Sezione 9: kernel

os16: devices(9) 3476

os16: dev_io(9) 3481
os16: dev_dsk(9) 3483
os16: dev_kmem(9) 3484
os16: dev_mem(9) 3485
os16: dev_tty(9) 3488

os16: diag(9) 3488

os16: fs(9) 3489

os16: fd_chmod(9) 3494
os16: fd_chown(9) 3496
os16: fd_close(9) 3498
os16: fd_dup(9) 3500
os16: fd_dup2(9) 3503
os16: fd_fcntl(9) 3503
os16: fd_lseek(9) 3506
os16: fd_open(9) 3509
os16: fd_read(9) 3514
os16: fd_reference(9) 3516
os16: fd_stat(9) 3518
os16: fd_write(9) 3518
os16: file_reference(9) 3520
os16: file_stdio_dev_make(9) 3521

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

os16: inode_alloc(9) 3523
os16: inode_check(9) 3525
os16: inode_dir_empty(9) 3527
os16: inode_file_read(9) 3529
os16: inode_file_write(9) 3530
os16: inode_free(9) 3532
os16: inode_fzones_read(9) 3534
os16: inode_fzones_write(9) 3535
os16: inode_get(9) 3536
os16: inode_put(9) 3538
os16: inode_reference(9) 3539
os16: inode_save(9) 3541
os16: inode_stdio_dev_make(9) 3542
os16: inode_truncate(9) 3544
os16: inode_zone(9) 3545
os16: path_chdir(9) 3547
os16: path_chmod(9) 3549
os16: path_chown(9) 3553
os16: path_device(9) 3555
os16: path_fix(9) 3556
os16: path_full(9) 3557
os16: path_inode(9) 3559
os16: path_inode_link(9) 3560
os16: path_link(9) 3563
os16: path_mkdir(9) 3565

os16: path_mknod(9) 3568
os16: path_mount(9) 3571
os16: path_stat(9) 3574
os16: path_umount(9) 3574
os16: path_unlink(9) 3574
os16: sb_inode_status(9) 3576
os16: sb_mount(9) 3578
os16: sb_reference(9) 3581
os16: sb_save(9) 3582
os16: sb_zone_status(9) 3584
os16: stat(9) 3584
os16: zone_alloc(9) 3589
os16: zone_free(9) 3590
os16: zone_read(9) 3591
os16: ibm_i86(9) 3592
os16: k_libc(9) 3602
os16: main(9) 3602
os16: memory(9) 3603
os16: proc(9) 3606
os16: isr_1C(9) 3613
os16: ivt_load(9) 3616
os16: proc_available(9) 3617
os16: proc_dump_memory(9) 3617
os16: proc_find(9) 3619

os16: proc_init(9) 3620
os16: proc_reference(9) 3622
os16: proc_sch_signals(9) 3623
os16: proc_sch_terminals(9) 3624
os16: proc_sch_timers(9) 3626
os16: proc_scheduler(9) 3627
os16: proc_sig_chld(9) 3629
os16: proc_sig_cont(9) 3630
os16: proc_sig_core(9) 3632
os16: proc_sig_ignore(9) 3633
os16: proc_sig_on(9) 3635
os16: proc_sig_status(9) 3636
os16: proc_sig_stop(9) 3638
os16: proc_sig_term(9) 3639
os16: proc_sys_exec(9) 3640
os16: proc_sys_exit(9) 3644
os16: proc_sys_fork(9) 3646
os16: proc_sys_kill(9) 3649
os16: proc_sys_seteuid(9) 3652
os16: proc_sys_setuid(9) 3654
os16: proc_sys_signal(9) 3656
os16: proc_sys_wait(9) 3658
os16: sysroutine(9) 3660
os16: tty(9) 3662

devices.h 3476 dev_dsk() 3483 dev_io() 3481
dev_kmem() 3484 dev_mem() 3485 dev_tty() 3488
diag.h 3488 fd_chmod() 3494 fd_chown() 3496
fd_close() 3498 fd_dup() 3500 fd_dup2() 3500
fd_fcntl() 3503 fd_lseek() 3506 fd_open() 3509
fd_read() 3514 fd_reference() 3516 fd_stat() 3584
fd_write() 3518 file_reference() 3520
file_stdio_dev_make() 3521 fs.h 3489 ibm_i86.h
3592 inode_alloc() 3523 inode_check() 3525
inode_dir_empty() 3527 inode_file_read() 3529
inode_file_write() 3530 inode_free() 3532
inode_fzones_read() 3534 inode_fzones_write()
3534 inode_get() 3536 inode_put() 3538
inode_reference() 3539 inode_save() 3541
inode_stdio_dev_make() 3542 inode_truncate()
3544 inode_zone() 3545 isr_1C 3613 isr_80 3613
ivt_load() 3616 k_libc.h 3602 main.h 3602 memory.h
3603 path_chdir() 3547 path_chmod() 3549
path_chown() 3553 path_device() 3555 path_fix()
3556 path_full() 3557 path_inode() 3559
path_inode_link() 3560 path_link() 3563
path_mkdir() 3565 path_mknod() 3568 path_mount()
3571 path_stat() 3584 path_umount() 3571
path_unlink() 3574 proc.h 3606 proc_available()
3617 proc_dump_memory() 3617 proc_find() 3619
proc_init() 3620 proc_reference() 3622
proc_scheduler() 3627 proc_sch_signals() 3623
proc_sch_terminals() 3624 proc_sch_timers() 3626
proc_sig_chld() 3629 proc_sig_cont() 3630

proc_sig_core() 3632 proc_sig_ignore() 3633
proc_sig_off() 3635 proc_sig_on() 3635
proc_sig_status() 3636 proc_sig_stop() 3638
proc_sig_term() 3639 proc_sys_exec() 3640
proc_sys_exit() 3644 proc_sys_fork() 3646
proc_sys_kill() 3649 proc_sys_seteuid() 3652
proc_sys_setuid() 3654 proc_sys_signal() 3656
proc_sys_wait() 3658 sb_inode_status() 3576
sb_mount() 3578 sb_reference() 3581 sb_save() 3582
sb_zone_status() 3576 sysroutine() 3660 tty.h 3663
zone_alloc() 3589 zone_free() 3589 zone_read()
3591 zone_write() 3591 _ivt_load() 3616

os16: devices(9)

«

Il file ‘kernel/devices.h’ [u0.2] descrive ciò che serve per la gestione dei dispositivi. Tuttavia, la definizione dei numeri di dispositivo è contenuta nel file ‘lib/sys/os16.h’ [u0.12], il quale viene incluso da ‘devices.h’.

Tabella u147.4. Classificazione dei dispositivi di os16.

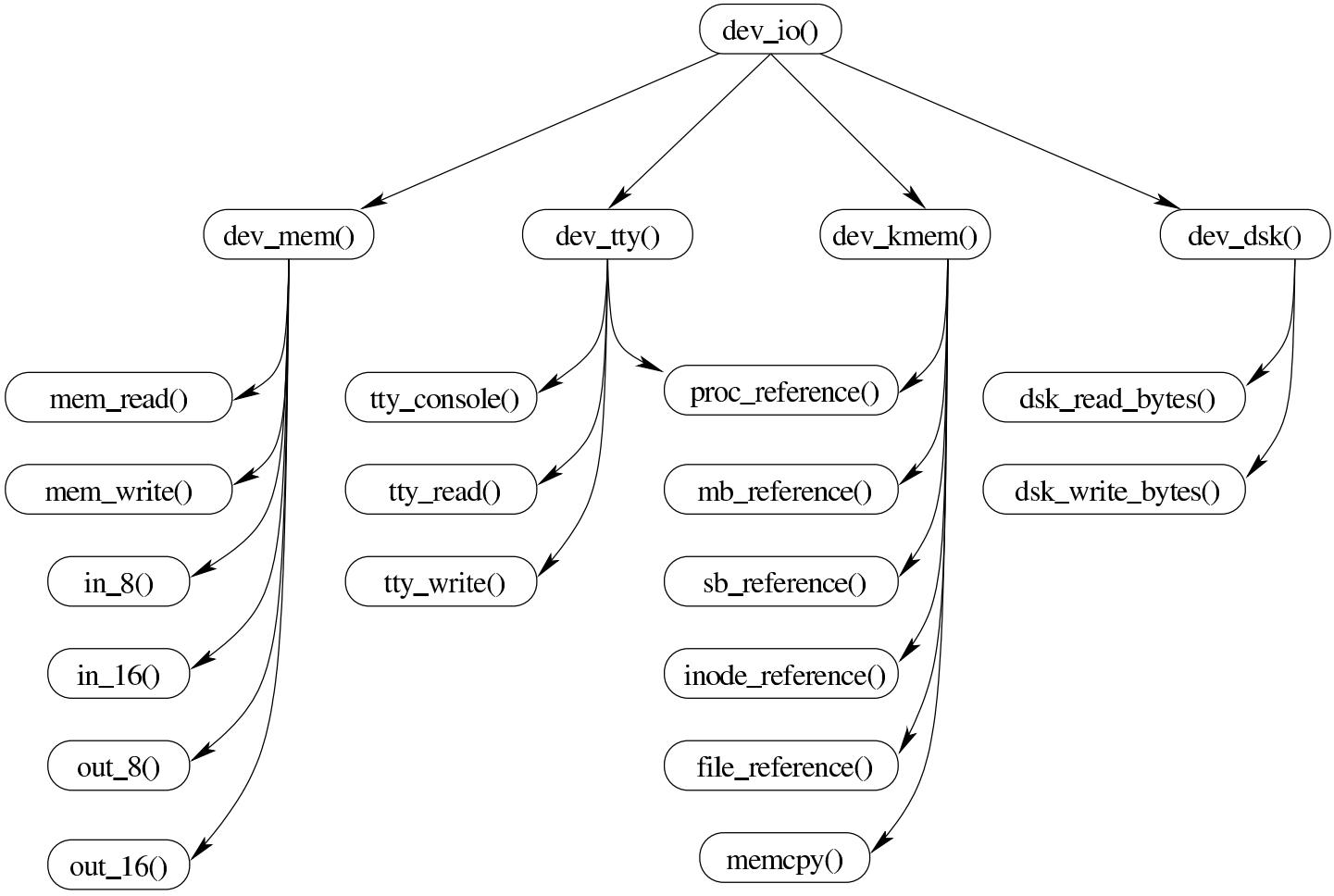
Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`DEV_MEM`	r/w	diretto	Permette l'accesso alla memoria, in modo indiscriminato, perché os16 non offre alcun tipo di protezione al riguardo.
`DEV_NULL`	r/w	nessuno	Consente la lettura e la scrittura, ma non si legge e non si scrive alcunché.
`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.

Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`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_DSK`***n***	r/w	diretto	Rappresenta l'unità a dischi ***n***. os16 non gestisce le partizioni.
`DEV_KMEM_PS`	r	diretto	Rappresenta la tabella contenente le informazioni sui processi. L'indirizzo di accesso indica il numero del processo di partenza; la dimensione da leggere dovrebbe essere abbastanza grande da contenere un processo, ma anche richiedendo una dimensione maggiore, se ne legge uno solo.
`DEV_KMEM_MMP`	r	sequenziale	Rappresenta la mappa della memoria, alla quale si può accedere solo dal suo principio. In pratica, l'indirizzo di accesso viene ignorato, mentre conta solo la quantità di byte richiesta.

Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`DEV_KMEM_SB`	r	diretto	Rappresenta la tabella dei super blocchi (per la gestione delle unità di memorizzazione). L'indirizzo di accesso serve a individuare il super blocco; la dimensione richiesta dovrebbe essere abbastanza grande da contenere un super blocco, ma anche richiedendo una dimensione maggiore, se ne legge uno solo.
`DEV_KMEM_INODE`	r	diretto	Rappresenta la tabella degli inode (per la gestione delle unità di memorizzazione). L'indirizzo di accesso serve a individuare l'inode; la dimensione richiesta dovrebbe essere abbastanza grande da contenere un inode, ma anche richiedendo una dimensione maggiore, se ne legge uno solo.

Dispositivo	Lettura e scrittura r/w	Accesso diretto o sequenziale	Annotazioni
`DEV_KMEM_FILE`	r	diretto	Rappresenta la tabella dei file (per la gestione delle unità di memorizzazione). L'indirizzo di accesso serve a individuare il file; la dimensione richiesta dovrebbe essere abbastanza grande da contenere le informazioni di un file, ma anche richiedendo una dimensione maggiore, se ne legge uno solo.
`DEV_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 u147.1. Interdipendenza tra la funzione ***dev_io()*** e le altre. I collegamenti con le funzioni ***major()*** e ***minor()*** sono omesse.



os16: dev_io(9)

«

NOME

‘**dev_io**’ - interfaccia di accesso ai dispositivi

SINTASSI

```
<kernel/devices.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 os16.

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/devices.h`’ [u0.2]

‘`kernel/devices/dev_io.c`’ [i160.2.2]

VEDERE ANCHE

dev_dsk(9) [i159.1.2], *dev_kmem(9)* [i159.1.3], *dev_mem(9)* [i159.1.4], *dev_tty(9)* [i159.1.5].

os16: dev_dsk(9)

NOME

‘**`dev_dsk`**’ - interfaccia di accesso alle unità di memorizzazione di massa

SINTASSI

```
<kernel/devices.h>
ssize_t dev_dsk (pid_t pid, dev_t device, int rw,
                 off_t offset, void *buffer,
                 size_t size, int *eof);
```

DESCRIZIONE

La funzione ***dev_dsk()*** consente di accedere alle unità di memorizzazione di massa, che per os16 si riducono ai soli dischetti da 1440 Kibyte.

Per il significato degli argomenti, il valore restituito e gli eventuali errori, si veda *dev_io(9)* [i159.1.1].

FILE SORGENTI

'`kernel/devices.h`' [u0.2]

'`kernel/devices/dev_io.c`' [i160.2.2]

'`kernel/devices/dev_dsk.c`' [i160.2.1]

os16: dev_kmem(9)

«

NOME

'**`dev_kmem`**' - interfaccia di accesso alle tabelle di dati del kernel, rappresentate in memoria

SINTASSI

```
<kernel/devices.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)* [i159.1.1].

FILE SORGENTI

'`kernel/devices.h`' [u0.2]

'`kernel/devices/dev_io.c`' [i160.2.2]

'`kernel/devices/dev_kmem.c`' [i160.2.3]

os16: dev_mem(9)

«

NOME

'**`dev_mem`**' - interfaccia di accesso alla memoria, in modo indiscriminato

SINTASSI

```
<kernel/devices.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)* [i159.1.1].

FILE SORGENTI

'`kernel/devices.h`' [u0.2]

'`kernel/devices/dev_io.c`' [i160.2.2]

'`kernel/devices/dev_mem.c`' [i160.2.4]

NOME

‘**dev_tty**’ - interfaccia di accesso alla console

SINTASSI

```
<kernel/devices.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 os16. Va osservato che il testo letto viene anche visualizzato automaticamente. Quando un processo non vuole mostrare il testo sullo schermo, deve provvedere a sovrascriverlo immediatamente, facendo arretrare il cursore preventivamente.

Per quanto non viene descritto qui, si veda *dev_io(9)* [i159.1.1].

FILE SORGENTI

‘kernel/devices.h’ [u0.2]

‘kernel/devices/dev_io.c’ [i160.2.2]

‘kernel/devices/dev_tty.c’ [i160.2.5]

os16: diag(9)

Il file ‘`kernel/diag.h`’ [u0.3] descrive alcune funzioni e macroistruzioni, per uso diagnostico. Lo scopo di queste è di mostrare o di rendere visualizzabile alcune informazioni interne alla gestione del kernel.

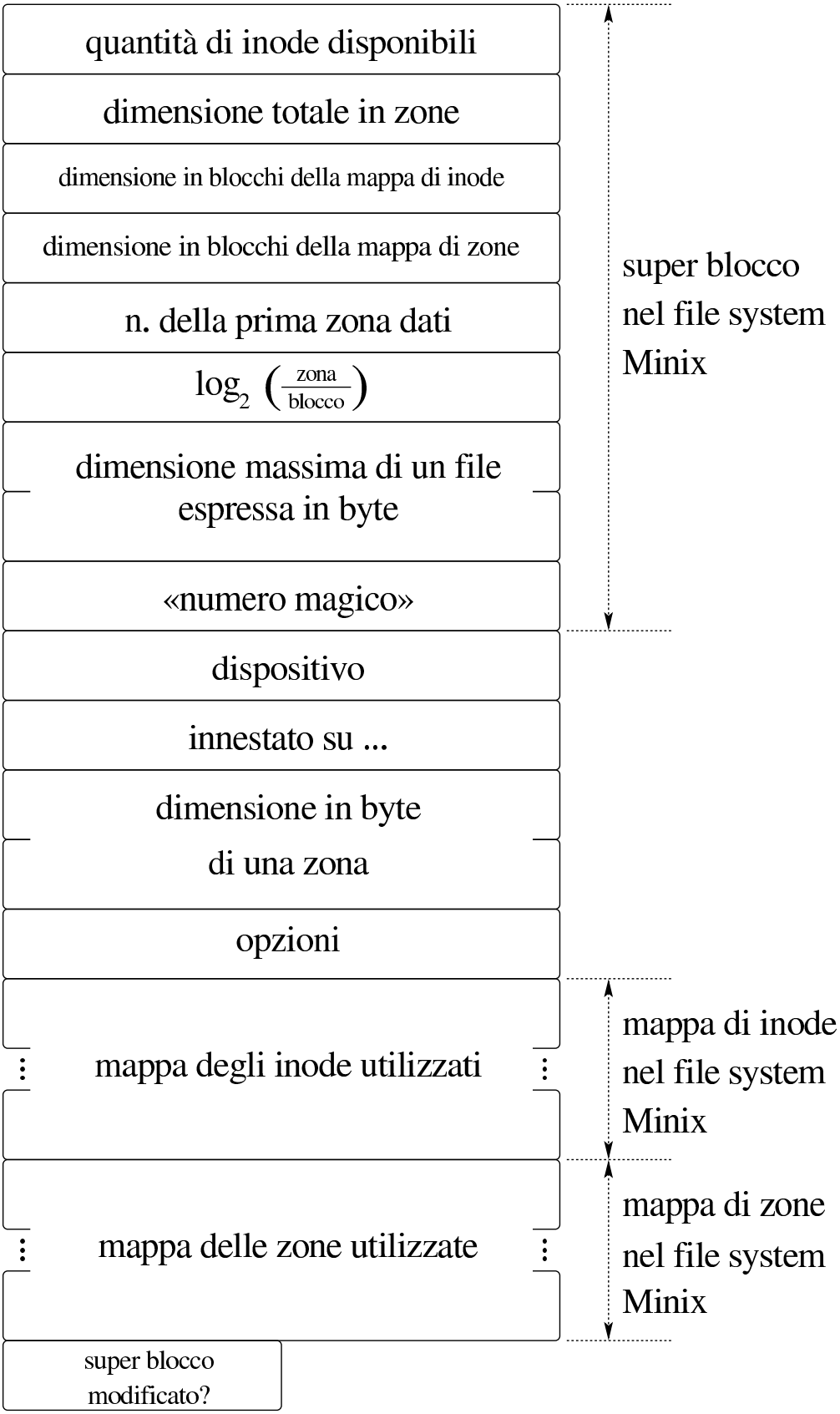
Alcune di queste funzioni sono usate, altre no. Per esempio durante il funzionamento interattivo del kernel vengono usate ***print_proc_list()***, ***print_segments()***, ***print_kmem()***, ***print_time()*** e ***print_mb_map()***.

os16: fs(9)

Il file ‘`kernel/fs.h`’ [u0.4] descrive ciò che serve per la gestione del file system, che per os16 corrisponde al tipo Minix 1.

La gestione del file system, a livello complessivo di sistema, è suddivisa in tre aspetti principali: super blocco, inode e file. Per ognuno di questi è prevista una tabella (di super blocchi, di inode e di file). Seguono delle figure che descrivono l’organizzazione di queste tabelle.

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

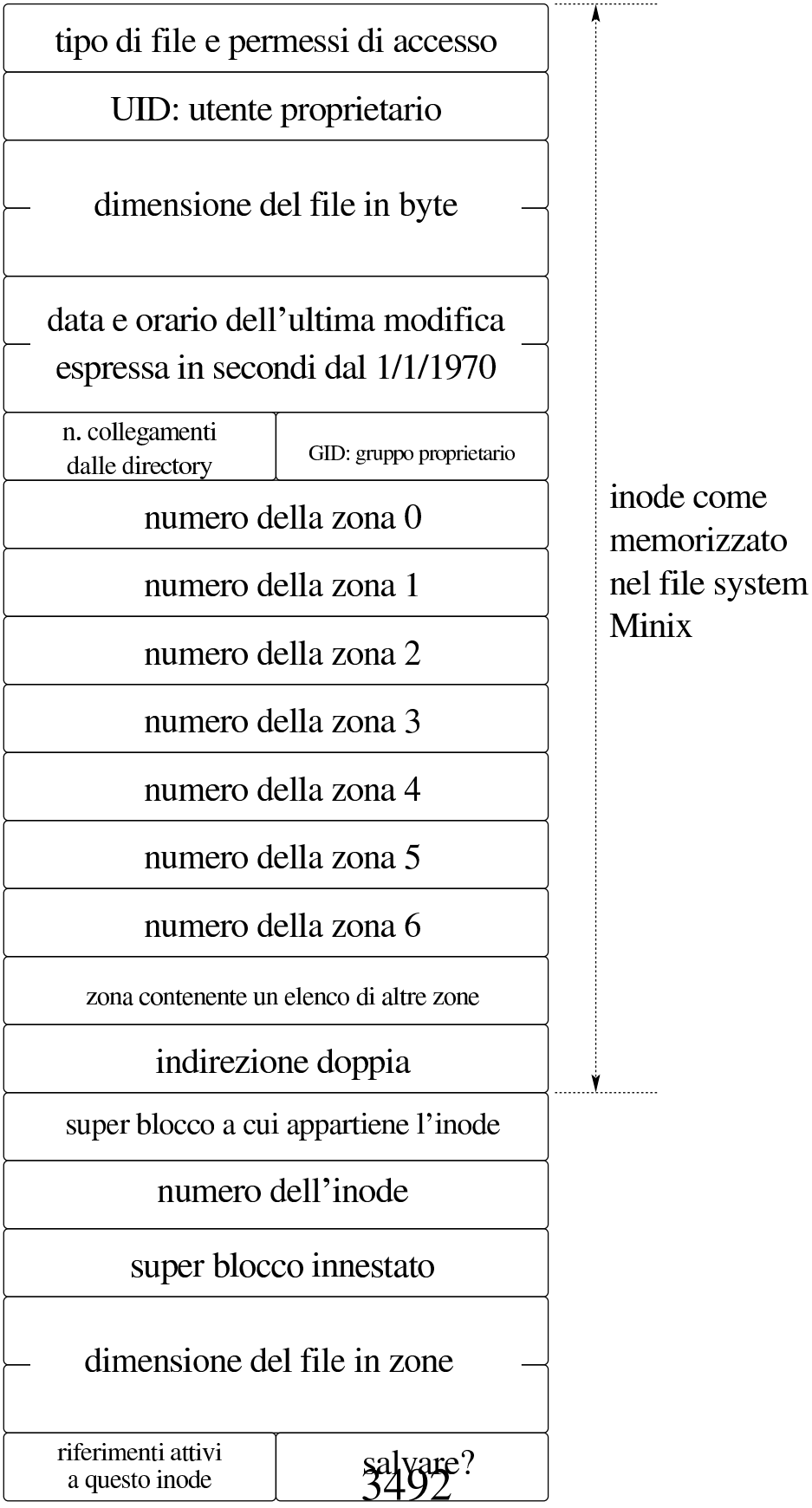


Listato u148.2. Struttura del tipo '**sb_t**', corrispondente agli elementi dell'array ***sb_table[]***.

```
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;
};
```

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

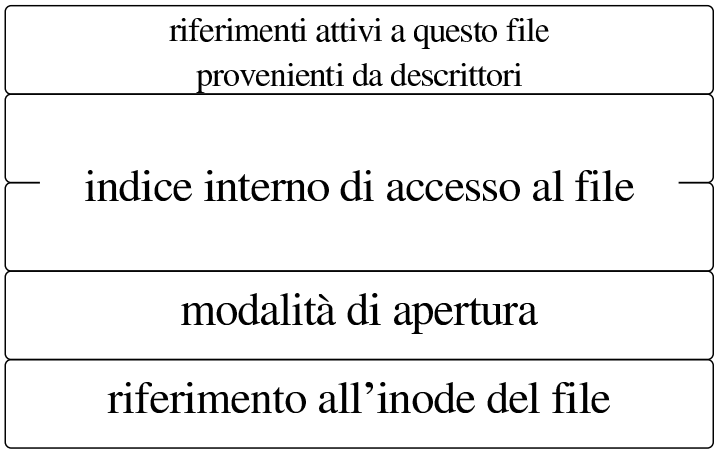


Listato u148.7. Struttura del tipo '**inode_t**', corrispondente agli elementi dell'array ***inode_table[]***.

```
<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;
};
```

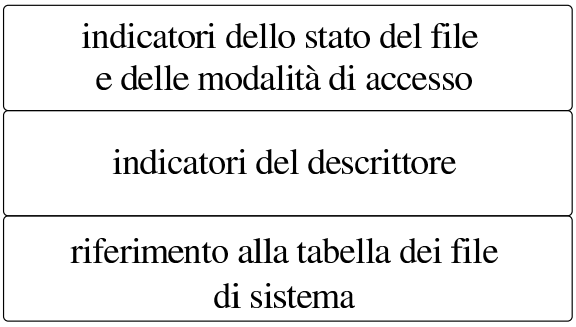
Figura u148.13. Struttura del tipo '**file_t**', corrispondente agli elementi dell'array ***file_table[]***.




```
typedef struct file file_t;

struct file {
  int       references;
  off_t     offset;
  int       oflags;
  inode_t *inode;
};
```

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;
};
```

os16: fd_chmod(9)

«

NOME

'**fd_chmod**' - cambiamento della modalità dei permessi di un descrittore di file

SINTASSI

```
<kernel/fs.h>
int fd_chmod (pid_t pid, int fdn, mode_t mode);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`int` ***fdn***	Numero del descrittore di file.
`mode_t` ***mode***	Modalità dei permessi, di cui si prendono in considerazione solo i 12 bit meno significativi.

DESCRIZIONE

La funzione ***fs_chmod()*** cambia la modalità dei permessi del file aperto con il descrittore numero ***fdn***, secondo il valore contenuto nel parametro ***mode***, di cui però si considerano solo gli ultimi 12 bit. L'operazione viene svolta per conto del processo ***pid***, il quale deve avere i privilegi necessari per poter intervenire così. La modifica della modalità dei permessi raggiunge l'inode del file a cui fa capo il descrittore in questione; pertanto l'inode viene necessariamente salvato dopo la modifica. Il fatto che il descrittore di file possa essere stato aperto in sola lettura, non impedisce la modifica dell'inode attuata da questa funzione.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_FCHMOD`**'. Nella libreria standard, si avvale di questa funzionalità *fchmod(2)* [u0.4].

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
−1	Operazione fallita, con aggiornamento della variabile ***errno*** del kernel.

ERRORI

Valore di *errno*	Significato
EBADF	Il descrittore di file ***fdn*** non è valido.
EACCES	Il processo elaborativo non ha i privilegi necessari nei confronti del file.

FILE SORGENTI

‘lib/sys/stat/fchmod.c’ [i161.13.2]

‘kernel/proc.h’ [u0.9]

‘kernel/proc/_isr.s’ [i160.9.1]

‘kernel/proc/sysroutine.c’ [i160.9.30]

‘kernel/fs.h’ [u0.4]

‘kernel/fs/fd_chmod.c’ [i160.4.1]

VEDERE ANCHE

fchmod(2) [u0.4], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7].

os16: fd_chown(9)

«

NOME

‘**fd_chown**’ - cambiamento della proprietà di un descrittore di file

SINTASSI

```
<kernel/fs.h>
int fd_chown (pid_t pid, int fdn, uid_t uid, gid_t gid);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`int` ***fdn***	Numero del descrittore di file.
`uid_t` ***uid***	Nuova proprietà che si intende attribuire al file.
`gid_t` ***gid***	Nuovo gruppo che si intenderebbe attribuire al file.

DESCRIZIONE

La funzione ***fs_chown()*** cambia la proprietà del file già aperto, individuato attraverso il suo descrittore. L'operazione viene svolta per conto del processo ***pid***, il quale deve avere i privilegi necessari per poter intervenire così: in pratica deve trattarsi di un processo con identità efficace pari a zero, perché os16 non considera la gestione dei gruppi. La modifica della proprietà raggiunge l'inode del file a cui fa capo il descrittore in questione; pertanto l'inode viene necessariamente salvato dopo la modifica. Il fatto che il descrittore di file possa essere stato aperto in sola lettura, non impedisce la modifica dell'inode attuata da questa funzione.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_FCHOWN`**'. Nella libreria standard, si avvale di questa funzionalità *fchown(2)* [u0.4].

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
−1	Operazione fallita, con aggiornamento della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il descrittore di file ***fdn*** non è valido.
`EACCES`	Il processo elaborativo non ha i privilegi necessari nei confronti del file.

FILE SORGENTI

‘`lib/unistd/fchown.c`’ [i161.17.16]

‘`kernel/proc.h`’ [u0.9]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/fd_chown.c`’ [i160.4.2]

VEDERE ANCHE

fchown(2) [u0.5], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7].

os16: fd_close(9)

«

NOME

‘**`fd_close`**’ - chiusura di un descrittore di file

SINTASSI

```
<kernel/fs.h>
int fd_close (pid_t pid, int fdn);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`int` ***fdn***	Numero del descrittore di file.

DESCRIZIONE

La funzione ***fd_close()*** chiude il descrittore di file specificato come argomento. Per ottenere questo risultato, oltre che intervenire nella tabella dei descrittori associata al processo elaborativo specificato come argomento, riduce il contatore dei riferimenti nella voce corrispondente della tabella dei file; se però questo contatore raggiunge lo zero, anche l'inode viene liberato, attraverso *inode_put(9)* [i159.3.24].

Questa funzione viene usata in modo particolare da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_CLOSE`**'. Nella libreria standard, si avvale di questa funzionalità *close(2)* [u0.7]. La funzione '**`fd_close`**' è comunque usata internamente al kernel, in tutte le occasioni in cui la chiusura di un descrittore deve avvenire in modo implicito.

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
−1	Operazione fallita, con aggiornamento della variabile ***errno*** del kernel.

ERRORI

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

FILE SORGENTI

‘`lib/unistd/close.c`’ [i161.17.5]

‘`kernel/proc.h`’ [u0.9]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/fd_close.c`’ [i160.4.3]

VEDERE ANCHE

close(2) [u0.7], *sysroutine(9)* [i159.8.28], *inode_put(9)* [i159.3.24].

os16: fd_dup(9)

«

NOME

‘**`fd_dup`**’, ‘**`fd_dup2`**’ - duplicazione di un descrittore di file

SINTASSI

```
<kernel/fs.h>
int fd_dup  (pid_t pid, int fdn_old, int fdn_min);
int fd_dup2 (pid_t pid, int fdn_old, int fdn_new);
```

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.
`int` ***fdn_new***	Numero di descrittore da usare per la copia.

DESCRIZIONE

Le funzioni ***fd_dup()*** e ***fd_dup2()*** duplicano un descrittore, nel senso che sdoppiano l'accesso a un file in due descrittori. La funzione ***fd_dup()***, per il duplicato da realizzare, cerca un descrittore libero, cominciando da ***fdn_min*** e continuando progressivamente, fino al primo disponibile. La funzione ***fd_dup2()***, invece, richiede di specificare esattamente il descrittore da usare per il duplicato, con la differenza che, se ***fdn_new*** è già utilizzato, prima della duplicazione viene chiuso.

In entrambi i casi, il descrittore ottenuto dalla copia, viene privato dell'indicatore '**`FD_CLOEXEC`**', ammesso che nel descrittore originale ci fosse.

Queste funzioni vengono usate da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di ti-

po '**SYS_DUP**' e '**SYS_DUP2**'. Inoltre, la funzione *fd_fcntl(9)* [i159.3.6] si avvale di ***fd_dup()*** per la duplicazione di un descrittore. Le funzioni della libreria standard che si avvalgono delle chiamate di sistema che poi raggiungono ***fd_dup()*** e ***fd_dup2()*** sono *dup(2)* [u0.8] e *dup2(2)* [u0.8].

VALORE RESTITUITO

Le due funzioni restituiscono il numero del descrittore prodotto dalla duplicazione. In caso di errore, invece, restituiscono il valore −1, aggiornando la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EINVAL`	Il valore di ***fdn_min*** o ***fdn_new*** è impossibile.
`EBADF`	Il valore di ***fdn_old*** non è valido.
`EMFILE`	Non è possibile allocare un nuovo descrittore.

FILE SORGENTI

'`lib/unistd/dup.c`' [i161.17.6]

'`lib/unistd/dup2.c`' [i161.17.7]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs.h`' [u0.4]

'`kernel/fs/fd_dup.c`' [i160.4.4]

'`kernel/fs/fd_dup2.c`' [i160.4.5]

VEDERE ANCHE

dup(2) [u0.8], *dup2(2)* [u0.8], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7].

os16: fd_dup2(9)

Vedere *fd_dup(9)* [i159.3.4].

os16: fd_fcntl(9)

NOME

'**fd_fcntl**' - configurazione e intervento sui descrittori di file

SINTASSI

```
<kernel/fs.h>
int fd_fcntl (pid_t pid, int fdn, int cmd, int arg);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`int` ***fdn***	Numero del descrittore a cui si fa riferimento.
`int` ***cmd***	Comando di ***fd_fcntl()***.
`int` ***arg***	Argomento eventuale del comando.

DESCRIZIONE

La funzione ***fd_fcntl()*** esegue un'operazione, definita dal parametro ***cmd***, sul descrittore ***fdn***. A seconda del tipo di opera-

zione richiesta, può essere preso in considerazione anche l'argomento corrispondente al parametro ***arg***. Il valore del parametro ***cmd*** che rappresenta l'operazione richiesta, va fornito in forma di costante simbolica, come descritto nell'elenco seguente. Tali macro-variabili derivano dalle dichiarazioni contenute nel file '`lib/sys/fcntl.h`'.

Sintassi	Descrizione

`fd_fcntl (`***pid***`, `***fdn***`,`
`F_DUPFD,`
`(int) `***fdn_min***`)`
	Richiede la duplicazione del descrittore di file ***fdn***, in modo tale che la copia abbia il numero di descrittore minore possibile, ma maggiore o uguale a quello indicato come argomento ***fdn_min***.

`fd_fcntl (`***pid***`, `***fdn***`,`
`F_GETFD, 0)`
`fd_fcntl (`***pid***`, `***fdn***`,`
`F_SETFD,`
`(int) `***fd_flags***`)`
	Rispettivamente, legge o imposta, gli indicatori del descrittore di file ***fdn*** (eventualmente noti come *fd flags*). È possibile impostare un solo indicatore, '**`FD_CLOEXEC`**', pertanto, al posto di ***fd_flags*** si può mettere solo la costante '**`FD_CLOEXEC`**'.

`fd_fcntl (`***pid***`, `***fdn***`,`
`F_GETFL, 0)`
`fd_fcntl (`***pid***`, `***fdn***`,`
`F_SETFL,`
`(int) `***fl_flags***`)`
	Rispettivamente, legge o imposta, gli indicatori dello stato del file, relativi al descrittore ***fdn***. Per impostare questi indicatori, vanno combinate delle costanti simboliche: '**`O_RDONLY`**', '**`O_WRONLY`**', '**`O_RDWR`**', '**`O_CREAT`**', '**`O_EXCL`**', '**`O_NOCTTY`**', '**`O_TRUNC`**'.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_FCNTL`**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***fd_fcntl()*** è *fcntl(2)* [u0.13].

VALORE RESTITUITO

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

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

`F_GETOWN`
`F_SETOWN`
`F_GETLK`
`F_SETLK`
`F_SETLKW`
	Si ottiene −1, in quanto si tratta di operazioni non realizzate in questa versione della funzione, per os16.

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

ERRORI

Valore di *errno*	Significato
`EBADF`	Il descrittore di file non è valido.
`EINVAL`	È stato richiesto un tipo di operazione non valido.
`EMFILE`	Non è possibile duplicare il descrittore, perché non ce ne sono di liberi.

FILE SORGENTI

‘`lib/fcntl/fcntl.c`’ [i161.4.2]

‘`kernel/proc.h`’ [u0.9]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/fd_fcntl.c`’ [i160.4.6]

VEDERE ANCHE

fcntl(2) [u0.13], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *fd_dup(9)* [i159.3.4].

os16: fd_lseek(9)

«

NOME

‘**`fd_lseek`**’ - riposizionamento dell’indice di accesso a un descrittore di file

SINTASSI

```
<kernel/fs.h>
off_t fd_lseek (pid_t pid, int fdn, off_t offset, int whence);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`int` ***fdn***	Numero del descrittore a cui si fa riferimento.
`int` ***cmd***	Comando di ***fd_fcntl()***.
`off_t` ***offset***	Scostamento, positivo o negativo, a partire dalla posizione indicata da ***whence***.
`int` ***whence***	Punto di riferimento iniziale a cui applicare lo scostamento.

DESCRIZIONE

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

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

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

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_LSEEK`**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***fd_lseek()*** è *lseek(2)* [u0.24].

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

'`lib/unistd/lseek.c`' [i161.17.27]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs.h`' [u0.4]

‘kernel/fs/fd_lseek.c’ [i160.4.7]

VEDERE ANCHE

lseek(2) [u0.24], *sysroutine(9)* [i159.8.28], *fd_reference(9)* [i159.3.10].

os16: fd_open(9)

«

NOME

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

SINTASSI

```
<kernel/fs.h>
int fd_open (pid_t pid, const char *path, int oflags,
             mode_t mode);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`char *`***path***	Il percorso assoluto del file da aprire.
`int` ***oflags***	Opzioni di apertura.
`mode_t` ***mode***	Tipo di file e modalità dei permessi, nel caso il file debba essere creato contestualmente.

DESCRIZIONE

La funzione ***fd_open()*** apre un file, indicato attraverso il percorso ***path***, in base alle opzioni rappresentate dagli indicatori ***oflags***.

A seconda del tipo di indicatori specificati, il parametro ***mode*** potrebbe essere preso in considerazione.

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

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

Opzione	Descrizione
`O_RDONLY`	Richiede un accesso in lettura.
`O_WRONLY`	Richiede un accesso in scrittura.

`O_RDWR`
 	
`O_RDONLY|O_WRONLY`
	Richiede un accesso in lettura e scrittura (la combinazione di '**`R_RDONLY`**' e di '**`O_WRONLY`**' è equivalente all'uso di '**`O_RDWR`**').

`O_CREAT`	Richiede di creare contestualmente il file, ma in tal caso va usato anche il parametro ***mode***.
`O_TRUNC`	Se file da aprire esiste già, richiede che questo sia ridotto preventivamente a un file vuoto.
`O_APPEND`	Fa in modo che le operazioni di scrittura avvengano sempre partendo dalla fine del file.

Quando si utilizza l'opzione ***O_CREAT***, è necessario stabilire la modalità dei permessi, attraverso la combinazione di macro-

variabili appropriate, come elencato nella tabella successiva. Tale combinazione va fatta con l'uso dell'operatore OR binario; per esempio: '**`S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH`**'. Va osservato che os16 non gestisce i gruppi di utenti, pertanto, la definizione dei permessi relativi agli utenti appartenenti al gruppo proprietario di un file, non ha poi effetti pratici nel controllo degli accessi per tale tipo di contesto.

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

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

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

Questa funzione viene usata principalmente da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_OPEN`**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***fd_open()*** è *open(2)* [u0.28].

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

'`lib/fcntl/open.c`' [i161.4.3]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs.h`' [u0.4]

'`kernel/fs/fd_open.c`' [i160.4.8]

VEDERE ANCHE

open(2) [u0.28], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_inode(9)* [i159.3.36], *path_full(9)* [i159.3.35], *path_inode_link(9)* [i159.3.37], *inode_truncate(9)* [i159.3.28], *inode_check(9)* [i159.3.16], *file_reference(9)* [i159.3.13], *fd_reference(9)* [i159.3.10].

os16: fd_read(9)

«

NOME

'**fd_read**' - lettura di descrittore di file

SINTASSI

```
<kernel/fs.h>
ssize_t fd_read (pid_t pid, int fdn, void *buffer,
                 size_t count,
                 int *eof);
```

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.
`void *`***buffer***	Area di memoria in cui scrivere ciò che viene letto dal descrittore di file.
`size_t` ***count***	Quantità di byte da leggere.
`int *`***eof***	Puntatore a una variabile in cui annotare, eventualmente, il raggiungimento della fine del file.

DESCRIZIONE

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

La funzione può leggere file normali, file di dispositivo e directory, trattandole però come se fossero dei file puri e semplici. Gli altri tipi di file non sono gestiti da os16.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_READ`**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***fd_read()*** è *read(2)* [u0.29].

VALORE RESTITUITO

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

ERRORI

Valore di *errno*	Significato
`EBADF`	Il numero del descrittore di file non è valido.
`EINVAL`	Il file non è aperto in lettura.
`E_FILE_TYPE_UNSUPPORTED`	Il file è di tipo non gestibile con os16.

FILE SORGENTI

‘`lib/unistd/read.c`’ [i161.17.28]

‘`kernel/proc.h`’ [u0.9]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/fd_read.c`’ [i160.4.9]

VEDERE ANCHE

read(2) [u0.29], *sysroutine(9)* [i159.8.28], *fd_reference(9)* [i159.3.10], *dev_io(9)* [i159.1.1], *inode_file_read(9)* [i159.3.18].

os16: 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`' [u0.4]

'`kernel/fs/fd_reference.c`' [i160.4.10]

VEDERE ANCHE

file_reference(9) [i159.3.13], *inode_reference(9)* [i159.3.25], *sb_reference(9)* [i159.3.47], *proc_reference(9)* [i159.8.7].

os16: fd_stat(9)

«

Vedere *stat(9)* [i159.3.50].

os16: fd_write(9)

«

NOME

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

SINTASSI

```
<kernel/fs.h>
ssize_t fd_write (pid_t pid, int fdn, const void *buffer,
                  size_t count);
```

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.
`const void *`***buffer***	Area di memoria da cui attingere i dati da scrivere nel descrittore di file.
`size_t` ***count***	Quantità di byte da scrivere.

DESCRIZIONE

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

Questa funzione viene usata principalmente da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_WRITE`**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***fd_write()*** è *write(2)* [u0.44].

VALORE RESTITUITO

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

ERRORI

Valore di ***errno***	Significato
`EBADF`	Il numero del descrittore di file non è valido.
`EINVAL`	Il file non è aperto in scrittura.
`EISDIR`	Il file è una directory.
`E_FILE_TYPE_UNSUPPORTED`	Il file è di tipo non gestibile con os16.
`EIO`	Errore di input-output.

FILE SORGENTI

'`lib/unistd/write.c`' [i161.17.36]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs.h`' [u0.4]

‘kernel/fs/fd_write.c’ [i160.4.12]

VEDERE ANCHE

write(2) [u0.44], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *dev_io(9)* [i159.1.1], *inode_file_write(9)* [i159.3.19].

os16: 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`’ [u0.4]

‘`kernel/fs/file_table.c`’ [i160.4.15]

‘`kernel/fs/file_reference.c`’ [i160.4.13]

VEDERE ANCHE

fd_reference(9) [i159.3.10], *inode_reference(9)* [i159.3.25], *sb_reference(9)* [i159.3.47], *proc_reference(9)* [i159.8.7].

os16: 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)* [i159.3.27], 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)* [i159.8.20], 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`’ [u0.4]

‘`kernel/fs/file_stdio_dev_make.c`’ [i160.4.14]

VEDERE ANCHE

proc_sys_exec(9) [i159.8.20], *inode_stdio_dev_make(9)* [i159.3.27], *file_reference(9)* [i159.3.13], *inode_put(9)* [i159.3.24].

os16: 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);
```

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***	Proprietà 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)* [i159.3.37], 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`' [u0.4]

'`kernel/fs/inode_alloc.c`' [i160.4.16]

VEDERE ANCHE

path_inode_link(9) [i159.3.37], *sb_reference(9)* [i159.3.47], *inode_get(9)* [i159.3.23], *inode_put(9)* [i159.3.24], *inode_truncate(9)* [i159.3.28], *inode_save(9)* [i159.3.26].

os16: 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);
```

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 os16 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.

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. 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***, tenendo conto del complesso dei permessi che lo 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`' [u0.4]

'`kernel/fs/inode_check.c`' [i160.4.17]

os16: 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`' [u0.4]

'`kernel/fs/inode_dir_empty.c`' [i160.4.18]

VEDERE ANCHE

inode_file_read(9) [i159.3.18].

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)* [i159.3.21], 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`' [u0.4]

'`kernel/fs/inode_file_read.c`' [i160.4.19]

VEDERE ANCHE

inode_fzones_read(9) [i159.3.21].

os16: 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)* [i159.3.21], per accedere ai contenuti del file, suddivisi in zone, secondo l'organizzazione del file system Minix 1, e di *zone_write(9)* [i159.3.53], per la riscrittura delle zone relative.

Per os16, 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`' [u0.4]

'`kernel/fs/inode_file_write.c`' [i160.4.20]

VEDERE ANCHE

inode_fzones_read(9) [i159.3.21], *zone_write(9)* [i159.3.53].

os16: 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)* [i159.3.24], per completare la cancellazione di un inode che non ha più collegamenti nel file system, nel momento in cui 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`' [u0.4]

'`kernel/fs/inode_free.c`' [i160.4.21]

VEDERE ANCHE

inode_save(9) [i159.3.26], *inode_alloc(9)* [i159.3.15].

«

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)* [i159.3.18] e *inode_file_write(9)* [i159.3.19], 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`' [u0.4]

'`kernel/fs/inode_fzones_read.c`' [i160.4.22]

'`kernel/fs/inode_fzones_write.c`' [i160.4.23]

VEDERE ANCHE

inode_file_read(9) [i159.3.18], *inode_file_write(9)* [i159.3.19], *zone_read(9)* [i159.3.53], *zone_write(9)* [i159.3.53].

os16: inode_fzones_write(9)

«

Vedere *inode_fzones_read(9)* [i159.3.21].

«

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 nel-

la 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`' [u0.4]

'`kernel/fs/inode_get.c`' [i160.4.24]

VEDERE ANCHE

offsetof(3) [u0.75], *inode_put(9)* [i159.3.24], *inode_reference(9)* [i159.3.25], *sb_reference(9)* [i159.3.47], *sb_inode_status(9)* [i159.3.45], *dev_io(9)* [i159.1.1].

«

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`’ [u0.4]

‘`kernel/fs/inode_put.c`’ [i160.4.25]

VEDERE ANCHE

inode_truncate(9) [i159.3.28], *inode_free(9)* [i159.3.20], *inode_save(9)* [i159.3.26].

os16: 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`’ [u0.4]

‘`kernel/fs/inode_reference.c`’ [i160.4.26]

VEDERE ANCHE

sb_reference(9) [i159.3.47], *file_reference(9)* [i159.3.13], *proc_reference(9)* [i159.8.7].

os16: 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`' [u0.4]

'`kernel/fs/inode_save.c`' [i160.4.27]

VEDERE ANCHE

sb_save(9) [i159.3.48], *dev_io(9)* [i159.1.1].

os16: 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)* [i159.3.14], 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`’ [u0.4]

‘`kernel/fs/inode_stdio_dev_make.c`’ [i160.4.28]

VEDERE ANCHE

file_stdio_dev_make(9) [i159.3.14], *inode_reference(9)* [i159.3.25].

os16: 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`’ [u0.4]

‘`kernel/fs/inode_truncate.c`’ [i160.4.30]

VEDERE ANCHE

zone_free(9) [i159.3.51], *sb_save(9)* [i159.3.48], *inode_save(9)* [i159.3.26].

os16: 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`’ [u0.4]

‘`kernel/fs/inode_zone.c`’ [i160.4.31]

VEDERE ANCHE

memset(3) [u0.72], *zone_alloc(9)* [i159.3.51], *zone_read(9)* [i159.3.53], *zone_write(9)* [i159.3.53].

os16: path_chdir(9)

«

NOME

‘**`path_chdir`**’ - cambiamento della directory corrente

SINTASSI

```
<kernel/fs.h>
int path_chdir (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 della nuova directory corrente, riferito alla directory corrente del processo ***pid***.

DESCRIZIONE

La funzione ***path_chdir()*** cambia la directory corrente del processo ***pid***, in modo che quella nuova corrisponda al percorso annotato nella stringa ***path***.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_CHDIR`**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***path_chdir()*** è *chdir(2)* [u0.3].

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`	Argomento non valido.
`EACCES`	Accesso negato.
`ENOTDIR`	Uno dei componenti del percorso non è una directory.
`ENOENT`	Uno dei componenti del percorso non esiste.

FILE SORGENTI

‘`lib/unistd/chdir.c`’ [i161.17.3]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs/path_chdir.c`’ [i160.4.32]

VEDERE ANCHE

chdir(2) [u0.3], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_full(9)* [i159.3.35], *path_inode(9)* [i159.3.36], *inode_put(9)* [i159.3.24].

os16: path_chmod(9)

«

NOME

‘**`path_chmod`**’ - cambiamento della modalità dei permessi di un file

SINTASSI

```
<kernel/fs.h>
int path_chmod (pid_t pid, const char *path, 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 su cui intervenire.
`mode_t` ***mode***	La modalità dei_permessi da applicare (contano solo i 12 bit meno significativi).

DESCRIZIONE

La funzione ***path_chmod()*** modifica la modalità dei permessi di accesso del file indicato, tramite il suo percorso, relativo eventualmente alla directory corrente del processo ***pid***.

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

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

os16 non considera i permessi SUID (*Set user id*), SGID (*Set group id*) e Sticky, che nella tabella non sono stati nemmeno annotati; inoltre, non tiene in considerazione i permessi legati al gruppo, perché non tiene traccia dei gruppi.

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
`EACCES`	Permesso negato.
`EBADF`	Il descrittore del file richiesto non è valido.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_CHMOD`**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***path_chmod()*** è *chmod(2)* [u0.4].

FILE SORGENTI

'`lib/sys/stat/chmod.c`' [i161.13.1]

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs.h`' [u0.4]

'`kernel/fs/path_chmod.c`' [i160.4.33]

VEDERE ANCHE

chmod(2) [u0.4], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_inode(9)* [i159.3.36].

NOME

‘**path_chown**’ - cambiamento della proprietà di un file

SINTASSI

```
<kernel/fs.h>
int path_chown (pid_t pid, const char *path, uid_t uid,
                gid_t gid);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`const char *`***path***	Il percorso del file su cui intervenire.
`uid_t` ***uid***	Utente a cui attribuire la proprietà del file.
`gid_t` ***gid***	Gruppo a cui associare il file.

DESCRIZIONE

La funzione ***path_chown()*** modifica la proprietà di un file, fornendo il numero UID e il numero GID. Il file viene indicato attraverso il percorso scritto in una stringa, relativo alla directory corrente del processo ***pid***.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo ‘**SYS_CHOWN**’. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***path_chown()*** è *chown(2)* [u0.5].

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

DIFETTI

Benché sia consentito di attribuire il numero del gruppo, os16 non valuta i permessi di accesso ai file, relativi a questi.

FILE SORGENTI

'`lib/unistd/chown.c`' [i161.17.4]

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs.h`' [u0.4]

'`kernel/fs/path_chown.c`' [i160.4.34]

VEDERE ANCHE

chown(2) [u0.5], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_inode(9)* [i159.3.36], *inode_save(9)* [i159.3.26], *inode_put(9)* [i159.3.24].

os16: 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 *path_mount(9)* [i159.8.28].

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`’ [u0.4]

‘`kernel/fs/path_device.c`’ [i160.4.35]

VEDERE ANCHE

proc_reference(9) [i159.8.7], *path_inode(9)* [i159.3.36], *inode_put(9)* [i159.3.24].

os16: 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`' [u0.4]

'`kernel/fs/path_fix.c`' [i160.4.36]

VEDERE ANCHE

strtok(3) [u0.120], *strcmp(3)* [u0.106], *strcat(3)* [u0.104], *strncat(3)* [u0.104], *strncpy(3)* [u0.108].

os16: 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`' [u0.4]

'`kernel/fs/path_full.c`' [i160.4.37]

VEDERE ANCHE

strtok(3) [u0.120], *strcmp(3)* [u0.106], *strcat(3)* [u0.104], *strncat(3)* [u0.104], *strncpy(3)* [u0.108], *path_fix(9)* [i159.3.34].

os16: 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 sul-

l'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`' [u0.4]

'`kernel/fs/path_inode.c`' [i160.4.38]

VEDERE ANCHE

proc_reference(9) [i159.8.7], *path_full(9)* [i159.3.35], *inode_get(9)* [i159.3.23], *inode_put(9)* [i159.3.24], *inode_check(9)* [i159.3.16], *inode_file_read(9)* [i159.3.18].

os16: 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, nel-

l'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`' [u0.4]

'`kernel/fs/path_inode_link.c`' [i160.4.39]

VEDERE ANCHE

proc_reference(9) [i159.8.7], *path_inode(9)* [i159.3.36],

inode_get(9) [i159.3.23], *inode_put(9)* [i159.3.24], *inode_save(9)* [i159.3.26], *inode_check(9)* [i159.3.16], *inode_alloc(9)* [i159.3.15], *inode_file_read(9)* [i159.3.18], *inode_file_write(9)* [i159.3.19].

os16: path_link(9)

«

NOME

'**path_link**' - creazione di un collegamento fisico

SINTASSI

```
<kernel/fs.h>
int path_link (pid_t pid, const char *path_old,
               const char *path_new);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`const char *`***path_old***	Il percorso del file originario.
`const char *`***path_new***	Il percorso del collegamento da creare.

DESCRIZIONE

La funzione ***path_link()*** produce un nuovo collegamento a un file già esistente. Va fornito il percorso del file già esistente, ***path_old*** e quello del file da creare, in qualità di collegamento, ***path_new***. L'operazione può avvenire soltanto se i due percorsi si trovano sulla stessa unità di memorizzazione e se ci sono i permessi di

scrittura necessari nella directory di destinazione per il processo ***pid***. Dopo l'operazione di collegamento, fatta in questo modo, non è possibile distinguere quale sia stato il file originale e quale sia invece il nome aggiunto.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_LINK`**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***path_link()*** è *link(2)* [u0.23].

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

'lib/unistd/link.c' [i161.17.26]

'lib/sys/os16/sys.s' [i161.12.15]

'kernel/proc/_isr.s' [i160.9.1]

'kernel/proc/sysroutine.c' [i160.9.30]

'kernel/fs/path_link.c' [i160.4.40]

'kernel/fs.h' [u0.4]

VEDERE ANCHE

link(2) [u0.23], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_inode(9)* [i159.3.36], *path_inode_link(9)* [i159.3.37], *inode_put(9)* [i159.3.24].

os16: path_mkdir(9)

«

NOME

'**path_mkdir**' - creazione di una directory

SINTASSI

```
<kernel/fs.h>
int path_mkdir (pid_t pid, const char *path, 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 della directory da creare.
`mode_t` ***mode***	Modalità dei permessi da attribuire alla nuova directory.

DESCRIZIONE

La funzione ***path_mkdir()*** crea una directory, indicata attraverso un percorso (parametro ***path()***) e specificando la modalità dei permessi (parametro ***mode***). Va osservato che il valore del parametro ***mode*** non viene preso in considerazione integralmente: di questo si considerano solo gli ultimi nove bit, ovvero quelli dei permessi di utenti, gruppi e altri utenti; inoltre, vengono tolti i bit presenti nella maschera dei permessi associata al processo.

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

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_MKDIR`**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***path_mkdir()*** è *mkdir(2)* [u0.25].

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘`lib/sys/stat/mkdir.c`’ [i161.13.4]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/path_mkdir.c`’ [i160.4.41]

VEDERE ANCHE

mkdir(2) [u0.25], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_inode(9)* [i159.3.36], *inode_file_write(9)* [i159.3.19], *inode_put(9)* [i159.3.24].

«

NOME

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

SINTASSI

```
<kernel/fs.h>
int path_mknod (pid_t pid, const char *path, mode_t mode,
                dev_t device);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`const char *`***path***	Il percorso del file da creare.
`mode_t` ***mode***	Tipo e modalità dei permessi del nuovo file.
`dev_t` ***device***	Numero di dispositivo nel caso il tipo sia riferito a un file di dispositivo.

DESCRIZIONE

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

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

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

Il valore del parametro ***mode*** va costruito combinando assieme delle macro-variabili definite nel file '`lib/sys/stat.h`', come descritto nella pagina di manuale *stat(2)* [u0.36], tenendo conto che os16 non può gestire file FIFO, collegamenti simbolici e socket di dominio Unix.

Il valore del parametro ***mode***, per la porzione che riguarda i permessi di accesso al file, viene comunque filtrato con la maschera dei permessi (*umask(2)* [u0.36]).

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_MKNOD`**'. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***path_mknod()*** è *mknod(2)* [u0.26].

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘`lib/sys/stat.h`’ [u0.13]

‘`lib/sys/stat/mknod.c`’ [i161.13.5]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/path_mknod.c`’ [i160.4.42]

VEDERE ANCHE

mknod(2) [u0.26], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_inode(9)* [i159.3.36], *inode_put(9)* [i159.3.24].

NOME

‘**path_mount**’, ‘**path_umount**’ - innesto e distacco di un file system

SINTASSI

```
<kernel/fs.h>
int path_mount  (pid_t pid, const char *path_dev,
                 const char *path_mnt, int options);
int path_umount (pid_t pid, const char *path_mnt);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo per conto del quale si agisce.
`const char *`***path_dev***	Il file di dispositivo dell’unità da innestare.
`const char *`***path_mnt***	Il percorso della directory di innesto.
`int` ***options***	Opzioni di innesto.

DESCRIZIONE

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

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

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

Queste funzioni vengono usate soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento delle chiamate di sistema di tipo '**`SYS_MOUNT`**' e '**`SYS_UMOUNT`**'. Le funzioni della libreria standard che si avvalgono delle chiamate di sistema che poi raggiungono ***path_mount()*** e ***path_umount()***, sono *mount(2)* [u0.27] e *umount(2)* [u0.27].

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘`lib/sys/os16/mount.c`’ [i161.12.12]

‘`lib/sys/os16/umount.c`’ [i161.12.16]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/path_mount.c`’ [i160.4.43]

‘`kernel/fs/path_umount.c`’ [i160.4.45]

VEDERE ANCHE

mount(2) [u0.27], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_device(9)* [i159.3.33], *path_inode(9)* [i159.3.36], *inode_put(9)* [i159.3.24], *sb_mount(9)* [i159.3.46].

os16: path_stat(9)

«

Vedere *stat(9)* [i159.3.50].

os16: path_umount(9)

«

Vedere *path_mount(9)* [i159.3.41].

os16: path_unlink(9)

«

NOME

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

SINTASSI

```
<kernel/fs.h>
int path_unlink (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 che rappresenta il file da cancellare.

DESCRIZIONE

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

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo

‘**SYS_UNLINK**’. La funzione della libreria standard che si avvale della chiamata di sistema che poi raggiunge ***path_unlink()*** è *unlink(2)* [u0.42].

VALORE RESTITUITO

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

ERRORI

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

FILE SORGENTI

‘`lib/unistd/unlink.c`’ [i161.17.35]

‘`lib/sys/os16/sys.s`’ [i161.12.15]

‘`kernel/proc/_isr.s`’ [i160.9.1]

‘`kernel/proc/sysroutine.c`’ [i160.9.30]

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/path_unlink.c`’ [i160.4.46]

VEDERE ANCHE

unlink(2) [u0.42], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_inode(9)* [i159.3.36], *inode_check(9)* [i159.3.16], *inode_file_read(9)* [i159.3.18], *inode_file_write(9)* [i159.3.19], *inode_put(9)* [i159.3.24].

os16: 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)* [i159.3.23]; 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`' [u0.4]

'`kernel/fs/sb_inode_status.c`' [i160.4.47]

'`kernel/fs/sb_zone_status.c`' [i160.4.52]

VEDERE ANCHE

inode_alloc(9) [i159.3.15], *zone_alloc(9)* [i159.3.51].

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)* [i159.8.6] per innestare il file system principale, e da *path_mount(9)* [i159.3.41] 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 os16.
`EUNKNOWN`	Errore imprevisto e sconosciuto.

FILE SORGENTI

‘`kernel/fs.h`’ [u0.4]

‘`kernel/fs/sb_mount.c`’ [i160.4.48]

VEDERE ANCHE

sb_reference(9) [i159.3.47], *dev_io(9)* [i159.1.1], *inode_get(9)* [i159.3.23].

os16: 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, resti-

tuisce 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`' [u0.4]

'`kernel/fs/sb_table.c`' [i160.4.51]

'`kernel/fs/sb_reference.c`' [i160.4.49]

VEDERE ANCHE

inode_reference(9) [i159.3.25], *file_reference(9)* [i159.3.13].

os16: 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`' [u0.4]

'`kernel/fs/sb_save.c`' [i160.4.50]

VEDERE ANCHE

inode_save(9) [i159.3.26], *dev_io(9)* [i159.1.1].

os16: sb_zone_status(9)

«

Vedere *sb_inode_status(9)* [i159.3.45].

os16: stat(9)

«

NOME

‘**fd_stat**’, ‘**path_stat**’ - interrogazione dello stato di un file

SINTASSI

```
<kernel/fs.h>
int fd_stat   (pid_t pid, int fdn, struct stat *buffer);
int path_stat (pid_t pid, const char *path,
               struct stat *buffer);
```

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.
`const char *`***path***	Il percorso che rappresenta il file.
`struct stat *`***buffer***	Area di memoria in cui scrivere le informazioni sul file.

DESCRIZIONE

Le funzioni ***fd_stat()*** e ***path_stat()*** raccolgono le informazioni disponibili sul file corrispondente al descrittore ***fdn*** del processo ***pid*** o al percorso ***path***, in una variabile strutturata di tipo ‘**struct stat**’, a cui punta ***buffer***. La struttura ‘**struct stat**’ è definita nel file ‘`lib/sys/stat.h`’ nel modo seguente:

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

Va osservato che il file system Minix 1, usato da os16, riporta esclusivamente la data e l'ora di modifica, pertanto le altre due date previste sono sempre uguali a quella di modifica.

Il membro ***st_mode***, oltre alla modalità dei permessi che si cambiano con *fd_chmod(9)* [i159.3.1], serve ad annotare anche il tipo di file. Nel file '`lib/sys/stat.h`' sono definite anche delle macro-variabili per individuare e facilitare la selezione dei bit che compongono le informazioni del membro ***st_mode***:

Modalità simbolica	Modalità numerica	Descrizione
S_IFMT	0170000_8	Maschera che raccoglie tutti i bit che individuano il tipo di file.
S_IFBLK	0060000_8	File di dispositivo a blocchi.
S_IFCHR	0020000_8	File di dispositivo a caratteri.
S_IFIFO	0010000_8	File FIFO, non gestito da os16.
S_IFREG	0100000_8	File puro e semplice.
S_IFDIR	0040000_8	Directory.
S_IFLNK	0120000_8	Collegamento simbolico, non gestito da os16.
S_IFSOCK	0140000_8	Socket di dominio Unix, non gestito da os16.

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

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

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

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

os16 non considera i permessi SUID (*Set user id*), SGID (*Set group id*) e Sticky; inoltre, non considera i permessi legati al gruppo, perché non tiene traccia dei gruppi.

Queste funzioni vengono usate soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento delle chiamate di sistema di tipo '**SYS_STAT**' e '**SYS_FSTAT**'. Le funzioni della libreria standard che si avvalgono delle chiamate di sistema che poi raggiungono ***fd_stat()*** e ***path_stat()***, sono *fstat(2)* [u0.36] e *stat(2)* [u0.36].

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

FILE SORGENTI

'`lib/sys/stat/fstat.c`' [i161.13.3]

'`lib/sys/stat/stat.c`' [i161.13.6]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/fs.h`' [u0.4]

‘kernel/fs/fd_stat.c’ [i160.4.11]

‘kernel/fs/path_stat.c’ [i160.4.44]

VEDERE ANCHE

fstat(2) [u0.36], *stat(2)* [u0.36], *sysroutine(9)* [i159.8.28], *proc_reference(9)* [i159.8.7], *path_inode(9)* [i159.3.36], *inode_put(9)* [i159.3.24].

os16: 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`' [u0.4]

'`kernel/fs/zone_alloc.c`' [i160.4.53]

'`kernel/fs/zone_free.c`' [i160.4.54]

VEDERE ANCHE

zone_write(9) [i159.3.53], *sb_save(9)* [i159.3.48].

os16: zone_free(9)

«

Vedere *zone_alloc(9)* [i159.3.51].

os16: 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`’ [u0.4]

‘`kernel/fs/zone_read.c`’ [i160.4.55]

‘`kernel/fs/zone_write.c`’ [i160.4.56]

VEDERE ANCHE

zone_alloc(9) [i159.3.51], *zone_free(9)* [i159.3.51].

os16: ibm_i86(9)

«

Il file ‘`kernel/ibm_i86.h`’ [u0.5] descrive le funzioni e le macroistruzioni per la gestione dell’hardware.

La sezione u144 descrive complessivamente queste funzioni e le tabelle successive sono tratte da lì.

Tabella u144.2. Funzioni e macroistruzioni di basso livello, dichiarate nel file di intestazione ‘`kernel/ibm_i86.h`’ e descritte nei file della directory ‘`kernel/ibm_i860/`’. Le macroistruzioni hanno argomenti di tipo numerico non precisato, purché in grado di rappresentare il valore necessario.

Funzione o macroistruzione	Descrizione

`void _int10_00 (uint16_t` ***video_mode*** `);`
 	
`void int10_00 (`***video_mode***`);`
	Imposta la modalità video della console.
	Questa funzione viene usata solo da ***con_init()***, per inizializzare la console; la modalità video è stabilita dalla macro-variabile ***IBM_I86_VIDEO_MODE***, dichiarata nel file ‘`kernel/ibm_i86.h`’.

`void _int10_02 (uint16_t` ***page*** `,`
`uint16_t` ***position*** `);`
`void int10_02 (`***page*** `,` ***position*** `);`
	Colloca il cursore in una posizione determinata dello schermo, relativo a una certa pagina video.
	Questa funzione viene usata solo da ***con_putc()***.

`void _int10_05 (uint16_t` ***page*** `);`
 	
`void int10_05 (`***page***`);`
	Seleziona la pagina attiva del video.
	Questa funzione viene usata solo da ***con_init()*** e ***con_select()***.

Funzione o macroistruzione	Descrizione

`void _int12 (void);`
 	
`void int12 (void);`
	Restituisce la quantità di memoria disponibile, in multipli di 1 024 byte.

`void _int13_00 (uint16_t` ***drive*** `);`
 	
`void int13_00 (` ***drive*** `);`
	Azzera lo stato dell'unità a disco indicata, rappresentata da un numero secondo le convenzioni del BIOS. Viene usata solo dalle funzioni '**`dsk_...()`**' che si occupano dell'accesso alle unità a disco.

`uint16_t _int13_02 (uint16_t` ***drive*** `,`
`uint16_t` ***sectors*** `,`
`uint16_t` ***cylinder*** `,`
`uint16_t` ***head*** `,`
`uint16_t` ***sector*** `,`
`void *`***buffer*** `);`
`void int13_02 (` ***drive*** `,` ***sectors*** `,`
cylinder `,` ***head*** `,`
sector `,` ***buffer*** `);`
	Legge dei settori da un'unità a disco. Questa funzione viene usata soltanto da ***dsk_read_sectors()***.

Funzione o macroistruzione	Descrizione
`uint16_t _int13_03 (uint16_t` ***drive*****`,`** `uint16_t` ***sectors*****`,`** `uint16_t` ***cylinder*****`,`** `uint16_t` ***head*****`,`** `uint16_t` ***sector*****`,`** `void *`***buffer***`);` `void int13_03 (`***drive*****`,`** ***sectors*****`,`** ***cylinder*****`,`** ***head*****`,`** ***sector*****`,`** ***buffer***`);`	Scrive dei settori in un'unità a disco. Questa funzione viene usata solo da ***dsk_write_sectors()***.
`uint16_t _int16_00 (void);` `void int16_00 (void);`	Legge un carattere dalla tastiera, rimuovendolo dalla memoria tampone relativa. Viene usata solo in alcune funzioni di controllo della console, denominate '**`con_...()`**'.
`uint16_t _int16_01 (void);` `void int16_01 (void);`	Verifica se è disponibile un carattere dalla tastiera: se c'è ne restituisce il valore, ma senza rimuoverlo dalla memoria tampone relativa, altrimenti restituisce zero. Viene usata solo dalle funzioni di gestione della console, denominate '**`con_...()`**'.

Funzione o macroistruzione	Descrizione

`void _int16_02 (void);`
 	
`void int16_02 (void);`
	Restituisce un valore con cui è possibile determinare quali funzioni speciali della tastiera risultano inserite (inserimento, fissamaiuscole, blocco numerico, ecc.). Al momento la funzione non viene usata.

`uint16_t _in_8 (uint16_t` ***port*****`);`
 	
`void in_8 (`***port*****`);`
	Legge un byte dalla porta di I/O indicata. Questa funzione viene usata da ***irq_on()***, ***irq_off()*** e ***dev_mem()***.

`uint16_t _in_16 (uint16_t` ***port*****`);`
 	
`void in_16 (`***port*****`);`
	Legge un valore a 16 bit dalla porta di I/O indicata. Questa funzione viene usata solo da ***dev_mem()***.

`void _out_8 (uint16_t` ***port*****`,`** `uint16_t` ***value*****`);`** `void out_8 (`***port*****`,` ***value*****`);`**	Scrive un byte nella porta di I/O indicata. Questa funzione viene usata da ***irq_on()***, ***irq_off()*** e ***dev_mem()***.
`void _out_16 (uint16_t` ***port*****`,`** `uint16_t` ***value*****`);`** `void out_16 (`***port*****`,` ***value*****`);`**	Scrive un valore a 16 bit nella porta indicata. Questa funzione viene usata solo da ***dev_mem()***.

Funzione o macroistruzione	Descrizione
`void cli (void);`	Azzera l'indicatore delle interruzioni, nel registro ***FLAGS***. La funzione serve a permettere l'uso dell'istruzione '**`CLI`**' dal codice in linguaggio C, ma in questa veste, viene usata solo dalla funzione ***proc_init()***.
`void sti (void);`	Attiva l'indicatore delle interruzioni, nel registro ***FLAGS***. La funzione serve a permettere l'uso dell'istruzione '**`STI`**' dal codice in linguaggio C, ma in questa veste, viene usata solo dalla funzione ***proc_init()***.
`void irq_on (unsigned int` ***irq***`);`	Abilita l'interruzione hardware indicata. Questa funzione viene usata solo da ***proc_init()***.
`void irq_off (unsigned int` ***irq***`);`	Disabilita l'interruzione hardware indicata. Questa funzione viene usata solo da ***proc_init()***.

Funzione o macroistruzione	Descrizione

`void _ram_copy (segment_t` ***org_seg*** `,`
`offset_t` ***org_off*** `,`
`segment_t` ***dst_seg*** `,`
`offset_t` ***dst_off*** `,`
`uint16_t` ***size*** `);`
`void ram_copy (` ***org_seg*** `,`
org_off `,`
dst_seg `,`
dst_off `,`
size `);`
	Copia una certa quantità di byte, da una posizione di memoria all'altra, specificando segmento e scostamento di origine e destinazione. Viene usata solo dalle funzioni '**`mem_...()`**'.

Tabella u144.3. Funzioni per l'accesso alla console, dichiarate nel file di intestazione '`kernel/ibm_i86.h`' e descritte nei file contenuti nella directory '`kernel/ibm_i86/`'.

Funzione	Descrizione
`int con_char_read (void);`	Legge un carattere dalla console, se questo è disponibile, altrimenti restituisce il valore zero. Questa funzione viene usata solo da ***proc_sch_terminals()***.
`int con_char_wait (void);`	Legge un carattere dalla console, ma se questo non è ancora disponibile, rimane in attesa, bloccando tutto il sistema operativo. Questa funzione non è utilizzata.

Funzione	Descrizione
`int con_char_ready (void);`	Verifica se è disponibile un carattere dalla console: se è così, restituisce un valore diverso da zero, corrispondente al carattere in attesa di essere prelevato. Questa funzione viene usata solo da ***proc_sch_terminals()***.
`void con_init (void);`	Inizializza la gestione della console. Questa funzione viene usata solo da ***tty_init()***.
`void con_select (int` ***console*** `);`	Seleziona la console desiderata, dove la prima si individua con lo zero. Questa funzione viene usata solo da ***tty_console()***.
`void con_putc (int` ***console*** `,` `int` ***c*** `);`	Visualizza il carattere indicato sullo schermo della console specificata, sulla posizione in cui si trova il cursore, facendolo avanzare di conseguenza e facendo scorrere il testo in alto, se necessario. Questa funzione viene usata solo da ***tty_write()***.
`void con_scroll (int` ***console*** `);`	Fa avanzare in alto il testo della console selezionata. Viene usata internamente, solo dalla funzione ***con_putc()***.

Tabella u144.6. Funzioni per l'accesso ai dischi, dichiarate nel file di intestazione '`kernel/ibm_i86.h`'.

Funzione	Descrizione
`void dsk_setup (void);`	Predispone il contenuto dell'array ***dsk_table[]***. Questa funzione viene usata soltanto da ***main()***.
`int dsk_reset (int` ***drive*** `);`	Azzera lo stato dell'unità corrispondente a ***dsk_table[drive].bios_drive***. Viene usata solo internamente, dalle altre funzioni '**`dsk_...()`**'.

`void dsk_sector_to_chs`
`(int` ***drive*** `,`
`unsigned int` ***sector*** `,`
`dsk_chs_t *`***chs*** `);`
	Modifica le coordinate della variabile strutturata a cui punta l'ultimo parametro, con le coordinate corrispondenti al numero di settore fornito. Viene usata solo internamente, dalle altre funzioni '**`dsk_...()`**'.

Funzione	Descrizione
`int dsk_read_sectors (int` ***drive*** `, unsigned int` ***start_sector*** `, void *`***buffer*** `, unsigned int` ***n_sectors*** `);`	Legge una sequenza di settori da un disco, mettendo i dati in memoria, a partire dalla posizione espressa da un puntatore generico. La funzione è ricorsiva, ma oltre che da se stessa, viene usata internamente da ***dsk_read_bytes()*** e da ***dsk_write_bytes()***.
`int dsk_write_sectors (int` ***drive*** `, unsigned int` ***start_sector*** `, void *`***buffer*** `, unsigned int` ***n_sectors*** `);`	Scrive una sequenza di settori in un disco, traendo i dati da un puntatore a una certa posizione della memoria. La funzione è ricorsiva, ma oltre che da se stessa, viene usata solo internamente da ***dsk_write_bytes()***.
`size_t dsk_read_bytes (int` ***drive*** `, off_t` ***offset*** `, void *`***buffer*** `, size_t count);`	Legge da una certa unità a disco una quantità specificata di byte, a partire dallo scostamento indicato (nel disco), il quale deve essere un valore positivo. Questa funzione viene usata solo da ***dev_dsk()***.

Funzione	Descrizione
`size_t dsk_write_bytes (int` ***drive*****,** `off_t` ***offset*****,** `void *`***buffer*****,** `size_t count);`	Scrive su una certa unità a disco una quantità specificata di byte, a partire dallo scostamento indicato (nel disco), il quale deve essere un valore positivo. Questa funzione viene usata solo da ***dev_dsk()***.

os16: k_libc(9)

«

Il file '`kernel/k_libc.h`' [u0.6] 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.

os16: main(9)

«

Il file '`kernel/main.h`' [u0.7] descrive la funzione ***main()*** del kernel e altre funzioni accessorie, assieme al codice iniziale necessario per mettere in funzione il kernel stesso.

Si rimanda alla sezione u143 che descrive dettagliatamente il codice iniziale del kernel.

os16: memory(9)

Il file ‘kernel/memory.h’ [u0.8] 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 u145. Le tabelle successive che sintetizzano l’uso delle funzioni di questo gruppo, sono tratte da quel capitolo.

Tabella u145.2. Funzioni per la gestione della mappa della memoria, dichiarate nel file di intestazione ‘kernel/memory.h’ e realizzate nella directory ‘kernel/memory/’.

Funzione	Descrizione
`uint16_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/’.

Funzione	Descrizione
`ssize_t mb_alloc (addr_t` ***address*****`,`** `size_t` ***size*****`);`**	Alloca la memoria a partire dall'indirizzo efficace indicato, per la quantità di byte richiesta (zero corrisponde a 10000_{16} byte). L'allocazione ha termine anticipatamente se si incontra un blocco già utilizzato. La funzione restituisce la dimensione allocata effettivamente.
`ssize_t mb_free (addr_t` ***address*****`,`** `size_t` ***size*****`);`**	Libera la memoria a partire dall'indirizzo efficace indicato, per la quantità di byte richiesta (zero corrisponde a 10000_{16} byte). Lo spazio viene liberato in ogni caso, anche se risulta già libero; tuttavia viene prodotto un avvertimento a video se si verifica tale ipotesi.

Funzione	Descrizione
`int mb_alloc_size (size_t` ***size*****`,`** `memory_t *`***allocated***`);`	Cerca e alloca un'area di memoria della dimensione richiesta, modificando la variabile strutturata di cui viene fornito il puntatore come secondo parametro. In pratica, l'indirizzo e l'estensione della memoria allocata effettivamente si trovano nella variabile strutturata in questione, mentre la funzione restituisce zero (se va tutto bene) o −1 se non è disponibile la memoria libera richiesta.

Tabella u145.3. Funzioni per le operazioni di lettura e scrittura in memoria, dichiarate nel file di intestazione '`kernel/memory.h`' e realizzate nella directory '`kernel/memory/`'.

Funzione	Descrizione
`void mem_copy (addr_t` ***orig*****`,`** `addr_t` ***dest*****`,`** `size_t` ***size***`);`	Copia la quantità richiesta di byte, dall'indirizzo di origine a quello di destinazione, espressi in modo efficace.

Funzione	Descrizione

`size_t mem_read (addr_t` ***start***`,`
`void *`***buffer***`,`
`size_t` ***size***`);`
	Legge dalla memoria, a partire dall'indirizzo indicato come primo parametro, la quantità di byte indicata come ultimo parametro. Ciò che viene letto va poi copiato nella memoria tampone corrispondente al puntatore generico indicato come secondo parametro.

`size_t mem_write (addr_t` ***start***`,`
`void *`***buffer***`,`
`size_t` ***size***`);`
	Scrive, in memoria, a partire dall'indirizzo indicato come primo parametro, la quantità di byte indicata come ultimo parametro. Ciò che viene scritto proviene dalla memoria tampone corrispondente al puntatore generico indicato come secondo parametro.

os16: proc(9)

«

Il file '`kernel/proc.h`' [u0.9] 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.

Figura u149.19. Struttura del tipo '**`proc_t`**', corrispondente agli elementi dell'array ***proc_table[]***.

Listato u149.20. Struttura del tipo '**`proc_t`**', corrispondente agli elementi dell'array ***proc_table[]***.

```
typedef struct {
  pid_t              ppid;
  pid_t              pgrp;
  uid_t              uid;
  uid_t              euid;
  uid_t              suid;
  dev_t              device_tty;
  char               path_cwd[PATH_MAX];
  inode_t           *inode_cwd;
  int                umask;
  unsigned long int sig_status;
  unsigned long int sig_ignore;
  clock_t            usage;
  unsigned int       status;
  int                wakeup_events;
  int                wakeup_signal;
  unsigned int       wakeup_timer;
  addr_t             address_i;
  segment_t          segment_i;
  size_t             size_i;
  addr_t             address_d;
  segment_t          segment_d;
  size_t             size_d;
  uint16_t           sp;
  int                ret;
  char               name[PATH_MAX];
  fd_t               fd[FOPEN_MAX];
} proc_t;
```

Tabella u149.21. Membri del tipo '**`proc_t`**'.

Membro	Contenuto
`ppid`	Numero del processo genitore: *parent process id*.
`pgrp`	Numero del gruppo di processi a cui appartiene quello della voce corrispondente: *process group*. Si tratta del numero del processo a partire dal quale viene definito il gruppo.
`uid`	Identità reale del processo della voce corrispondente: *user id*. Si tratta del numero dell'utente, secondo la classificazione del file '`/etc/passwd`', per conto del quale il processo è stato avviato. Tuttavia, i privilegi del processo dipendono dall'identità efficace, definita dal membro '**`euid`**'.
`euid`	Identità efficace del processo della voce corrispondente: *effective user id*. Si tratta del numero dell'utente, secondo la classificazione del file '`/etc/passwd`', per conto del quale il processo è in funzione; pertanto, il processo ha i privilegi di quell'utente.
`suid`	Identità salvata: *saved user id*. Si tratta del valore che aveva ***euid*** prima di cambiare identità.
`device_tty`	Terminale di controllo, espresso attraverso il numero del dispositivo.

Membro	Contenuto

`path_cwd`
 	
`inode_cwd`
	Entrambi i membri rappresentano la directory corrente del processo: nel primo caso in forma di percorso, ovvero di stringa, nel secondo in forma di puntatore a inode rappresentato in memoria.

`umask`	Maschera dei permessi associata al processo: i permessi attivi nella maschera vengono tolti in fase di creazione di un file o di una directory.
`sig_status`	Segnali inviati al processo e non ancora trattati: ogni segnale si associa a un bit differente del valore del membro ***sig_status***; un bit a uno indica che il segnale corrispondente è stato ricevuto e non ancora trattato.
`sig_ignore`	Segnali che il processo ignora: ogni segnale da ignorare si associa a un bit differente del valore del membro ***sig_ignore***; un bit a uno indica che quel segnale va ignorato.
`usage`	Tempo di utilizzo della CPU, da parte del processo, espresso in impulsi del temporizzatore, il quale li produce alla frequenza di circa 18,2 Hz.

Membro	Contenuto
`status`	Stato del processo, rappresentabile attraverso una macro-variabile simbolica, definita nel file '`proc.h`'. Per os16, gli stati possibili sono: «inesistente», quando si tratta di una voce libera della tabella dei processi; «creato», quando un processo è appena stato creato; «pronto», quando un processo è pronto per essere eseguito, «in esecuzione», quando il processo è in funzione; «sleeping», quando un processo è in attesa di qualche evento; «zombie», quando un processo si è concluso, ha liberato la memoria, ma rimangono le sue tracce perché il genitore non ha ancora recepito la sua fine.
`wakeup_events`	Eventi attesi per il risveglio del processo, ammesso che si trovi nello stato si attesa. Ogni tipo di evento che può essere atteso corrisponde a un bit e si rappresenta con una macro-variabile simbolica, dichiarata nel file '`lib/sys/os16.h`'.
`wakeup_signal`	Ammesso che il processo sia in attesa di un segnale, questo membro esprime il numero del segnale atteso.
`wakeup_timer`	Ammesso che il processo sia in attesa dello scadere di un conto alla rovescia, questo membro esprime il numero di secondi che devono ancora trascorrere.

Membro	Contenuto

`address_i`
`segment_i`
`size_i`
	Il valore di questi membri descrive la memoria utilizzata dal processo per le istruzioni (il segmento codice). Le informazioni sono in parte ridondanti, perché conoscendo ***segment_i*** si ottiene facilmente ***address_i*** e viceversa, ma ciò consente di ridurre i calcoli nelle funzioni che ne fanno uso.

`address_d`
`segment_d`
`size_d`
	Il valore di questi membri descrive la memoria utilizzata dal processo per i dati (il segmento usato per le variabili statiche e per la pila). Anche in questo caso, le informazioni sono in parte ridondanti, ma ciò consente di semplificare il codice nelle funzioni che ne fanno uso.

`sp`	Indice della pila dei dati, nell'ambito del segmento dati del processo. Il valore è significativo quando il processo è nello stato di pronto o di attesa di un evento. Quando invece un processo era attivo e viene interrotto, questo valore viene aggiornato.
`ret`	Rappresenta il valore restituito da un processo terminato e passato nello stato di «zombie».
`name`	Il nome del processo, rappresentato dal nome del programma avviato.
`fd`	Tabella dei descrittori dei file relativi al processo.

NOME

‘**isr_1C**’, ‘**isr_80**’ - routine di gestione delle interruzioni

DESCRIZIONE

La routine ‘**isr_1C**’ del file ‘`kernel/proc/_isr.s`’ viene eseguita a ogni impulso del temporizzatore, proveniente dal sistema delle interruzioni hardware; la routine ‘**isr_80**’, in modo analogo, viene eseguita in corrispondenza dell’interruzione software 80_{16} Perché ciò avvenga, nella tabella IVT, nelle voci che riguardano l’interruzione $1C_{16}$ e 80_{16}, si trova l’indirizzo corrispondente alle routine in questione. La configurazione della tabella IVT avviene per mezzo della funzione *ivt_load(9)* [i159.8.2].

La routine ‘**isr_1C**’ prevede il salvataggio dei registri principali nella pila dei dati in funzione al momento dell’interruzione. Quindi vengono modificati i registri che definiscono l’area dati (ES e DS) e successivamente ciò permette di intervenire su delle variabili locali: viene incrementato il contatore degli impulsi del temporizzatore; viene incrementato il contatore dei secondi, se il contatore degli impulsi è divisibile per 18 senza dare resto; vengono salvati l’indice e il segmento della pila dei dati, in due variabili locali.

Dalla verifica del valore del segmento in cui si colloca la pila dei dati del processo interrotto, la routine verifica se si tratta di un processo comune o del kernel. Se si tratta di un processo comune, si scambia la pila con quella del kernel. Per questo la routine si avvale della variabile ***_ksp*** (*kernel stack pointer*), usata anche dalla funzione *proc_scheduler(9)* [i159.8.11]. Sempre se si tratta

dell'interruzione di un processo diverso dal kernel, viene chiamata la funzione ***proc_scheduler()***, già citata, fornendo come argomenti il puntatore alla variabile che contiene l'indice della pila e il puntatore alla variabile che contiene il segmento di memoria che ospita la pila dei dati. Al termine viene scambiata nuovamente la pila dei dati, usando come valori quanto contenuto nelle variabili che prima sono servite per salvare l'indice e il segmento della pila.

Poi, indipendentemente dal tipo di processo, vengono ripristinati i registri accumulati in precedenza nella pila e viene restituito il controllo, concludendo il lavoro dell'interruzione.

Va osservato che la funzione ***proc_scheduler()*** riceve l'indice e il segmento della pila dei dati attraverso dei puntatori a variabili scalari. Pertanto, tale funzione è perfettamente in grado di sostituire questi valori, con quelli della pila di un altro processo. Per questo, quando al ritorno della funzione viene ripristinata la pila sulla base di tali variabili, si ha uno scambio di processi. Il ripristino successivo dalla pila dei registri, completa il procedimento di sostituzione dei processi.

La routine '**`isr_80`**' viene attivata da un'interruzione software, dovuta a una chiamata di sistema. Questa routine si distingue leggermente da '**`isr_1C`**', in quanto non si occupa di tenere conto del tempo trascorso, ma ha la necessità di recuperare dalla pila del processo interrotto, i valori che hanno origine dalla chiamata di sistema. Si tratta sempre del numero della chiamata di sistema, del puntatore al messaggio trasmesso con la chiamata e della sua lunghezza.

Si può osservare anche un'altra differenza importante, per cui, se

l'interruzione riguarda il processo del kernel, l'indice della pila dello stesso viene conservato nella variabile ***_ksp***. Questo fatto è importante, perché prima di abilitare la gestione delle interruzioni, è necessario che il kernel stesso ne provochi una, in modo da poter salvare la prima volta l'indice della propria pila.

Successivamente, indipendentemente dal processo interrotto, si chiama la funzione *sysroutine(9)* [i159.8.28], alla quale si passano come argomenti, oltre che i puntatori all'indice e al segmento della pila dei dati del processo interrotto, anche gli argomenti della chiamata di sistema.

La funzione ***sysroutine()*** si avvale a sua volta della funzione ***proc_scheduler()***, pertanto anche in questo caso la pila dei dati che viene ripristinata successivamente può risultare differente da quella del processo interrotto originariamente, comportando anche in questo caso lo scambio del processo con un altro.

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_table.c`' [i160.9.29]

'`kernel/proc/_isr.s`' [i160.9.1]

VEDERE ANCHE

ivt_load(9) [i159.8.2], *sys(2)* [u0.37], *proc_scheduler(9)* [i159.8.11], *sysroutine(9)* [i159.8.28].

«

NOME

‘**ivt_load**’ - caricamento della tabella IVT

SINTASSI

```
<kernel/proc.h>
void _ivt_load (void);
```

```
<kernel/proc.h>
void  ivt_load (void);
```

DESCRIZIONE

La funzione ***_ivt_load()***, ovvero la macroistruzione corrispondente ***ivt_load()***, modifica la tabella IVT del BIOS, in modo che nella posizione corrispondente all'interruzione $1C_{16}$ ci sia il puntatore alla routine *isr_1C(9)* [i159.8.1], e che in corrispondenza dell'interruzione 80_{16} ci sia il puntatore alla routine *isr_80(9)* [i159.8.1].

Questa funzione viene usata una volta sola, all'interno di *main(9)* [u0.6].

FILE SORGENTI

‘kernel/proc.h’ [u0.9]

‘kernel/proc/_isr.s’ [i160.9.1]

‘kernel/proc/_ivt_load.s’ [i160.9.2]

VEDERE ANCHE

sys(2) [u0.37], *isr_80(9)* [i159.8.1], *proc_scheduler(9)* [i159.8.11], *sysroutine(9)* [i159.8.28].

os16: 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)* [i159.8.6], *proc_sig_chld(9)* [i159.8.12], *proc_sys_wait(9)* [i159.8.27].

FILE SORGENTI

‘`kernel/proc.h`’ [u0.9]

‘`kernel/proc/proc_table.c`’ [i160.9.29]

‘`kernel/proc/proc_available.c`’ [i160.9.3]

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)* [i159.8.6], 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`’ [u0.9]

‘`kernel/proc/proc_sig_core.c`’ [i160.9.14]

VEDERE ANCHE

fd_open(9) [i159.3.8], *fd_write(9)* [i159.3.12], *fd_close(9)* [i159.3.3].

os16: proc_find(9)

«

NOME

‘**`proc_find`**’ - localizzazione di un processo sulla base dell’indirizzo del segmento dati

SINTASSI

```
<kernel/proc.h>
pid_t proc_find (segment_t segment_d);
```

ARGOMENTI

Argomento	Descrizione
`segment_t` ***segment_d***	Indirizzo del segmento da cercare nella tabella dei processi, come allocato per il segmento dati.

DESCRIZIONE

La funzione ***proc_find()*** scandisce la tabella dei processi, alla ricerca di quel processo il cui segmento dati corrisponde al valore fornito come argomento. Ciò serve per sapere chi sia il processo interrotto, del quale si conosce il valore che, prima dell’interruzione, aveva il registro DS (*data segment*).

Questa funzione viene usata da *proc_scheduler(9)* [i159.8.11] e da *sysroutine(9)* [i159.8.28].

VALORE RESTITUITO

La funzione restituisce il numero del processo trovato e non è ammissibile che la ricerca possa fallire. Infatti, se così fosse, si produrrebbe un errore fatale, con avvertimento a video, tale da arrestare il funzionamento del kernel.

FILE SORGENTI

‘`kernel/proc.h`’ [u0.9]

‘`kernel/proc/proc_table.c`’ [i160.9.29]

‘`kernel/proc/proc_find.c`’ [i160.9.5]

os16: proc_init(9)

«

NOME

‘**`proc_init`**’ - inizializzazione della gestione complessiva dei processi elaborativi

SINTASSI

```
<kernel/proc.h>
extern uint16_t _etext;
void proc_init (void);
```

ARGOMENTI

Argomento	Descrizione
`extern uint16_t` ***_etext;***	La variabile ***_etext*** viene fornita dal compilatore e rappresenta l'indirizzo in cui l'area codice si è conclusa. Il valore di ***_etext*** si riferisce a un'area codice che inizia dall'indirizzo zero, pertanto questo dato viene usato per conoscere la dimensione dell'area codice del kernel.

DESCRIZIONE

La funzione ***proc_init()*** viene usata una volta sola, dalla funzione *main(9)* [u0.6], per predisporre la gestione dei processi. Per la precisione svolge le operazioni seguenti:

- carica la tabella IVT, in modo che le interruzioni software $1C_{16}$ e 80_{16} siano dirette correttamente al codice che deve gestirle;
- programma il temporizzatore interno, in modo da produrre una frequenza di circa 18,2 Hz;
- inizializza la tabella dei processi in modo che tutti gli alloggiamenti previsti risultino liberi;
- innesta il file system principale, presupponendo che possa trattarsi soltanto della prima unità a dischetti;
- inizializza correttamente le voci del processo zero, ovvero quelle del kernel, segnando anche come allocata la porzione di memoria utilizzata dal kernel e lo spazio iniziale usato dal BIOS (tabella IVT e BDA);
- abilita le interruzioni hardware del temporizzatore, della tastiera e dell'unità a dischetti: le altre interruzioni hardware rimangono disabilitate.

FILE SORGENTI

‘kernel/proc.h’ [u0.9]

‘kernel/proc/proc_table.c’ [i160.9.29]

‘kernel/proc/proc_init.c’ [i160.9.6]

VEDERE ANCHE

ivt_load(9) [i159.8.2], *proc_available(9)* [i159.8.3], *sb_mount(9)* [i159.3.46].

os16: 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`' [u0.9]

'`kernel/proc/proc_table.c`' [i160.9.29]

'`kernel/proc/proc_reference.c`' [i160.9.7]

os16: 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.

Va comunque osservato che os16 si limita a gestire le azioni predefinite, pertanto si può soltanto attivare o inibire i segnali, salvo i casi in cui questi non possono essere mascherati.

Questa funzione viene usata soltanto da *proc_scheduler(9)* [i159.8.11], ogni volta che ci si prepara allo scambio con un altro processo.

FILE SORGENTI

‘`kernel/proc.h`’ [u0.9]

‘`kernel/proc/proc_scheduler.c`’ [i160.9.11]

‘`kernel/proc/proc_sch_signals.c`’ [i160.9.8]

VEDERE ANCHE

proc_sig_term(9) [i159.8.19], *proc_sig_core(9)* [i159.8.14], *proc_sig_chld(9)* [i159.8.12], *proc_sig_cont(9)* [i159.8.13], *proc_sig_stop(9)* [i159.8.18].

os16: 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 verifica che

effettivamente è stato digitato un carattere, dopo aver determinato a quale terminale virtuale si riferisce, determina se per quel terminale era già stato accumulato un carattere, e se è effettivamente così, sovrascrive quel carattere ma annota anche che l'inserimento precedente è stato perduto.

Successivamente verifica se quel terminale virtuale è associato a un gruppo di processi; se è così e se il carattere corrisponde alla combinazione [*Ctrl c*], invia il segnale SIGINT a tutti i processi di quel gruppo, ma senza poi accumulare il carattere.

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.

Alla fine, scandisce tutti i processi sospesi in attesa di input dal terminale, risvegliandoli (ogni processo deve poi verificare se effettivamente c'è un carattere per sé oppure no, e se non c'è dovrebbe rimettersi in attesa).

Questa funzione viene usata soltanto da *proc_scheduler(9)* [i159.8.11], ogni volta che ci si prepara allo scambio con un altro processo.

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_scheduler.c`' [i160.9.11]

'`kernel/proc/proc_sch_terminals.c`' [i160.9.9]

«

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 os16 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)* [i159.8.11], ogni volta che ci si prepara allo scambio con un altro processo.

FILE SORGENTI

‘`kernel/proc.h`’ [u0.9]

‘`kernel/proc/proc_scheduler.c`’ [i160.9.11]

‘`kernel/proc/proc_sch_timers.c`’ [i160.9.10]

NOME

‘**proc_scheduler**’ - schedulatore

SINTASSI

```
<kernel/proc.h>
void proc_scheduler (uint16_t *sp, segment_t *segment_d);
```

ARGOMENTI

Argomento	Descrizione
`extern uint16_t` ***_ksp*** `;`	L’indice della pila del kernel.
`uint16_t *`***sp***	Puntatore all’indice della pila del processo interrotto.
`segment_t *`***segment_d***	Puntatore al segmento dati del processo interrotto.

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 funzione determina qual è il processo interrotto, scandendo la tabella dei processi alla ricerca di quello il cui segmento dati corrisponde al valore ***segment_d***. Per questo si avvale di *proc_find(9)* [i159.8.5].

Successivamente verifica se ci sono processi in attesa di un evento del temporizzatore o del terminale, inoltre verifica se ci sono processi con segnali in attesa di essere presi in considerazione. per fare questo si avvale di *proc_sch_timers(9)* [i159.8.10], *proc_sch_terminals(9)*

[i159.8.9] e *proc_sch_signals(9)* [i159.8.8], che provvedono a fare ciò che serve in presenza degli eventi di propria competenza.

Si occupa quindi di annotare il tempo di CPU utilizzato dal processo appena sospeso, misurato in unità di tempo a cui si riferisce il tipo '**clock_t**'.

Successivamente scandisce la tabella dei processi alla ricerca di un altro processo da mettere in funzione, al posto di quello sospeso. Se trova un processo pronto per questo lo elegge a processo attivo, declassando quello sospeso a processo pronto ma in attesa, inoltre aggiorna i valori per le variabili ***sp** e ***segment_d**.

Al termine salva nella variabile globale ***_ksp*** il valore dell'indice della pila del kernel, come appare nelle informazioni della tabella dei processi e poi manda il messaggio «EOI» (*end of interrupt* al «PIC 1» (*programmable interrupt controller*).

Questa funzione viene usata dalla routine *isr_1C(9)* [i159.8.1] del file '`kernel/proc/_isr.s`' e dalla funzione *sysroutine(9)* [i159.8.28].

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/proc/proc_scheduler.c`' [i160.9.11]

VEDERE ANCHE

proc_find(9) [i159.8.5], *proc_sch_timers(9)* [i159.8.10], *proc_sch_signals(9)* [i159.8.8], *proc_sch_terminals(9)* [i159.8.9].

«

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)* [i159.8.8].

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_sig_chld.c`' [i160.9.12]

VEDERE ANCHE

proc_sig_status(9) [i159.8.17], *proc_sig_ignore(9)* [i159.8.15], *proc_sig_off(9)* [i159.8.16].

os16: proc_sig_cont(9)

«

NOME

'**`proc_sig_cont`**' - ripresa di un processo sospeso in attesa di qualcosa

SINTASSI

```
<kernel/proc.h>
void proc_sig_cont (pid_t pid, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Numero del processo considerato.
`int` ***sig***	Numero del segnale: deve trattarsi esclusivamente di quanto corrispondente a SIGCONT.

DESCRIZIONE

La funzione ***proc_sig_cont()*** si occupa di verificare che il processo specificato con il parametro ***pid*** abbia ricevuto precedentemente un segnale SIGCONT e che questo non sia stato disabilitato. In tal caso, assegna al processo lo status di «pronto» (‘**`PROC_READY`**’), ammesso che non si trovasse già in questa situazione.

Lo scopo del segnale SIGCONT è quindi quello di far riprendere un processo che in precedenza fosse stato sospeso attraverso un segnale SIGSTOP, SIGTSTP, SIGTTIN oppure SIGTTOU.

Questa funzione viene usata soltanto da *proc_sch_signals(9)* [i159.8.8].

FILE SORGENTI

‘`kernel/proc.h`’ [u0.9]

‘`kernel/proc/proc_sig_cont.c`’ [i160.9.13]

VEDERE ANCHE

proc_sig_status(9) [i159.8.17], *proc_sig_ignore(9)* [i159.8.15], *proc_sig_off(9)* [i159.8.16].

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)* [i159.8.4].

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 os16 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)* [i159.8.8].

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_sig_core.c`' [i160.9.14]

VEDERE ANCHE

proc_sig_status(9) [i159.8.17], *proc_sig_ignore(9)* [i159.8.15], *proc_sig_off(9)* [i159.8.16], *proc_dump_memory(9)* [i159.8.4].

os16: 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)* [i159.8.12], *proc_sig_cont(9)* [i159.8.13], *proc_sig_core(9)* [i159.8.14], *proc_sig_stop(9)* [i159.8.18] e *proc_sig_term(9)* [i159.8.19], 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`’ [u0.9]

‘`kernel/proc/proc_sig_ignore.c`’ [i160.9.15]

VEDERE ANCHE

proc_sig_chld(9) [i159.8.12], *proc_sig_cont(9)* [i159.8.13], *proc_sig_core(9)* [i159.8.14], *proc_sig_stop(9)* [i159.8.18]m *proc_sig_term(9)* [i159.8.19].

os16: 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)*

[i159.8.12], *proc_sig_cont(9)* [i159.8.13], *proc_sig_core(9)* [i159.8.14], *proc_sig_stop(9)* [i159.8.18] e *proc_sig_term(9)* [i159.8.19].

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 *proc_sys_exit(9)* [i159.8.21] e *proc_sys_kill(9)* [i159.8.23].

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_sig_on.c`' [i160.9.17]

'`kernel/proc/proc_sig_off.c`' [i160.9.16]

VEDERE ANCHE

proc_sys_exit(9) [i159.8.21], *proc_sys_kill(9)* [i159.8.23], *proc_sig_chld(9)* [i159.8.12], *proc_sig_cont(9)* [i159.8.13], *proc_sig_core(9)* [i159.8.14], *proc_sig_stop(9)* [i159.8.18], *proc_sig_term(9)* [i159.8.19].

os16: 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)* [i159.8.12], *proc_sig_cont(9)* [i159.8.13], *proc_sig_core(9)* [i159.8.14], *proc_sig_stop(9)* [i159.8.18] e *proc_sig_term(9)* [i159.8.19], 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`’ [u0.9]

‘`kernel/proc/proc_sig_status.c`’ [i160.9.18]

VEDERE ANCHE

proc_sig_chld(9) [i159.8.12], *proc_sig_cont(9)* [i159.8.13], *proc_sig_core(9)* [i159.8.14], *proc_sig_stop(9)* [i159.8.18], *proc_sig_term(9)* [i159.8.19].

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)* [i159.8.8].

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_sig_stop.c`' [i160.9.19]

VEDERE ANCHE

proc_sig_status(9) [i159.8.17], *proc_sig_ignore(9)* [i159.8.15], *proc_sig_off(9)* [i159.8.16].

os16: 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)* [i159.8.8].

FILE SORGENTI

'`kernel/proc.h`' [u0.9]

'`kernel/proc/proc_sig_term.c`' [i160.9.20]

VEDERE ANCHE

proc_sig_status(9) [i159.8.17], *proc_sig_ignore(9)* [i159.8.15], *proc_sig_off(9)* [i159.8.16], *proc_sys_exit(9)* [i159.8.21].

os16: 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 (uint16_t *sp, segment_t *segment_d,
                   pid_t pid, const char *path,
                   unsigned int argc, char *arg_data,
                   unsigned int envc, char *env_data);
```

ARGOMENTI

Argomento	Descrizione
`uint16_t *`***sp***	Puntatore alla variabile contenente l'indice della pila dei dati del processo che ha eseguito la chiamata di sistema *execve(2)* [u0.10].
`segment_t *`***segment_d***	Puntatore alla variabile contenente il valore del segmento dati del processo che ha eseguito la chiamata di sistema *execve(2)* [u0.10].
`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***.

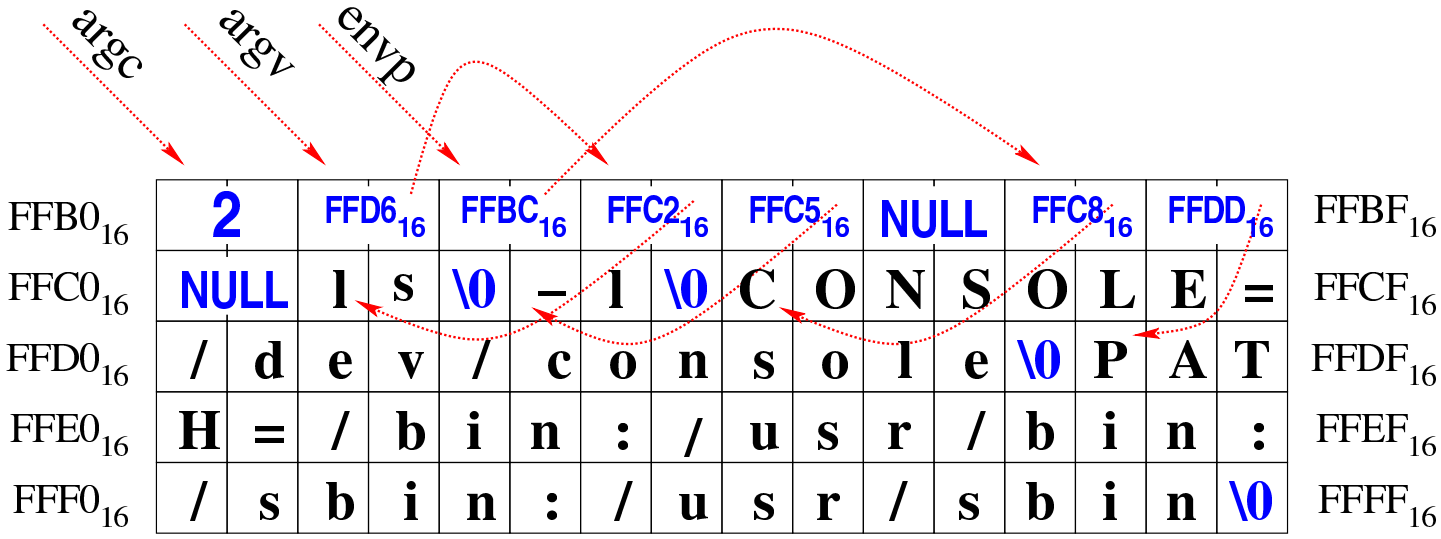
DESCRIZIONE

La funzione ***proc_sys_exec()*** serve a mettere in pratica la chiamata di sistema *execve(2)* [u0.10], 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, dividendo se necessario l'area codice da quella dei dati, 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 u159.145. 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.

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)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_EXEC`**'.

FILE SORGENTI

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

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

‘kernel/proc/proc_sys_exec.c’ [i160.9.21]

VEDERE ANCHE

execve(2) [u0.10], *sys(2)* [u0.37], *isr_80(9)* [i159.8.1], *sysroutine(9)* [i159.8.28], *proc_scheduler(9)* [i159.8.11], *path_inode(9)* [i159.3.36], *inode_check(9)* [i159.3.16], *inode_put(9)* [i159.3.24], *inode_file_read(9)* [i159.3.18], *dev_io(9)* [i159.1.1], *fd_close(9)* [i159.3.3].

os16: proc_sys_exit(9)

«

NOME

‘**proc_sys_exit**’ - chiusura di un processo elaborativo

SINTASSI

```
<kernel/proc.h>
void proc_sys_exit (pid_t pid, int status);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il numero del processo elaborativo da concludere.
`int` ***status***	Il valore di uscita del processo da concludere.

DESCRIZIONE

La funzione ***proc_sys_exit()*** conclude il processo indicato come argomento, chiudendo tutti i descrittori di file che risultano ancora aperti e liberando la memoria. Precisamente compie i passaggi seguenti:

- aggiorna la tabella dei processi indicando per questo lo stato di «zombie» e annotando il valore di uscita;
- chiude i descrittori di file che risultano aperti;
- chiude l'inode della directory corrente;
- se si tratta del processo principale di un gruppo di processi, allora chiude anche il terminale di controllo;
- libera la memoria utilizzata dal processo, verificando comunque che l'area usata per il codice non sia abbinata anche a un altro processo, nel qual caso l'area del codice verrebbe preservata;
- se ci sono dei processi figli di quello che si va a chiudere, questi vengono abbandonati e affidati al processo numero uno ('**`init`**');
- se sono stati abbandonati dei processi, invia il segnale SIGCHLD al processo numero uno ('**`init`**');
- invia al processo genitore il segnale, in modo che possa valutare, eventualmente, il valore di uscita del processo ormai defunto.

Questa funzione viene usata principalmente da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_EXIT`**', e anche dalle funzioni *proc_sig_core(9)* [i159.8.14] e *proc_sig_term(9)* [i159.8.19].

FILE SORGENTI

'`lib/unistd/_exit.c`' [i161.17.1]

'`lib/stdlib/_Exit.c`' [i161.10.1]

'`lib/sys/os16/sys.s`' [i161.12.15]

‘kernel/proc.h’ [u0.9]

‘kernel/proc/_isr.s’ [i160.9.1]

‘kernel/proc/sysroutine.c’ [i160.9.30]

‘kernel/proc/proc_sys_exit.c’ [i160.9.22]

VEDERE ANCHE

_exit(2) [u0.2], *sys(2)* [u0.37], *isr_80(9)* [i159.8.1], *sysroutine(9)* [i159.8.28], *proc_scheduler(9)* [i159.8.11], *proc_sig_core(9)* [i159.8.14], *proc_sig_term(9)* [i159.8.19], *fd_close(9)* [i159.3.3], *inode_put(9)* [i159.3.24], *proc_sig_on(9)* [i159.8.16].

os16: proc_sys_fork(9)

«

NOME

‘**proc_sys_fork**’ - sdoppiamento di un processo elaborativo

SINTASSI

```
<kernel/proc.h>
pid_t proc_sys_fork (pid_t ppid, uint16_t sp);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***ppid***	Il numero del processo che chiede di creare un figlio uguale a se stesso.
`uint16_t` ***sp***	Indice della pila del processo da duplicare.

DESCRIZIONE

La funzione ***proc_sys_fork()*** crea un duplicato del processo chiamante, il quale diventa figlio dello stesso. Precisamente, la funzione compie i passaggi seguenti:

- cerca un alloggiamento libero nella tabella dei processi e procede solo se questo risulta disponibile effettivamente;
- per sicurezza, analizza i processi che risultano essere defunti (zombie) e ne inizializza i valori delle allocazioni in memoria;
- alloca la memoria necessaria a ottenere la copia del processo, tenendo conto che se il processo originario divide l'area codice da quella dei dati, è necessario allocare soltanto lo spazio per l'area dati, in quanto quella del codice può essere condivisa (essendo usata soltanto in lettura);
- compila le informazioni necessarie nella tabella dei processi, relative al nuovo processo da produrre, dichiarandolo come figlio di quello chiamante;
- incrementa il contatore di utilizzo dell'inode che rappresenta la directory corrente, in quanto un nuovo processo la va a utilizzare;
- duplica i descrittori di file già aperti per il processo da duplicare, incrementando di conseguenza il contatore dei riferimenti nella tabella dei file;
- modifica i valori dei registri di segmento nella pila dei dati riferita al processo nuovo, per renderli coerenti con la nuova collocazione in memoria;
- mette il nuovo processo nello stato di pronto, annotandolo così nella tabella dei processi;
- restituisce il numero del nuovo processo: nel processo figlio, invece, non restituisce alcunché.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_FORK`**'.

VALORE RESTITUITO

La funzione restituisce al processo chiamante il numero del processo figlio, mentre il risultato che si ottiene nel processo figlio che si trova a riprendere il funzionamento dallo stesso punto, è semplicemente zero. Ciò consente di distinguere quale sia il processo genitore e quale è invece il figlio. Se la funzione non è in grado di portare a termine il lavoro di duplicazione dei processi, restituisce il valore −1, aggiornando di conseguenza la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`ENOMEM`	Non c'è memoria sufficiente, oppure la tabella dei processi è occupata completamente.

FILE SORGENTI

'`lib/unistd/fork.c`' [i161.17.17]

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/proc/proc_sys_fork.c`' [i160.9.23]

VEDERE ANCHE

fork(2) [u0.14], *sys(2)* [u0.37], *isr_80(9)* [i159.8.1], *sysroutine(9)* [i159.8.28], *proc_scheduler(9)* [i159.8.11], *dev_io(9)* [i159.1.1].

NOME

‘**proc_sys_kill**’ - invio di un segnale a uno o più processi elaborativi

SINTASSI

```
<kernel/proc.h>
int proc_sys_kill (pid_t pid_killer, pid_t pid_target, int sig);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid_killer***	Il numero del processo per conto del quale si invia il segnale.
`pid_t` ***pid_target***	Il numero del processo che dovrebbe ricevere il segnale.
`int` ***sig***	Il numero del segnale da inviare.

DESCRIZIONE

La funzione ***proc_sys_kill()*** invia il segnale ***sig*** al processo numero ***pid_target***, ammesso che il processo ***pid_killer*** abbia i privilegi necessari a farlo. Tuttavia, se il numero ***pid_target*** è zero o −1, si richiede alla funzione l’invio del segnale a un insieme di processi. La tabella successiva descrive i vari casi.

Identità efficace del processo ***pid_killer***	Valore di ***pid_target***	Effetto.
--	< -1	Il valore di ***pid_target*** non è ammissibile: si ottiene un errore.
0	-1	Viene inviato il segnale ***sig*** a tutti i processi con UID ≥ 2 (si esclude il kernel e il processo numero uno, '`init`').
> 0	-1	Viene inviato il segnale ***sig*** a tutti i processi con UID ≥ 1 (si esclude il kernel) la cui identità efficace coincide con quella di ***pid_killer***.
--	0	Viene inviato il segnale ***sig*** a tutti i processi che appartengono allo stesso gruppo di ***pid_target***.
--	> 0	Viene inviato il segnale ***sig*** al processo ***pid_target***, purché l'identità reale o efficace del processo ***pid_killer*** sia uguale all'identità reale o salvata del processo ***pid_target***.

Si osservi che il preteso invio di un segnale pari a zero, ovvero di un segnale nullo, non produce alcun effetto, ma la funzione segnala comunque di avere completato l'operazione con successo.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo

'**SYS_KILL**'.

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
−1	Operazione fallita, con aggiornamento della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`ESRCH`	Il processo ***pid_target*** non esiste, non è un processo che possa ricevere segnali, oppure il valore dato non è interpretabile in alcun modo.
`EPERM`	Il processo ***pid_killer*** non ha i privilegi necessari a inviare il segnale a ***pid_target***.

FILE SORGENTI

'`lib/signal/kill.c`' [i161.8.1]

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/proc/proc_sys_kill.c`' [i160.9.24]

VEDERE ANCHE

kill(2) [u0.22], *sys(2)* [u0.37], *isr_80(9)* [i159.8.1], *sysroutine(9)* [i159.8.28], *proc_scheduler(9)* [i159.8.11], *proc_sig_on(9)* [i159.8.16].

«

NOME

'**proc_sys_seteuid**' - modifica dell'identità efficace

SINTASSI

```
<kernel/proc.h>
int proc_sys_seteuid (pid_t pid, uid_t euid);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Processo su cui intervenire per il cambiamento di identità efficace.
`uid_t` ***euid***	Nuova identità efficace richiesta.

DESCRIZIONE

La funzione ***proc_sys_seteuid()*** modifica l'identità efficace del processo ***pid***, purché si verifichino certe condizioni:

- se il processo ***pid*** è zero, l'identità efficace viene modificata senza altre verifiche;
- se l'identità efficace che ha già il processo coincide con quella nuova richiesta, non viene apportata alcuna modifica (per ovvi motivi);
- se la nuova identità efficace corrisponde all'identità reale del processo, oppure se corrisponde alla sua identità salvata, allora la modifica di quella efficace ha luogo come richiesto;
- in tutti gli altri casi si ottiene un errore.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_SETEUID`**'.

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
−1	Operazione fallita, con aggiornamento della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EPERM`	Il processo ***pid*** non può cambiare l'identità efficace con il valore richiesto.

FILE SORGENTI

'`lib/unistd/seteuid.c`' [i161.17.30]

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/proc/proc_sys_seteuid.c`' [i160.9.25]

VEDERE ANCHE

seteuid(2) [u0.33], *sys(2)* [u0.37], *isr_80(9)* [i159.8.1], *sysroutine(9)* [i159.8.28], *proc_scheduler(9)* [i159.8.11], *proc_sys_setuid(9)* [i159.8.25].

«

NOME

'**proc_sys_setuid**' - modifica dell'identità

SINTASSI

```
<kernel/proc.h>
int proc_sys_setuid (pid_t pid, uid_t uid);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Processo su cui intervenire per il cambiamento di identità.
`uid_t` ***uid***	Nuova identità richiesta.

DESCRIZIONE

La funzione ***proc_sys_setuid()*** modifica l'identità del processo ***pid***, oppure tutti i tipi di identità, a seconda di certe condizioni:

- se l'identità efficace del processo ***pid*** è zero, viene modificata l'identità reale, quella salvata e quella efficace, utilizzando il nuovo valore ***uid***;
- se l'identità efficace del processo coincide già con quella del valore richiesto ***uid***, non viene apportata alcuna modifica e la funzione si conclude con successo;
- se l'identità reale o quella salvato del processo ***pid*** coincide con l'identità richiesta ***uid***, allora viene modificata l'identità efficace del processo con il valore ***uid***;
- in tutti gli altri casi si ottiene un errore.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**`SYS_SETUID`**'.

VALORE RESTITUITO

Valore	Significato
0	Operazione conclusa con successo.
−1	Operazione fallita, con aggiornamento della variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`EPERM`	Il processo ***pid*** non può cambiare identità come richiesto.

FILE SORGENTI

'`lib/unistd/setuid.c`' [i161.17.32]

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/proc/proc_sys_setuid.c`' [i160.9.26]

VEDERE ANCHE

setuid(2) [u0.33], *sys(2)* [u0.37], *isr_80(9)* [i159.8.1], *sysroutine(9)* [i159.8.28], *proc_scheduler(9)* [i159.8.11], *proc_sys_seteuid(9)* [i159.8.24].

«

NOME

‘**proc_sys_signal**’ - modifica della configurazione dei segnali

SINTASSI

```
<kernel/proc.h>
sighandler_t proc_sys_signal (pid_t pid, int sig,
                              sighandler_t handler);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Processo su cui intervenire per il cambiamento di configurazione.
`int` ***sig***	Segnale da riconfigurare.
`sighandler_t` ***handler***	Nuova azione da associare al segnale. Si possono solo usare i valori corrispondenti a ‘**SIG_IGN**’ e ‘**SIG_DFL**’, con cui, rispettivamente, si inibisce il segnale o gli si attribuisce l’azione predefinita.

DESCRIZIONE

La funzione ***proc_sys_signal()*** ha il compito di modificare il comportamento del processo nel caso fosse ricevuto il segnale specificato. Teoricamente, il parametro ***handler*** potrebbe riferirsi a una funzione da eseguire allo scattare del segnale; tuttavia, os16 non è in grado di gestire questa evenienza e per ***handler*** si può specificare soltanto il valore corrispondente all’azione predefinita o a quella di inibizione del segnale.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo '**SYS_SIGNAL**'.

VALORE RESTITUITO

La funzione restituisce il valore di ***handler*** abbinato precedentemente al processo. Se si verifica un errore, restituisce '**SIG_ERR**' e aggiorna la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
EINVAL	La combinazione degli argomenti non è valida.

FILE SORGENTI

'lib/signal/signal.c' [i161.8.2]

'lib/sys/os16/sys.s' [i161.12.15]

'kernel/proc.h' [u0.9]

'kernel/proc/_isr.s' [i160.9.1]

'kernel/proc/sysroutine.c' [i160.9.30]

'kernel/proc/proc_sys_signal.c' [i160.9.27]

VEDERE ANCHE

signal(2) [u0.34], *sys(2)* [u0.37], *isr_80(9)* [i159.8.1], *sysroutine(9)* [i159.8.28], *proc_scheduler(9)* [i159.8.11], *proc_sys_kill(9)* [i159.8.23].

NOME

‘**proc_sys_wait**’ - attesa per la morte di un processo figlio

SINTASSI

```
<kernel/proc.h>
pid_t proc_sys_wait (pid_t pid, int *status);
```

ARGOMENTI

Argomento	Descrizione
`pid_t` ***pid***	Il processo che intende mettersi in attesa della morte di un proprio figlio.
`int *`***status***	Puntatore a una variabile atta a contenere il valore di uscita di un processo figlio defunto.

DESCRIZIONE

La funzione ***proc_sys_wait()*** ha il compito di mettere il processo ***pid*** in pausa, fino alla morte di uno dei propri processi figli.

Per realizzare questo compito, la funzione scandisce inizialmente la tabella dei processi alla ricerca di figli di ***pid***. Se tra questi ne esiste già uno defunto, allora aggiorna ****status*** con il valore di uscita di quello, liberando definitivamente la tabella dei processi dalle tracce di questo figlio. Se invece, pur avendo trovato dei figli, questi risultano ancora tutti in funzione, mette il processo ***pid*** in pausa, in attesa di un segnale SIGCHLD.

Questa funzione viene usata soltanto da *sysroutine(9)* [i159.8.28], in occasione del ricevimento di una chiamata di sistema di tipo ‘**SYS_WAIT**’.

VALORE RESTITUITO

La funzione restituisce il numero PID del processo defunto, se c'è, aggiornando anche ***status** con il valore di uscita dello stesso processo. Se invece il processo ***pid*** è stato messo in attesa, allora restituisce zero, mentre se non ci sono proprio figli di ***pid***, restituisce −1 e aggiorna la variabile ***errno*** del kernel.

ERRORI

Valore di ***errno***	Significato
`ECHILD`	Non ci sono figli del processo ***pid*** e a nulla servirebbe attendere.

FILE SORGENTI

'`lib/sys/wait/wait.c`' [i161.15.1]

'`lib/sys/os16/sys.s`' [i161.12.15]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

'`kernel/proc/proc_sys_wait.c`' [i160.9.28]

VEDERE ANCHE

wait(2) [u0.43], *sys(2)* [u0.37], *isr_80(9)* [i159.8.1], *sysroutine(9)* [i159.8.28], *proc_available(9)* [i159.8.3], *proc_scheduler(9)* [i159.8.11], *proc_sys_fork(9)* [i159.8.22], *proc_sys_kill(9)* [i159.8.23].

«

NOME

‘**sysroutine**’ - attuazione delle chiamate di sistema

SINTASSI

```
<kernel/proc.h>
void sysroutine (uint16_t *sp, segment_t *segment_d,
                 uint16_t syscallnr,
                 uint16_t msg_off, uint16_t msg_size);
```

ARGOMENTI

Argomento	Descrizione
`uint16_t` ***sp***	Puntatore all’indice della pila dei dati del processo che ha emesso la chiamata di sistema.
`segment_t` ***segment_d***	Puntatore al valore del segmento dati del processo che ha emesso la chiamata di sistema.
`uint16_t` ***syscallnr***	Il numero della chiamata di sistema.
`uint16_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 ***segment_d**.
`uint16_t` ***msg_size***	La lunghezza del messaggio della chiamata di sistema.

DESCRIZIONE

La funzione ***sysroutine()*** viene chiamata esclusivamente dalla routine *isr_80(9)* [i159.8.1], a seguito di una chiamata di sistema.

Inizialmente, la funzione individua il processo elaborativo corrispondente a quello che utilizza il segmento dati ****segment_d*** e l'**indirizzo efficace** dell'area di memoria contenente il messaggio della chiamata di sistema, traducendo le informazioni contenute in ***msg_off*** e ****segment_d***.

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 efficace 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 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 anche riscritto a partire dall'indirizzo efficace 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)* [i159.8.11], 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/os16/sys.s`' [i161.12.15]

'`kernel/proc.h`' [u0.9]

'`kernel/proc/_isr.s`' [i160.9.1]

'`kernel/proc/sysroutine.c`' [i160.9.30]

VEDERE ANCHE

sys(2) [u0.37], *isr_80(9)* [i159.8.1], *proc_scheduler(9)* [i159.8.11], *dev_io(9)* [i159.1.1], *path_chdir(9)* [i159.3.30], *path_chmod(9)* [i159.3.31], *path_chown(9)* [i159.3.32], *fd_close(9)* [i159.3.3], *fd_dup(9)* [i159.3.4], *fd_dup2(9)* [i159.3.4], *proc_sys_exec(9)* [i159.8.20], *proc_sys_exit(9)* [i159.8.21], *fd_chmod(9)* [i159.3.1], *fd_chown(9)* [i159.3.2], *fd_fcntl(9)* [i159.3.6], *proc_sys_fork(9)* [i159.8.22], *fd_stat(9)* [i159.3.50], *proc_sys_kill(9)* [i159.8.23], *path_link(9)* [i159.3.38], *fd_lseek(9)* [i159.3.7], *path_mkdir(9)* [i159.3.39], *path_mknod(9)* [i159.3.40], *path_mount(9)* [i159.3.41], *fd_open(9)* [i159.3.8], *fd_read(9)* [i159.3.9], *proc_sys_seteuid(9)* [i159.8.24], *proc_sys_setuid(9)* [i159.8.25], *proc_sys_signal(9)* [i159.8.26], *path_stat(9)* [i159.3.50], *path_umount(9)* [i159.3.41], *path_unlink(9)* [i159.3.44], *proc_sys_wait(9)* [i159.8.27], *fd_write(9)* [i159.3.12].

Il file ‘`kernel/tty.h`’ [u0.10] descrive le funzioni per la gestione dei terminali virtuali.

Per la descrizione dell’organizzazione della gestione dei terminali virtuali di os16, si rimanda alla sezione u146. La tabella successiva che sintetizza l’uso delle funzioni di questo gruppo, è tratta da quel capitolo.

Tabella u146.1. Funzioni per la gestione dei terminali, dichiarate nel file di intestazione ‘`kernel/tty.h`’.

Funzione	Descrizione
`void tty_init (void);`	Inizializza la gestione dei terminali. Viene usata una volta sola nella funzione ***main()*** del kernel.
`tty_t *tty_reference (dev_t` ***device*** `);`	Restituisce il puntatore a un elemento della tabella dei terminali. Se come numero di dispositivo si indica lo zero, si ottiene il riferimento a tutta la tabella; se non viene trovato il numero di dispositivo cercato, si ottiene il puntatore nullo.

Funzione	Descrizione
`dev_t tty_console (dev_t` ***device*** `);`	Seleziona la console indicata attraverso il numero di dispositivo che costituisce l'unico parametro. Se viene dato un valore a zero, si ottiene solo di conoscere qual è la console attiva. La console selezionata viene anche memorizzata in una variabile statica, per le chiamate successive della funzione. Se viene indicato un numero di dispositivo non valido, si seleziona implicitamente la prima console.
`int tty_read (dev_t` ***device*** `);`	Legge un carattere dal terminale specificato attraverso il numero di dispositivo. Per la precisione, il carattere viene tratto dal campo relativo contenuto nella tabella dei terminali. Il carattere viene restituito dalla funzione come valore intero comune; se si ottiene zero significa che non è disponibile alcun carattere.

Funzione	Descrizione
`void tty_write (dev_t` ***device*****,** `int` ***c*****);**	Scrive sullo schermo del terminale rappresentato dal numero di dispositivo, il carattere fornito come secondo parametro.