dei processi. In modo particolare, in questo file si definisce il tipo derivato ‘**`proc_t`**’, con cui si realizza la tabella dei processi.

Si veda in particolare la spiegazione contenuta nella sezione 84.4 al riguardo della gestione dei processi.

93.20.1 os32: proc_available(9)

NOME

‘**`proc_available`**’ - inizializzazione di un processo libero

SINTASSI

```
<kernel/proc.h>
void proc_available (pid_t pid);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo da inizializzare.

DESCRIZIONE

La funzione ***proc_available()*** si limita a inizializzare, con valori appropriati, i dati di un processo nella tabella relativa, in modo che risulti correttamente uno spazio libero per le allocazioni successive.

Questa funzione viene usata da *proc_init(9)* [93.20.3], *proc_sig_chld(9)* [93.20.11], *s_wait(9)* [93.12].

FILE SORGENTI

‘kernel/proc.h’ [94.14]

‘kernel/proc/proc_public.c’ [94.14.5]

‘kernel/proc/proc_available.c’ [94.14.1]

93.20.2 os32: proc_dump_memory(9)

«

NOME

‘**proc_dump_memory**’ - copia di una porzione di memoria in un file

SINTASSI

```
<kernel/proc.h>
void proc_dump_memory (pid_t pid, addr_t address, size_t size,
                       char *name);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`addr_t` ***address***	Indirizzo efficace della memoria.
`size_t` ***size***	Quantità di byte da trascrivere, a partire dall’indirizzo efficace.
`char *`***name***	Nome del file da creare.

DESCRIZIONE

La funzione ***proc_dump_memory()*** salva in un file una porzione di memoria, secondo le coordinate fornita dagli argomenti.

Viene usata esclusivamente da *proc_sig_core(9)* [93.20.3], quando si riceve un segnale per cui è necessario scaricare la memoria di un processo. In quel caso, se il processo eliminato ha i permessi per scrivere nella directory radice, vengono creati due file: uno con l'immagine del segmento codice (‘`/core.i`’) e l'altro con l'immagine del segmento dati (‘`/core.d`’).

FILE SORGENTI

‘`kernel/proc.h`’ [94.14]

‘`kernel/proc/proc_sig_core.c`’ [94.14.14]

93.20.3 os32: proc_init(9)

NOME

‘**`proc_init`**’ - inizializzazione della gestione complessiva dei processi elaborativi

SINTASSI

```
<kernel/proc.h>
extern uint32_t _k_start;
extern uint32_t _k_end;
extern uint32_t _k_text_end;
extern uint32_t _k_data_end;
extern uint32_t _k_bss_end;
extern uint32_t _k_stack_top;
extern uint32_t _k_stack_bottom;
void proc_init (void);
```

ARGOMENTI

Argomento	Descrizione
`extern uint32_t` ***_k_start*** `;`	La variabile ***_k_start*** viene fornita dal file di configurazione '`kernel.ld`' e rappresenta l'indirizzo iniziale del kernel.
`extern uint32_t` ***_k_end*** `;`	La variabile ***_k_end*** viene fornita dal file di configurazione '`kernel.ld`' e rappresenta l'indirizzo finale del kernel, dati e pila inclusi.
`extern uint32_t` ***_k_text_end*** `;`	La variabile ***_k_text_end*** viene fornita dal file di configurazione '`kernel.ld`' e rappresenta l'indirizzo finale del codice del kernel.
`extern uint32_t` ***_k_data_end*** `;`	La variabile ***_k_data_end*** viene fornita dal file di configurazione '`kernel.ld`' e rappresenta l'indirizzo finale dei dati kernel, esclusa però l'area BSS e la pila.

Argomento	Descrizione
`extern uint32_t` ***_k_bss_end*** `;`	La variabile ***_k_bss_end*** viene fornita dal file di configurazione '`kernel.ld`' e rappresenta l'indirizzo finale dell'area BSS del kernel, ma coincide con ***_k_end***.
`extern uint32_t` ***_k_stack_top*** `;`	La variabile ***_k_stack_top*** viene fornita dal file '`kernel/main/stack.s`' e rappresenta l'indirizzo iniziale della pila dei dati del kernel.
`extern uint32_t` ***_k_stack_bottom*** `;`	La variabile ***_k_stack_bottom*** viene fornita dal file '`kernel/main/stack.s`' e rappresenta l'indirizzo finale della pila dei dati del kernel.

DESCRIZIONE

La funzione ***proc_init()*** viene usata una volta sola, dalla funzione *kmain(9)* [93.13], per predisporre la gestione dei processi. Per la precisione si occupa di:

- predisporre la tabella GDT, attraverso la chiamata della funzione ***gdt()***;
- impostare il temporizzatore in modo da fornire impulsi alla frequenza dichiarata nella macro-variabile ***CLOCKS_PER_SEC***, pari a 100 Hz;
- predisporre la tabella IDT, attraverso la chiamata della funzione ***idt()***;
- azzerare la tabella dei processi, inserendovi però i dati relativi al kernel (il processo zero);

- allocare la memoria già utilizzata dal kernel;
- attivare selettivamente le interruzioni hardware desiderate;
- attivare la gestione delle unità PATA;
- innestare il file system principale.

FILE SORGENTI

‘kernel/proc.h’ [94.14]

‘kernel/proc/proc_public.c’ [94.14.5]

‘kernel/proc/proc_init.c’ [94.14.3]

VEDERE ANCHE

proc_available(9) [93.20.1], *sb_mount(9)* [93.6.27].

93.20.4 os32: proc_print(9)

«

NOME

‘**proc_print**’ - visualizzazione diagnostica dei in corso

SINTASSI

```
<kernel/proc.h>
void proc_print (void);
```

DESCRIZIONE

La funzione ***proc_print()*** visualizza sinteticamente i processi in corso, per fini diagnostici. Viene usata nella funzione ***kmain()***, quando il kernel è in modalità di funzionamento interattivo.

FILE SORGENTI

‘kernel/proc.h’ [94.14]

‘kernel/proc/proc_print.c’ [94.14.4]

93.20.5 os32: proc_reference(9)

«

NOME

‘**proc_reference**’ - puntatore alla voce che rappresenta un certo processo

SINTASSI

```
<kernel/proc.h>
proc_t *proc_reference (pid_t pid);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo cercato nella tabella relativa.

DESCRIZIONE

La funzione ***proc_reference()*** serve a produrre il puntatore all’elemento dell’array ***proc_table[]*** che contiene i dati del processo indicato per numero come argomento.

Viene usata dalle funzioni che non fanno parte del gruppo di ‘`kernel/proc.h`’.

VALORE RESTITUITO

Restituisce il puntatore all’elemento della tabella ***proc_table[]*** che rappresenta il processo richiesto. Se il numero del processo richiesto non può esistere, la funzione restituisce il puntatore nullo ‘**NULL**’.

FILE SORGENTI

‘`kernel/proc.h`’ [94.14]

‘kernel/proc/proc_public.c’ [94.14.5]

‘kernel/proc/proc_reference.c’ [94.14.6]

93.20.6 os32: proc_sch_net(9)

«

NOME

‘**proc_sch_net**’ - operazioni di routine relative alla gestione della rete

SINTASSI

```
<kernel/proc.h>
void proc_sch_net (void);
```

DESCRIZIONE

La funzione ***proc_sch_net()*** ha il compito di scandire le interfacce di rete alla ricerca di operazioni da concludere e di pacchetti da acquisire; inoltre si occupa di aggiornare lo stato della gestione TCP e di risvegliare i processi in attesa di eventi relativi alla rete, se se ne presenta il motivo.

Questa funzione viene usata soltanto da *proc_scheduler(9)* [93.20.10], ogni volta che ci si prepara allo scambio con un altro processo.

FILE SORGENTI

‘kernel/proc.h’ [94.14]

‘kernel/proc/proc_scheduler.c’ [94.14.11]

‘kernel/proc/proc_sch_net.c’ [94.14.7]

VEDERE ANCHE

proc_scheduler(9) [93.20.10], *proc_sch_signals(9)* [93.20.7], *proc_sch_terminals(9)* [93.20.8], *proc_sch_timers(9)* [93.20.9].

93.20.7 os32: proc_sch_signals(9)

NOME

'**proc_sch_signals**' - verifica dei segnali dei processi

SINTASSI

```
<kernel/proc.h>
void proc_sch_signals (void);
```

DESCRIZIONE

La funzione ***proc_sch_signals()*** ha il compito di scandire tutti i processi della tabella ***proc_table[]***, per verificare lo stato di attivazione dei segnali e procedere di conseguenza.

Dal punto di vista pratico, la funzione si limita a scandire i numeri PID possibili, demandando ad altre funzioni il compito di fare qualcosa nel caso fosse attivato l'indicatore di un segnale.

Questa funzione viene usata soltanto da *proc_scheduler(9)* [93.20.10], ogni volta che ci si prepara allo scambio con un altro processo.

FILE SORGENTI

'kernel/proc.h' [94.14]

'kernel/proc/proc_scheduler.c' [94.14.11]

'kernel/proc/proc_sch_signals.c' [94.14.8]

VEDERE ANCHE

proc_sig_term(9) [93.20.20], *proc_sig_core(9)* [93.20.13], *proc_sig_chld(9)* [93.20.11], *proc_sig_cont(9)* [93.20.12], *proc_sig_stop(9)* [93.20.19].

93.20.8 os32: proc_sch_terminals(9)

«

NOME

'**proc_sch_terminals**' - acquisizione di un carattere dal terminale attivo

SINTASSI

```
<kernel/proc.h>
void proc_sch_terminals (void);
```

DESCRIZIONE

La funzione ***proc_sch_terminals()*** ha il compito di verificare la presenza di un carattere digitato dalla console. Se c'è effettivamente un carattere digitato, dopo aver determinato a quale terminale virtuale si riferisce, lo accumula nella sua riga di inserimento.

Successivamente verifica se quel terminale virtuale è associato a un gruppo di processi; se è così e se il carattere corrisponde a ***VINTR*** (di norma si tratta di ciò che viene prodotto dalla combinazione di tasti [*Ctrl c*]), invia il segnale SIGINT al processo più interno del gruppo, il quale dovrebbe corrispondere presumibilmente a quello in primo piano.

Indipendentemente dal fatto che il terminale appartenga a un gruppo di processi, controlla che il carattere inserito sia stato

ottenuto, rispettivamente, con le combinazioni di tasti [*Ctrl q*], [*Ctrl r*], [*Ctrl s*] e [*Ctrl t*], nel qual caso attiva la console virtuale relativa (dalla prima alla quarta), evitando di accumulare il carattere.

Se il carattere ricevuto è tale da fare intendere la conclusione di un inserimento (per esempio il carattere <*NL*>, prodotto dalla pressione di [*Invio*]), la funzione scandisce tutti i processi sospesi in attesa di input dal terminale, risvegliandoli (ogni processo deve poi verificare se effettivamente c'è qualcosa da trarre dal terminale per sé oppure no, e se non c'è dovrebbe rimettersi in attesa).

Questa funzione viene usata soltanto da *proc_scheduler(9)* [93.20.10], ogni volta che ci si prepara allo scambio con un altro processo.

FILE SORGENTI

'`kernel/proc.h`' [94.14]

'`kernel/proc/proc_scheduler.c`' [94.14.11]

'`kernel/proc/proc_sch_terminals.c`' [94.14.9]

93.20.9 os32: proc_sch_timers(9)

NOME

'**`proc_sch_timers`**' - verifica dell'incremento del contatore del tempo

SINTASSI

```
<kernel/proc.h>
void proc_sch_timers (void);
```

DESCRIZIONE

La funzione ***proc_sch_timers()*** verifica che il calendario si sia incrementato di almeno una unità temporale (per os32 è un secondo soltanto) e se è così, va a risvegliare tutti i processi sospesi in attesa del passaggio di un certo tempo. Tali processi, una volta messi effettivamente in funzione, devono verificare che sia trascorsa effettivamente la quantità di tempo desiderata, altrimenti devono rimettersi a riposo in attesa del tempo rimanente.

Questa funzione viene usata soltanto da *proc_scheduler(9)* [93.20.10], ogni volta che ci si prepara allo scambio con un altro processo.

FILE SORGENTI

‘`kernel/proc.h`’ [94.14]

‘`kernel/proc/proc_scheduler.c`’ [94.14.11]

‘`kernel/proc/proc_sch_timers.c`’ [94.14.10]

93.20.10 os32: proc_scheduler(9)

«

NOME

‘**`proc_scheduler`**’ - schedulatore

SINTASSI

```
<kernel/proc.h>
extern uint32_t _ksp;
extern uint32_t proc_stack_pointer;
extern uint16_t proc_stack_segment_selector;
extern pid_t    proc_current;
void proc_scheduler (void);
```

ARGOMENTI

Argomento	Descrizione
`extern uint32_t` ***_ksp*** `;`	La variabile ***_ksp*** contiene l'indice della pila del kernel.
`extern uint32_t` ***proc_stack_pointer*** `;`	La variabile ***proc_stack_pointer*** contiene l'indice della pila del processo interrotto, dal punto di vista del segmento dati del processo stesso.
`extern uint16_t` ***proc_stack_segment_selector*** `;`	La variabile ***proc_stack_segment_selector*** contiene il valore del registro ***SS*** (*stack segment*) relativo al processo sospeso.
`extern pid_t` ***proc_current*** `;`	La variabile ***proc_current*** contiene il numero del processo corrente, appena sospeso.

DESCRIZIONE

La funzione ***proc_scheduler()*** viene avviata a seguito di un'interruzione hardware, dovuta al temporizzatore, oppure a seguito di un'interruzione software, dovuta a una chiamata di sistema.

La prima cosa che fa la funzione consiste nel verificare che il valore dell'indice della pila del processo interrotto non superi lo spazio disponibile per la pila stessa. Diversamente il proces-

so viene eliminato forzatamente, con una segnalazione adeguata sul terminale attivo. Si ottiene comunque una segnalazione se l'indice si avvicina pericolosamente al limite.

Successivamente la funzione svolge delle operazioni che riguardano tutti i processi: aggiorna i contatori dei processi che attendono lo scadere di un certo tempo; verifica la presenza di segnali e predispone le azioni relative; raccoglie l'input dai terminali. Quindi aggiorna il tempo di utilizzo della CPU del processo appena interrotto.

Poi inizia la ricerca di un altro processo, candidato a essere ripreso al posto di quello interrotto. La ricerca inizia dal processo successivo a quello interrotto, senza considerare alcun criterio di precedenza. Il ciclo termina se la ricerca incontra di nuovo il processo di partenza. All'interno di questo ciclo di ricerca, se si incontra un processo pronto per essere messo in funzione, lo si scambia con quello interrotto: in pratica si salva il valore attuale dell'indice della pila, si scambiano gli stati e si aggiornano i valori di ***proc_current***, ***proc_stack_segment_selector*** e ***proc_stack_pointer***, in modo da ottenere effettivamente lo scambio all'uscita dalla funzione. Alla fine del ciclo, occorre verificare se esiste effettivamente un processo successivo attivato, perché in caso contrario, si lascia il controllo direttamente al kernel. In fine, si salva il valore accumulato in precedenza dell'indice della pila del kernel, nella variabile ***_ksp***.

Questa funzione viene usata dalla routine '**`irq_timer`**' del file '`kernel/ibm_i386/isr.s`' e dalla funzione *sysroutine(9)* [93.20.28].

FILE SORGENTI

'kernel/proc.h' [94.14]

'kernel/ibm_i386/isr.s' [94.6.21]

'kernel/proc/sysroutine.c' [94.14.28]

'kernel/proc/proc_scheduler.c' [94.14.11]

VEDERE ANCHE

proc_sch_timers(9) [93.20.9], *proc_sch_signals(9)* [93.20.7], *proc_sch_terminals(9)* [93.20.8].

93.20.11 os32: proc_sig_chld(9)

«

NOME

'**proc_sig_chld**' - procedura associata alla ricezione di un segnale SIGCHLD

SINTASSI

```
<kernel/proc.h>
void proc_sig_chld (pid_t parent, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***parent***	Numero del processo considerato, il quale potrebbe avere ricevuto un segnale SIGCHLD.
`int` ***sig***	Numero del segnale: deve trattarsi esclusivamente di quanto corrispondente a SIGCHLD.

DESCRIZIONE

La funzione ***proc_sig_chld()*** si occupa di verificare che il processo specificato con il parametro ***parent*** abbia ricevuto precedentemente un segnale SIGCHLD. Se risulta effettivamente così, allora va a verificare se tale segnale risulta ignorato per quel processo: se è preso in considerazione verifica ancora se quel processo è sospeso proprio in attesa di un segnale SIGCHLD. Se si tratta di un processo che sta attendendo tale segnale, allora viene risvegliato, altrimenti, sempre ammesso che comunque il segnale non sia ignorato, la funzione elimina tutti i processi figli di ***parent***, i quali risultano già defunti, ma non ancora rimossi dalla tabella dei processi (pertanto processi «zombie»).

In pratica, se il processo ***parent*** sta attendendo un segnale SIGCHLD, significa che al risveglio si aspetta di verificare la morte di uno dei suoi processi figli, in modo da poter ottenere il valore di uscita con cui questo si è concluso. Diversamente, non c'è modo di informare il processo ***parent*** di tali conclusioni, per cui a nulla servirebbe continuare a mantenerne le tracce nella tabella dei processi.

Questa funzione viene usata soltanto da *proc_sch_signals(9)* [93.20.7].

FILE SORGENTI

'`kernel/proc.h`' [94.14]

'`kernel/proc/proc_sig_chld.c`' [94.14.12]

VEDERE ANCHE

proc_sig_status(9) [93.20.18], *proc_sig_ignore(9)* [93.20.15], *proc_sig_off(9)* [93.20.17].

93.20.12 os32: proc_sig_cont(9)

«

NOME

‘**proc_sig_cont**’ - ripresa di un processo sospeso in attesa di qualcosa

SINTASSI

```
<kernel/proc.h>
void proc_sig_cont (pid_t pid, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Numero del processo considerato.
`int` ***sig***	Numero del segnale: deve trattarsi esclusivamente di quanto corrispondente a SIGCONT.

DESCRIZIONE

La funzione ***proc_sig_cont()*** si occupa di verificare che il processo specificato con il parametro ***pid*** abbia ricevuto precedentemente un segnale SIGCONT e che questo non sia stato disabilitato. In tal caso, assegna al processo lo status di «pronto» (‘**PROC_READY**’), ammesso che non si trovasse già in questa situazione.

Lo scopo del segnale SIGCONT è quindi quello di far riprendere un processo che in precedenza fosse stato sospeso attraverso un segnale SIGSTOP, SIGTSTP, SIGTTIN oppure SIGTTOU.

Questa funzione viene usata soltanto da *proc_sch_signals(9)* [93.20.7].

FILE SORGENTI

‘kernel/proc.h’ [94.14]

‘kernel/proc/proc_sig_cont.c’ [94.14.13]

VEDERE ANCHE

proc_sig_status(9) [93.20.18], *proc_sig_ignore(9)* [93.20.15], *proc_sig_off(9)* [93.20.17].

93.20.13 os32: proc_sig_core(9)

«

NOME

‘**proc_sig_core**’ - chiusura di un processo e scarico della memoria su file

SINTASSI

```
<kernel/proc.h>
void proc_sig_core (pid_t pid, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Numero del processo considerato.
`int` ***sig***	Numero del segnale: deve trattarsi di un segnale a cui si associa in modo predefinito la conclusione e lo scarico della memoria.

DESCRIZIONE

La funzione ***proc_sig_core()*** si occupa di verificare che il processo specificato con il parametro ***pid*** abbia ricevuto precedentemente un segnale tale da richiedere la conclusione e lo scarico

della memoria del processo stesso, e che il segnale in questione non sia stato disabilitato. In tal caso, la funzione chiude il processo, ma prima ne scarica la memoria su uno o due file, avvalendosi per questo della funzione *proc_dump_memory(9)* [93.20.2].

Un segnale atto a produrre lo scarico della memoria, potrebbe essere prodotto anche a seguito di un errore rilevato dalla CPU, come una divisione per zero. Tuttavia, il kernel di os32 non riesce a intrappolare errori di questo tipo, dato che dalla tabella IVT vengono presi in considerazione soltanto l'impulso del temporizzatore e le chiamate di sistema. In altri termini, se un programma produce effettivamente un errore così grave da essere rilevato dalla CPU, al sistema operativo non arriva alcuna comunicazione. Pertanto, tali segnali possono essere soltanto provocati deliberatamente.

Lo scarico della memoria, nell'eventualità di un errore così grave, dovrebbe servire per consentire un'analisi dello stato del processo nel momento del verificarsi di un errore fatale. Sotto questo aspetto, va anche considerato che l'area dati dei processi è priva di etichette che possano agevolare l'interpretazione dei contenuti e, di conseguenza, non ci sono strumenti che consentano tale attività.

Questa funzione viene usata soltanto da *proc_sch_signals(9)* [93.20.7].

FILE SORGENTI

'`kernel/proc.h`' [94.14]

'`kernel/proc/proc_sig_core.c`' [94.14.14]

VEDERE ANCHE

proc_sig_status(9) [93.20.18], *proc_sig_ignore(9)* [93.20.15], *proc_sig_off(9)* [93.20.17], *proc_dump_memory(9)* [93.20.2].

93.20.14 os32: proc_sig_handler(9)

«

NOME

'**proc_sig_handler**' - attivazione di una funzione a seguito del recepimento di un segnale

SINTASSI

```
<kernel/proc.h>
void proc_sig_handler (pid_t pid, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Numero del processo considerato.
`int` ***sig***	Numero del segnale da verificare.

DESCRIZIONE

La funzione ***proc_sig_handler()*** verifica se, per un certo processo ***pid***, il segnale ***sig*** risulti associato a una funzione. Se è così, modifica la pila dei dati del processo in modo da far si che alla sua ripresa, venga azionata la funzione programmata, prima di riprendere con l'attività precedente. Contestualmente, il segnale ***sig*** viene liberato dall'associazione a una funzione.

Questa funzione viene usata da *proc_sig_cont(9)* [93.20.12], *proc_sig_core(9)* [93.20.13], *proc_sig_stop(9)* [93.20.19] e

proc_sig_term(9) [93.20.20], per verificare se un segnale sia stato associato a una funzione; prima di decidere di eseguire l'azione predefinita, in mancanza di tale associazione.

FILE SORGENTI

'`kernel/proc.h`' [94.14]

'`kernel/proc/proc_sig_handler.c`' [94.14.15]

VEDERE ANCHE

proc_sig_cont(9) [93.20.12], *proc_sig_core(9)* [93.20.13], *proc_sig_stop(9)* [93.20.19]m *proc_sig_term(9)* [93.20.20].

93.20.15 os32: proc_sig_ignore(9)

NOME

'**`proc_sig_ignore`**' - verifica dello stato di inibizione di un segnale

SINTASSI

```
<kernel/proc.h>
int proc_sig_ignore (pid_t pid, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Numero del processo considerato.
`int` ***sig***	Numero del segnale da verificare.

DESCRIZIONE

La funzione ***proc_sig_ignore()*** verifica se, per un certo processo ***pid***, il segnale ***sig*** risulti inibito.

Questa funzione viene usata da *proc_sig_chld(9)* [93.20.11], *proc_sig_cont(9)* [93.20.12], *proc_sig_core(9)* [93.20.13], *proc_sig_stop(9)* [93.20.19] e *proc_sig_term(9)* [93.20.20], per verificare se un segnale sia stato inibito, prima di applicarne le conseguenze, nel caso fosse stato ricevuto.

VALORE RESTITUITO

Valore	Significato
1	Il segnale risulta bloccato (inibito).
0	Il segnale è abilitato regolarmente.

FILE SORGENTI

‘`kernel/proc.h`’ [94.14]

‘`kernel/proc/proc_sig_ignore.c`’ [94.14.16]

VEDERE ANCHE

proc_sig_chld(9) [93.20.11], *proc_sig_cont(9)* [93.20.12], *proc_sig_core(9)* [93.20.13], *proc_sig_stop(9)* [93.20.19]m *proc_sig_term(9)* [93.20.20].

93.20.16 os32: proc_sig_off(9)

«

Vedere *proc_sig_on(9)* [93.20.17].

93.20.17 os32: proc_sig_on(9)

«

NOME

‘**`proc_sig_on`**’, ‘**`proc_sig_off`**’ - registrazione o cancellazione di un segnale per un processo

SINTASSI

```
<kernel/proc.h>
void proc_sig_on  (pid_t pid, int sig);
void proc_sig_off (pid_t pid, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Numero del processo considerato.
`int` ***sig***	Numero del segnale da registrare o da cancellare.

DESCRIZIONE

La funzione ***proc_sig_on()*** annota per il processo ***pid*** la ricezione del segnale ***sig***; la funzione ***proc_sig_off()*** procede invece in senso opposto, cancellando quel segnale.

La funzione ***proc_sig_off()*** viene usata quando l'azione prevista per un segnale che risulta ricevuto è stata eseguita, allo scopo di riportare l'indicatore di quel segnale in una condizione di riposo. Si tratta delle funzioni *proc_sig_chld(9)* [93.20.11], *proc_sig_cont(9)* [93.20.12], *proc_sig_core(9)* [93.20.13], *proc_sig_stop(9)* [93.20.19] e *proc_sig_term(9)* [93.20.20].

La funzione ***proc_sig_on()*** viene usata quando risulta acquisito un segnale o quando il contesto lo deve produrre, per annotarlo. Si tratta delle funzioni *s__exit(9)* [93.12] e *s_kill(9)* [93.12].

FILE SORGENTI

'`kernel/proc.h`' [94.14]

'`kernel/proc/proc_sig_on.c`' [94.14.18]

‘kernel/proc/proc_sig_off.c’ [94.14.17]

VEDERE ANCHE

proc_sig_chld(9) [93.20.11], *proc_sig_cont(9)* [93.20.12], *proc_sig_core(9)* [93.20.13], *proc_sig_stop(9)* [93.20.19], *proc_sig_term(9)* [93.20.20].

93.20.18 os32: proc_sig_status(9)

«

NOME

‘**proc_sig_status**’ - verifica dello stato di ricezione di un segnale

SINTASSI

```
<kernel/proc.h>
int proc_sig_status (pid_t pid, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Numero del processo considerato.
`int` ***sig***	Numero del segnale da verificare.

DESCRIZIONE

La funzione ***proc_sig_status()*** verifica se, per un certo processo ***pid***, il segnale ***sig*** risulti essere stato ricevuto (registrato).

Questa funzione viene usata da *proc_sig_chld(9)* [93.20.11], *proc_sig_cont(9)* [93.20.12], *proc_sig_core(9)* [93.20.13], *proc_sig_stop(9)* [93.20.19] e *proc_sig_term(9)* [93.20.20], per

verificare se un segnale è stato ricevuto effettivamente, prima di applicarne eventualmente le conseguenze.

VALORE RESTITUITO

Valore	Significato
1	Il segnale risulta ricevuto.
0	Il segnale risulta cancellato.

FILE SORGENTI

‘`kernel/proc.h`’ [94.14]

‘`kernel/proc/proc_sig_status.c`’ [94.14.19]

VEDERE ANCHE

proc_sig_chld(9) [93.20.11], *proc_sig_cont(9)* [93.20.12], *proc_sig_core(9)* [93.20.13], *proc_sig_stop(9)* [93.20.19], *proc_sig_term(9)* [93.20.20].

93.20.19 os32: proc_sig_stop(9)

NOME

‘**`proc_sig_stop`**’ - sospensione di un processo

SINTASSI

```
<kernel/proc.h>
void proc_sig_stop (pid_t pid, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Numero del processo considerato.
`int` ***sig***	Numero del segnale: deve trattarsi di SIGSTOP, SIGTSTP, SIGTTIN o SIGTTOU.

DESCRIZIONE

La funzione ***proc_sig_stop()*** si occupa di verificare che il processo specificato con il parametro ***pid*** abbia ricevuto precedentemente un segnale SIGSTOP, SIGTSTP, SIGTTIN o SIGTTOU, e che questo non sia stato disabilitato. In tal caso, sospende il processo, lasciandolo in attesa di un segnale (SIGCONT).

Questa funzione viene usata soltanto da *proc_sch_signals(9)* [93.20.7].

FILE SORGENTI

‘`kernel/proc.h`’ [94.14]

‘`kernel/proc/proc_sig_stop.c`’ [94.14.20]

VEDERE ANCHE

proc_sig_status(9) [93.20.18], *proc_sig_ignore(9)* [93.20.15], *proc_sig_off(9)* [93.20.17].

93.20.20 os32: proc_sig_term(9)

«

NOME

‘**`proc_sig_term`**’ - conclusione di un processo

SINTASSI

```
<kernel/proc.h>
void proc_sig_term (pid_t pid, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Numero del processo considerato.
`int` ***sig***	Numero del segnale: deve trattarsi di un segnale per cui si associa la conclusione del processo, ma senza lo scarico della memoria.

DESCRIZIONE

La funzione ***proc_sig_term()*** si occupa di verificare che il processo specificato con il parametro ***pid*** abbia ricevuto precedentemente un segnale per cui si prevede generalmente la conclusione del processo. Inoltre, la funzione verifica che il segnale non sia stato inibito, con l'eccezione che per il segnale SIGKILL un'eventuale inibizione non viene considerata (in quanto segnale non mascherabile). Se il segnale risulta ricevuto e valido, procede con la conclusione del processo.

Questa funzione viene usata soltanto da *proc_sch_signals(9)* [93.20.7].

FILE SORGENTI

'`kernel/proc.h`' [94.14]

'`kernel/proc/proc_sig_term.c`' [94.14.21]

VEDERE ANCHE

proc_sig_status(9) [93.20.18], *proc_sig_ignore(9)* [93.20.15], *proc_sig_off(9)* [93.20.17].

93.20.21 os32: proc_sys_exec(9)

«

NOME

'**proc_sys_exec**' - sostituzione di un processo esistente con un altro, ottenuto dal caricamento di un file eseguibile

SINTASSI

```
<kernel/proc.h>
int proc_sys_exec (pid_t pid, const char *path,
                   unsigned int argc, char *arg_data,
                   unsigned int envc, char *env_data)
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo corrispondente.
`const char *`***path***	Il percorso assoluto del file da caricare ed eseguire.
`unsigned int` ***argc***	La quantità di argomenti per l'avvio del nuovo processo, incluso il nome del processo stesso.
`char *`***arg_data***	Una sequenza di stringhe (dopo la terminazione di una inizia la successiva), ognuna contenente un argomento da passare al processo.
`unsigned int` ***envc***	La quantità di variabili di ambiente da passare al nuovo processo.
`char *`***env_data***	Una sequenza di stringhe (dopo la terminazione di una inizia la successiva), ognuna contenente l'assegnamento di una variabile di ambiente.

I parametri ***arg_data*** e ***env_data*** sono stringhe multiple, nel senso che sono separate le une dalle altre dal codice nullo di terminazione. Per sapere quante sono effettivamente le stringhe da cercare a partire dai puntatori che costituiscono effettivamente questi due parametri, si usano ***argc*** e ***envc***.

DESCRIZIONE

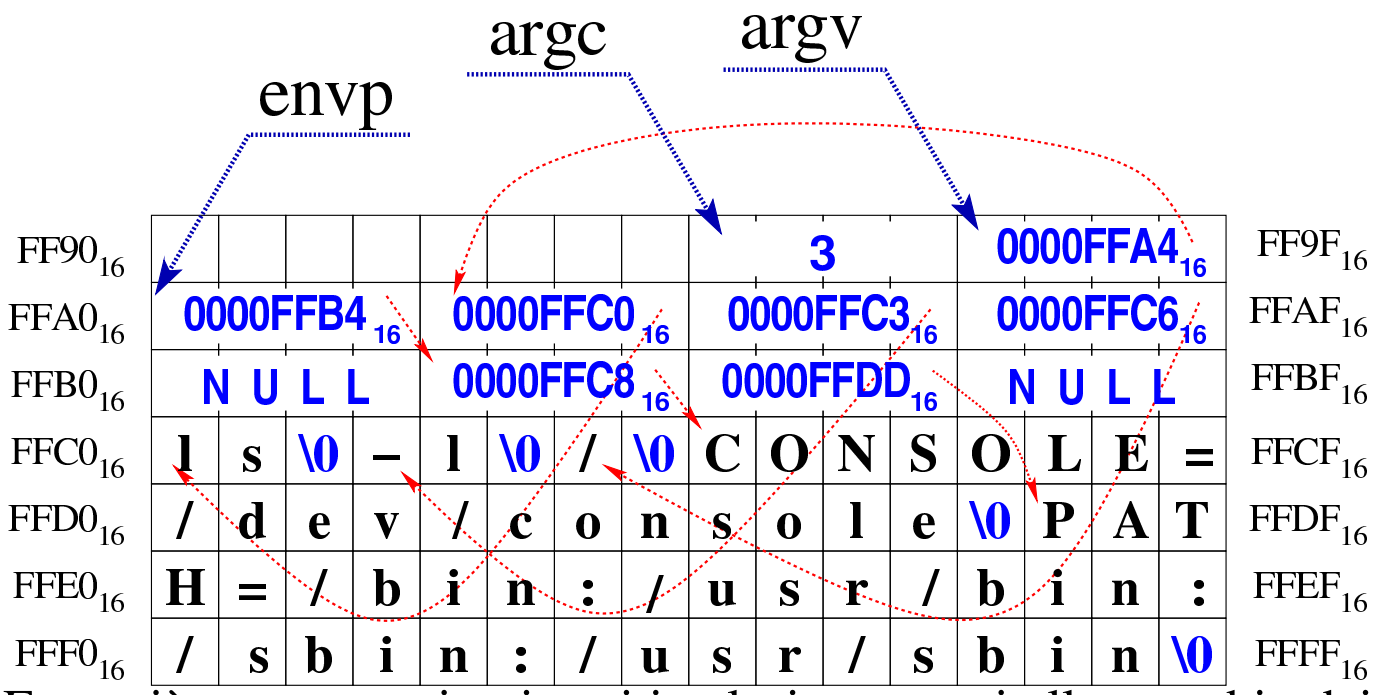
La funzione ***proc_sys_exec()*** serve a mettere in pratica la chiamata di sistema *execve(2)* [87.14], destinata a rimpiazzare il processo in corso con un nuovo processo, caricato da un file eseguibile.

La funzione ***proc_sys_exec()***, dopo aver verificato che si tratti

effettivamente di un file eseguibile valido e che ci siano i permessi per metterlo in funzione, procede all'allocazione della memoria, quindi legge il file e copia opportunamente le componenti di questo nelle aree di memoria allocate.

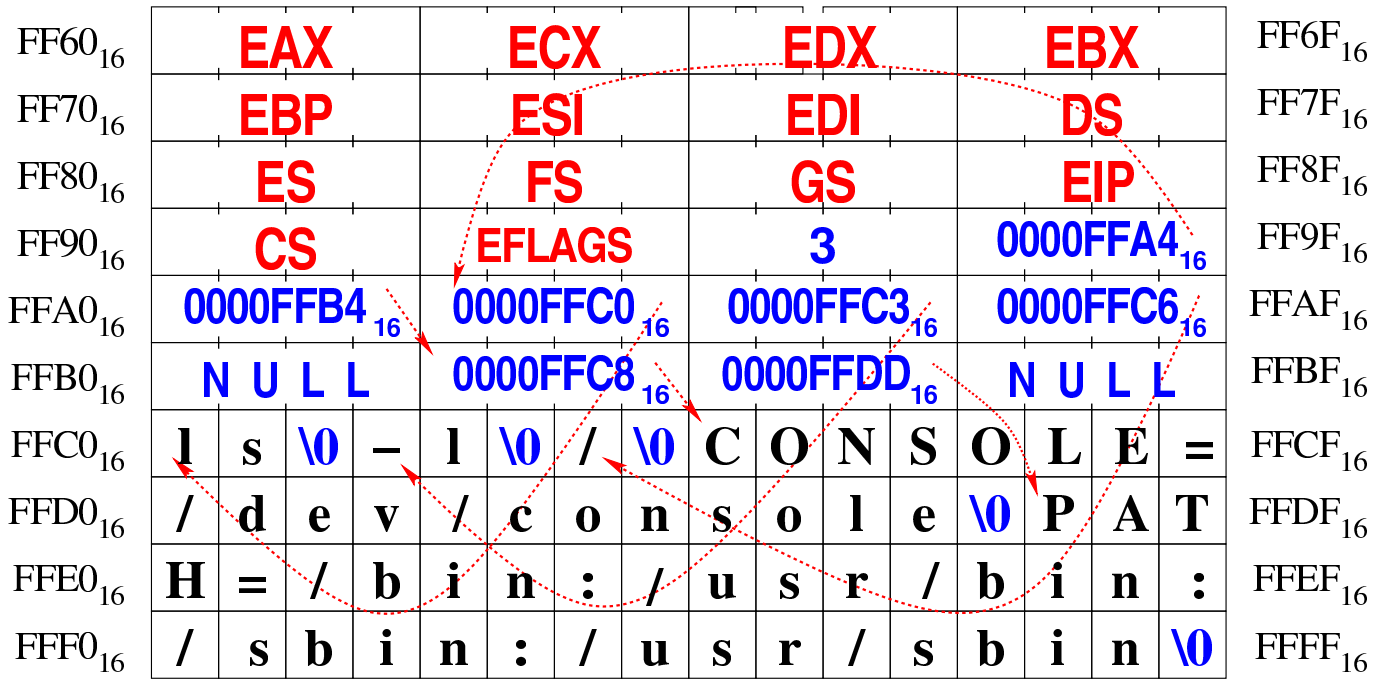
Terminato il caricamento del file, viene ricostruita in memoria la pila dei dati del nuovo processo. Prima si mettono sul fondo le stringhe delle variabili di ambiente e quelle degli argomenti della chiamata, quindi si aggiungono i puntatori alle stringhe delle variabili di ambiente, ricostruendo così l'array noto convenzionalmente come '**`envp[]`**', continuando con l'aggiunta dei puntatori alle stringhe degli argomenti della chiamata, per riprodurre l'array '**`argv[]`**'. Per ricostruire gli argomenti della chiamata della funzione ***main()*** dell'applicazione, vanno però aggiunti ancora: il puntatore all'inizio dell'array delle stringhe che descrivono le variabili di ambiente, il puntatore all'array delle stringhe che descrivono gli argomenti della chiamata e il valore che rappresenta la quantità di argomenti della chiamata.

Figura 84.69. Caricamento degli argomenti della chiamata della funzione ***main()***.



Fatto ciò, vanno aggiunti tutti i valori necessari allo scambio dei processi, costituiti dai vari registri da rimpiazzare.

Figura 84.70. Completamento della pila con i valori dei registri.



Superato il problema della ricostruzione della pila dei dati, la funzione ***proc_sys_exec()*** predispone i descrittori di standard input, standard output e standard error, quindi libera la memoria usata dal processo chiamante e ne rimpiazza i dati nella tabella dei processi con quelli del nuovo processo caricato.

Questa funzione viene usata soltanto da *sysroutine(9)* [93.20.28], in occasione del ricevimento di una chiamata di sistema di tipo '**SYS_EXEC**'.

FILE SORGENTI

'lib/unistd/execve.c' [95.30.14]

'lib/sys/os32/sys.s' [95.21.7]

'kernel/proc.h' [94.14]

'kernel/ibm_i386/isr.s' [94.6.21]

'kernel/proc/sysroutine.c' [94.14.28]

'kernel/proc/proc_sys_exec.c' [94.14.22]

VEDERE ANCHE

execve(2) [87.14], *sys(2)* [87.56], *sysroutine(9)* [93.20.28], *proc_scheduler(9)* [93.20.10], *path_inode(9)* [93.6.41], *inode_check(9)* [93.6.8], *inode_put(9)* [93.6.20], *inode_file_read(9)* [93.6.10], *dev_io(9)* [93.4.1].

93.20.22 os32: proc_timer_init(9)

«

NOME

'**proc_timer_init**' - inizializzazione del generatore di impulsi (temporizzatore)

SINTASSI

```
<kernel/proc.h>
void proc_timer_init (clock_t freq);
```

ARGOMENTI

Argomento	Descrizione
`clock_t` ***freq***	Frequenza in Hz che si intende produrre. Tuttavia, tutto il sistema è organizzato per gestire una frequenza di 100 Hz, il quale diventa praticamente obbligatorio.

DESCRIZIONE

La funzione ***proc_timer_init()*** viene chiamata esclusivamente dalla funzione *proc_init(9)* [93.20.3], per inizializzare il generatore di impulsi alla frequenza di ***CLOCKS_PER_SEC*** Hz, pari a 100 Hz.

FILE SORGENTI

‘`kernel/proc.h`’ [94.14]

‘`kernel/proc/proc_timer_init.c`’ [94.14.23]

VEDERE ANCHE

kmain(9) [93.13], *proc_init(9)* [93.20.3].

93.20.23 os32: proc_wakeup(9)

«

NOME

‘**`proc_wakeup_pipe_read`**’, ‘**`proc_wakeup_pipe_write`**’, ‘**`proc_wakeup_terminal`**’ - risveglio dei processi in attesa di un condotto o di un terminale.

SINTASSI

```
<kernel/proc.h>
void proc_wakeup_pipe_read (inode_t *inode);
void proc_wakeup_pipe_write (inode_t *inode);
void proc_wakeup_terminal (void);
```

ARGOMENTI

Argomento	Descrizione
`inode_t *inode`	Riferimento a una voce della tabella degli inode, con cui si identifica il condotto per il quale si attende di poter leggere o scrivere.

DESCRIZIONE

Le funzioni ***proc_wakeup_...()*** hanno lo scopo di scandire la tabella dei processi, alla ricerca di quelli da risvegliare, a seguito di un evento che richiede o può richiedere la loro attenzione. Le funzioni ***proc_wakeup_pipe_...()*** risvegliano esclusivamente i processi che sono in attesa di accedere a un condotto identificato attraverso un inode, mentre ***proc_wakeup_terminal()*** va a risvegliare tutti i processi in attesa del terminale, anche quelli che probabilmente non sono interessati direttamente da un input disponibile da tastiera.

FILE SORGENTI

‘`kernel/proc.h`’ [94.14]

‘`kernel/proc/proc_wakeup_pipe_read.c`’ [94.14.24]

‘`kernel/proc/proc_wakeup_pipe_write.c`’ [94.14.25]

‘`kernel/proc/proc_wakeup_terminal.c`’ [94.14.26]

VEDERE ANCHE

proc_sch_terminals(9) [93.20.8].

93.20.24 os32: proc_wakeup_pipe_read(9)

Vedere *proc_wakeup(9)* [93.20.23].

93.20.25 os32: proc_wakeup_pipe_write(9)

Vedere *proc_wakeup(9)* [93.20.23].

93.20.26 os32: proc_wakeup_terminal(9)

Vedere *proc_wakeup(9)* [93.20.23].

93.20.27 os32: ptr(9)

NOME

'**ptr**' - conversione di un puntatore relativo all'area dati di un processo in un puntatore valido per il kernel

SINTASSI

```
<kernel/proc.h>
void *ptr (pid_t pid, void *p);
```

DESCRIZIONE

La funzione ***ptr()*** ha il compito di convertire un puntatore proveniente dall'area dati di un processo in un puntatore valido dal punto di vista del kernel. In pratica, il puntatore corrispondente al parametro ***p*** va inteso come riferito a un certo processo ***pid***; la funzione restituisce un puntatore equivalente, ma valido per il kernel.

FILE SORGENTI

'kernel/proc.h' [94.14]

'kernel/proc/ptr.c' [94.14.27]

VEDERE ANCHE

sysroutine(9) [93.20.28].

93.20.28 os32: sysroutine(9)

«

NOME

'**s_sysroutine**' - attuazione delle chiamate di sistema

SINTASSI

```
<kernel/proc.h>
extern pid_t proc_current;
void sysroutine (uint32_t syscallnr, uint32_t msg_off,
                 uint32_t msg_size);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***proc_current***	Il numero del processo interrotto.
`uint32_t` ***syscallnr***	Il numero della chiamata di sistema.
`uint32_t` ***msg_off***	Nonostante il tipo di variabile, si tratta del **puntatore** alla posizione di memoria in cui inizia il messaggio con gli argomenti della chiamata di sistema, ma tale puntatore è valido solo nell'ambito del segmento dati del processo interrotto.
`uint32_t` ***msg_size***	La lunghezza del messaggio della chiamata di sistema.

DESCRIZIONE

La funzione ***sysroutine()*** viene chiamata esclusivamente dalla routine '**`isr_128`**', contenuta nel file '`kernel/ibm_i386/isr.s`' [94.6.21], a seguito di una chiamata di sistema.

Inizialmente, la funzione individua l'indirizzo corrispondente alla posizione del messaggio proveniente dal processo, dal punto di vista del kernel; in pratica traduce ***msg_off*** in qualcosa di adatto al kernel.

Attraverso un'unione di variabili strutturate, tutti i tipi di messaggi gestibili per le chiamate di sistema vengono dichiarati assieme in un'unica area di memoria. Successivamente, la funzione deve trasferire il messaggio, dall'indirizzo calcolato precedentemente all'inizio dell'unione in questione.

Quando la funzione è in grado di accedere ai dati del messaggio, procede con una grande struttura di selezione, sulla base del tipo di messaggio, quindi esegue ciò che è richiesto, avvalendosi prevalentemente di altre funzioni, interpretando il messaggio in modo diverso a seconda del tipo di chiamata.

Il messaggio originale viene poi sovrascritto con le informazioni prodotte dall'azione richiesta, in particolare viene trasferito anche il valore della variabile ***errno*** del kernel, in modo che possa essere recepita anche dal processo che ha eseguito la chiamata, in caso di esito erroneo. Pertanto, il messaggio viene riscritto a partire dall'indirizzo da cui era stato copiato precedentemente, in modo da renderlo disponibile effettivamente al processo chiamante.

Quando la funzione ***sysroutine()*** ha finito il suo lavoro, chiama a sua volta *proc_scheduler(9)* [93.20.10], perché con l'occasione

provveda eventualmente alla sostituzione del processo attivo con un altro che si trovi nello stato di pronto.

VALORE RESTITUITO

La funzione non restituisce alcun valore, in quanto tutto ciò che c'è da restituire viene trasmesso con la riscrittura del messaggio, nell'area di memoria originale.

FILE SORGENTI

'`lib/sys/os32/sys.s`' [95.21.7]

'`kernel/proc.h`' [94.14]

'`kernel/ibm_i386/isr.s`' [94.6.21]

'`kernel/proc/sysroutine.c`' [94.14.28]

VEDERE ANCHE

sys(2) [87.56], *proc_scheduler(9)* [93.20.10], *dev_io(9)* [93.4.1], *lib_s(9)* [93.12].

93.21 os32: route(9)

«

Il file '`kernel/net/route.h`' [94.12.33] descrive le funzioni per la gestione degli instradamenti IPv4 secondo os32.

Per la descrizione sulla gestione degli instradamenti IPv4 secondo il sistema os32, si rimanda alla sezione 84.10.3. La tabella successiva che sintetizza l'uso delle funzioni di questo gruppo, è tratta da lì.

Tabella 84.128. Funzioni per la gestione della tabella degli instradamenti.

Funzione	Descrizione
`void route_init (void);`	Inizializza la tabella ***route_table[]***, predisponendo la voce ***route_table[0]*** per l'interfaccia locale *loopback* [94.12.34].
`void route_sort (void);`	Riordina la tabella degli instradamenti [94.12.39].
`h_addr_t route_remote_to_local (h_addr_t` ***remote***`);`	Restituisce l'indirizzo IPv4 locale, più adatto per intrattenere una connessione con l'indirizzo remoto fornito come argomento. Questo tipo di analisi viene determinato partendo dalla tabella degli instradamenti, per determinare l'indirizzo IPv4 locale dell'interfaccia interessata dal collegamento [94.12.37].

Funzione	Descrizione
`h_addr_t route_remote_to_router (h_addr_t` ***remote*** `);`	Restituisce l'indirizzo IPv4 del router da utilizzare per raggiungere l'indirizzo IPv4 remoto specificato. Se l'indirizzo restituito è pari a −1 significa che non è stata ottenuta alcuna voce corrispondente, mentre se si ottiene zero significa che non c'è bisogno di router per raggiungere la destinazione [94.12.38].

93.22 os32: screen(9)

«

Il file '`kernel/driver/screen.h`' [94.4.30] descrive le funzioni per la gestione dello schermo, in relazione alla gestione complessiva dei terminali virtuali.

Per la descrizione dell'organizzazione della gestione dello schermo di os32, si rimanda alla sezione 84.7.5.2. La tabella successiva che sintetizza l'uso delle funzioni di questo gruppo, è tratta da lì.

Tabella 84.89. Funzioni per la gestione dello schermo, dichiarate nel file di intestazione '`kernel/driver/screen.h`' e descritte nei file contenuti nella directory '`kernel/driver/screen/`'.

Funzione	Descrizione
`int screen_clear (screen_t *screen);`	Ripulisce il contenuto dello schermo selezionato, riposizionando il cursore all'inizio.
`screen_t *screen_current (void);`	Restituisce il puntatore alla voce della tabella ***screen_table[]*** che descrive lo schermo virtuale attivo.
`void screen_init (void);`	Inizializza la gestione degli schermi virtuali, ripulendoli e collocando il cursore all'inizio. Questa funzione viene usata da ***tty_init()***.
`int screen_newline (screen_t *screen);`	Produce sullo schermo virtuale selezionato un avanzamento alla riga successiva. Ciò può comportare semplicemente il riposizionamento del cursore, oppure lo scorrimento in avanti del contenuto, quando il cursore si trova già sull'ultima riga visualizzabile.

Funzione	Descrizione
`int screen_number (screen_t *`***screen***`);`	Restituisce il numero dello schermo corrispondente al puntatore fornito, purché questo sia valido. È in pratica l'opposto della funzione ***screen_pointer()***.
`screen_t *screen_pointer (int` ***scrn***`);`	Restituisce il puntatore alla voce della tabella ***screen_table[]*** che descrive lo schermo virtuale indicato per numero. È in pratica l'opposto della funzione ***screen_number()***.
`int screen_putc (screen_t *`***screen***`,` `int` ***c***`);`	Colloca sullo schermo virtuale individuato dal puntatore che costituisce il primo parametro, il carattere richiesto come secondo. Se il carattere in questione è *<CR>* o *<LF>*, si produce un avanzamento alla riga successiva, mentre con un carattere *<BS>* si produce un arretramento del cursore.

Funzione	Descrizione
`uint16_t screen_cell (c, attributo);`	Si tratta di una macroistruzione che produce il valore corretto per una cella dello schermo VGA, contenente sia l'informazione sul carattere, sia quella dell'attributo associato.
`int screen_scroll (screen_t *screen);`	Fa scorrere in avanti lo schermo, di una riga, ricollocando di conseguenza il cursore.
`int screen_select (screen_t *screen);`	Seleziona lo schermo indicato come schermo attivo, facendone apparire il contenuto sullo schermo VGA reale.
`void screen_update (screen_t *screen);`	Aggiorna la memoria VGA sulla base della copia che rappresenta lo schermo virtuale attivo. L'aggiornamento implica anche la collocazione del cursore visibile in corrispondenza della posizione attuale.

93.23 os32: tcp(9)

I file '`kernel/net/tcp.h`' [94.12.40] e '`kernel/net/udp.h`' [94.12.53] descrivono le funzioni per la gestione dei protocolli TCP e UDP, secondo os32.

Per la descrizione sulla gestione dei protocolli TCP e UDP da parte di os32, si rimanda alla sezione 84.12. La tabella successiva che sintetizza l'uso delle funzioni di questo gruppo, è tratta da lì.

Tabella 84.132. Funzioni per la gestione dei protocolli TCP e UDP.

Funzione	Descrizione

`int tcp_tx_raw (h_port_t` ***sport***`,`
`h_port_t` ***dport***`,`
`uint32_t` ***seq***`,`
`uint32_t` ***ack_seq***`,`
`int` ***flags***`,`
`h_addr_t` ***saddr***`,`
`h_addr_t` ***daddr***`,`
`const void *`***buffer***`,`
`size_t` ***size***`);`
	Costruisce un pacchetto TCP, utilizzando i dati forniti come argomenti della chiamata; quindi lo trasmette attraverso la funzione ***ip_tx()*** che a sua volta provvede a imbustarlo in un pacchetto IP prima della trasmissione effettiva. Si tratta comunque di una funzione usata soltanto per fare dei test di funzionamento [94.12.50].

`void`
`tcp_show`
`(h_addr_t` ***src***`,`
`h_addr_t` ***dst***`,`
`const struct tcphdr *`***tcphdr***`);`
	Funzione diagnostica realizzata per visualizzare alcune informazioni su un pacchetto TCP, di cui si conosce il contenuto e gli indirizzi IPv4. Questa funzione viene usata prevalentemente da ***tcp()***, quando si attiva la macro-variabile ***DEBUG*** [94.12.46].

Funzione	Descrizione
`int tcp_tx_rst (void *ip_packet);`	Sulla base di un pacchetto IP ricevuto con un contenuto TCP, trasmette un pacchetto TCP di azzeramento (RST) [94.12.51].
`int tcp_tx_sock (void *sock_item);`	Trasmette quanto contenuto nella coda del socket indicato come argomento, ammesso che il socket sia nella condizione di poter trasmettere [94.12.52].
`int tcp_tx_ack (void *sock_item);`	Trasmette un pacchetto TCP vuoto contenente la conferma di quanto ricevuto in precedenza, sulla base dello stato attuale del socket indicato come argomento [94.12.49].
`int tcp_rx_ack (void *sock_item,` `void *packet);`	Verifica che il pacchetto indicato come secondo parametro, contenga una conferma valida per il socket specificato come primo parametro [94.12.44].

Funzione	Descrizione
`int tcp_rx_data (void *sock_item, void *packet);`	Legge il contenuto di un pacchetto TCP e lo copia all'interno della memoria tampone del socket rappresentata da ***sock_table[s].tcp.recv_data*** (dove ***s*** è l'indice del socket considerato) [94.12.45].
`int tcp_connect (void *sock_item);`	Fa in modo di mettere il socket in connessione, ammesso che ciò sia possibile. Questa funzione serve a ***s_connect()*** [94.12.43].
`int tcp_close (void *sock_item);`	Fa in modo di mettere il socket nello stato di chiusura, ammesso che ciò sia possibile. Questa funzione serve a ***s_close()*** [94.12.42].
`int tcp (void);`	Viene chiamata da ***proc_sch_net()*** e si occupa di gestire lo stato di tutte le connessioni TCP in essere in quel momento [94.12.41].

Funzione	Descrizione

`int udp_tx (h_port_t` ***sport***`,`
`h_port_t` ***dport***`,`
`h_addr_t` ***saddr***`,`
`h_addr_t` ***daddr***`,`
`const void *`***buffer***`,`
`size_t` ***size***`);`
	Assembla un pacchetto UDP e lo trasmette (dopo aver costruito a sua volta il pacchetto IPv4 complessivo) [94.12.54].

93.24 os32: tty(9)

Il file ‘`kernel/driver/tty.h`’ [94.4.42] descrive le funzioni per la gestione dei terminali virtuali.

Per la descrizione dell’organizzazione della gestione dei terminali virtuali di os32, si rimanda alla sezione 84.7.5. La tabella successiva che sintetizza l’uso delle funzioni di questo gruppo, è tratta da lì.

Tabella 84.85. Funzioni per l’accesso al terminale, dichiarate nel file di intestazione ‘`kernel/driver/tty.h`’ e descritte nei file contenuti nella directory ‘`kernel/driver/tty/`’.

Funzione	Descrizione
`dev_t tty_console (dev_t` ***device*** `);`	Seleziona un terminale virtuale, rendendolo attivo, specificandone il numero del dispositivo. La funzione restituisce il dispositivo attivo in precedenza e se le viene fornito solo il valore zero, il terminale virtuale non cambia, ma si ottiene comunque di conoscere qual è quello attuale.
`void tty_init (void);`	Inizializza la gestione dei terminali virtuali, popolando anche la tabella ***tty_table[]*** con i valori predefiniti. Questa funzione viene usata una volta sola all'interno di ***kmain()***.
`int tty_read (dev_t` ***device*** `);`	Legge un carattere dal terminale virtuale specificato attraverso il numero di dispositivo. La lettura avviene solo se l'input da tastiera risulta concluso, altrimenti la funzione restituisce il valore −1.

Funzione	Descrizione
`tty_t *tty_reference (dev_t` ***device*** `);`	Restituisce il puntatore alla voce della tabella ***tty_table[]*** contenente le informazioni sul terminale virtuale indicato attraverso il numero di dispositivo. Se il dispositivo indicato non è valido, si ottiene il puntatore nullo; se viene richiesto il dispositivo indefinito, si ottiene il puntatore all'inizio della tabella.
`void tty_write (dev_t` ***device*** `, int` ***c*** `);`	Scrive un carattere sullo schermo del terminale specificato.