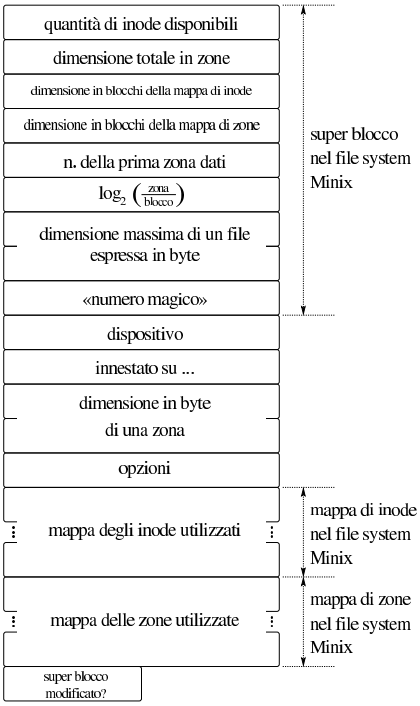
La gestione del file system è suddivisa in diversi file contenuti nella directory '`kernel/fs/`', facenti capo al file di intestazione '`kernel/fs.h`'.

File «kernel/fs/sb_...»

I file '`kernel/fs/sb_...`' descrivono le funzioni per la gestione dei super blocchi, distinguibili perché iniziano tutte con il prefisso '**`sb_`**'. Tra questi file si dichiara l'array ***sb_table[]***, il quale rappresenta una tabella le cui righe sono rappresentate da elementi di tipo '**`sb_t`**' (il tipo '**`sb_t`**' è definito nel file '`kernel/fs.h`'). Per uniformare l'accesso alla tabella, la funzione ***sb_reference()*** permette di ottenere il puntatore a un elemento dell'array ***sb_table[]***, specificando il numero del dispositivo cercato.

Figura u148.1. Struttura del tipo '**sb_t**', corrispondente agli elementi dell'array ***sb_table[]***.



Listato u148.2. 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;
};
```

Il super blocco rappresentato dal tipo '**sb_t**' include anche le mappe delle zone e degli inode impegnati. Queste mappe hanno una dimensione fissa in memoria, mentre nel file system reale possono essere di dimensione minore. La tabella di super blocchi, contiene le informazioni dei dispositivi di memorizzazione innestati nel sistema. L'innesto si concretizza nel riferimento a un inode, contenuto nella tabella degli inode (descritta in un altro capitolo), il quale rappresenta la directory di un'altra unità, su cui tale innesto è avvenuto. Naturalmente, l'innesto del file system principale rappresenta un caso particolare.

Tabella u148.3. Funzioni per la gestione dei dispositivi di memorizzazione di massa, a livello di super blocco, definite nei file '`kernel/fs/sb_...`'.

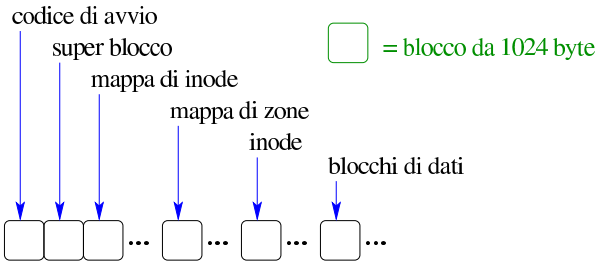
Funzione	Descrizione
`sb_t *sb_reference (dev_t` ***device***`);`	Restituisce il riferimento a un elemento della tabella dei super blocchi, in base al numero del dispositivo di memorizzazione. Se il dispositivo cercato non risulta già innestato, si ottiene il puntatore nullo; se si chiede il dispositivo zero, si ottiene il puntatore al primo elemento della tabella.

`sb_t *sb_mount (dev_t` ***device***`,`
`inode_t **`***inode_mnt***`,`
`int` ***options***`);`
	Innesta il dispositivo rappresentato numericamente dal primo parametro, sulla directory corrispondente all'inode a cui punta il secondo parametro, con le opzioni del terzo parametro. Quando si tratta del primo innesto del file system principale, la directory è quella dello stesso file system, pertanto, in tal caso, ****inode_mnt*** è inizialmente un puntatore nullo e deve essere modificato dalla funzione stessa.

`int sb_save (sb_t *`***sb***`);`	Salva il super blocco nella sua unità di memorizzazione, se questo risulta modificato. In questo caso, il super blocco include anche le mappe degli inode e delle zone.
`int sb_zone_status (sb_t *`***sb***`,` `zno_t` ***zone***`);`	Restituisce uno se la zona rappresentata dal secondo parametro è impegnata nel super blocco a cui si riferisce il primo parametro; diversamente restituisce zero.
`int sb_inode_status (sb_t *`***sb***`,` `ino_t` ***ino***`);`	Restituisce uno se l'inode rappresentato dal secondo parametro è impegnato nel super blocco a cui si riferisce il primo parametro; diversamente restituisce zero.

File «kernel/fs/zone_...»

Nel file system Minix 1, si distinguono i concetti di blocco e zona di dati, con il vincolo che la zona ha una dimensione multipla del blocco. Il contenuto del file system, dopo tutte le informazioni amministrative, è organizzato in zone; in altri termini, i blocchi di dati si raggiungono in qualità di zone.



La zona rimane comunque un tipo di blocco, potenzialmente più grande (ma sempre multiplo) del blocco vero e proprio, che si numera a partire dall'inizio dello spazio disponibile, con la differenza che

è utile solo per raggiungere i blocchi di dati. Nel super blocco del file system si trova l'informazione del numero della prima zona che contiene dati, in modo da non dover ricalcolare questa informazione ogni volta.

I file '`kernel/fs/zone_...`' descrivono le funzioni per la gestione del file system a zone.

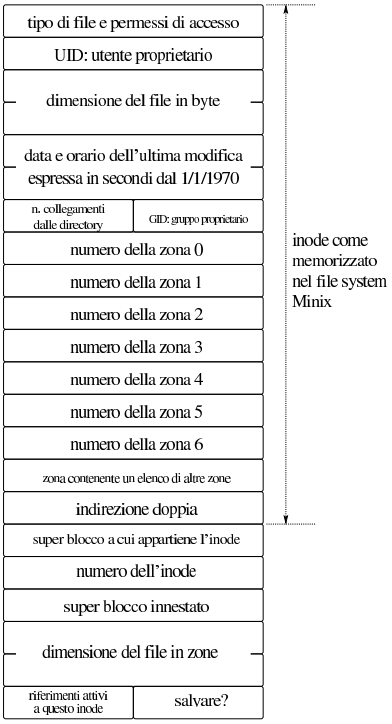
Tabella u148.5. Funzioni per la gestione delle zone, definite nei file '`kernel/fs/zone_...`'.

Funzione	Descrizione
`zno_t zone_alloc (sb_t *sb);`	Alloca una zona, restituendo il numero della stessa. In pratica, cerca la prima zona libera nel file system a cui si riferisce il super blocco ***sb** e la segna come impegnata, restituendone il numero.
`int zone_free (sb_t *sb, zno_t zone);`	Libera una zona, impegnata precedentemente.
`int zone_read (sb_t *sb, zno_t zone, void *buffer);`	Legge il contenuto di una zona, memorizzandolo a partire dalla posizione di memoria rappresentato da ***buffer***.
`int zone_write (sb_t *sb, zno_t zone, void *buffer);`	Sovrascrive una zona, utilizzando il contenuto della memoria a partire dalla posizione rappresentata da ***buffer***.

File «kernel/fs/inode_...»

I file '`kernel/fs/inode_...`' descrivono le funzioni per la gestione dei file, in forma di inode. In uno di questi file viene dichiarata la tabella degli inode in uso nel sistema, rappresentata dall'array ***inode_table[]*** e per individuare un certo elemento dell'array si usa preferibilmente la funzione ***inode_reference()***. Gli elementi della tabella degli inode sono di tipo '**`inode_t`**' (definito nel file '`kernel/fs.h`'); una voce della tabella rappresenta un inode utilizzato se il campo dei riferimenti (***references***) ha un valore maggiore di zero.

Figura u148.6. Struttura del tipo '**`inode_t`**', corrispondente agli elementi dell'array ***inode_table[]***.



Listato u148.7. Struttura del tipo '**`inode_t`**', corrispondente agli elementi dell'array ***inode_table[]***.

```
<verbatimpre width="60">
<![CDATA[
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;
};
```

Figura u148.8. Collegamento tra la tabella degli inode e quella dei super blocchi.

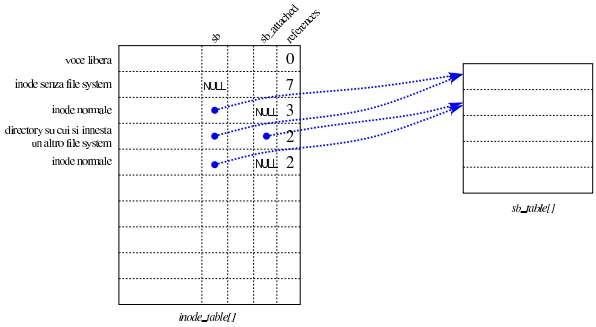


Tabella u148.9. Funzioni per la gestione dei file in forma di inode, definite nei file '`kernel/fs/inode_...`'.

Funzione	Descrizione
`inode_t *` `inode_reference (dev_t device, ino_t ino);`	Restituisce il puntatore a un inode, rappresentato in pratica da un elemento dell'array ***inode_table[]***, corrispondente a quello con il numero di dispositivo e di inode indicati come argomenti. Se entrambi gli argomenti sono a zero, si ottiene il puntatore al primo elemento; se entrambi i valori sono pari a `-1`, si ottiene il puntatore al primo elemento libero; se viene indicato il dispositivo zero e l'inode numero uno, si ottiene il puntatore all'elemento corrispondente alla directory radice del file system principale.

Funzione	Descrizione

`inode_t *`
`inode_alloc (dev_t` ***device***`,`
`mode_t` ***mode***`,`
`uid_t` ***uid***`);`
	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.

`inode_t *`
`inode_get (dev_t` ***device***`,`
`ino_t` ***ino***`);`
	Restituisce il puntatore all'inode rappresentato dal numero di dispositivo e di inode, indicati come argomenti. Se l'inode è già presente nella tabella degli inode, la cosa si risolve nell'incremento di una unità del numero dei riferimenti di tale inode; se invece l'inode non è ancora presente, questo viene caricato dal suo file system nella tabella e gli viene attribuito inizialmente un riferimento attivo.

`int inode_put (inode_t *`***inode***`);`	Rilascia un inode che non serve più. Ciò comporta la riduzione del contatore dei riferimenti nella tabella degli inode, tenendo conto che se tale valore raggiunge lo zero, si provvede anche al suo salvataggio nel file system (ammesso che l'inode della tabella risulti modificato, rispetto alla versione presente nel file system). La funzione restituisce zero in caso di successo, oppure `-1` in caso contrario.
`int inode_save (inode_t *`***inode***`);`	Salva l'inode nel file system, se questo risulta modificato.

`blkcnt_t`
`inode_fzones_read (inode_t *`***inode***`,`
`zno_t` ***zone_start***`,`
`void *`***buffer***`,`
`blkcnt_t` ***blkcnt***`);`
	Legge da un file, identificato attraverso il puntatore all'inode (della tabella di inode), una certa quantità di zone, a partire da una certa zona relativa al file, mettendo il risultato della lettura a partire dalla posizione di memoria rappresentata da un puntatore generico. La funzione restituisce la quantità di zone lette con successo.

Funzione	Descrizione

`ssize_t`
`inode_file_read (inode_t *`***inode***`,`
`off_t` ***offset***`,`
`void *`***buffer***`,`
`size_t` ***count***`,`
`int *`***eof***`);`
	Legge il contenuto di un file, individuato da un inode già caricato nella tabella relativa, aggiornando eventualmente una variabile contenente l'indicatore di fine file. La funzione restituisce la quantità di byte letti con successo, oppure il valore `-1` in caso di problemi.

`ssize_t`
`inode_file_write (inode_t *`***inode***`,`
`off_t` ***offset***`,`
`void *`***buffer***`,`
`size_t` ***count***`);`
	Scrive una certa quantità di byte nel file individuato da un inode già caricato nella tabella relativa. La funzione restituisce la quantità di byte scritti effettivamente, oppure il valore `-1` in caso di problemi.

`zno_t inode_zone (inode_t *`***inode***`,`
`zno_t` ***fzone***`,`
`int` ***write***`);`
	Restituisce il numero di zona effettivo, corrispondente a un numero di zona relativo a un certo file di un certo inode. Se il parametro ***write*** è pari a zero, si intende che la zona deve esistere, quindi se questa non c'è, si ottiene semplicemente un valore pari a zero; se invece l'ultimo parametro è pari a uno, nel caso la zona cercata fosse attualmente mancante, verrebbe creata al volo nel file system.

`int` `inode_truncate (inode_t *`***inode***`);`	Riduce la dimensione del file a cui si riferisce l'inode a zero. In pratica fa sì che le zone allocate del file siano liberate. La funzione restituisce zero se l'operazione si conclude con successo, oppure `-1` in caso di problemi.

`int`
`inode_check (inode_t *`***inode***`,`
`mode_t` ***type***`,`
`int` ***perm***`,`
`uid_t` ***uid***`);`
	Verifica che l'inode sia di un certo tipo e abbia i permessi di accesso necessari a un certo utente. Nel parametro ***type*** si possono indicare più tipi validi. La funzione restituisce zero in caso di successo, ovvero di compatibilità, mentre restituisce `-1` se il tipo o i permessi non sono adatti.

`int` `inode_dir_empty (inode_t *`***inode***`);`	Verifica se la directory a cui si riferisce l'inode è effettivamente una directory ed è vuota, nel qual caso restituisce il valore uno, altrimenti restituisce zero.

Fasi dell'innesto di un file system

L'innesto e il distacco di un file system, coinvolge simultaneamente la tabella dei super blocchi e quella degli inode. Si distinguono due situazioni fondamentali: l'innesto del file system principale e quello di un file system ulteriore.

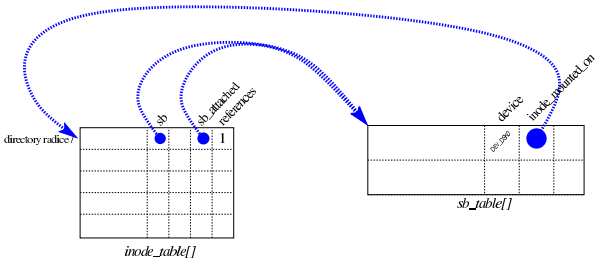
Quando si tratta dell'innesto del file system principale, la tabella dei super blocchi è priva di voci e quella degli inode non contiene

riferimenti a file system. La funzione ***sb_mount()*** viene chiamata indicando, come riferimento all'inode di innesto, il puntatore a una variabile puntatore contenente il valore nullo:

```
…
    inode_t *inode;
    sb_t    *sb;
    …
    inode = NULL;
    sb = sb_mount (DEV_DSK0, &inode, MOUNT_DEFAULT);
    …
```

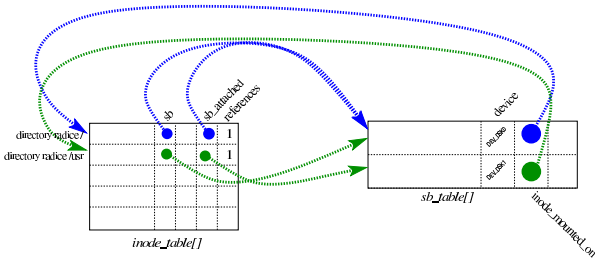
La funzione ***sb_mount()*** carica il super blocco nella tabella relativa, ma trovando il riferimento all'inode di innesto nullo, provvede a caricare l'inode della directory radice dello stesso dispositivo, creando un collegamento incrociato tra le tabelle dei super blocchi e degli inode, come si vede nella figura successiva.

Figura u148.11. Collegamento tra la tabella degli inode e quella dei super blocchi, quando si innesta il file system principale.



Per innestare un altro file system, occorre prima disporre dell'inode di una directory (appropriata) nella tabella degli inode, quindi si può caricare il super blocco del nuovo file system, creando il collegamento tra directory e file system innestato.

Figura u148.12. Innesto di un file system nella directory '`/usr/`'.

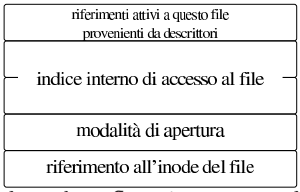


File «kernel/fs/file_...»

I file '`kernel/fs/file_…`' descrivono le funzioni per la gestione della tabella dei file, la quale si collega a sua volta a quella degli inode. In realtà, le funzioni di questo gruppo sono in numero molto limitato, perché l'intervento nella tabella dei file avviene prevalentemente per opera di funzioni che gestiscono i descrittori.

La tabella dei file è rappresentata dall'array ***file_table[]*** e per individuare un certo elemento dell'array si usa preferibilmente la funzione ***file_reference()***. Gli elementi della tabella dei file sono di tipo '**`file_t`**' (definito nel file '`kernel/fs.h`'); una voce della tabella rappresenta un file aperto se il campo dei riferimenti (***references***) ha un valore maggiore di zero.

Figura u148.13. 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;
};
```

Nel membro ***oflags*** si annotano esclusivamente opzioni relative alla

modalità di apertura del file: lettura, scrittura o entrambe; pertanto si possono usare le macro-variabili ***O_RDONLY***, ***O_WRONLY*** e ***O_RDWR***, come dichiarato nel file di intestazione '`lib/fcntl.h`'. Il membro ***offset*** rappresenta l'indice interno di accesso al file, per l'operazione successiva di lettura o scrittura al suo interno. Il membro ***references*** è un contatore dei riferimenti a questa tabella, da parte di descrittori di file.

La tabella dei file si collega a quella degli inode, attraverso il membro ***inode***. Più voci della tabella dei file possono riferirsi allo stesso inode, perché hanno modalità di accesso differenti, oppure soltanto per poter distinguere l'indice interno di lettura e scrittura. Va osservato che le voci della tabella di inode potrebbero essere usate direttamente e non avere elementi corrispondenti nella tabella dei file.

Figura u148.14. Collegamento tra la tabella dei file e quella degli inode.

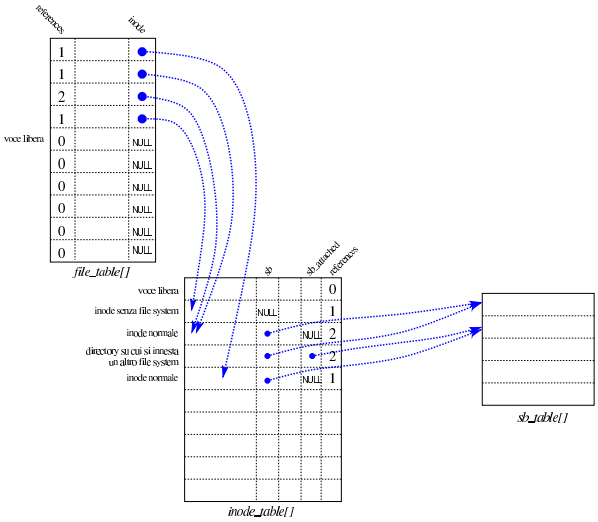


Tabella u148.15. Funzioni fatte esclusivamente per la gestione della tabella dei file ***file_table[]***.

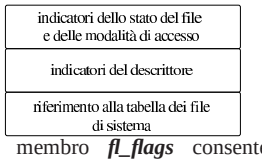
Funzione	Descrizione
`file_t *` `file_reference (int` ***`fno`***`);`	Restituisce il puntatore all'elemento ***fno***-esimo della tabella dei file. Se ***fno*** è un valore negativo, viene restituito il puntatore a una voce libera della tabella.

`file_t *`
`file_stdio_dev_make (dev_t` ***`device`***`,`
`mode_t` ***`mode`***`,`
`int` ***`oflags`***`);`
	Crea una voce per l'accesso a un file di dispositivo standard di input-output, restituendo il puntatore alla voce stessa.

Descrittori di file

Le tabelle di super blocchi, inode e file, riguardano il sistema nel complesso. Tuttavia, l'accesso normale ai file avviene attraverso il concetto di «descrittore», il quale è un file aperto da un certo processo elaborativo. Nel file '`kernel/fs.h`' si trova la dichiarazione e descrizione del tipo derivato '**`fd_t`**', usato per costruire una tabella di descrittori, ma tale tabella non fa parte della gestione del file system, bensì è incorporata nella tabella dei processi elaborativi. Pertanto, ogni processo ha una propria tabella di descrittori di file.

Figura u148.16. Struttura del tipo '**fd_t**', con cui si costituiscono gli elementi delle tabelle dei descrittori di file, una per ogni processo.

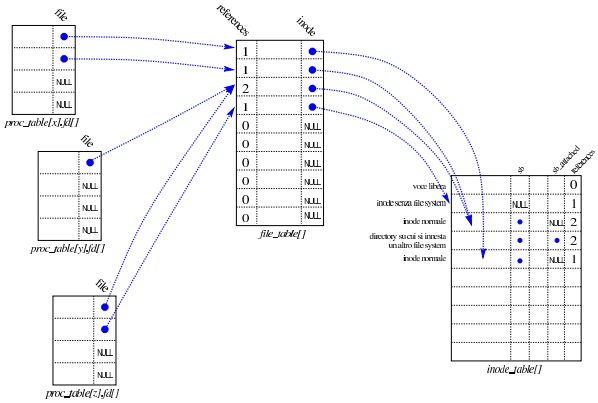



```
typedef struct fd fd_t;
struct fd {
    int     fl_flags;
    int     fd_flags;
    file_t *file;
};
```

Il membro ***fl_flags*** consente di annotare indicatori del tipo '**O_RDONLY**', '**O_WRONLY**', '**O_RDWR**', '**O_CREAT**', '**O_EXCL**', '**O_NOCTTY**', '**O_TRUNC**' e '**O_APPEND**', come dichiarato nella libreria standard, nel file di intestazione '`lib/fcntl.h`'. Tali indicatori si combinano assieme con l'operatore binario OR. Altri tipi di opzione che sarebbero previsti nel file '`lib/fcntl.h`', sono privi di effetto nella gestione del file system di os16.

Il membro ***fd_flags*** serve a contenere, eventualmente, l'opzione '**FD_CLOEXEC**', definita nel file '`lib/fcntl.h`'. Non sono previste altre opzioni di questo tipo.

Figura u148.17. Collegamento tra le tabelle dei descrittori e la tabella complessiva dei file. La tabella ***proc_table[x].fd[]*** rappresenta i descrittori di file del processo elaborativo ***x***.



File «kernel/fs/path_...»

«

I file '`kernel/fs/path_...`' descrivono le funzioni che fanno riferimento a file o directory attraverso una stringa che ne descrive il percorso.

Tabella u148.18. Funzioni per la gestione dei file, a cui si fa riferimento attraverso un percorso, senza indicazioni sul processo elaborativo.

Funzione	Descrizione
`int path_fix (char *path);`	Verifica il percorso indicato semplificandolo, quindi sovrascrive il percorso originario con quello riveduto e corretto. Un percorso assoluto rimane assoluto; un percorso relativo rimane relativo, mancando qualunque indicazione sulla directory corrente.
`int path_full (const char *path, const char *path_cwd, char *full_path);`	Ricostruisce un percorso assoluto, usando come riferimento la directory corrente indicata in ***path_cwd***, salvandolo in ***path_full***.

Tabella u148.19. Funzioni per la gestione dei file, a cui si fa riferimento attraverso un percorso, tenendo conto del processo elaborativo per conto del quale si svolge l'operazione. Del processo elaborativo si considera soprattutto l'identità efficace, per conoscerne i privilegi e determinare se è data effettivamente la facoltà di eseguire l'azione richiesta.

Funzione	Descrizione
`int path_chdir (pid_t pid, const char *path);`	Cambia la directory corrente, utilizzando il nuovo percorso indicato. È l'equivalente della funzione standard ***chdir()*** (sezione u0.3).
`int path_chmod (pid_t pid, const char *path, mode_t mode);`	Cambia la modalità di accesso al file indicato. È l'equivalente della funzione standard ***chmod()*** (sezione u0.4).
`int path_chown (pid_t pid, const char *path, uid_t uid, gid_t gid);`	Cambia l'utente e il gruppo proprietari del file (va però ricordato che os16 non considera i gruppi, anche se nel file system sono annotati). È l'equivalente della funzione standard ***chown()*** (sezione u0.5).
`dev_t path_device (pid_t pid, const char *path);`	Restituisce il numero del dispositivo di un file di dispositivo; pertanto, il percorso deve fare riferimento a un file di dispositivo, per poter ottenere un risultato valido.
`inode_t * path_inode (pid_t pid, const char *path);`	Apre l'inode del file indicato tramite il percorso, purché il processo ***pid*** abbia i permessi di accesso («x») alle directory che vi conducono. La funzione restituisce il puntatore all'inode aperto, oppure il puntatore nullo se non può eseguire l'operazione.
`inode_t *path_inode_link (pid_t pid, const char *path, inode_t *inode, mode_t mode);`	Crea un collegamento fisico con il nome fornito in ***path***, riferito all'inode a cui punta ***inode***, ma se ***inode*** fosse un puntatore nullo, verrebbe semplicemente creato un file vuoto con un nuovo inode. Si richiede inoltre che il processo ***pid*** abbia i permessi di accesso per tutte le directory che portano al file da collegare e che nell'ultima ci sia anche il permesso di scrittura, dovendo intervenire su tale directory in questo modo. Se la funzione riesce nel proprio intento, restituisce il puntatore a ciò che descrive l'inode collegato o creato.
`int path_link (pid_t pid, const char *path_old, const char *path_new);`	Crea un collegamento fisico. È l'equivalente della funzione standard ***link()*** (sezione u0.23).

Funzione	Descrizione
`int path_mkdir (pid_t pid, const char *path, mode_t mode);`	Crea una directory, con la modalità dei permessi indicata. È l'equivalente della funzione standard ***mkdir()*** (sezione u0.25).
`int path_mknod (pid_t pid, const char *path, mode_t mode, dev_t device);`	Crea un file vuoto, con il tipo e i permessi specificati da ***mode***; se si tratta di un file di dispositivo, viene preso in considerazione anche il parametro ***device***, per specificare il numero primario e secondario dello stesso. Va osservato che con questa funzione è possibile creare una directory priva delle voci '.' e '..'. È l'equivalente della funzione standard ***mknod()*** (sezione u0.26).
`int path_stat (pid_t pid, const char *path, struct stat *buffer);`	Aggiorna la variabile strutturata a cui punta ***buffer***, con le informazioni sul file specificato. È l'equivalente della funzione standard ***stat()*** (sezione u0.36).
`int path_unlink (pid_t pid, const char *path);`	Cancella un file o una directory, purché questa sia vuota. È l'equivalente della funzione standard ***unlink()*** (sezione u0.42).
`int path_mount (pid_t pid, const char *path_dev, const char *path_mnt, int options);`	Innesta il dispositivo corrispondente a ***path_dev***, nella directory ***path_mnt*** (tenendo conto della directory corrente del processo ***pid***), con le opzioni specificate, per conto dell'utente ***uid***. Le opzioni disponibili sono solo '**`MOUNT_DEFAULT`**' e '**`MOUNT_RO`**', come dichiarato nel file di intestazione '`lib/sys/os16.h`'.
`int path_umount (pid_t pid, const char *path_mnt);`	Stacca l'unità innestata nella directory indicata, purché nulla al suo interno sia attualmente in uso.

File «kernel/fs/fd_...»

I file '`kernel/fs/fd_...`' descrivono le funzioni che fanno riferimento a file o directory attraverso il numero di descrittore, riferito a sua volta a un certo processo elaborativo. Pertanto, il numero del processo e il numero del descrittore sono i primi due parametri obbligatori di tutte queste funzioni.

Tabella u148.20. Funzioni per la gestione dei file, a cui si fa riferimento attraverso il descrittore, relativamente a un certo processo elaborativo. La funzione ***fd_open()*** fa eccezione, in quanto apre un descrittore, ma per identificare il file non ancora aperto, ne richiede il percorso.

Funzione	Descrizione
`int fd_chmod (pid_t pid, int fdn, mode_t mode);`	Cambia la modalità dei permessi (solo gli ultimi 12 bit del parametro ***mode*** vengono considerati). È l'equivalente della funzione standard ***fchmod()*** (sezione u0.4).

Funzione	Descrizione
`int fd_chown (pid_t pid, int fdn, uid_t uid, gid_t gid);`	Cambia la proprietà (utente e gruppo). È l'equivalente della funzione standard ***fchown()*** (sezione u0.5).
`int fd_close (pid_t pid, int fdn);`	Chiude il descrittore di file. È l'equivalente della funzione standard ***close()*** (sezione u0.7).
`int fd_dup (pid_t pid, int fdn_old, int fdn_min);`	Duplica il descrittore ***fdn_old***, creandone un altro con numero maggiore o uguale a ***fdn_min*** (viene scelto il primo libero a partire da ***fdn_num***). È l'equivalente della funzione standard ***dup()*** (sezione u0.8).
`int fd_dup2 (pid_t pid, int fdn_old, int fdn_new);`	Duplica il descrittore ***fdn_old***, creandone un altro con numero ***fdn_new***. Se però ***fdn_new*** è già aperto, prima della duplicazione questo viene chiuso. È l'equivalente della funzione standard ***dup2()*** (sezione u0.8).
`int fd_fcntl (pid_t pid, int fdn, int cmd, int arg);`	Svolge il compito della funzione standard ***fcntl()*** (sezione u0.13).
`off_t fd_lseek (pid_t pid, int fdn, off_t offset, int whence);`	Riposiziona l'indice interno di accesso del descrittore di file. È l'equivalente della funzione standard ***lseek()*** (sezione u0.24).
`int fd_open (pid_t pid, const char *path, int oflags, mode_t mode);`	Apre un descrittore, fornendo però il percorso del file. È l'equivalente della funzione standard ***open()*** (sezione u0.28).
`ssize_t fd_read (pid_t pid, int fdn, void *buffer, size_t count, int *eof);`	Legge da un descrittore, aggiornando eventualmente la variabile ****eof*** in caso di fine del file. È l'equivalente della funzione standard ***read()*** (sezione u0.29).
`fd_t *fd_reference (pid_t pid, int *fdn);`	Produce il puntatore ai dati del descrittore ****fdn***. Se ****fdn*** è minore di zero, si ottiene il riferimento al primo descrittore libero, aggiornando anche ****fdn*** stesso.
`int fd_stat (pid_t pid, int fdn, struct stat *buffer);`	Svolge il compito della funzione standard ***fstat()*** (sezione u0.36).
`ssize_t fd_write (pid_t pid, int fdn, const void *buffer, size_t count);`	Scrive nel descrittore. È l'equivalente della funzione standard ***write()*** (sezione u0.44).