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].
`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.
`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.
`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.

Funzione	Descrizione
`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`'.

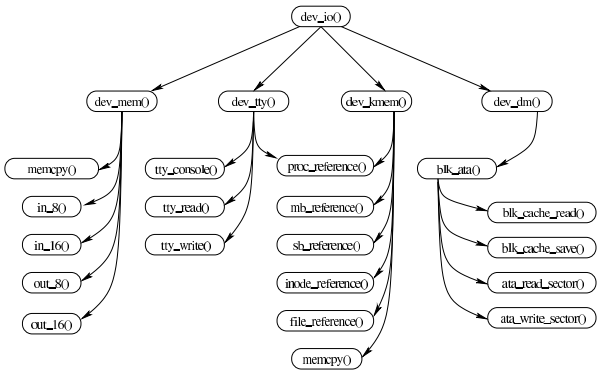
Tabella 84.82. Classificazione dei dispositivi di os32.

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.
`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.
`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.
`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.
`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)

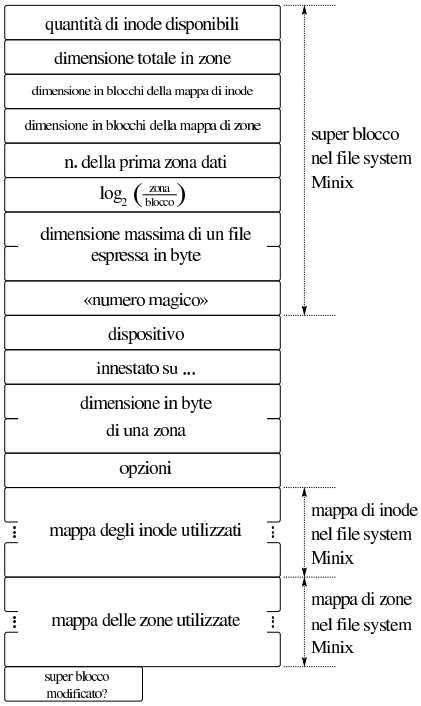
Il file ‘`kernel/dm.h`’ [94.4] descrive ciò che serve per la gestione delle unità di memorizzazione, in generale. È disponibile soltanto la funzione ***dm_init()*** per la predisposizione iniziale della tabella ***dm_table[]***.

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.

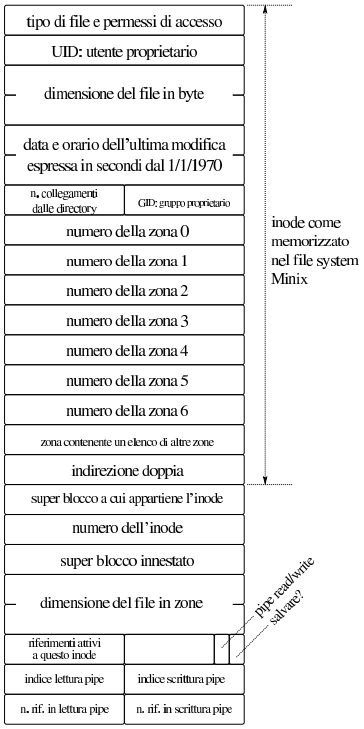
Figura 84.95. Struttura del tipo ‘**`sb_t`**’, corrispondente agli elementi dell’array ***sb_table[]***.



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[]***.



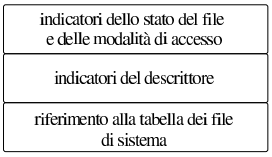
Listato 84.101. Struttura del tipo '**inode_t**', corrispondente agli elementi dell'array ***inode_table[]***.

```
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.

Funzione o macroistruzione	Descrizione
`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.
`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.
`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.

Funzione	Descrizione
`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].
`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.

`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].

Funzione	Descrizione
`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].
`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].
`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].

Funzione	Descrizione
`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].
`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].
`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 processo 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 di-

sabilitato. 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

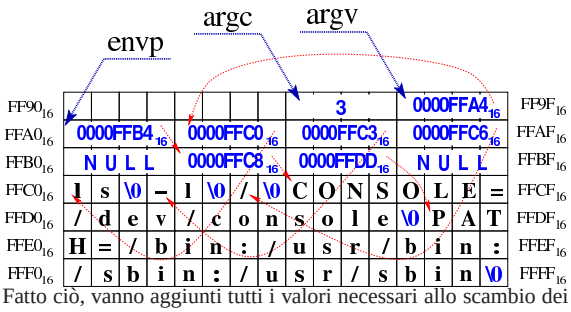
La funzione ***proc_sys_exec()*** serve a mettere in pratica la chiamata di sistema *execve(2)* [87.14], destinata a rimpiazzare il processo in corso con un nuovo processo, caricato da un file eseguibile.

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

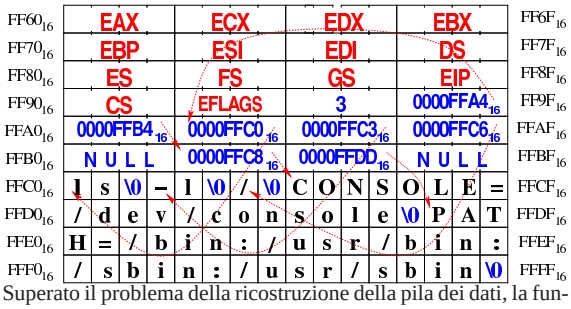
variabili di ambiente, ricostruendo così l'array noto convenzionalmente come '**`envp[]`**', continuando con l'aggiunta dei puntatori alle stringhe degli argomenti della chiamata, per riprodurre l'array '**`argv[]`**'. Per ricostruire gli argomenti della chiamata della funzione ***main()*** dell'applicazione, 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].
`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.
`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()***.

Funzione	Descrizione
`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.
`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].

Funzione	Descrizione
`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].
`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].
`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].
`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.
`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.