nucleo

Generato da Doxygen 1.9.1

Capitolo 1

Pagina Principale

Dispensa: https://calcolatori.iet.unipi.it/resources/nucleo.pdf

Capitolo 2

Indice dei moduli

2.1 Moduli

Questo è l'elenco di tutti i moduli:

Capitolo 3

Indice delle strutture dati

3.1 Strutture dati

Queste sono le strutture dati con una loro breve descrizione:

Capitolo 4

Indice dei file

4.1 Elenco dei file

Questo è un elenco dei file documentati con una loro breve descrizione:

Capitolo 5

Documentazione dei moduli

5.1 Modulo I/O

Diagramma di collaborazione per Modulo I/O:

Moduli

- Memoria Dinamica
- Console
- Interfacce ATA
- Inizializzazione
- Gestione errori

5.1.1 Descrizione dettagliata

Dispensa: https://calcolatori.iet.unipi.it/resources/modulo-io.pdf

5.2 Memoria Dinamica

Diagramma di collaborazione per Memoria Dinamica:

Funzioni

- void * operator new (size_t s)

  Alloca un oggetto nello heap I/O.
- void * operator new (size_t s, std::align_val_t a)

  Alloca un oggetto nello heap I/O, con vincoli di allineamento.
- void operator delete (void *p)

  Dealloca un oggetto restituendolo all'heap I/O.

Variabili

- natl ioheap_mutex

  Indice del semaforo di mutua esclusione per lo heap I/O.

5.2.1 Descrizione dettagliata

Dal momento che le funzioni del modulo I/O sono eseguite con le interruzioni esterne mascherabili abilitate, dobbiamo proteggere lo heap I/O con un semaforo di mutua esclusione.

5.2.2 Documentazione delle funzioni

5.2.2.1 operator delete()

```
void operator delete (
            void * p )
```

Dealloca un oggetto restituendolo all'heap I/O.

Parametri

p	puntatore all'oggetto

Definizione alla linea 64 del file io.cpp.

5.2.2.2 operator new() [1/2]

```
void* operator new (
            size_t s )
```

Alloca un oggetto nello heap I/O.

Parametri

s	dimensione dell'oggetto

Restituisce

puntatore all'oggetto (nullptr se heap esaurito)

Definizione alla linea 35 del file io.cpp.

5.2.2.3 operator new() [2/2]

```
void* operator new (
            size_t s,
            std::align_val_t a )
```

Alloca un oggetto nello heap I/O, con vincoli di allineamento.

Parametri

s	dimensione dell'oggetto
a	allineamento richiesto

Restituisce

puntatore all'oggetto (nullptr se heap esaurito)

Definizione alla linea 51 del file io.cpp.

5.3 Console

Diagramma di collaborazione per Console:

Strutture dati

- struct des_console

  Descrittore della console.

Funzioni

- void c_writeconsole (const char *buff, natq quanti)

  Parte C+++ della primitiva writeconsole()
- void startkbd_in (des_console *d, char *buff, natq dim)

  Avvia una operazione di lettura dalla tastiera.
- natq c_readconsole (char *buff, natq quanti)

  Parte C++ della primitiva readconsole()
- void estern_kbd (int)

  Processo esterno associato alla tastiera.
- void c_iniconsole (natb cc)

  Parte C++ della primitiva iniconsole()
- bool kbd_init ()

  Inizializza la tastiera.
- bool vid_init ()

  Inizializza il video (modalità testo)
- bool console_init ()

  Inizializza la console (tastiera + video)

Variabili

- des_console console

  Unica istanza di des_console.
- const int KBD_IRQ = 1

  Piedino dell'APIC per le richieste di interruzione della tastiera.

5.3.1 Descrizione dettagliata

Per *console* intendiamo l'unione della tastiera e del video (modalità testo).

5.3.2 Documentazione delle funzioni

5.3.2.1 c_iniconsole()

```
void c_iniconsole (
            natb cc )
```

Parte C++ della primitiva iniconsole()

Parametri

cc	Attributo colore per il video

Definizione alla linea 216 del file io.cpp.

5.3.2.2 c_readconsole()

```
natq c_readconsole (
            char * buff,
            natq quanti )
```

Parte C++ della primitiva readconsole()

Parametri

buff	buffer che deve ricevere i caratteri letti
quanti	dimensione di *buff*

Restituisce

numero di caratteri effettivamente letti

Definizione alla linea 141 del file io.cpp.

5.3.2.3 c_writeconsole()

```
void c_writeconsole (
            const char * buff,
            natq quanti )
```

Parte C+++ della primitiva writeconsole()

Parametri

buff	buffer contenente i caratteri da scrivere
quanti	numero di caratteri da scrivere

Definizione alla linea 101 del file io.cpp.

5.3.2.4 console_init()

```
bool console_init ( )
```

Inizializza la console (tastiera + video)

Restituisce

true in caso di successo, false altrimenti

Definizione alla linea 256 del file io.cpp.

5.3.2.5 kbd_init()

```
bool kbd_init ( )
```

Inizializza la tastiera.

Restituisce

true in caso di successo, false altrimenti

Definizione alla linea 227 del file io.cpp.

5.3.2.6 startkbd_in()

```
void startkbd_in (
            des_console * d,
            char * buff,
            natq dim )
```

Avvia una operazione di lettura dalla tastiera.

Parametri

d	descrittore della console
buff	buffer che deve ricevere i caratteri
dim	dimensione di *buff*

Definizione alla linea 128 del file io.cpp.

5.3.2.7 vid_init()

```
bool vid_init ( )
```

Inizializza il video (modalità testo)

Restituisce

true in caso di successo, false altrimenti

Definizione alla linea 246 del file io.cpp.

5.4 Interfacce ATA

Diagramma di collaborazione per Interfacce ATA:

Strutture dati

- struct des_ata
  Descrittore di interfaccia ATA.

Funzioni

- bool prepare_prd (natb *vett, natb quanti)
  Prepara i descrittori per il Bus Mastering.
- void starthd_in (des_ata *d, natb vetti[ ], natl primo, natb quanti)
  Avvia una operazione di ingresso dall'hard disk.
- void c_readhd_n (natb vetti[ ], natl primo, natb quanti)
  Parte C++ della primitiva readhd_n().
- void starthd_out (des_ata *d, natb vetto[ ], natl primo, natb quanti)
  Avvia una operazione di uscita verso l'hard disk.
- void c_writehd_n (natb vetto[ ], natl primo, natb quanti)
  Parte C++ della primitiva writehd_n().
- void dmastarthd_in (des_ata *d, natb vetti[ ], natl primo, natb quanti)
  Avvia una operazione di ingresso in DMA dall'hard disk.
- void c_dmareadhd_n (natb vetti[ ], natl primo, natb quanti)
  Parte C++ della primitiva dmareadhd_n().
- void dmastarthd_out (des_ata *d, natb vetto[ ], natl primo, natb quanti)
  Avvia una operazione di uscita in DMA verso l'hard disk.
- void c_dmawritehd_n (natb vetto[ ], natl primo, natb quanti)
  Parte C++ della primitiva dmawritehd_n().
- void estern_hd (int)
  Processo esterno per le richieste di interruzione dell'hard disk.
- bool hd_init ()
  Inizializza la gestione dell'hard disk.

Variabili

- des_ata hard_disk

  Descrittore dell'unico hard disk installato nel sistema.
- natl hd_prd [MAX_PRD ∗2]

  Array dei descrittori per il Bus Mastering.
- const natb HD_IRQ = 14

  Piedino dell'APIC per le richieste di interruzione dell'hard disk.

5.4.1 Descrizione dettagliata

5.4.2 Documentazione delle funzioni

5.4.2.1 c_dmareadhd_n()

```
void c_dmareadhd_n (
            natb vetti[],
            natl primo,
            natb quanti )
```

Parte C++ della primitiva dmareadhd_n().

Parametri

vetti	buffer che dovrà ricevere i settori letti
primo	LBA del primo settore da leggere
quanti	numero di settori da leggere

Definizione alla linea 434 del file io.cpp.

5.4.2.2 c_dmawritehd_n()

```
void c_dmawritehd_n (
            natb vetto[],
            natl primo,
            natb quanti )
```

Parte C++ della primitiva dmawritehd_n().

Parametri

vetto	buffer che contiene i settori da scrivere
primo	LBA del primo settore da scrivere
quanti	numero di settori da scrivere

Definizione alla linea 484 del file io.cpp.

5.4.2.3 c_readhd_n()

```
void c_readhd_n (
            natb vetti[],
            natl primo,
            natb quanti )
```

Parte C++ della primitiva readhd_n().

Parametri

vetti	buffer che dovrà ricevere i settori letti
primo	LBA del primo settore da leggere
quanti	numero di settori da leggere

Definizione alla linea 351 del file io.cpp.

5.4.2.4 c_writehd_n()

```
void c_writehd_n (
            natb vetto[],
            natl primo,
            natb quanti )
```

Parte C++ della primitiva writehd_n().

Parametri

vetto	buffer che contiene i settori da scrivere
primo	LBA del primo settore da scrivere
quanti	numero di settori da scrivere

Definizione alla linea 389 del file io.cpp.

5.4.2.5 dmastarthd_in()

```
void dmastarthd_in (
            des_ata * d,
            natb vetti[],
            natl primo,
            natb quanti )
```

Avvia una operazione di ingresso in DMA dall'hard disk.

Parametri

d	descrittore dell'hard disk
vetti	buffer che dovrà ricevere i settori letti
primo	LBA del primo settore da leggere
quanti	numero di settori da leggere

Definizione alla linea 413 del file io.cpp.

5.4.2.6 dmastarthd_out()

```
void dmastarthd_out (
             des_ata * d,
             natb vetto[],
             natl primo,
             natb quanti )
```

Avvia una operazione di uscita in DMA verso l'hard disk.

Parametri

d	descrittore dell'hard disk
vetto	buffer che contiene i settori da scrivere
primo	LBA del primo settore da scrivere
quanti	numero di settori da scrivere

Definizione alla linea 463 del file io.cpp.

5.4.2.7 hd_init()

```
bool hd_init ( )
```

Inizializza la gestione dell'hard disk.

Restituisce

true in caso di successo, false altrimenti

Definizione alla linea 539 del file io.cpp.

5.4.2.8 prepare_prd()

```
bool prepare_prd (
             natb * vett,
             natb quanti )
```

Prepara i descrittori per il Bus Mastering.

Parametri

vett	buffer coinvolto nel trasferimento DMA
quanti	numero di settori da trasferire

Restituisce

false se i PRD non sono sufficienti per trasferire tutti i byte richiesti, true altrimenti

Definizione alla linea 309 del file io.cpp.

5.4.2.9 starthd_in()

```
void starthd_in (
            des_ata * d,
            natb vetti[],
            natl primo,
            natb quanti )
```

Avvia una operazione di ingresso dall'hard disk.

Parametri

d	descrittore dell'hard disk
vetti	buffer che dovrà ricevere i settori letti
primo	LBA del primo settore da leggere
quanti	numero di settori da leggere

Definizione alla linea 338 del file io.cpp.

5.4.2.10 starthd_out()

```
void starthd_out (
            des_ata * d,
            natb vetto[],
            natl primo,
            natb quanti )
```

Avvia una operazione di uscita verso l'hard disk.

Parametri

d	descrittore dell'hard disk
vetto	buffer che contiene i settori da scrivere
primo	LBA del primo settore da scrivere
quanti	numero di settori da scrivere

Definizione alla linea 375 del file io.cpp.

5.5 Inizializzazione

Diagramma di collaborazione per Inizializzazione:

Funzioni

- bool fill_io_gates ()

  Riempie i gate della IDT relativi alle primitive fornite dal modulo I/O.
- void main (natq sem_io)

  Corpo del processo main I/O.

Variabili

- char end [ ]

  Ultimo indirizzo utilizzato dal modulo I/O (fornito dal collegatore)

5.5.1 Descrizione dettagliata

5.5.2 Documentazione delle funzioni

5.5.2.1 fill_io_gates()

```
bool fill_io_gates ( )
```

Riempie i gate della IDT relativi alle primitive fornite dal modulo I/O.

Restituisce

true in caso di successo, false altrimenti

5.5.2.2 main()

```
void main (
            natq sem_io )
```

Corpo del processo main I/O.

Il modulo sistema crea il processo main I/O all'avvio e gli cede il controllo, passandogli l'indice di un semaforo di sincronizzazione. Il modulo I/O deve eseguire una sem_signal() su questo semaforo quando ha terminato la fase di inizializzazione.

Parametri

sem↩ _io	indice del semaforo di sincronizzazione

Definizione alla linea 598 del file io.cpp.

5.6 Gestione errori

Diagramma di collaborazione per Gestione errori:

Funzioni

- void panic (const char ∗msg)

  Segnala un errore fatale nel modulo I/O.
- natq c_getiomeminfo ()

  Parte C++ della primitiva getiomeminfo()

5.6.1 Descrizione dettagliata

5.6.2 Documentazione delle funzioni

5.6.2.1 c_getiomeminfo()

```
natq c_getiomeminfo ( )
```

Parte C++ della primitiva getiomeminfo()

Restituisce

byte liberi nello heap I/O

Definizione alla linea 638 del file io.cpp.

5.6.2.2 panic()

```
void panic (
            const char * msg )
```

Segnala un errore fatale nel modulo I/O.

Parametri

msg	messaggio da inviare al log (severità LOG_ERR)

Definizione alla linea 629 del file io.cpp.

5.7 Modulo sistema

Diagramma di collaborazione per Modulo sistema:

Moduli

- Memoria Dinamica
- Processi
- Semafori
- Timer
- Eccezioni
- Frame
- Paginazione
- Inizializzazione
- Gestione errori

5.7.1 Descrizione dettagliata

5.8 Memoria Dinamica

Diagramma di collaborazione per Memoria Dinamica:

Funzioni

- void * operator new (size_t size)

  Alloca nello heap.
- void * operator new (size_t size, std::align_val_t align)

  Alloca nello heap con allienamento.
- void operator delete (void *p)

  Dealloca un oggetto restituendolo allo heap.

5.8.1 Descrizione dettagliata

Per la gestione della memoria dinamica del modulo sistema usiamo le fuzioni fornite da libce e ci limitiamo a ridefinire gli operatori new/delete in modo che le usino. La zona di memoria usata come heap per il modulo sistema è definita durante l'inizializzazione (si vedano la chiamata a heap_init() in main() e in main_sistema()).

5.8.2 Documentazione delle funzioni

5.8.2.1 operator delete()

```
void operator delete (
             void * p )
```

Dealloca un oggetto restituendolo allo heap.

Parametri

p	puntatore all'oggetto da deallocare

Definizione alla linea 45 del file sistema.cpp.

5.8.2.2 operator new() [1/2]

```
void* operator new (
            size_t size )
```

Alloca nello heap.

Parametri

size	dimensione dell'oggetto da allocare

Restituisce

puntatore all'oggetto (nullptr se heap pieno)

Definizione alla linea 27 del file sistema.cpp.

5.8.2.3 operator new() [2/2]

```
void* operator new (
            size_t size,
            std::align_val_t align )
```

Alloca nello heap con allienamento.

Parametri

size	dimensione dell'oggetto da allocare
align	allineamento richiesto

Restituisce

indirizzo dell'oggetto (nullptr se heap pieno)

Definizione alla linea 37 del file sistema.cpp.

5.9 Processi

Diagramma di collaborazione per Processi:

Moduli

- Creazione e distruzione dei processi

Strutture dati

- struct des_proc

  Descrittore di processo.

Tipi enumerati (enum)

- enum {
  I_RAX , **I_RCX** , **I_RDX** , **I_RBX** ,
  I_RSP , **I_RBP** , **I_RSI** , **I_RDI** ,
  I_R8 , **I_R9** , **I_R10** , **I_R11** ,
  I_R12 , **I_R13** , **I_R14** , **I_R15** }

  Indici delle copie dei registri nell'array contesto.

Funzioni

- void inserimento_lista (des_proc ∗&p_lista, des_proc ∗p_elem)

  Inserimento in lista ordinato (per priorità)
- des_proc ∗ rimozione_lista (des_proc ∗&p_lista)

  Estrazione del processo a maggiore priorità
- void inspronti ()

  Inserisce esecuzione in testa alla lista pronti.
- void schedulatore (void)

  Sceglie il prossimo processo da mettere in esecuzione.
- des_proc ∗ des_p (natw id)

  Trova il descrittore di processo dato l'id.
- void dummy (natq)

  Corpo del processo dummy.

Variabili

- const natl DUMMY_PRIORITY = 0

  Priorità del processo dummy.
- const int N_REG = 16

  Numero di registri nel campo contesto del descrittore di processo.
- des_proc ∗ proc_table [MAX_PROC]

  Tabella che associa l'id di un processo al corrispondente des_proc.
- natl processi

  Numero di processi utente attivi.
- des_proc ∗ esecuzione

  Coda esecuzione (contiene sempre un solo elemento)
- des_proc ∗ pronti

  Coda pronti (vuota solo quando dummy è in esecuzione)

Funzioni usate dal processo dummy

- void end_program ()

  Esegue lo shutdown del sistema.
- void halt ()

  Esegue l'istruzione `hlt`.

5.9.1 Descrizione dettagliata

Dispensa: https://calcolatori.iet.unipi.it/resources/processi.pdf

5.9.2 Documentazione delle funzioni

5.9.2.1 des_p()

```
des_proc* des_p (
            natw id )
```

Trova il descrittore di processo dato l'id.

Errore fatale se *id* non corrisponde ad alcun processo.

Parametri

id	id del processo

Restituisce

descrittore di processo corrispondente

Definizione alla linea 192 del file sistema.cpp.

5.9.2.2 halt()

```
void halt ( )
```

Esegue l'istruzione `hlt`.

Mette in pausa il processore in attesa di una richiesta di interruzione esterna

5.9.2.3 inserimento_lista()

```
void inserimento_lista (
            des_proc *& p_lista,
            des_proc * p_elem )
```

Inserimento in lista ordinato (per priorità)

Parametri

p_lista	lista in cui inserire
p_elem	elemento da inserire

Nota

a parità di priorità favorisce i processi già in coda

Definizione alla linea 126 del file sistema.cpp.

5.9.2.4 rimozione_lista()

```
des_proc* rimozione_lista (
            des_proc *& p_lista )
```

Estrazione del processo a maggiore priorità

Parametri

p_lista	lista da cui estrarre

Restituisce

processo a più alta priorità

Definizione alla linea 152 del file sistema.cpp.

5.9.2.5 schedulatore()

```
void schedulatore (
            void  )
```

Sceglie il prossimo processo da mettere in esecuzione.

Nota

Modifica solo la variabile esecuzione. Il processo andrà effettivamente in esecuzione solo alla prossima `call carica_stato; iretq`

Definizione alla linea 178 del file sistema.cpp.

5.9.3 Documentazione delle variabili

5.9.3.1 proc_table

```
des_proc* proc_table[MAX_PROC]
```

Tabella che associa l'id di un processo al corrispondente des_proc.

I des_proc sono allocati dinamicamente nello heap del sistema (si veda crea_processo).

Definizione alla linea 97 del file sistema.cpp.

5.10 Semafori

Diagramma di collaborazione per Semafori:

Strutture dati

- struct des_sem

  Descrittore di semaforo.

Funzioni

- int liv_chiamante ()

  Restituisce il livello a cui si trovava il processore al momento in cui è stata invocata la primitiva.
- natl alloca_sem ()

  Alloca un nuovo semaforo.
- bool sem_valido (natl sem)

  Verifica un id di semaforo.
- void c_sem_ini (int val)

  Parte C++ della primitiva sem_ini().
- void c_sem_wait (natl sem)

  Parte C++ della primitiva sem_wait().
- void c_sem_signal (natl sem)

  Parte C++ della primitiva sem_signal().

Variabili

- des_sem array_dess [MAX_SEM ∗2]

  Array dei descrittori di semaforo.
- natl sem_allocati_utente = 0

  Numero di semafori allocati per il livello utente.
- natl sem_allocati_sistema = 0

  Numero di semafori allocati per il livello sistema (moduli sistema e I/O)

5.10.1 Descrizione dettagliata

Dispensa: https://calcolatori.iet.unipi.it/resources/semafori.pdf

5.10.2 Documentazione delle funzioni

5.10.2.1 alloca_sem()

```
natl alloca_sem ( )
```

Alloca un nuovo semaforo.

Restituisce

id del nuovo semaforo (0xFFFFFFFF se esauriti)

Definizione alla linea 275 del file sistema.cpp.

5.10.2.2 c_sem_ini()

```
void c_sem_ini (
            int val )
```

Parte C++ della primitiva sem_ini().

Parametri

val	numero di gettoni iniziali

Definizione alla linea 317 del file sistema.cpp.

5.10.2.3 c_sem_signal()

```
void c_sem_signal (
            natl sem )
```

Parte C++ della primitiva sem_signal().

Parametri

sem	id di semaforo

Definizione alla linea 351 del file sistema.cpp.

5.10.2.4 c_sem_wait()

```
void c_sem_wait (
            natl sem )
```

Parte C++ della primitiva sem_wait().

Parametri

sem	id di semaforo

Definizione alla linea 330 del file sistema.cpp.

5.10.2.5 liv_chiamante()

```
int liv_chiamante ( )
```

Restituisce il livello a cui si trovava il processore al momento in cui è stata invocata la primitiva.

Avvertimento

funziona solo nelle routine di risposta ad una interruzione (INT, esterna o eccezione) se è stata chiamata `salva_stato`.

Definizione alla linea 255 del file sistema.cpp.

5.10.2.6 sem_valido()

```
bool sem_valido (
            natl sem )
```

Verifica un id di semaforo.

Parametri

sem	id da verificare

Restituisce

true se sem è l'id di un semaforo allocato; false altrimenti

Definizione alla linea 303 del file sistema.cpp.

5.10.3 Documentazione delle variabili

5.10.3.1 array_dess

```
des_sem array_dess[MAX_SEM *2]
```

Array dei descrittori di semaforo.

I primi MAX_SEM semafori di array_dess sono per il livello utente, gli altri MAX_SEM sono per il livello sistema.

Definizione alla linea 247 del file sistema.cpp.

5.11 Timer

Diagramma di collaborazione per Timer:

Strutture dati

- struct richiesta

  Richiesta al timer.

Funzioni

- void inserimento_lista_attesa (richiesta *p)

  Inserisce un processo nella coda delle richieste al timer.
- void c_delay (natl n)

  Parte C++ della primitiva delay.
- void c_driver_td (void)

  Driver del timer.

Variabili

- richiesta * sospesi

  Coda dei processi sospesi.

5.11.1 Descrizione dettagliata

Dispensa: https://calcolatori.iet.unipi.it/resources/delay.pdf

5.11.2 Documentazione delle funzioni

5.11.2.1 c_delay()

```
void c_delay (
            natl n )
```

Parte C++ della primitiva delay.

Parametri

n	numero di intervalli di tempo

Definizione alla linea 421 del file sistema.cpp.

5.11.2.2 inserimento_lista_attesa()

```
void inserimento_lista_attesa (
            richiesta * p )
```

Inserisce un processo nella coda delle richieste al timer.

Parametri

p	richiesta da inserire

Definizione alla linea 395 del file sistema.cpp.

5.12 Eccezioni

Diagramma di collaborazione per Eccezioni:

Funzioni

- void gestore_eccezioni (int tipo, natq errore, vaddr rip)

  Gestore generico di eccezioni.

Variabili

- natb start [ ]

  Primo indirizzo del codice di sistema.
- natb end [ ]

  Ultimo indirizzo del codice sistema (fornito dal collegatore)

5.12.1 Descrizione dettagliata

5.12.2 Documentazione delle funzioni

5.12.2.1 gestore_eccezioni()

```
void gestore_eccezioni (
            int tipo,
            natq errore,
            vaddr rip )
```

Gestore generico di eccezioni.

Funzione chiamata da tutti i gestori di eccezioni in sistema.s, tranne quello per il Non-Maskable-Interrupt.

Parametri

tipo	tipo dell'eccezione
errore	eventuale codice di errore aggiuntivo
rip	istruction pointer salvato in pila

Definizione alla linea 480 del file sistema.cpp.

5.13 Frame

Diagramma di collaborazione per Frame:

Strutture dati

- struct des_frame
  Descrittore di frame.

Funzioni

- void init_frame ()
  Inizializza la parte M2 e i descrittori di frame.
- paddr alloca_frame ()
  Estrae un frame dalla lista dei frame liberi.
- void rilascia_frame (paddr f)
  Restiuisce un frame alla listsa dei frame liberi.

Variabili

- natq const N_FRAME = MEM_TOT / DIM_PAGINA
  Numero totale di frame (M1 + M2)
- natq N_M1
  Numero di frame in M1.
- natq N_M2
  Numero di frame in M2.
- des_frame vdf [N_FRAME]
  Array dei descrittori di frame.
- natq primo_frame_libero
  Testa della lista dei frame liberi.
- natq num_frame_liberi
  Numero di frame nella lista dei frame liberi.

5.13.1 Descrizione dettagliata

5.13.2 Documentazione delle funzioni

5.13.2.1 alloca_frame()

```
paddr alloca_frame ( )
```

Estrae un frame dalla lista dei frame liberi.

Restituisce

indirizzo fisico del frame estratto, o 0 se la lista è vuota

Definizione alla linea 597 del file sistema.cpp.

5.13.2.2 init_frame()

```
void init_frame ( )
```

Inizializza la parte M2 e i descrittori di frame.

Funzione chiamata in fase di inizializzazione.

Definizione alla linea 550 del file sistema.cpp.

5.13.2.3 rilascia_frame()

```
void rilascia_frame (
            paddr f )
```

Restiuisce un frame alla listsa dei frame liberi.

Parametri

f	indirizzo fisico del frame da restituire

Definizione alla linea 613 del file sistema.cpp.

5.14 Paginazione

Diagramma di collaborazione per Paginazione:

Moduli

- Parti della memoria virtuale dei processi
- Funzioni necessarie per map() e unmap()

Funzioni

- bool in_utn_c (vaddr v)

  Controlla che un indirizzo appartenga alla zona utente/condivisa.
- bool c_access (vaddr begin, natq dim, bool writeable, bool shared=true)

  Parte C++ della primitiva access()
- void c_trasforma (vaddr ind_virt)

  Parte C++ della primitiva trasforma()

5.14.1 Descrizione dettagliata

Dispensa: https://calcolatori.iet.unipi.it/resources/paginazione-implementazione.pdf

5.14.2 Documentazione delle funzioni

5.14.2.1 c_access()

```
bool c_access (
              vaddr begin,
              natq dim,
              bool writeable,
              bool shared = true )
```

Parte C++ della primitiva access()

Primitiva utilizzata dal modulo I/O per controllare che i buffer passati dal livello utente siano accessibili dal livello utente (problema del Cavallo di Troia) e non possano causare page fault nel modulo I/O (bit P tutti a 1 e scrittura permessa quando necessario).

Parametri

begin	base dell'intervallo da controllare
dim	dimensione dell'intervallo da controllare
writeable	se true, l'intervallo deve essere anche scrivibile
shared	se true, l'intevallo deve trovarsi in utente/condivisa

Restituisce

true se i vincoli sono rispettati, false altrimenti

Definizione alla linea 751 del file sistema.cpp.

5.14.2.2 c_trasforma()

```
void c_trasforma (
            vaddr ind_virt )
```

Parte C++ della primitiva trasforma()

Traduce *ind_virt* usando il TRIE del processo puntato da esecuzione.

Parametri

ind_virt	indirizzo virtuale da tradurre

Definizione alla linea 781 del file sistema.cpp.

5.14.2.3 in_utn_c()

```
bool in_utn_c (
            vaddr v )
```

Controlla che un indirizzo appartenga alla zona utente/condivisa.

Parametri

v	indirizzo virtuale da controllare

Restituisce

true sse *v* appartiene alla parte utente/condivisa, false altrimenti

Definizione alla linea 733 del file sistema.cpp.

5.15 Parti della memoria virtuale dei processi

Diagramma di collaborazione per Parti della memoria virtuale dei processi:

Variabili

- static const natq PART_SIZE = dim_region(MAX_LIV - 1)

  Granularità delle parti della memoria virtuale.

- const vaddr ini_sis_c = norm(I_SIS_C * PART_SIZE)

  base di sistema/condivisa
- const vaddr ini_sis_p = norm(I_SIS_P * PART_SIZE)

  base di sistema/privata
- const vaddr ini_mio_c = norm(I_MIO_C * PART_SIZE)

  base di modulo IO/condivisa
- const vaddr ini_utn_c = norm(I_UTN_C * PART_SIZE)

  base di utente/condivisa
- const vaddr ini_utn_p = norm(I_UTN_P * PART_SIZE)

  base di utente/privata
- const vaddr fin_sis_c = ini_sis_c + PART_SIZE * N_SIS_C

  limite di sistema/condivisa
- const vaddr fin_sis_p = ini_sis_p + PART_SIZE * N_SIS_P

  limite di sistema/privata
- const vaddr fin_mio_c = ini_mio_c + PART_SIZE * N_MIO_C

  limite di modulo IO/condivisa
- const vaddr fin_utn_c = ini_utn_c + PART_SIZE * N_UTN_C

  limite di utente/condivisa
- const vaddr fin_utn_p = ini_utn_p + PART_SIZE * N_UTN_P

  limite di utente/privata

5.15.1 Descrizione dettagliata

Le parti hanno dimensioni multiple della dimensione della pagina di livello massimo (PART_SIZE), sono allineate naturalmente e non si sovrappongono. In questo modo possiamo definire le varie parti semplicemente specificando un intervallo di entrate della tabella radice. Per esempio, la parte sistema/condivisa usa N_SIS_C entrate a partire da I_SIS_C e contiene tutti e soli gli indirizzi la cui traduzione passa da queste entrate.

5.16 Funzioni necessarie per map() e unmap()

Diagramma di collaborazione per Funzioni necessarie per map() e unmap():

Funzioni

- paddr alloca_tab ()

  Alloca un frame libero destinato a contenere una tabella.
- void rilascia_tab (paddr f)

  Dealloca un frame che contiene una tabella.
- void inc_ref (paddr f)

  Incrementa il contatore delle entrate valide di una tabella.
- void dec_ref (paddr f)

  Decrementa il contatore delle entrate valide di una tabella.
- natl get_ref (paddr f)

  Legge il contatore delle entrate valide di una tabella.

5.16.1 Descrizione dettagliata

Le funzioni map() e unmap() di libce richiedono la definizione di alcune funzioni per l'allocazione e la deallocazione delle tabelle. Le definiamo qui, utilizzando i descrittori di frame. In particolare, se un frame contiene una tabella, il campo nvalide del suo descrittore (des_frame) è il contatore delle entrate valide della tabella (entrate della tabella con P == 1).

5.16.2 Documentazione delle funzioni

5.16.2.1 alloca_tab()

```
paddr alloca_tab ( )
```

Alloca un frame libero destinato a contenere una tabella.

Azzera tutta la tabella e il suo contatore di entrate di valide.

Restituisce

indirizzo fisico della tabella

Definizione alla linea 678 del file sistema.cpp.

5.16.2.2 dec_ref()

```
void dec_ref (
            paddr f )
```

Decrementa il contatore delle entrate valide di una tabella.

Parametri

f	indirizzo fisico della tabella

Definizione alla linea 714 del file sistema.cpp.

5.16.2.3 get_ref()

```
natl get_ref (
            paddr f )
```

Legge il contatore delle entrate valide di una tabella.

Parametri

f	indirizzo fisico della tabella

Restituisce

valore del contatore

Definizione alla linea 723 del file sistema.cpp.

5.16.2.4 inc_ref()

```
void inc_ref (
             paddr f )
```

Incrementa il contatore delle entrate valide di una tabella.

Parametri

f	indirizzo fisico della tabella

Definizione alla linea 706 del file sistema.cpp.

5.16.2.5 rilascia_tab()

```
void rilascia_tab (
             paddr f )
```

Dealloca un frame che contiene una tabella.

Avvertimento

La funzione controlla che la tabella non contenga entrate valide e causa un errore fatale in caso contrario.

Parametri

f	indirizzo fisico della tabella

Definizione alla linea 695 del file sistema.cpp.

5.17 Creazione e distruzione dei processi

Diagramma di collaborazione per Creazione e distruzione dei processi:

Variabili

- des_proc * a_p [apic::MAX_IRQ]

  Associazione IRQ -> processo esterno che lo gestisce.
- des_proc *const ESTERN_BUSY = reinterpret_cast<des_proc*>(1UL)

  Valore da inserire in a_p per gli IRQ che sono gestiti da driver.

Distruzione della pila sistema corrente

Quando dobbiamo eliminare una pila sistema dobbiamo stare attenti a non eliminare proprio quella che stiamo usando. Questo succede durante una terminate_p() o abort_p(), quando si tenta di distrugguere proprio il processo che ha invocato la primitiva.

Per fortuna, se stiamo terminando il processo corrente, vuol dire anche che stiamo per metterne in esecuzione un altro e possiamo dunque usare la pila sistema di quest'ultimo. Operiamo dunque nel seguente modo:

- all'ingresso nel sistema (in salva_stato), salviamo il valore di esecuzione in esecuzione_precedente; questo è il processo a cui appartiene la pila sistema che stiamo usando;
- in distruggi_processo(), se esecuzione è uguale a esecuzione_precedente (stiamo distruggendo proprio il processo a cui appartiene la pila corrente), *non* distruggiamo la pila sistema e settiamo la variabile ultimo_terminato;
- in carica_stato, dopo aver cambiato pila, se ultimo_terminato è settato, distruggiamo la pila sistema di esecuzione_precedente.

- des_proc * esecuzione_precedente

  Processo che era in esecuzione all'entrata nel modulo sistema.
- paddr ultimo_terminato

  Se diverso da zero, indirizzo fisico della root_tab dell'ultimo processo terminato o abortito.
- void distruggi_pila_precedente ()

  Distrugge la pila sistema del processo uscente e rilascia la sua tabella radice.

Funzioni di supporto alla creazione e distruzione dei processi

- natl alloca_proc_id (des_proc *p)

  Alloca un id di processo.
- void rilascia_proc_id (natw id)

  Rilascia un id di processo non più utilizzato.
- void init_root_tab (paddr dest)

  Inizializza la tabella radice di un nuovo processo.
- void clear_root_tab (paddr dest)

  Ripulisce la tabella radice di un processo.
- bool crea_pila (paddr root_tab, vaddr bottom, natq size, natl liv)

  Crea una pila processo.
- void distruggi_pila (paddr root_tab, vaddr bottom, natq size)

  Distrugge una pila processo.
- des_proc * crea_processo (void f(natq), natq a, int prio, char liv)

  Funzione interna per la creazione di un processo.
- void distruggi_processo (des_proc *p)

  Dealloca tutte le risorse allocate da crea_processo()
- bool load_handler (natq tipo, natq irq)

  Carica un handler nella IDT.

Primitive per la creazione e distruzione dei processi (parte C++)

- void c_activate_p (void f(natq), natq a, natl prio, natl liv)

  Parte C++ della primitiva activate_p()
- void c_terminate_p (bool logmsg)

  Parte C++ della pritimiva terminate_p()
- void c_abort_p (bool selfdump)

  Parte C++ della primitiva abort_p()
- void c_activate_pe (void f(natq), natq a, natl prio, natl liv, natb irq)

  Parte C++ della primitiva activate_pe().

5.17.1 Descrizione dettagliata

5.17.2 Documentazione delle funzioni

5.17.2.1 alloca_proc_id()

```
natl alloca_proc_id (
            des_proc * p )
```

Alloca un id di processo.

Parametri

p	descrittore del processo a cui assegnare l'id

Restituisce

id del processo (0xFFFFFFFF se terminati)

Definizione alla linea 809 del file sistema.cpp.

5.17.2.2 c_abort_p()

```
void c_abort_p (
            bool selfdump )
```

Parte C++ della primitiva abort_p()

Fuziona come la terminate_p(), ma invia anche un warning al log. La funzione va invocata quando si vuole terminare un processo segnalando che c'è stato un errore.

Parametri

selfdump	se true mostra sul log lo stato del processo

Definizione alla linea 1218 del file sistema.cpp.

5.17.2.3 c_activate_p()

```
void c_activate_p (
             void  fnatq,
             natq a,
             natl prio,
             natl liv )
```

Parte C++ della primitiva activate_p()

Parametri

f	corpo del processo
a	parametro per il corpo del processo
prio	priorità del processo
liv	livello del processo (LIV_UTENTE o LIV_SISTEMA)

Definizione alla linea 1147 del file sistema.cpp.

5.17.2.4 c_activate_pe()

```
void c_activate_pe (
             void  fnatq,
             natq a,
             natl prio,
             natl liv,
             natb irq )
```

Parte C++ della primitiva activate_pe().

Parametri

f	corpo del processo
a	parametro per il corpo del processo
prio	priorità del processo
liv	livello del processo (LIV_UTENTE o LIV_SISTEMA)
irq	IRQ gestito dal processo

Definizione alla linea 1238 del file sistema.cpp.

5.17.2.5 c_terminate_p()

```
void c_terminate_p (
            bool logmsg )
```

Parte C++ della pritimiva terminate_p()

Parametri

logmsg	set true invia un messaggio sul log

Definizione alla linea 1193 del file sistema.cpp.

5.17.2.6 clear_root_tab()

```
void clear_root_tab (
            paddr dest )
```

Ripulisce la tabella radice di un processo.

Parametri

dest	indirizzo fisico della tabella

Definizione alla linea 866 del file sistema.cpp.

5.17.2.7 crea_pila()

```
bool crea_pila (
            paddr root_tab,
            vaddr bottom,
            natq size,
            natl liv )
```

Crea una pila processo.

Parametri

root_tab	indirizzo fisico della radice del TRIE del processo
bottom	indirizzo virtuale del bottom della pila
size	dimensione della pila (in byte)
liv	livello della pila (LIV_UTENTE o LIV_SISTEMA)

Restituisce

true se la creazione ha avuto successo, false altrimenti

Definizione alla linea 882 del file sistema.cpp.

5.17.2.8 crea_processo()

```
des_proc* crea_processo (
             void  fnatq,
             natq a,
             int prio,
             char liv )
```

Funzione interna per la creazione di un processo.

Parte comune a activate_p() e activate_pe(). Alloca un id per il processo e crea e inizializza il descrittore di processo, la pila sistema e, per i processi di livello utente, la pila utente. Crea l'albero di traduzione completo per la memoria virtuale del processo.

Parametri

f	corpo del processo
a	parametro per il corpo del processo
prio	priorità del processo
liv	livello del processo (LIV_UTENTE o LIV_SISTEMA)

Restituisce

puntatore al nuovo descrittore di processo (nullptr in caso di errore)

Definizione alla linea 928 del file sistema.cpp.

5.17.2.9 distruggi_pila()

```
void distruggi_pila (
             paddr root_tab,
             vaddr bottom,
             natq size )
```

Distrugge una pila processo.

Funziona indifferentemente per pile utente o sistema.

Parametri

root_tab	indirizzo fisico della radice del TRIE del processo
bottom	indirizzo virtuale del bottom della pila
size	dimensione della pila (in byte)

Definizione alla linea 905 del file sistema.cpp.

5.17.2.10 distruggi_pila_precedente()

```
void distruggi_pila_precedente ( )
```

Distrugge la pila sistema del processo uscente e rilascia la sua tabella radice.

Chiamata da distruggi_processo() oppure da carica_stato se distruggi_processo() aveva rimandato la distruzione della pila. Dealloca la pila sistema e le traduzioni corrispondenti nel TRIE di radice ultimo_terminato. La distruggi_processo() aveva già eliminato tutte le altre traduzioni, quindi la funzione può anche deallocare la radice del TRIE.

Definizione alla linea 1127 del file sistema.cpp.

5.17.2.11 distruggi_processo()

```
void distruggi_processo (
            des_proc * p )
```

Dealloca tutte le risorse allocate da crea_processo()

Dealloca l'id, il descrittore di processo, l'eventuale pila utente, comprese le tabelle che la mappavano nella memoria virtuale del processo. Per la pila sistema si veda sopra esecuzione_precedente.

Definizione alla linea 1053 del file sistema.cpp.

5.17.2.12 init_root_tab()

```
void init_root_tab (
            paddr dest )
```

Inizializza la tabella radice di un nuovo processo.

Parametri

dest	indirizzo fisico della tabella

Definizione alla linea 848 del file sistema.cpp.

5.17.2.13 load_handler()

```
bool load_handler (
             natq tipo,
             natq irq )
```

Carica un handler nella IDT.

Fa in modo che l'handler *irq* -esimo sia associato alle richieste di interruzione provenienti dal piedino *irq* dell'APIC. L'assozione avviene tramite l'entrata *tipo* della IDT.

Parametri

tipo	tipo dell'interruzione a cui associare l'handler
irq	piedino dell'APIC associato alla richiesta di interruzione

5.17.2.14 rilascia_proc_id()

```
void rilascia_proc_id (
             natw id )
```

Rilascia un id di processo non più utilizzato.

Parametri

id	id da rilasciare

Definizione alla linea 833 del file sistema.cpp.

5.17.3 Documentazione delle variabili

5.17.3.1 esecuzione_precedente

```
des_proc* esecuzione_precedente
```

Processo che era in esecuzione all'entrata nel modulo sistema.

La salva_stato ricorda quale era il processo in esecuzione al momento dell'entrata nel sistema e lo scrive in questa variabile.

Definizione alla linea 1108 del file sistema.cpp.

5.17.3.2 ESTERN_BUSY

```
des_proc* const ESTERN_BUSY = reinterpret_cast<des_proc*>(1UL)
```

Valore da inserire in a_p per gli IRQ che sono gestiti da driver.

Nel nucleo base questo accade solo per l'IRQ del timer.

Definizione alla linea 800 del file sistema.cpp.

5.17.3.3 ultimo_terminato

```
paddr ultimo_terminato
```

Se diverso da zero, indirizzo fisico della root_tab dell'ultimo processo terminato o abortito.

La carica_stato legge questo indirizzo per sapere se deve distruggere la pila del processo uscente, dopo aver effettuato il passaggio alla pila del processo entrante.

Definizione alla linea 1116 del file sistema.cpp.

5.18 Inizializzazione

Diagramma di collaborazione per Inizializzazione:

Moduli

- Caricamento dei moduli I/O e utente

Funzioni

- natl crea_dummy ()
  Crea il processo dummy.
- void main_sistema (natq)
  Corpo del processo main_sistema.
- natl crea_main_sistema ()
  Crea il processo main_sistema.
- bool crea_spazio_condiviso (paddr root_tab, boot64_modinfo *mod)
  Crea le parti utente/condivisa e io/condivisa.
- void salta_a_main ()
  Funzione di supporto pr avviare il primo processo (definita in sistema.s)
- void c_fill_gate (natb tipo, void routine(), int liv)
  Parte C++ della primitiva fill_gate().
- void main (boot64_info *info)
  Prima parte dell'inizializzazione; crea i primi processi.

Variabili

- des_proc init

  Un primo des_proc, allocato staticamente, da usare durante l'inizializzazione.
- const natl DELAY = 59659

  Periodo del timer di sistema.
- paddr tss_punt_nucleo

  Indirizzo fisico del puntatore alla pila sistema nel segmento TSS.
- void(∗ io_entry)(natq)

  Entry point del modulo IO.
- void(∗ user_entry)(natq)

  Entry point del modulo utente.

Costanti per lo heap di sistema

- const natq HEAP_START = 1∗MiB

  Indirizzo base dello heap di sistema.
- const natq HEAP_SIZE = 1∗MiB

  Dimensione dello heap di sistema.

5.18.1 Descrizione dettagliata

L'inizializzazione è in due parti:

1. la funzione main() inizializza le strutture dati necessarie alla creazione dei primi processi, quindi crea dummy e main_sistema, cedendo il controllo a quest'ultimo (salta_a_main);
2. main_sistema si occupa della seconda parte, che inizializza il modulo I/O e crea il processo main utente, quindi termina cedendo il controllo a quest'ultimo.

5.18.2 Documentazione delle funzioni

5.18.2.1 c_fill_gate()

```
void c_fill_gate (
            natb tipo,
            void  routine(),
            int liv )
```

Parte C++ della primitiva fill_gate().

Parametri

tipo	tipo del gate da riempire
routine	funzione da associare al gate
liv	DPL del gate (LIV_UTENTE o LIV_SISTEMA)

Definizione alla linea 1392 del file sistema.cpp.

5.18.2.2 crea_dummy()

```
natl crea_dummy ( )
```

Crea il processo dummy.

Restituisce

id del processo

Definizione alla linea 1338 del file sistema.cpp.

5.18.2.3 crea_main_sistema()

```
natl crea_main_sistema ( )
```

Crea il processo main_sistema.

Restituisce

id del processo

Definizione alla linea 1358 del file sistema.cpp.

5.18.2.4 crea_spazio_condiviso()

```
bool crea_spazio_condiviso (
            paddr root_tab,
            boot64_modinfo * mod )
```

Crea le parti utente/condivisa e io/condivisa.

Nota

Setta le variabili user_entry e io_entry.

Parametri

root_tab	indirizzo fisico della tabella radice
mod	informazioni sui moduli caricati dal boot loader

Restituisce

true in caso di successo, false altrimenti

5.18.2.5 main()

```
void main (
            boot64_info * info )
```

Prima parte dell'inizializzazione; crea i primi processi.

Parametri

info	informazioni passata dal boot loader di libce

Definizione alla linea 1419 del file sistema.cpp.

5.18.2.6 main_sistema()

```
void main_sistema (
            natq  )
```

Corpo del processo main_sistema.

Si occupa della seconda parte dell'inizializzazione.

Definizione alla linea 1511 del file sistema.cpp.

5.18.3 Documentazione delle variabili

5.18.3.1 io_entry

```
void(* io_entry) (natq) (
            natq  )
```

Entry point del modulo IO.

Nota

Inizializzato da crea_spazio_condiviso().

Definizione alla linea 1503 del file sistema.cpp.

5.18.3.2 user_entry

```
void(* user_entry) (natq) (
            natq  )
```

Entry point del modulo utente.

Nota

Inizializzato da crea_spazio_condiviso().

Definizione alla linea 1508 del file sistema.cpp.

5.19 Caricamento dei moduli I/O e utente

Diagramma di collaborazione per Caricamento dei moduli I/O e utente:

Strutture dati

- struct copy_segment

  Oggetto da usare con map() per caricare un segmento ELF in memoria virtuale.

Funzioni

- vaddr carica_modulo (boot64_modinfo *mod, paddr root_tab, natq flags, natq heap_size)

  Carica un modulo in M2.
- vaddr carica_IO (boot64_modinfo *mod, paddr root_tab)

  Mappa il modulo I/O.
- vaddr carica_utente (boot64_modinfo *mod, paddr root_tab)

  Mappa il modulo utente.
- paddr copy_segment::operator() (vaddr)

  Funzione chiamata da map().

5.19.1 Descrizione dettagliata

All'avvio troviamo il contenuto dei file `sistema`, `io` e `utente` già copiati in memoria dal primo boot loader (quello realizzato da QEMU stesso). Il secondo boot loader (quello della libce) ha caricato nella posizione finale solo il modulo sistema. Ora il modulo sistema deve rendere operativi anche i moduli IO e utente, interpretando i file e creando le necessarie traduzioni nella memoria virtuale. Per farlo dobbiamo esaminare le righe di tipo PT_LOAD delle tabelle di programma dei due file. Ciascuna di queste righe ci indica:

- una parte *S* del file da mappare in memoria, espressa come offset dall'inizio del file e dimensone in byte (*S* è un segmento ELF);
- l'indirizzo virtuale *V* a cui mappare *S* e il numero di byte di memoria virtuale da occupare (può essere più grande della dimensione di *S*, e in quel caso la parte eccedente deve essere azzerata)
- i diritti di accesso (lettura, scrittura, esecuzione) da garantire.

Per poter creare le traduzioni da *V* a *S* dobbiamo copiare *S* da dove si trova ora in dei frame di M2, per almeno due motivi: la copia attuale di *S* potrebbe non essere allineata correttamente; inoltre, potremmo non avere spazio per azzerare l'eventuale parte eccedente.

5.19.2 Documentazione delle funzioni

5.19.2.1 carica_IO()

```
vaddr carica_IO (
            boot64_modinfo * mod,
            paddr root_tab )
```

Mappa il modulo I/O.

Parametri

mod	informazioni sul modulo caricato dal boot loader
root_tab	indirizzo fisico della radice del TRIE

Restituisce

indirrizzo virtuale dell'entry point del modulo I/O, o zero in caso di errore

Definizione alla linea 1736 del file sistema.cpp.

5.19.2.2 carica_modulo()

```
vaddr carica_modulo (
            boot64_modinfo * mod,
            paddr root_tab,
            natq flags,
            natq heap_size )
```

Carica un modulo in M2.

Copia il modulo in M2, lo mappa al suo indirizzo virtuale e aggiunge lo heap dopo l'ultimo indirizzo virtuale usato.

Parametri

mod	informazioni sul modulo caricato dal boot loader
root_tab	indirizzo fisico della radice del TRIE
flags	BIT_US per rendere il modulo accessibile da livello utente, altrimenti 0
heap_size	dimensione dello heap (in byte)

Restituisce

indirizzo virtuale dell'entry point del modulo, o zero in caso di errore

Definizione alla linea 1659 del file sistema.cpp.

5.19.2.3 carica_utente()

```
vaddr carica_utente (
            boot64_modinfo * mod,
            paddr root_tab )
```

Mappa il modulo utente.

Parametri

mod	informazioni sul modulo caricato dal boot loader
root_tab	indirizzo fisico della radice del TRIE

Restituisce

indirrizzo virtuale dell'entry point del modulo utente, o zero in caso di errore

Definizione alla linea 1748 del file sistema.cpp.

5.19.2.4 operator()()

```
paddr copy_segment::operator() (
            vaddr v )
```

Funzione chiamata da map().

Copia il prossimo frame di un segmento in un frame di M2.

Parametri

v	indirizzo virtuale da mappare

Restituisce

indirizzo fisico del frame di M2

Definizione alla linea 1620 del file sistema.cpp.

5.20 Gestione errori

Diagramma di collaborazione per Gestione errori:

Funzioni

- void panic (const char *msg)

  Ferma il sistema e stampa lo stato di tutti i processi.
- void c_io_panic ()

  Parte C++ della primitiva io_panic()
- void c_nmi ()

  Routine di risposta a un non-maskable-interrupt.
- void c_do_log (log_sev sev, const char *buf, natl quanti)

  Parte C++ della primitiva do_log().
- void c_getmeminfo ()

  Parte C++ della primitiva getmeminfo().

Variabili

- int MAX_LOG = 5

  Massimo livello ammesso per la severità dei messaggi del log.

Funzioni di supporto per il backtrace

- natq read_mem (void *token, vaddr v)

  Callback invocata dalla funzione cfi_backstep() per leggere dalla pila di un qualunque processo.
- void backtrace (des_proc *p, log_sev sev, const char *msg)

  Invia sul log il backtrace di un processo.
- void process_dump (des_proc *p, log_sev sev)

  Invia sul log lo stato di un processo.

5.20.1 Descrizione dettagliata

5.20.2 Documentazione delle funzioni

5.20.2.1 backtrace()

```
void backtrace (
            des_proc * p,
            log_sev sev,
            const char * msg )
```

Invia sul log il backtrace di un processo.

Parametri

p	descrittore del processo
sev	severità dei messaggi da inviare al log
msg	primo messaggio da inviare (intestazione)

Definizione alla linea 1917 del file sistema.cpp.

5.20.2.2 c_do_log()

```
void c_do_log (
            log_sev sev,
            const char * buf,
            natl quanti )
```

Parte C++ della primitiva do_log().

Parametri

sev	severità del messaggio
buf	buffer che contiene il messaggio
quanti	lunghezza del messaggio in byte

Definizione alla linea 1856 del file sistema.cpp.

5.20.2.3 c_io_panic()

```
void c_io_panic ( )
```

Parte C++ della primitiva io_panic()

Il modulo I/O può usare questa primitiva per segnalare un errore fatale.

Definizione alla linea 1827 del file sistema.cpp.

5.20.2.4 c_nmi()

```
void c_nmi ( )
```

Routine di risposta a un non-maskable-interrupt.

La routine ferma il sistema e stampa lo stato di tutti i processi.

Nota

Il sito dell'autocorrezione invia un nmi se il programma da testare non termina entro il tempo prestabilito.

Definizione alla linea 1838 del file sistema.cpp.

5.20.2.5 panic()

```
void panic (
            const char * msg )
```

Ferma il sistema e stampa lo stato di tutti i processi.

Parametri

msg	messaggio da inviare al log (severità LOG_ERR)

Definizione alla linea 1797 del file sistema.cpp.

5.20.2.6 process_dump()

```
void process_dump (
            des_proc * p,
            log_sev sev )
```

Invia sul log lo stato di un processo.

Parametri

p	descrittore del processo
sev	severità dei messaggi da inviare al log

Definizione alla linea 1979 del file sistema.cpp.

5.20.2.7 read_mem()

```
natq read_mem (
            void * token,
            vaddr v )
```

Callback invocata dalla funzione cfi_backstep() per leggere dalla pila di un qualunque processo.

Parametri

token	(opaco) descrittore del processo di cui si sta producendo il backtrace
v	indirizzo virtuale da leggere

Restituisce

natq letto da *v* nella memoria virtuale del processo (0 se non mappato)

Definizione alla linea 1900 del file sistema.cpp.

5.21 Memoria dinamica

Diagramma di collaborazione per Memoria dinamica:

Funzioni

- void * operator new (size_t s)

  alloca un oggetto nello heap utente
- void operator delete (void *p)

  dealloca un oggetto restituendolo allo heap utente.

Variabili

- natl userheap_mutex

  Semaforo di mutua esclusione per lo heap utente.

5.21.1 Descrizione dettagliata

Dal momento che il modulo utente è eseguito con le interruzioni esterne mascherabili abilitate, dobbiamo proteggere lo heap con un semaforo di mutua esclusione.

5.21.2 Documentazione delle funzioni

5.21.2.1 operator delete()

```
void operator delete (
            void * p )
```

dealloca un oggetto restituendolo allo heap utente.

Parametri

p	puntatore all'oggetto

Definizione alla linea 84 del file lib.cpp.

5.21.2.2 operator new()

```
void* operator new (
            size_t s )
```

alloca un oggetto nello heap utente

Parametri

s	dimensione dell'oggetto

Restituisce

puntatore all'oggetto (nullptr se heap esaurito)

Definizione alla linea 70 del file lib.cpp.

5.22 Gestione errori

Diagramma di collaborazione per Gestione errori:

Funzioni

- void panic (const char *msg)
  Termina il processo corrente.

5.22.1 Descrizione dettagliata

5.22.2 Documentazione delle funzioni

5.22.2.1 panic()

```
void panic (
            const char * msg )
```

Termina il processo corrente.

Gli errori nel modulo utente non sono mai fatali.

Definizione alla linea 99 del file lib.cpp.

5.23 Inizializzazione

Diagramma di collaborazione per Inizializzazione:

Funzioni

- void lib_init () __attribute__((constructor))
  Inizializza la libreria utente.

5.23.1 Descrizione dettagliata

5.23.2 Documentazione delle funzioni

5.23.2.1 lib_init()

```
void lib_init ( ) const
```

Inizializza la libreria utente.

In particolare, alloca il semaforo di mutua esclusione dello heap utente e inizializza lo heap stesso.

Nota

La stringa `__attribute__((constructor))` è una estensione di gcc. Dice al compilatore di aggiungere un puntatore a questa funzione alla tabella init_array del file ELF. Nel nostro caso, queste funzioni vengono poi chiamate da start (usando la funzione ctors() di libce) prima di invocare main().

5.24 Modulo utente

Diagramma di collaborazione per Modulo utente:

Moduli

- Memoria dinamica
- Gestione errori
- Inizializzazione
- Funzioni di utilità generale

Variabili

- natb end [ ]

  ultimo indirizzo usato dal modulo utente (fornito dal collegatore)

5.24.1 Descrizione dettagliata

5.25 Funzioni di utilità generale

Diagramma di collaborazione per Funzioni di utilità generale:

Funzioni

- int printf (const char *fmt,...) __attribute__((format(printf

  Formatta un messaggio e lo scrive sul video.
- void pause ()

  Attende la pressione di un carattere.
- natl getpid ()

  Id del processo corrente.

Variabili

- char pause_buf [1]

  buffer della funzione pause()

5.25.1 Descrizione dettagliata

5.25.2 Documentazione delle funzioni

5.25.2.1 getpid()

```
natl getpid ( )
```

Id del processo corrente.

Per implementare getpid() usiamo getmeminfo().

Restituisce

id del processo corrente

Definizione alla linea 49 del file lib.cpp.

5.25.2.2 pause()

```
int void pause ( )
```

Attende la pressione di un carattere.

Non possiamo usare la funzione pause() di libce, perché non abbiamo accesso diretto ai registri della tastiera. Invece, usiamo readconsole() per leggere un singolo carattere.

Definizione alla linea 40 del file lib.cpp.

5.25.2.3 printf()

```
int printf (
            const char * fmt,
             ...  )
```

Formatta un messaggio e lo scrive sul video.

Non possiamo usare la funzione printf di libce, perché non abbiamo accesso diretto alla memoria video. Invece, formattiamo il messaggio in un buffer e poi lo scriviamo tramite writeconsole().

Si possono usare gli stessi formati della `printf(3)` della libreria standard del C, con l'esclusione di quelli relativi ai tipi `float` e `double`.

Parametri

fmt	stringa di formato
...	argomenti richiesti da *fmt*

Restituisce

numero di caratteri scritti

Definizione alla linea 15 del file lib.cpp.

Capitolo 6

Documentazione delle classi

6.1 Riferimenti per la struct copy_segment

Oggetto da usare con map() per caricare un segmento ELF in memoria virtuale.

Membri pubblici

- paddr operator() (vaddr)

  Funzione chiamata da map().

Campi

- paddr mod_beg

  base del segmento in memoria fisica
- paddr mod_end

  limite del segmento in memoria fisica
- vaddr virt_beg

  indirizzo virtuale della base del segmento

6.1.1 Descrizione dettagliata

Oggetto da usare con map() per caricare un segmento ELF in memoria virtuale.

Definizione alla linea 1597 del file sistema.cpp.

6.2 Riferimenti per la struct des_ata

Descrittore di interfaccia ATA.

Campi

- natb comando
  Ultimo comando inviato all'interfaccia.
- natl mutex
  Indice di un semaforo di mutua esclusione.
- natl sincr
  Indice di un semaforo di sincronizzazione.
- natb cont
  Quanti settori resta da leggere o scrivere.
- natb * punt
  Da dove leggere/dove scrivere il prossimo settore.

6.2.1 Descrizione dettagliata

Descrittore di interfaccia ATA.

Definizione alla linea 278 del file io.cpp.

6.3 Riferimenti per la struct des_console

Descrittore della console.

Campi

- natl mutex
  Semaforo di mutua esclusione per l'accesso alla console.
- natl sincr
  Semafor di sincronizzazione (per le letture da tastiera)
- char * punt
  Dove scrivere il prossimo carattere letto.
- natq cont
  Quanti caratteri resta da leggere.
- natq dim
  Dimensione del buffer passato a readconsole()

6.3.1 Descrizione dettagliata

Descrittore della console.

Definizione alla linea 81 del file io.cpp.

6.4 Riferimenti per la struct des_frame

Descrittore di frame.

Campi

- union {
  natw nvalide
  numero di entrate valide (se il frame contiene una tabella)
  natl prossimo_libero
  prossimo frame libero (se il frame è libero)
  };

6.4.1 Descrizione dettagliata

Descrittore di frame.

Definizione alla linea 519 del file sistema.cpp.

6.5 Riferimenti per la struct des_proc

Descrittore di processo.

Diagramma di collaborazione per des_proc:

Campi

- natw id
  identificatore numerico del processo
- natw livello
  livello di privilegio (LIV_UTENTE o LIV_SISTEMA)
- natl precedenza
  precedenza nelle code dei processi
- vaddr punt_nucleo
  indirizzo della base della pila sistema
- natq contesto [N_REG]
  copia dei registri generali del processore
- paddr cr3
  radice del TRIE del processo
- des_proc * puntatore
  prossimo processo in coda

Informazioni utili per il per debugging

- void(* corpo)(natq)
  parametro `f` passato alla `activate_p/_pe` che ha creato questo processo
- natq parametro
  parametro `a` passato alla `activate_p/_pe` che ha creato questo processo

6.5.1 Descrizione dettagliata

Descrittore di processo.

Definizione alla linea 66 del file sistema.cpp.

6.6 Riferimenti per la struct des_sem

Descrittore di semaforo.

Diagramma di collaborazione per des_sem:

Campi

- int counter
- des_proc * pointer

  coda di processi bloccati sul semaforo

6.6.1 Descrizione dettagliata

Descrittore di semaforo.

Definizione alla linea 235 del file sistema.cpp.

6.6.2 Documentazione dei campi

6.6.2.1 counter

```
int des_sem::counter
```

se >= 0, numero di gettoni contenuti; se < 0, il valore assoluto è il numero di processi in coda

Definizione alla linea 238 del file sistema.cpp.

6.7 Riferimenti per la struct meminfo

Informazioni di debug.

Campi

- natl heap_libero
  numero di byte liberi nello heap di sistema
- natl num_frame_liberi
  numero di frame liberi in M2
- natl pid
  id del processo corrente

6.7.1 Descrizione dettagliata

Informazioni di debug.

Questa struttura contiene delle informazioni che sono usate nei testi d'esame per eseguire alcuni controlli.

Definizione alla linea 83 del file sys.h.

6.8 Riferimenti per la struct richiesta

Richiesta al timer.

Diagramma di collaborazione per richiesta:

Campi

- natl d_attesa
  tempo di attesa aggiuntivo rispetto alla richiesta precedente
- richiesta * p_rich
  puntatore alla richiesta successiva
- des_proc * pp
  descrittore del processo che ha effettuato la richiesta

6.8.1 Descrizione dettagliata

Richiesta al timer.

Definizione alla linea 380 del file sistema.cpp.

Capitolo 7

Documentazione dei file

7.1 Riferimenti per il file include/costanti.h

File incluso da tutti i moduli, sia nella parte C++ che nella parte assembler.

Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:

Definizioni

limiti modificabili

Queste costanti definiscono alcune dimensioni o valori che possono essere modificati liberamente, entro limiti ragionevoli.

- #define MAX_SEM 1024UL
  massimo numero di semafori per livello
- #define MAX_PROC 1024UL
  massimo numero di processi
- #define MIN_EXT_PRIO 1024UL
  priorità minima dei processi esterni
- #define MEM_TOT (32∗MiB)
  dimensione della memoria fisica
- #define DIM_USR_HEAP (1∗MiB)
  dimensione dello heap utente
- #define DIM_USR_STACK (64∗KiB)
  dimensione degli stack utente
- #define DIM_IO_HEAP (1∗MiB)
  dimensione dello heap del modulo I/O
- #define DIM_SYS_STACK (4∗KiB)
  dimensione degli stack sistema
- #define MAX_PRD 16
  numero massimo di PRD usati da dmaread/dmawrite

Primitive comuni

Tipi delle primitive dichiarate in `sys.h`

- #define TIPO_A 0x20
  activate_p()
- #define TIPO_T 0x21

terminate_p()
- #define TIPO_SI 0x22
  sem_ini()
- #define TIPO_W 0x23
  sem_wait()
- #define TIPO_S 0x24
  sem_signal()
- #define TIPO_D 0x25
  delay()
- #define TIPO_L 0x26
  do_log()
- #define TIPO_GMI 0x27
  getmeminfo()

Primitive riservate per il modulo I/O

Tipi delle primitive dichiarate in `sysio.h`

- #define TIPO_APE 0x30
  activate_pe()
- #define TIPO_WFI 0x31
  wfi()
- #define TIPO_FG 0x32
  fill_gate()
- #define TIPO_AB 0x33
  abort_p()
- #define TIPO_IOP 0x34
  io_panic()
- #define TIPO_TRA 0x35
  trasforma()
- #define TIPO_ACC 0x36
  access()

Primitive fornite dal modulo I/O

Tipi delle primitive dichiarate in `io.h`

- #define IO_TIPO_HDR 0x40
  readhd_n()
- #define IO_TIPO_HDW 0x41
  writehd_n()
- #define IO_TIPO_DMAHDR 0x42
  dmareadhd_n()
- #define IO_TIPO_DMAHDW 0x43
  dmawritehd_n()
- #define IO_TIPO_RCON 0x44
  readconsole()
- #define IO_TIPO_WCON 0x45
  writeconsole()
- #define IO_TIPO_INIC 0x46
  iniconsole()
- #define IO_TIPO_GMI 0x47
  getiomeminfo()

Tipi delle interruzioni esterne

- #define INTR_TIPO_KBD 0x50
  tastiera
- #define INTR_TIPO_HD 0x60

hard disk

- #define INTR_TIPO_TIMER 0xFE
  timer (prio massima)

Suddivisione della memoria virtuale.

I nomi di queste costanti seguono lo schema {I,N}_{SIS,MIO,UTN}_{C,P}, dove:

- *I = Indice della prima entrata in root_tab*
- *N = Numero di entrate in root_tab*
- *SIS = SIStema*
- *MIO = Modulo IO*
- *UTN = modulo UTeNte*
- *C = Condiviso*
- *P = Privato*

- #define I_SIS_C 0
  prima entrata sistema/condivisa
- #define I_SIS_P 1
  prima entrata sistema/privata
- #define I_MIO_C 2
  prima entrata modulo IO/condivisa
- #define I_UTN_C 256
  prima entrata utente/condivisa
- #define I_UTN_P 384
  prima entrata utente/privata
- #define N_SIS_C 1
  numero entrate sistema/condivisa
- #define N_SIS_P 1
  numero entrate sistema/privata
- #define N_MIO_C 1
  numero entrate modulo IO/condivisa
- #define N_UTN_C 128
  numero entrate utente/convidisa
- #define N_UTN_P 128
  numero entrate utente/privata

Costanti non modificabili

Non modificare la defizione di queste costanti.

- #define MIN_PROC_ID 0
  minimo id di processo
- #define MAX_PROC_ID (MAX_PROC - 1)
  massimo id di processo
- #define MAX_PRIORITY (MIN_EXT_PRIO - 1)
  priorità massima dei processi (non esterni)
- #define MIN_PRIORITY 0x1
  priorità minima dei processi
- #define MAX_EXT_PRIO (MIN_EXT_PRIO + 0xFE)
  priorità massima dei processi esterni

7.1.1 Descrizione dettagliata

File incluso da tutti i moduli, sia nella parte C++ che nella parte assembler.

7.2 Riferimenti per il file include/io.h

Primitive fornite dal modulo I/O.

Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:

Funzioni

- void iniconsole (natb cc)
  Inizializza la console (video e tastiera)
- natq readconsole (char *buff, natq quanti)
  Lettura da tastiera.
- void writeconsole (const char *buff, natq quanti)
  Scrive caratteri sul video.
- void readhd_n (void *vetti, natl primo, natb quanti)
  Lettura di settori dall'hard disk.
- void writehd_n (const void *vetto, natl primo, natb quanti)
  Scrittura di settori sull'hard disk.
- void dmareadhd_n (void *vetti, natl primo, natb quanti)
  Lettura di settori dall'hard disk (in DMA).
- void dmawritehd_n (const void *vetto, natl primo, natb quanti)
  Scrittura di settori sull'hard disk (in DMA).

Funzioni di supporto al debugging

- natq getiomeminfo ()
  Informazioni di debug.

7.2.1 Descrizione dettagliata

Primitive fornite dal modulo I/O.

Queste primitive possono essere usate dal modulo utente. Sono eseguite a livello sistema, ma con le interruzioni esterne mascherabili abilitate.

7.2.2 Documentazione delle funzioni

7.2.2.1 dmareadhd_n()

```
void dmareadhd_n (
            void * vetti,
            natl primo,
            natb quanti )
```

Lettura di settori dall'hard disk (in DMA).

Parametri

vetti	buffer destinato a ricevere i dati letti
primo	LBA del primo settore da leggere
quanti	numero di settori da leggere

7.2.2.2 dmawritehd_n()

```
void dmawritehd_n (
            const void * vetto,
            natl primo,
            natb quanti )
```

Scrittura di settori sull'hard disk (in DMA).

Parametri

vetto	buffer contenente i dati da scrivere
primo	LBA del primo settore da scrivere
quanti	numero di settori da scrivere

7.2.2.3 getiomeminfo()

```
natq getiomeminfo ( )
```

Informazioni di debug.

Questa primitiva è usata in alcuni testi d'esame per eseguire dei controlli.

Restituisce

quantià di byte liberi nello heap del modulo I/O

7.2.2.4 iniconsole()

```
void iniconsole (
            natb cc )
```

Inizializza la console (video e tastiera)

Ripulisce il video (modalità testo) e setta il codice di attributo colore.

Parametri

cc	codice di attributo colore

7.2.2.5 readconsole()

```
natq readconsole (
            char * buff,
            natq quanti )
```

Lettura da tastiera.

Legge una riga da tastiera, terminata da a-capo. L'a-capo è sostituito con il terminatore di stringa.

Nota

La primitiva non legge mai più di *quanti* caratteri, quindi può restituire una stringa incompleta (non terminata) se l'a-capo non è ricevuto entro i primi *quanti* caratteri. Siccome il terminatore non è contato tra i caratteri ricevuti, il caso di stringa incompleta può essere riconosciuto perché è l'unico caso in cui il valore restituito è uguale a *quanti*.

Parametri

buff	buffer dove ricevere la riga
quanti	numero massimo di caratteri da leggere

Restituisce

numero di caratteri effettivamente letti, escluso l'a-capo

7.2.2.6 readhd_n()

```
void readhd_n (
            void * vetti,
            natl primo,
            natb quanti )
```

Lettura di settori dall'hard disk.

Parametri

vetti	buffer destinato a ricevere i dati letti
primo	LBA del primo settore da leggere
quanti	numero di settori da leggere

7.2.2.7 writeconsole()

```
void writeconsole (
            const char * buff,
            natq quanti )
```

Scrive caratteri sul video.

La scrittura avviene a partire dalla posizione corrente del cursore.

Parametri

buff	buffer contenente i caratteri da scrivere
quanti	numero di caratteri da scrivere

7.2.2.8 writehd_n()

```
void writehd_n (
            const void * vetto,
            natl primo,
            natb quanti )
```

Scrittura di settori sull'hard disk.

Parametri

vetto	buffer contenente i dati da scrivere
primo	LBA del primo settore da scrivere
quanti	numero di settori da scrivere

7.3 Riferimenti per il file include/sys.h

Primitive comuni definite dal modulo sistema.

Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:

Strutture dati

- struct meminfo

  Informazioni di debug.

Funzioni

- natl activate_p (void f(natq), natq a, natl prio, natl liv)

  Crea un nuovo processo.
- void terminate_p ()

  Termina il processo corrente.
- natl sem_ini (int val)

  Crea un nuovo semaforo.
- void sem_wait (natl sem)

  Estrae un gettone da un semaforo.
- void sem_signal (natl sem)

  Inserisce un gettone in un semaforo.
- void delay (natl n)

  Sospende il processo corrente.

Funzioni di supporto al debugging

- void do_log (log_sev sev, const char *buf, natl quanti)

  Invia un messaggio al log.
- meminfo getmeminfo ()

  Estrae informazioni di debug.

7.3.1 Descrizione dettagliata

Primitive comuni definite dal modulo sistema.

Queste primitive possono essere usate sia dal modulo utente che dal modulo I/O. Alcune sono usate anche dal modulo sistema stesso.

7.3.2 Documentazione delle funzioni

7.3.2.1 activate_p()

```
natl activate_p (
            void  fnatq,
            natq a,
            natl prio,
            natl liv )
```

Crea un nuovo processo.

Il nuovo processo eseguirà *f(a)* con priorità *prio* e a livello *liv*.

Un processo non può usare questa primitiva per creare un processo a priorità o livello maggiori dei propri.

Parametri

f	corpo del processo
a	parametro per il corpo del processo
prio	priorità del processo
liv	livello del processo (LIV_UTENTE o LIV_SISTEMA)

Restituisce

id del nuovo processo, o 0xFFFFFFFF in caso di errore

7.3.2.2 delay()

```
void delay (
            natl n )
```

Sospende il processo corrente.

Parametri

n	numero di intervalli di tempo

7.3.2.3 do_log()

```
void do_log (
            log_sev sev,
            const char * buf,
            natl quanti )
```

Invia un messaggio al log.

Questa primitiva è usata dai moduli I/O e utente per inviare i propri messaggi al log di sistema.

Parametri

sev	severità del messaggio
buf	buffer contenente il messaggio
quanti	lunghezza del messaggio

7.3.2.4 getmeminfo()

```
meminfo getmeminfo ( )
```

Estrae informazioni di debug.

Restituisce

struttura contenente le informazioni

7.3.2.5 sem_ini()

```
natl sem_ini (
             int val )
```

Crea un nuovo semaforo.

Parametri

val	numero di gettoni iniziali

Restituisce

id del nuovo semaforo, o 0xFFFFFFFF in caso di errore

7.3.2.6 sem_signal()

```
void sem_signal (
             natl sem )
```

Inserisce un gettone in un semaforo.

Parametri

sem	id del semaforo

7.3.2.7 sem_wait()

```
void sem_wait (
             natl sem )
```

Estrae un gettone da un semaforo.

Parametri

sem	id del semaforo.

7.3.2.8 terminate_p()

```
void terminate_p ( )
```

Termina il processo corrente.

I processi devono invocare questa primitiva per poter terminare.

7.4 Riferimenti per il file include/sysio.h

Primitive realizzate dal modulo sistema e riservate al modulo I/O.

Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:

Funzioni

- natl activate_pe (void f(int), int a, natl prio, natl liv, natb irq)
  Crea un processo esterno.
- void wfi ()
  Attende la prossima richiesta di interruzione.
- void abort_p ()
  Abortisce il processo corrente.
- void io_panic ()
  Errore fatale nel modulo I/O.
- paddr trasforma (void *ff)
  Traduzione da indirizzo virtuale a fisico.
- bool access (const void *start, natq dim, bool writeable, bool shared=true)
  Verifica dei problemi di Cavallo di Troia.
- bool fill_gate (natl tipo, vaddr f)
  Riempi un gate della IDT.

7.4.1 Descrizione dettagliata

Primitive realizzate dal modulo sistema e riservate al modulo I/O.

7.4.2 Documentazione delle funzioni

7.4.2.1 abort_p()

```
void abort_p ( )
```

Abortisce il processo corrente.

Usata dalle primitive di I/O quando rilevano un errore nei parametri ricevuti dall'utente.

7.4.2.2 access()

```
bool access (
            const void * start,
            natq dim,
            bool writeable,
            bool shared = true )
```

Verifica dei problemi di Cavallo di Troia.

La primitiva controlla che tutti gli indirizzi in [*start*, *start* + *dim*) siano accessibili da livello utente. In particolare, tutti gli indirizzi dell'intervallo devono essere mappati.

Parametri

start	base dell'intervallo da controllare
dim	dimensione in byte dell'intervallo da controllare
writeable	se true, l'intervallo deve essere anche scrivibile
shared	se true, l'intervallo deve trovarsi nella parte utente/condivisa

Restituisce

true se i vincoli sono rispettati, false altrimenti

7.4.2.3 activate_pe()

```
natl activate_pe (
            void  fint,
            int a,
            natl prio,
            natl liv,
            natb irq )
```

Crea un processo esterno.

La primitiva crea un processo esterno e lo associa ad una particolare richiesta di interruzione (IRQ).

Il processo eseguirà *f(a)* con priorità *prio* a livello *liv*.

Il tipo da associare alla richiesta *irq* è passato indirettamente tramite il parametro *prio*. In particolare, il tipo è ottenuto da *prio* sottraendo MIN_EXT_PRIO.

Parametri

f	corpo del processo
a	parametro per il corpo del processo
prio	priorità del processo
liv	livello del processo (LIV_UTENTE o LIV_SISTEMA)
irq	IRQ gestito dal processo

Restituisce

id del nuovo processo, o 0xFFFFFFFF in caso di errore

7.4.2.4 fill_gate()

```
bool fill_gate (
            natl tipo,
            vaddr f )
```

Riempi un gate della IDT.

Il gate sarà in ogni caso di tipo trap. Il gate non deve essere già occupato. Il tipo deve essere compreso tra 0x40 e 0x4F (inclusi).

Parametri

tipo	tipo del gate da riempire
f	funzione da associare al gate

Restituisce

false in caso di errore, true altrimenti

7.4.2.5 io_panic()

```
void io_panic ( )
```

Errore fatale nel modulo I/O.

Invocando questa primitiva il modulo I/O informa il modulo sistema di aver rilevato un errore fatale (per esempio, non è riuscito ad inizializzare la console).

7.4.2.6 trasforma()

```
paddr trasforma (
            void * ff )
```

Traduzione da indirizzo virtuale a fisico.

Usa il TRIE del processo corrente.

Parametri

ff	indirizzo virtuale da tradurre

Restituisce

indirizzo fisico corrispondente (0 se non mappato)

7.5 Riferimenti per il file io/io.cpp

Parte C++ del modulo I/O.

Grafo delle dipendenze di inclusione per io.cpp:

Strutture dati

- struct des_console

  Descrittore della console.
- struct des_ata

  Descrittore di interfaccia ATA.

Funzioni

- void * operator new (size_t s)

  Alloca un oggetto nello heap I/O.
- void * operator new (size_t s, std::align_val_t a)

  Alloca un oggetto nello heap I/O, con vincoli di allineamento.
- void operator delete (void *p)

  Dealloca un oggetto restituendolo all'heap I/O.
- void c_writeconsole (const char *buff, natq quanti)

  Parte C+++ della primitiva writeconsole()
- void startkbd_in (des_console *d, char *buff, natq dim)

  Avvia una operazione di lettura dalla tastiera.
- natq c_readconsole (char *buff, natq quanti)

  Parte C++ della primitiva readconsole()
- void estern_kbd (int)

  Processo esterno associato alla tastiera.
- void c_iniconsole (natb cc)

  Parte C++ della primitiva iniconsole()
- bool kbd_init ()

  Inizializza la tastiera.
- bool vid_init ()

  Inizializza il video (modalità testo)
- bool console_init ()

  Inizializza la console (tastiera + video)
- bool prepare_prd (natb *vett, natb quanti)

  Prepara i descrittori per il Bus Mastering.
- void starthd_in (des_ata *d, natb vetti[ ], natl primo, natb quanti)

  Avvia una operazione di ingresso dall'hard disk.
- void c_readhd_n (natb vetti[ ], natl primo, natb quanti)

  Parte C++ della primitiva readhd_n().
- void starthd_out (des_ata *d, natb vetto[ ], natl primo, natb quanti)

  Avvia una operazione di uscita verso l'hard disk.
- void c_writehd_n (natb vetto[ ], natl primo, natb quanti)

  Parte C++ della primitiva writehd_n().
- void dmastarthd_in (des_ata *d, natb vetti[ ], natl primo, natb quanti)

  Avvia una operazione di ingresso in DMA dall'hard disk.
- void c_dmareadhd_n (natb vetti[ ], natl primo, natb quanti)

  Parte C++ della primitiva dmareadhd_n().
- void dmastarthd_out (des_ata *d, natb vetto[ ], natl primo, natb quanti)

  Avvia una operazione di uscita in DMA verso l'hard disk.
- void c_dmawritehd_n (natb vetto[ ], natl primo, natb quanti)

  Parte C++ della primitiva dmawritehd_n().
- void estern_hd (int)

Processo esterno per le richieste di interruzione dell'hard disk.

- bool hd_init ()

  Inizializza la gestione dell'hard disk.
- bool fill_io_gates ()

  Riempie i gate della IDT relativi alle primitive fornite dal modulo I/O.
- void main (natq sem_io)

  Corpo del processo main I/O.
- void panic (const char *msg)

  Segnala un errore fatale nel modulo I/O.
- natq c_getiomeminfo ()

  Parte C++ della primitiva getiomeminfo()

Variabili

- natl ioheap_mutex

  Indice del semaforo di mutua esclusione per lo heap I/O.
- des_console console

  Unica istanza di des_console.
- const int KBD_IRQ = 1

  Piedino dell'APIC per le richieste di interruzione della tastiera.
- des_ata hard_disk

  Descrittore dell'unico hard disk installato nel sistema.
- natl hd_prd [MAX_PRD *2]

  Array dei descrittori per il Bus Mastering.
- const natb HD_IRQ = 14

  Piedino dell'APIC per le richieste di interruzione dell'hard disk.
- char end [ ]

  Ultimo indirizzo utilizzato dal modulo I/O (fornito dal collegatore)

7.5.1 Descrizione dettagliata

Parte C++ del modulo I/O.

7.6 Riferimenti per il file sistema/sistema.cpp

Parte C++ del modulo sistema.

Grafo delle dipendenze di inclusione per sistema.cpp:

Strutture dati

- struct des_proc

  Descrittore di processo.
- struct des_sem

  Descrittore di semaforo.
- struct richiesta

  Richiesta al timer.
- struct des_frame

  Descrittore di frame.
- struct copy_segment

  Oggetto da usare con map() per caricare un segmento ELF in memoria virtuale.

Tipi enumerati (enum)

- enum {
  I_RAX , **I_RCX** , **I_RDX** , **I_RBX** ,
  I_RSP , **I_RBP** , **I_RSI** , **I_RDI** ,
  I_R8 , **I_R9** , **I_R10** , **I_R11** ,
  I_R12 , **I_R13** , **I_R14** , **I_R15** }

  Indici delle copie dei registri nell'array contesto.

Funzioni

- void ∗ operator new (size_t size)

  Alloca nello heap.
- void ∗ operator new (size_t size, std::align_val_t align)

  Alloca nello heap con allienamento.
- void operator delete (void ∗p)

  Dealloca un oggetto restituendolo allo heap.
- void inserimento_lista (des_proc ∗&p_lista, des_proc ∗p_elem)

  Inserimento in lista ordinato (per priorità)
- des_proc ∗ rimozione_lista (des_proc ∗&p_lista)

  Estrazione del processo a maggiore priorità
- void inspronti ()

  Inserisce esecuzione in testa alla lista pronti.
- void schedulatore (void)

  Sceglie il prossimo processo da mettere in esecuzione.
- des_proc ∗ des_p (natw id)

  Trova il descrittore di processo dato l'id.
- void dummy (natq)

  Corpo del processo dummy.
- int liv_chiamante ()

  Restituisce il livello a cui si trovava il processore al momento in cui è stata invocata la primitiva.
- natl alloca_sem ()

  Alloca un nuovo semaforo.
- bool sem_valido (natl sem)

  Verifica un id di semaforo.
- void c_sem_ini (int val)

  Parte C++ della primitiva sem_ini().
- void c_sem_wait (natl sem)

  Parte C++ della primitiva sem_wait().
- void c_sem_signal (natl sem)

  Parte C++ della primitiva sem_signal().
- void inserimento_lista_attesa (richiesta ∗p)

  Inserisce un processo nella coda delle richieste al timer.
- void c_delay (natl n)

  Parte C++ della primitiva delay.
- void c_driver_td (void)

  Driver del timer.
- void gestore_eccezioni (int tipo, natq errore, vaddr rip)

  Gestore generico di eccezioni.
- void init_frame ()

  Inizializza la parte M2 e i descrittori di frame.

- paddr alloca_frame ()

  Estrae un frame dalla lista dei frame liberi.
- void rilascia_frame (paddr f)

  Restiuisce un frame alla listsa dei frame liberi.
- paddr alloca_tab ()

  Alloca un frame libero destinato a contenere una tabella.
- void rilascia_tab (paddr f)

  Dealloca un frame che contiene una tabella.
- void inc_ref (paddr f)

  Incrementa il contatore delle entrate valide di una tabella.
- void dec_ref (paddr f)

  Decrementa il contatore delle entrate valide di una tabella.
- natl get_ref (paddr f)

  Legge il contatore delle entrate valide di una tabella.
- bool in_utn_c (vaddr v)

  Controlla che un indirizzo appartenga alla zona utente/condivisa.
- bool c_access (vaddr begin, natq dim, bool writeable, bool shared=true)

  Parte C++ della primitiva access()
- void c_trasforma (vaddr ind_virt)

  Parte C++ della primitiva trasforma()
- natl crea_dummy ()

  Crea il processo dummy.
- void main_sistema (natq)

  Corpo del processo main_sistema.
- natl crea_main_sistema ()

  Crea il processo main_sistema.
- bool crea_spazio_condiviso (paddr root_tab, boot64_modinfo *mod)

  Crea le parti utente/condivisa e io/condivisa.
- void salta_a_main ()

  Funzione di supporto pr avviare il primo processo (definita in sistema.s)
- void c_fill_gate (natb tipo, void routine(), int liv)

  Parte C++ della primitiva fill_gate().
- void main (boot64_info *info)

  Prima parte dell'inizializzazione; crea i primi processi.
- vaddr carica_modulo (boot64_modinfo *mod, paddr root_tab, natq flags, natq heap_size)

  Carica un modulo in M2.
- vaddr carica_IO (boot64_modinfo *mod, paddr root_tab)

  Mappa il modulo I/O.
- vaddr carica_utente (boot64_modinfo *mod, paddr root_tab)

  Mappa il modulo utente.
- void panic (const char *msg)

  Ferma il sistema e stampa lo stato di tutti i processi.
- void c_io_panic ()

  Parte C++ della primitiva io_panic()
- void c_nmi ()

  Routine di risposta a un non-maskable-interrupt.
- void c_do_log (log_sev sev, const char *buf, natl quanti)

  Parte C++ della primitiva do_log().
- void c_getmeminfo ()

  Parte C++ della primitiva getmeminfo().

Funzioni usate dal processo dummy

- void end_program ()
  Esegue lo shutdown del sistema.
- void halt ()
  Esegue l'istruzione `hlt`.

Funzioni di supporto alla creazione e distruzione dei processi

- natl alloca_proc_id (des_proc *p)
  Alloca un id di processo.
- void rilascia_proc_id (natw id)
  Rilascia un id di processo non più utilizzato.
- void init_root_tab (paddr dest)
  Inizializza la tabella radice di un nuovo processo.
- void clear_root_tab (paddr dest)
  Ripulisce la tabella radice di un processo.
- bool crea_pila (paddr root_tab, vaddr bottom, natq size, natl liv)
  Crea una pila processo.
- void distruggi_pila (paddr root_tab, vaddr bottom, natq size)
  Distrugge una pila processo.
- des_proc * crea_processo (void f(natq), natq a, int prio, char liv)
  Funzione interna per la creazione di un processo.
- void distruggi_processo (des_proc *p)
  Dealloca tutte le risorse allocate da crea_processo()
- bool load_handler (natq tipo, natq irq)
  Carica un handler nella IDT.

Primitive per la creazione e distruzione dei processi (parte C++)

- void c_activate_p (void f(natq), natq a, natl prio, natl liv)
  Parte C++ della primitiva activate_p()
- void c_terminate_p (bool logmsg)
  Parte C++ della pritimiva terminate_p()
- void c_abort_p (bool selfdump)
  Parte C++ della primitiva abort_p()
- void c_activate_pe (void f(natq), natq a, natl prio, natl liv, natb irq)
  Parte C++ della primitiva activate_pe().

Funzioni di supporto per il backtrace

- natq read_mem (void *token, vaddr v)
  Callback invocata dalla funzione cfi_backstep() per leggere dalla pila di un qualunque processo.
- void backtrace (des_proc *p, log_sev sev, const char *msg)
  Invia sul log il backtrace di un processo.
- void process_dump (des_proc *p, log_sev sev)
  Invia sul log lo stato di un processo.

Variabili

- const natl DUMMY_PRIORITY = 0

  Priorità del processo dummy.
- const int N_REG = 16

  Numero di registri nel campo contesto del descrittore di processo.
- des_proc * proc_table [MAX_PROC]

  Tabella che associa l'id di un processo al corrispondente des_proc.
- natl processi

  Numero di processi utente attivi.
- des_proc * esecuzione

  Coda esecuzione (contiene sempre un solo elemento)
- des_proc * pronti

  Coda pronti (vuota solo quando dummy è in esecuzione)
- des_sem array_dess [MAX_SEM *2]

  Array dei descrittori di semaforo.
- natl sem_allocati_utente = 0

  Numero di semafori allocati per il livello utente.
- natl sem_allocati_sistema = 0

  Numero di semafori allocati per il livello sistema (moduli sistema e I/O)
- richiesta * sospesi

  Coda dei processi sospesi.
- natb start [ ]

  Primo indirizzo del codice di sistema.
- natb end [ ]

  Ultimo indirizzo del codice sistema (fornito dal collegatore)
- natq const N_FRAME = MEM_TOT / DIM_PAGINA

  Numero totale di frame (M1 + M2)
- natq N_M1

  Numero di frame in M1.
- natq N_M2

  Numero di frame in M2.
- des_frame vdf [N_FRAME]

  Array dei descrittori di frame.
- natq primo_frame_libero

  Testa della lista dei frame liberi.
- natq num_frame_liberi

  Numero di frame nella lista dei frame liberi.
- static const natq PART_SIZE = dim_region(MAX_LIV - 1)

  Granularità delle parti della memoria virtuale.
- const vaddr ini_sis_c = norm(I_SIS_C * PART_SIZE)

  base di sistema/condivisa
- const vaddr ini_sis_p = norm(I_SIS_P * PART_SIZE)

  base di sistema/privata
- const vaddr ini_mio_c = norm(I_MIO_C * PART_SIZE)

  base di modulo IO/condivisa
- const vaddr ini_utn_c = norm(I_UTN_C * PART_SIZE)

  base di utente/condivisa
- const vaddr ini_utn_p = norm(I_UTN_P * PART_SIZE)

  base di utente/privata
- const vaddr fin_sis_c = ini_sis_c + PART_SIZE * N_SIS_C

limite di sistema/condivisa

- const vaddr fin_sis_p = ini_sis_p + PART_SIZE ∗ N_SIS_P

  limite di sistema/privata
- const vaddr fin_mio_c = ini_mio_c + PART_SIZE ∗ N_MIO_C

  limite di modulo IO/condivisa
- const vaddr fin_utn_c = ini_utn_c + PART_SIZE ∗ N_UTN_C

  limite di utente/condivisa
- const vaddr fin_utn_p = ini_utn_p + PART_SIZE ∗ N_UTN_P

  limite di utente/privata
- des_proc ∗ a_p [apic::MAX_IRQ]

  Associazione IRQ -> processo esterno che lo gestisce.
- des_proc ∗const ESTERN_BUSY = reinterpret_cast<des_proc∗>(1UL)

  Valore da inserire in a_p per gli IRQ che sono gestiti da driver.
- des_proc init

  Un primo des_proc, allocato staticamente, da usare durante l'inizializzazione.
- const natl DELAY = 59659

  Periodo del timer di sistema.
- paddr tss_punt_nucleo

  Indirizzo fisico del puntatore alla pila sistema nel segmento TSS.
- void(∗ io_entry)(natq)

  Entry point del modulo IO.
- void(∗ user_entry)(natq)

  Entry point del modulo utente.
- int MAX_LOG = 5

  Massimo livello ammesso per la severità dei messaggi del log.

Costanti per lo heap di sistema

- const natq HEAP_START = 1∗MiB

  Indirizzo base dello heap di sistema.
- const natq HEAP_SIZE = 1∗MiB

  Dimensione dello heap di sistema.

Distruzione della pila sistema corrente

Quando dobbiamo eliminare una pila sistema dobbiamo stare attenti a non eliminare proprio quella che stiamo usando. Questo succede durante una terminate_p() o abort_p(), quando si tenta di distrugguere proprio il processo che ha invocato la primitiva.

Per fortuna, se stiamo terminando il processo corrente, vuol dire anche che stiamo per metterne in esecuzione un altro e possiamo dunque usare la pila sistema di quest'ultimo. Operiamo dunque nel seguente modo:

- all'ingresso nel sistema (in salva_stato), salviamo il valore di esecuzione in esecuzione_precedente; questo è il processo a cui appartiene la pila sistema che stiamo usando;
- in distruggi_processo(), se esecuzione è uguale a esecuzione_precedente (stiamo distruggendo proprio il processo a cui appartiene la pila corrente), *non* distruggiamo la pila sistema e settiamo la variabile ultimo_terminato;
- in carica_stato, dopo aver cambiato pila, se ultimo_terminato è settato, distruggiamo la pila sistema di esecuzione_precedente.

- des_proc ∗ esecuzione_precedente

  Processo che era in esecuzione all'entrata nel modulo sistema.
- paddr ultimo_terminato

  Se diverso da zero, indirizzo fisico della root_tab dell'ultimo processo terminato o abortito.
- void distruggi_pila_precedente ()

  Distrugge la pila sistema del processo uscente e rilascia la sua tabella radice.

7.6.1 Descrizione dettagliata

Parte C++ del modulo sistema.

7.7 Riferimenti per il file utente/all.h

File da includere nei programmi utente.

Grafo delle dipendenze di inclusione per all.h:

7.8 Riferimenti per il file utente/lib.cpp

Libreria collegata con i programmi utente.

Grafo delle dipendenze di inclusione per lib.cpp:

Funzioni

- int printf (const char *fmt,...) __attribute__((format(printf
  Formatta un messaggio e lo scrive sul video.
- void pause ()
  Attende la pressione di un carattere.
- natl getpid ()
  Id del processo corrente.
- void * operator new (size_t s)
  alloca un oggetto nello heap utente
- void operator delete (void *p)
  dealloca un oggetto restituendolo allo heap utente.
- void panic (const char *msg)
  Termina il processo corrente.
- void lib_init () __attribute__((constructor))
  Inizializza la libreria utente.

Variabili

- char pause_buf [1]
  buffer della funzione pause()
- natl userheap_mutex
  Semaforo di mutua esclusione per lo heap utente.
- natb end [ ]
  ultimo indirizzo usato dal modulo utente (fornito dal collegatore)

7.8.1 Descrizione dettagliata

Libreria collegata con i programmi utente.

7.9 Riferimenti per il file utente/lib.h

Funzioni di libreria per il modulo utente.

Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:

Funzioni

- int printf (const char *fmt,...) __attribute__((format(printf

  Formatta un messaggio e lo scrive sul video.
- void pause ()

  Attende la pressione di un carattere.
- natl getpid ()

  Id del processo corrente.

7.9.1 Descrizione dettagliata

Funzioni di libreria per il modulo utente.

Indice analitico