

libce

Generato da Doxygen 1.9.1

1 Pagina Principale	1
2 Indice dei moduli	3
2.1 Moduli	3
3 Indice dei namespace	5
3.1 Lista dei namespace	5
4 Indice delle strutture dati	7
4.1 Strutture dati	7
5 Indice dei file	9
5.1 Elenco dei file	9
6 Documentazione dei moduli	15
6.1 Costanti	15
6.1.1 Descrizione dettagliata	15
6.2 Tipi base	15
6.2.1 Descrizione dettagliata	16
6.3 Tipi dipendenti dall'architettura	16
6.3.1 Descrizione dettagliata	16
6.4 Funzioni di utilità generale	17
6.4.1 Descrizione dettagliata	18
6.4.2 Documentazione dei tipi enumerati	18
6.4.2.1 log_sev	18
6.4.3 Documentazione delle funzioni	18
6.4.3.1 flog()	18
6.4.3.2 fpanic()	19
6.4.3.3 int_cast()	19
6.4.3.4 max()	20
6.4.3.5 memcpy()	20
6.4.3.6 memset()	21
6.4.3.7 min()	21
6.4.3.8 printf()	22
6.4.3.9 ptr_cast()	22
6.4.3.10 setseed()	23
6.4.3.11 snprintf()	23
6.4.3.12 strlen()	24
6.4.3.13 voidptr_cast()	24
6.5 Funzioni per la memoria dinamica	25
6.5.1 Descrizione dettagliata	25
6.5.2 Documentazione delle funzioni	26
6.5.2.1 alloc()	26
6.5.2.2 alloc_aligned()	26

6.5.2.3 dealloc()	27
6.5.2.4 disponibile()	27
6.5.2.5 heap_init()	27
6.6 Funzioni per l'interazione con le interfacce di I/O.	28
6.6.1 Descrizione dettagliata	28
6.7 Funzioni per leggere/scrivere nello spazio di I/O.	28
6.7.1 Descrizione dettagliata	28
6.7.2 Documentazione delle funzioni	28
6.7.2.1 inputb()	28
6.7.2.2 inputbw()	29
6.7.2.3 inputl()	29
6.7.2.4 inputw()	29
6.7.2.5 outputb()	31
6.7.2.6 outputbw()	31
6.7.2.7 outputl()	31
6.7.2.8 outputw()	32
6.8 Funzioni legate al meccanismo delle interruzioni.	32
6.8.1 Descrizione dettagliata	32
6.8.2 Documentazione delle funzioni	33
6.8.2.1 apic_send_EOI()	33
6.8.3 Documentazione delle variabili	33
6.8.3.1 BIT_IF	33
6.8.3.2 BIT_TF	33
6.9 Funzioni per la manipolazione della tabella IDT.	33
6.9.1 Descrizione dettagliata	34
6.9.2 Documentazione delle funzioni	34
6.9.2.1 gate_init()	34
6.9.2.2 gate_present()	34
6.10 Eccezioni.	35
6.10.1 Descrizione dettagliata	35
6.10.2 Documentazione delle funzioni	35
6.10.2.1 log_exception()	35
6.10.3 Documentazione delle variabili	36
6.10.3.1 PF_PROT	36
6.10.3.2 PF_RES	36
6.10.3.3 PF_USER	36
6.10.3.4 PF_WRITE	37
6.10.3.5 SE_EXT	37
6.10.3.6 SE_IDT	37
6.10.3.7 SE_TI	37
6.11 Funzioni di uso meno generale, usate dal nucleo.	38
6.11.1 Descrizione dettagliata	38

6.11.2 Documentazione delle funzioni	38
6.11.2.1 do_log()	38
6.11.2.2 find_eh_frame()	38
6.11.2.3 reboot()	39
6.11.2.4 vsnprintf()	39
6.12 Costanti usate da boot64.	40
6.12.1 Descrizione dettagliata	40
6.12.2 Documentazione delle definizioni	40
6.12.2.1 NUM_GDT_ENTRIES	41
6.13 Funzioni per la memoria virtuale.	41
6.13.1 Descrizione dettagliata	43
6.13.2 Documentazione delle funzioni	43
6.13.2.1 base()	43
6.13.2.2 copy_des()	44
6.13.2.3 dim_region()	44
6.13.2.4 extr_IND_FISICO()	45
6.13.2.5 get_entry()	45
6.13.2.6 i_tab()	45
6.13.2.7 invalida_entrata_TLB()	46
6.13.2.8 limit()	46
6.13.2.9 loadCR3()	47
6.13.2.10 map()	47
6.13.2.11 norm()	48
6.13.2.12 readCR2()	48
6.13.2.13 readCR3()	48
6.13.2.14 set_des()	49
6.13.2.15 set_entry()	49
6.13.2.16 set_IND_FISICO()	50
6.13.2.17 trasforma()	50
6.13.2.18 unmap()	50
6.13.3 Documentazione delle variabili	52
6.13.3.1 VADDR_MSBIT	52
7 Documentazione dei namespace	53
7.1 Riferimenti per il namespace apic	53
7.1.1 Descrizione dettagliata	53
7.1.2 Documentazione delle funzioni	53
7.1.2.1 init()	54
7.1.2.2 set_MIRQ()	54
7.1.2.3 set_TRGM()	54
7.1.2.4 set_VECT()	55
7.2 Riferimenti per il namespace bm	55

7.2.1 Descrizione dettagliata	55
7.2.2 Documentazione delle funzioni	55
7.2.2.1 find()	56
7.2.2.2 init()	56
7.2.2.3 prepare()	56
7.3 Riferimenti per il namespace hd	57
7.3.1 Descrizione dettagliata	57
7.3.2 Documentazione dei tipi enumerati	57
7.3.2.1 cmd	57
7.3.3 Documentazione delle funzioni	58
7.3.3.1 input_sect()	58
7.3.3.2 output_sect()	58
7.3.3.3 start_cmd()	58
7.4 Riferimenti per il namespace kbd	59
7.4.1 Descrizione dettagliata	59
7.4.2 Documentazione delle funzioni	59
7.4.2.1 char_read()	60
7.4.2.2 char_read_intr()	60
7.4.2.3 conv()	60
7.4.2.4 get_code()	61
7.5 Riferimenti per il namespace pci	61
7.5.1 Descrizione dettagliata	61
7.5.2 Documentazione delle funzioni	62
7.5.2.1 decode_class()	62
7.5.2.2 find_class()	62
7.5.2.3 find_dev()	63
7.5.2.4 next()	63
7.5.2.5 read_confb()	64
7.5.2.6 read_confl()	64
7.5.2.7 read_confw()	65
7.5.2.8 write_confb()	65
7.5.2.9 write_confl()	66
7.5.2.10 write_confw()	66
7.6 Riferimenti per il namespace serial	67
7.6.1 Descrizione dettagliata	67
7.6.2 Documentazione delle funzioni	67
7.6.2.1 in1()	67
7.6.2.2 in2()	68
7.6.2.3 out1()	68
7.6.2.4 out2()	68
7.7 Riferimenti per il namespace std	68
7.7.1 Descrizione dettagliata	68

7.8 Riferimenti per il namespace svga	69
7.8.1 Descrizione dettagliata	69
7.8.2 Documentazione delle funzioni	69
7.8.2.1 config()	69
7.9 Riferimenti per il namespace timer	69
7.9.1 Descrizione dettagliata	70
7.9.2 Documentazione delle funzioni	70
7.9.2.1 start0()	70
7.9.2.2 start2()	70
7.10 Riferimenti per il namespace vid	71
7.10.1 Descrizione dettagliata	71
7.10.2 Documentazione delle funzioni	71
7.10.2.1 char_put()	71
7.10.2.2 char_write()	72
7.10.2.3 clear()	72
7.10.2.4 cols()	72
7.10.2.5 pos()	73
7.10.2.6 rows()	73
7.10.2.7 str_write()	73
8 Documentazione delle classi	75
8.1 Riferimenti per la classe tab_iter	75
8.1.1 Descrizione dettagliata	76
8.1.2 Documentazione dei costruttori e dei distruttori	76
8.1.2.1 tab_iter()	76
8.1.3 Documentazione delle funzioni membro	76
8.1.3.1 down()	76
8.1.3.2 get_e()	77
8.1.3.3 get_l()	77
8.1.3.4 get_tab()	77
8.1.3.5 get_v()	77
8.1.3.6 is_leaf()	78
8.1.3.7 operator bool()	78
8.1.3.8 right()	78
8.1.3.9 up()	78
8.1.3.10 valid_interval()	78
9 Documentazione dei file	81
9.1 Riferimenti per il file include/libce.h	81
9.1.1 Descrizione dettagliata	88
9.2 Riferimenti per il file include/vm.h	88
9.2.1 Descrizione dettagliata	90

Capitolo 1

Pagina Principale

Questa libreria contiene alcune funzioni per l'accesso a basso livello ad una macchina PC-compatibile (in particolare, una macchina QEMU con processore Intel/AMD a 64 bit, bus PCI a 32 bit e periferiche ISA).

La libreria è usata per i seguenti scopi durante il corso di Calcolatori Elettronici:

- per supportare un boot loader (*boot64*, incluso nella libreria stessa) che porta il processore nella modalità a 64 bit partendo dal modo protetto a 32 bit; in questo caso la libreria è compilata a 32 bit;
- per supportare i moduli del *nucleo* introdotto nella seconda parte del corso (sia il modulo *sistema*, sia i moduli *io* e *utente*);
- per supportare programmi che accedono a basso livello senza usare il nucleo. Questi programmi sono detti "bare" (nudi) nella documentazione, e durante il corso si distinguono in:
 - *esempiIO*, piccoli programmi che sono utilizzati per spiegare il funzionamento delle periferiche ISA, del bus PCI e delle interruzioni;
 - esercizi di traduzione da C++ ad Assembler (soprattutto nel caso in cui la macchina ospite non sia Intel/AMD, ma sia ad esempio un Mac M1/M2/...);

Si noti che *boot64* viene usato per il bootstrap sia del nucleo che dei programmi bare.

I file nella directory *bare* sono invece specifici dei programmi bare e non sono usati dai moduli del nucleo.

Per compilare la libreria è sufficiente eseguire *make*. Si otterranno i seguenti file:

- *boot.bin* (il bootloader *boot64*)
- *libce32.a* (libreria compilata a 32 bit, usata per costruire *boot.bin*)
- *libce64.a* (libreria compilata a 64 bit, usata per tutto il resto)

I file oggetto della libreria si trovano in *build*, suddivisi in varie sottodirectory.

La libreria contiene anche gli script *compile*, *boot* e *debug* e un file *libce-debug.py* che contiene alcune estensioni per *gdb*, usate da *debug*.

Capitolo 2

Indice dei moduli

2.1 Moduli

Questo è l'elenco di tutti i moduli:

Costanti	15
Tipi base	15
Tipi dipendenti dall'architettura	16
Funzioni di utilità generale	17
Funzioni per la memoria dinamica.	25
Funzioni per l'interazione con le interfacce di I/O.	28
Funzioni per leggere/scrivere nello spazio di I/O.	28
Funzioni legate al meccanismo delle interruzioni.	32
Funzioni per la manipolazione della tabella IDT.	33
Eccezioni.	35
Funzioni di uso meno generale, usate dal nucleo.	38
Costanti usate da boot64.	40
Funzioni per la memoria virtuale.	41

Capitolo 3

Indice dei namespace

3.1 Lista dei namespace

Questa è l'elenco dei namespace documentati, con una loro breve descrizione:

apic	Funzioni per interagire con l'APIC	53
bm	Funzioni per il PCI BUS Mastering ATA	55
hd	Funzioni per interagire con l'hard disk	57
kbd	Funzioni per interagire con la tastiera	59
pci	Funzioni per il bus PCI	61
serial	Funzioni per interagire con le due porte seriali	67
std	Tipo standard per la definizione della new con allineamento	68
svga	Funzioni per interagire con il video in modalità grafica	69
timer	Funzioni per interagire con il timer	69
vid	Funzioni per interagire con il video in modalità testo	71

Capitolo 4

Indice delle strutture dati

4.1 Strutture dati

Queste sono le strutture dati con una loro breve descrizione:

tab_iter	Iteratore per la visita di un TRIE	75
--------------------------	--	--------------------

Capitolo 5

Indice dei file

5.1 Elenco dei file

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

internal.h	??
apic/in.cpp	??
apic/init.cpp	??
apic/IOREGSEL-IOWIN.cpp	??
apic/out.cpp	??
apic/read_irr.cpp	??
apic/read_isr.cpp	??
apic/read_rth.cpp	??
apic/read_rtl.cpp	??
apic/send_EOI.cpp	??
apic/set_MIRQ.cpp	??
apic/set_TRGM.cpp	??
apic/set_VECT.cpp	??
apic/write_rth.cpp	??
apic/write_rtl.cpp	??
as32/disable_8259.s	??
as32/inputb.s	??
as32/inputbw.s	??
as32/inputl.s	??
as32/inputw.s	??
as32/loadCR3.s	??
as32/outputb.s	??
as32/outputbw.s	??
as32/outputl.s	??
as32/outputw.s	??
as32/readCR3.s	??
as64/_dso_handle.s	??
as64/cli.s	??
as64/componi_gate.s	??
as64/ctors.s	??
as64/idt_pointer.s	??
as64/ignore_pic.s	??
as64/inputb.s	??
as64/inputbw.s	??
as64/inputl.s	??

as64/inputw.s	??
as64/invalida_entrata_TLB.s	??
as64/invalida_TLB.s	??
as64/loadCR3.s	??
as64/outputb.s	??
as64/outputbw.s	??
as64/outputl.s	??
as64/outputw.s	??
as64/readCR2.s	??
as64/readCR3.s	??
as64/sti.s	??
bare/alloca_tab.cpp	??
bare/autocorr_printf.cpp	??
bare/cout.cpp	??
bare/dec_ref.cpp	??
bare/delete.cpp	??
bare/endl.cpp	??
bare/esecuzione.cpp	??
bare/exc_ac.s	??
bare/exc_bound_re.s	??
bare/exc_breakpoint.s	??
bare/exc_coproc_so.s	??
bare/exc_debug.s	??
bare/exc_dev_na.s	??
bare/exc_divide_error.s	??
bare/exc_double_fault.s	??
bare/exc_fp.s	??
bare/exc_int_tipo_pf.s	??
bare/exc_invalid_opcode.s	??
bare/exc_invalid_tss.s	??
bare/exc_mc.s	??
bare/exc_nmi.s	??
bare/exc_overflow.s	??
bare/exc_prot_fault.s	??
bare/exc_segm_fault.s	??
bare/exc_simd.s	??
bare/exc_stack_fault.s	??
bare/exc_unknown.s	??
bare/gestore_eccezioni.cpp	??
bare/get_ref.cpp	??
bare/inc_ref.cpp	??
bare/init_all.cpp	??
bare/init_idt.s	??
bare/new.cpp	??
bare/panic.cpp	??
bare/pause_caller.cpp	??
bare/rilascia_tab.cpp	??
bare/stack_end.s	??
bare/start64.s	??
bm/ack.cpp	??
bm/bm.cpp	??
bm/find.cpp	??
bm/init.cpp	??
bm/prepare.cpp	??
bm/start.cpp	??
boot64/boot.cpp	??
boot64/boot.s	??
boot64/mboot.h	??

cfi/backstep.cpp	??
cfi/cfi_internal.h	??
cfi/find_eh_frame.cpp	??
cfi/interp_exec.cpp	??
cfi/interp_init.cpp	??
cfi/interp_jump.cpp	??
cfi/interp_save_cie.cpp	??
cfi/interp_snapshot.cpp	??
cfi/read_sleb128.cpp	??
cfi/read_uleb128.cpp	??
exc/eccezioni.cpp	??
exc/log_exception.cpp	??
hd/ack_intr.cpp	??
hd/disable_intr.cpp	??
hd/enable_intr.cpp	??
hd/input.sect.cpp	??
hd/output.sect.cpp	??
hd/set_lba.cpp	??
hd/start_cmd.cpp	??
hd/wait_for_br.cpp	??
heap/alloc.cpp	??
heap/alloc_aligned.cpp	??
heap/dealloc.cpp	??
heap/delete_array.cpp	??
heap/delete_size.cpp	??
heap/delete_size_align.cpp	??
heap/disponibile.cpp	??
heap/free_interna.cpp	??
heap/heap_init.cpp	??
heap/memlibera.cpp	??
heap/new_aligned.cpp	??
heap/new_array.cpp	??
include/boot.h	??
include/cfi.h	??
include/elf64.h	??
include/libce.h	??
File che va incluso sempre per primo	81
include/vm.h	??
File da includere (dopo libce.h) per usare le funzioni della memoria virtuale	88
kbd/char_read.cpp	??
kbd/char_read_intr.cpp	??
kbd/conv.cpp	??
kbd/disable_intr.cpp	??
kbd/drain.cpp	??
kbd/enable_intr.cpp	??
kbd/get_code.cpp	??
kbd/init.cpp	??
kbd/reboot.cpp	??
kbd/reset.cpp	??
kbd/shift.cpp	??
kbd/tab.cpp	??
kbd/tabmai.cpp	??
kbd/tabmin.cpp	??
pci/decode_class.cpp	??
pci/find_class.cpp	??
pci/find_dev.cpp	??
pci/make_CAP.cpp	??
pci/next.cpp	??

pci/read_confb.cpp	??
pci/read_confl.cpp	??
pci/read_confw.cpp	??
pci/write_confb.cpp	??
pci/write_confl.cpp	??
pci/write_confw.cpp	??
piix3/ioapic_disable.cpp	??
piix3/ioapic_enable.cpp	??
serial/in1.cpp	??
serial/in2.cpp	??
serial/init1.cpp	??
serial/init2.cpp	??
serial/out1.cpp	??
serial/out2.cpp	??
svga/config.cpp	??
svga/framebuffer.cpp	??
svga/init.cpp	??
svga/reg.cpp	??
tab_iter/down.cpp	??
tab_iter/next.cpp	??
tab_iter/next_post.cpp	??
tab_iter/post.cpp	??
tab_iter/right.cpp	??
tab_iter/tab_iter.cpp	??
tab_iter/up.cpp	??
timer/disable_spk.cpp	??
timer/enable_spk.cpp	??
timer/start0.cpp	??
timer/start2.cpp	??
util/__cxa_atexit.cpp	??
util/arith64.cpp	??
util/do_log.cpp	??
util/flog.cpp	??
util/fpanic.cpp	??
util/gate_init.cpp	??
util/gate_present.cpp	??
util/idt.cpp	??
util/index.cpp	??
util/MAX_LOG.cpp	??
util/memcpy.cpp	??
util/memset.cpp	??
util/pause.cpp	??
util/printf.cpp	??
util/puts.cpp	??
util/random.cpp	??
util/seed.cpp	??
util/setseed.cpp	??
util/snprintf.cpp	??
util/strlen.cpp	??
util/strnlen.cpp	??
util/vsnprintf.cpp	??
vid/attr.cpp	??
vid/char_put.cpp	??
vid/char_write.cpp	??
vid/clear.cpp	??
vid/cols.cpp	??
vid/cursor.cpp	??
vid/pos.cpp	??

vid/rows.cpp	??
vid/scroll.cpp	??
vid/str_write.cpp	??
vid/video.cpp	??
vid/xy.cpp	??
vm/copy_des.cpp	??
vm/set_des.cpp	??
vm/set_entry.cpp	??
vm/trasforma.cpp	??

Capitolo 6

Documentazione dei moduli

6.1 Costanti

Definizioni

- #define [KiB](#) 1024ULL
kibibyte
- #define [MiB](#) (1024*[KiB](#))
mibibyte
- #define [GiB](#) (1024*[MiB](#))
gibibyte
- #define [DIM_PAGINA](#) (4*[KiB](#))
dimensione in byte di una pagina o di un frame
- #define [DIM_BLOCK](#) 512ULL
dimensione in byte di un blocco dell'hard disk
- #define [LIV_SISTEMA](#) 0
DPL del livello sistema.
- #define [LIV_UTENTE](#) 3
DPL del livello utente.

6.1.1 Descrizione dettagliata

Alcune costanti di uso comune. Usiamo delle macro in modo da poterle usare anche in assembler.

6.2 Tipi base

Moduli

- [Tipi dipendenti dall'architettura](#)

Ridefinizioni di tipo (typedef)

- using `ioaddr` = unsigned short
indirizzo nello spazio di I/O
- using `natb` = unsigned char
naturale su un byte
- using `natw` = unsigned short
naturale su due byte
- using `natl` = unsigned int
naturale su 4 byte

6.2.1 Descrizione dettagliata

Alcuni tipi di uso comune, più altri la cui definizione è richiesta dallo standard.

6.3 Tipi dipendenti dall'architettura

Ridefinizioni di tipo (typedef)

- using `natq` = unsigned long
naturale su 8 byte (64bit)
- using `vaddr` = unsigned long
indirizzo virtuale (64bit)
- using `paddr` = unsigned long
indirizzo fisico (64bit)

Tipi richiesti dallo standard (64bit)

- typedef unsigned long `size_t`
tipo restituito da sizeof
- typedef long `ssize_t`
tipo che può contenere una dimensione o un errore
- typedef long `ptrdiff_t`
tipo che può contenere il risultato della sottrazione tra due puntatori
- typedef long `intmax_t`
tipo intero più capiente supportato dal sistema
- typedef unsigned long `uintmax_t`
tipo intero senza segno più capiente supportato dal sistema
- typedef unsigned long `uintptr_t`
tipo senza segno che può contenere il valore di un puntatore

6.3.1 Descrizione dettagliata

Questi tipi sono definiti in modo diverso se la compilazione è a 32 bit (usata dal boot loader) o a 64 bit. Per `natq`, `vaddr` e `paddr` vogliamo che la dimensione in byte sia sempre 8, indipendentemente dalla modalità a 32 o 64 bit. Gli altri tipi (`size_t`, `ssize_t`, ...) hanno definizioni diverse decise dallo standard.

6.4 Funzioni di utilità generale

Moduli

- [Funzioni per la memoria dinamica.](#)

Tipi enumerati (enum)

- enum [log_sev](#) {
[LOG_DEBUG](#) , [LOG_INFO](#) , [LOG_WARN](#) , [LOG_ERR](#) ,
[LOG_USR](#) }

Livello di severità del messaggio inviato al log (usato per colorare i messaggi ove previsto)

Funzioni

- template<typename T >
T [max](#) (T a, T b)
Restituisce il massimo tra due valori confrontabili.
- template<typename T >
T [min](#) (T a, T b)
Restituisce il minimo tra due valori confrontabili.
- void * [memset](#) (void *dest, int c, [size_t](#) n)
Scrive lo stesso valore in tutti i byte di un intervallo.
- void * [memcpy](#) (void *dest, const void *src, [size_t](#) n)
Copia un intervallo di memoria su un altro (i due intervalli non possono sovrapporsi).
- [size_t](#) [strlen](#) (const char str[])
Calcola la lunghezza di una stringa.
- int [printf](#) (const char *fmt,...)
Scrittura formattata su schermo.
- int [snprintf](#) (char *buf, [natl](#) n, const char *fmt,...)
Scrittura formattata su un buffer in memoria.
- void [flog](#) ([log_sev](#) sev, const char *fmt,...)
Invio di un messaggio formattato sul log.
- void [pause](#) ()
Stampa un messaggio e attende che venga premuto il tasto ESC.
- void [fpanic](#) (const char *fmt,...)
Invia un messaggio sul log (severità ERR) ed esegue lo shutdown.
- template<typename To , typename From >
static To * [ptr_cast](#) (From v)
Converte da intero a puntatore non void.
- template<typename From >
static void * [voidptr_cast](#) (From v)
Converte da intero a puntatore a void.
- template<typename To , typename From >
static To [int_cast](#) (From *p)
Converte da puntatore a intero.
- static [natq](#) [allinea](#) ([natq](#) v, [natq](#) a)
Restituisce il più piccolo multiplo di a maggiore o uguale a v.
- template<typename T >
static T * [allinea_ptr](#) (T *p, [natq](#) a)
Restituisce il più piccolo puntatore a T allineato ad a e maggiore o uguale a p.
- long int [random](#) ()
Restituisce un intero pseudo-causale.
- void [setseed](#) ([natl](#) seed)
Imposta il seme iniziale del generatore di numeri pseudo-casuali.

6.4.1 Descrizione dettagliata

Le funzioni non inline sono definite nella directory `util`, ciascuna nel proprio file.

6.4.2 Documentazione dei tipi enumerati

6.4.2.1 `log_sev`

enum `log_sev`

Livello di severità del messaggio inviato al log (usato per colorare i messaggi ove previsto)

Valori del tipo enumerato

LOG_DEBUG	debugging
LOG_INFO	informazione
LOG_WARN	avviso
LOG_ERR	errore
LOG_USR	messaggio proveniente da livello utente

Definizione alla linea 193 del file `libce.h`.

6.4.3 Documentazione delle funzioni

6.4.3.1 `flog()`

```
void flog (
    log_sev sev,
    const char * fmt,
    ... )
```

Invio di un messaggio formattato sul log.

Si possono usare gli stessi operatori di `printf`.

Nella configurazione di default il messaggio viene visualizzato sul terminale dal quale è stato lanciato QEMU. Il messaggio sarà preceduto dall'id del processo che lo ha inviato (se disponibile) e da una stringa di tre lettere che identifica la severità del messaggio.

Parametri

<i>sev</i>	severità del messaggio
<i>fmt</i>	stringa di formato
...	argomenti richiesti da <i>fmt</i>

Nota

si veda [printf](#) per il significato di `__attribute__`

Definizione alla linea 6 del file `flog.cpp`.

6.4.3.2 fpanic()

```
void fpanic (
    const char * fmt,
    ... )
```

Invia un messaggio sul log (severità ERR) ed esegue lo shutdown.

Si possono usare gli stessi operatori di [printf](#).

Parametri

<i>fmt</i>	stringa di formato
...	argomenti richiesti da <i>fmt</i>

Nota

si veda [printf](#) per il significato di `__attribute__`

Definizione alla linea 4 del file `fpanic.cpp`.

6.4.3.3 int_cast()

```
template<typename To , typename From >
static To int_cast (
    From * p ) [inline], [static]
```

Converte da puntatore a intero.

Il tipo *To* deve essere sufficientemente grande per contenere l'indirizzo. In caso contrario la funzione chiama [fpanic\(\)](#).

Parametri dei template

<i>To</i>	un tipo intero
<i>From</i>	tipo degli oggetti puntati

Parametri

<i>p</i>	puntatore da convertire
----------	-------------------------

Restituisce

valore di p convertito a intero

Definizione alla linea 295 del file libce.h.

6.4.3.4 max()

```
template<typename T >
T max (
    T a,
    T b )
```

Restituisce il massimo tra due valori confrontabili.

Parametri dei template

<i>T</i>	tipo dei valori (deve definire <)
----------	-----------------------------------

Parametri

<i>a</i>	primo valore da confrontare
<i>b</i>	secondo valore da confrontare

Restituisce

massimo tra a e b

Definizione alla linea 120 del file libce.h.

6.4.3.5 memcpy()

```
void* memcpy (
    void * dest,
    const void * src,
    size_t n )
```

Copia un intervallo di memoria su un altro (i due intervalli non possono sovrapporsi).

Parametri

<i>dest</i>	base dell'intervallo di destinazione
<i>src</i>	base dell'intervallo sorgente
<i>n</i>	dimensione in byte dei due intervalli

Restituisce

dest

Definizione alla linea 3 del file memcpy.cpp.

6.4.3.6 memset()

```
void* memset (
    void * dest,
    int c,
    size_t n )
```

Scrive lo stesso valore in tutti i byte di un intervallo.

Parametri

<i>dest</i>	indirizzo base dell'intervallo
<i>c</i>	valore da scrivere
<i>n</i>	dimensione in byte dell'intervallo

Restituisce

dest

Definizione alla linea 3 del file memset.cpp.

6.4.3.7 min()

```
template<typename T >
T min (
    T a,
    T b )
```

Restituisce il minimo tra due valori confrontabili.

Parametri dei template

<i>T</i>	tipo dei valori (deve definire <)
----------	-----------------------------------

Parametri

<i>a</i>	primo valore da confrontare
<i>b</i>	secondo valore da confrontare

Restituisce

minimo tra *a* e *b*

Definizione alla linea 128 del file libce.h.

6.4.3.8 printf()

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

Scrittura formattata su schermo.

Segue la stessa sintassi della `printf(3)` della libreria standard del C, con l'esclusione dei parametri di tipo `float` e `double`.

Parametri

<i>fmt</i>	la stringa di formato
...	argomenti richiesti da <i>fmt</i>

Restituisce

il numero di caratteri stampati

Nota

La strana stringa `__attribute__((format(printf, 1, 2)))` introduce una estensione di gcc. In questo caso dice al compilatore che *fmt* (argomento 1) segue la sintassi della `printf(3)` standard e che la funzione si aspetta un numero variabile di argomenti a partire da quello in seconda posizione . Ad ogni invocazione di questa funzione con una stringa *fmt* nota a tempo di compilazione, il compilatore controllerà che gli argomenti attuali corrispondano in tipo e numero con quelli richiesti da *fmt*, emettendo dei warning in caso contrario.

Definizione alla linea 3 del file printf.cpp.

6.4.3.9 ptr_cast()

```
template<typename To , typename From >
static To* ptr_cast (
    From v ) [inline], [static]
```

Converte da intero a puntatore non void.

L'intero deve essere allineato correttamente e deve poter essere contenuto in un puntatore (4 byte a 32 bit, 8 byte a 64 bit) In caso contrario la funzione chiama [fpanic\(\)](#).

Parametri dei template

<i>To</i>	tipo degli oggetti puntati
<i>From</i>	un tipo intero

Parametri

<i>v</i>	valore (di tipo <i>From</i>) da convertire
----------	---

Restituisce

puntatore a *To*

Definizione alla linea 245 del file libce.h.

6.4.3.10 `setseed()`

```
void setseed (
    natl seed )
```

Imposta il seme iniziale del generatore di numeri pseudo-casuali.

Parametri

<i>seed</i>	valore del seme
-------------	-----------------

Definizione alla linea 3 del file setseed.cpp.

6.4.3.11 `snprintf()`

```
int snprintf (
    char * buf,
    natl n,
    const char * fmt,
    ... )
```

Scrittura formattata su un buffer in memoria.

Segue la stessa sintassi della `snprintf(3)` della libreria standard del C, con l'esclusione dei parametri di tipo `float` e `double`.

Parametri

<i>buf</i>	buffer di destinazione
<i>n</i>	dimensione in byte del buffer di destinazione
<i>fmt</i>	stringa di formato
	argomenti richiesti da <i>fmt</i>

Restituisce

il numero di caratteri dell'output completo (se $\geq n$ l'output è stato troncato)

Nota

si veda [printf](#) per il significato di `__attribute__`

Definizione alla linea 3 del file `snprintf.cpp`.

6.4.3.12 strlen()

```
size_t strlen (
    const char str[] )
```

Calcola la lunghezza di una stringa.

Parametri

<i>str</i>	stringa terminata con <code>\0</code>
------------	---------------------------------------

Restituisce

lunghezza della stringa (escluso il terminatore)

6.4.3.13 voidptr_cast()

```
template<typename From >
static void* voidptr_cast (
    From v ) [inline], [static]
```

Converte da intero a puntatore a void.

L'intero deve poter essere contenuto in un puntatore (4 byte a 32 bit, 8 byte a 64 bit) In caso contrario la funzione chiama [fpanic\(\)](#).

Parametri dei template

<i>From</i>	un tipo intero
-------------	----------------

Parametri

<i>v</i>	valore (di tipo <i>From</i>) da convertire
----------	---

Restituisce

puntatore a void

Definizione alla linea 272 del file libce.h.

6.5 Funzioni per la memoria dinamica.

Funzioni

- void `heap_init` (void *start, `size_t` size, `natq` initmem=0)
Inizializza un intervallo di memoria in modo che possa essere usata con `alloc()`, `alloc_aligned()` e `dealloc()`.
- void * `alloc` (`size_t` dim)
Alloca una zona di memoria nello heap.
- void * `alloc_aligned` (`size_t` dim, `std::align_val_t` align)
Alloca una zona di memoria nello heap, con vincoli di allineamento.
- void `dealloc` (void *p)
Dealloca una zona di memoria, restituendola allo heap.
- `size_t` `disponibile` ()
Memoria libera nello heap.

Overload standard.

Overloading degli operatori di default normalmente forniti dalla libreria standard del C++. Si limitano a richiamare in modo appropriato `operator new` e `operator delete`, che devono essere definiti a parte.

- void * `operator new[]` (`size_t` s)
versione di new per l'allocazione di array
- void * `operator new[]` (`size_t` s, `std::align_val_t` a)
versione di new per l'allocazione di array con allineamento
- void `operator delete` (void *p, `size_t` s)
versione di delete con dimensione esplicita
- void `operator delete` (void *p, `size_t` s, `std::align_val_t`)
versione di delete con dimensione esplicita e allineamento
- void `operator delete[]` (void *p)
versione di delete per la deallocazione degli array
- void `operator delete[]` (void *p, `size_t` s)
versione di delete per la deallocazione degli array con dimensione esplicita

6.5.1 Descrizione dettagliata

Queste funzioni definiscono un semplice allocatore di memoria che può essere usato per implementare gli operatori `new` e `delete`. La zona di memoria da usare come heap va prima inizializzata con `heap_init()`.

Le funzioni sono definite nella directory `heap`. Alcuni dettagli sull'implementazione si trovano nei commenti del file `internal.h`.

6.5.2 Documentazione delle funzioni

6.5.2.1 `alloc()`

```
void* alloc (
    size_t dim )
```

Alloca una zona di memoria nello heap.

Parametri

<i>dim</i>	dimensione in byte della zona da allocare
------------	---

Restituisce

puntatore alla zona allocata se disponibile,
`nullptr` altrimenti

Definizione alla linea 8 del file `alloc.cpp`.

6.5.2.2 `alloc_aligned()`

```
void* alloc_aligned (
    size_t dim,
    std::align_val_t align )
```

Alloca una zona di memoria nello heap, con vincoli di allineamento.

L'indirizzo restituito sarà multiplo dell'allineamento richiesto.

Parametri

<i>dim</i>	dimensione in byte della zona da allocare
<i>align</i>	allineamento richiesto

Restituisce

puntatore alla zona allocata se disponibile, `nullptr` altrimenti

Definizione alla linea 3 del file `alloc_aligned.cpp`.

6.5.2.3 dealloc()

```
void dealloc (
    void * p )
```

Dealloca una zona di memoria, restituendola allo heap.

Parametri

<i>p</i>	puntatore alla zona da deallocare.
----------	------------------------------------

Il puntatore deve essere stato precedentemente ottenuto tramite [alloc\(\)](#) o [alloc_aligned\(\)](#).

Definizione alla linea 5 del file `dealloc.cpp`.

6.5.2.4 disponibile()

```
size_t disponibile ( )
```

Memoria libera nello heap.

Restituisce

quantità di memoria attualmente libera nello heap

Definizione alla linea 3 del file `disponibile.cpp`.

6.5.2.5 heap_init()

```
void heap_init (
    void * start,
    size_t size,
    natq initmem = 0 )
```

Inizializza un intervallo di memoria in modo che possa essere usata con [alloc\(\)](#), [alloc_aligned\(\)](#) e [dealloc\(\)](#).

L'intervallo deve essere accessibile in lettura e scrittura. L'allocatore mantiene una lista di chunk liberi e i descrittori dei chunk sono allocati nell'intervallo stesso.

È possibile passare, come terzo parametro, un indirizzo da assegnare direttamente alla testa della lista dei chunk liberi, prima di aggiungervi il chunk che descrive il nuovo intervallo. Questa funzionalità è utile soprattutto per permettere al nucleo di aggiungere al proprio heap la memoria già usata dal boot loader.

Parametri

<i>start</i>	base dell'intervallo
<i>size</i>	dimensione in byte dell'intervallo
<i>initmem</i>	se != 0, valore iniziale della lista dei chunk liberi

Definizione alla linea 3 del file `heap_init.cpp`.

6.6 Funzioni per l'interazione con le interfacce di I/O.

Moduli

- [Funzioni per leggere/scrivere nello spazio di I/O.](#)

6.6.1 Descrizione dettagliata

Queste sono le funzioni definite e usate in `esempiO` e nei moduli `sistema` e `IO` del nucleo. Le funzioni di ogni interfaccia sono raggruppate nel proprio namespace e definite in una directory che ha lo stesso nome del namespace.

6.7 Funzioni per leggere/scrivere nello spazio di I/O.

Funzioni

- `natb inputb (ioaddr reg)`
Legge un byte da una porta di I/O.
- `natw inputw (ioaddr reg)`
Legge una word (2 byte) da una porta di I/O.
- `natl inputl (ioaddr reg)`
Legge un long (4 byte) da una porta di I/O.
- `void inputbw (ioaddr reg, natw vetto[], int n)`
Legge una successione di word (2 byte) da una porta di I/O.
- `void outputb (natb a, ioaddr reg)`
Invia un byte ad una porta di I/O.
- `void outputw (natw a, ioaddr reg)`
Invia una word (2 byte) ad una porta di I/O.
- `void outputl (natl a, ioaddr reg)`
Invia un long (4 byte) ad una porta di I/O.
- `void outputbw (natw vetto[], int n, ioaddr reg)`
Invia una successione di word (2 byte) ad una porta di I/O.

6.7.1 Descrizione dettagliata

Funzioni generiche per l'accesso allo spazio di I/O. Dal momento che usano le istruzioni `IN` e `OUT` sono definite in assembler. Le definizioni si trovano nella directory `as64` (versione a 64 bit) e `as32` (versione a 32bit)

6.7.2 Documentazione delle funzioni

6.7.2.1 inputb()

```
natb inputb (  
    ioaddr reg )
```

Legge un byte da una porta di I/O.

Parametri

<i>reg</i>	indirizzo della porta nello spazio di I/O
------------	---

Restituisce

byte letto

6.7.2.2 inputbw()

```
void inputbw (
    ioaddr reg,
    natw vetti[],
    int n )
```

Legge una successione di word (2 byte) da una porta di I/O.

Parametri

<i>reg</i>	indirizzo della porta nello spazio di I/O
<i>vetti</i>	buffer in cui ricevere le word lette
<i>n</i>	numero di word da leggere

6.7.2.3 inputl()

```
natl inputl (
    ioaddr reg )
```

Legge un long (4 byte) da una porta di I/O.

Parametri

<i>reg</i>	indirizzo della porta nello spazio di I/O
------------	---

Restituisce

long letto

6.7.2.4 inputw()

```
natw inputw (
    ioaddr reg )
```

Legge una word (2 byte) da una porta di I/O.

Parametri

<i>reg</i>	indirizzo della porta nello spazio di I/O
------------	---

Restituisce

word letta

6.7.2.5 outputb()

```
void outputb (
    natb a,
    ioaddr reg )
```

Invia un byte ad una porta di I/O.

Parametri

<i>a</i>	byte da inviare
<i>reg</i>	indirizzo della porta nello spazio di I/O

6.7.2.6 outputbw()

```
void outputbw (
    natw vetto[],
    int n,
    ioaddr reg )
```

Invia una successione di word (2 byte) ad una porta di I/O.

Parametri

<i>vetto</i>	buffer contenente le word da inviare
<i>n</i>	numero di word da inviare
<i>reg</i>	indirizzo della porta nello spazio di I/O

6.7.2.7 outputl()

```
void outputl (
    natl a,
    ioaddr reg )
```

Invia un long (4 byte) ad una porta di I/O.

Parametri

<i>a</i>	long da inviare
<i>reg</i>	indirizzo della porta nello spazio di I/O

6.7.2.8 outputw()

```
void outputw (
    natw a,
    ioaddr reg )
```

Invia una word (2 byte) ad una porta di I/O.

Parametri

<i>a</i>	word da inviare
<i>reg</i>	indirizzo della porta nello spazio di I/O

6.8 Funzioni legate al meccanismo delle interruzioni.

Moduli

- [Funzioni per la manipolazione della tabella IDT.](#)
- [Eccezioni.](#)

Funzioni

- void [apic_send_EOI](#) ()
Invia l'End Of Interrupt.

Variabili

- const natq [BIT_IF](#) = 1UL << 9
Interrupt Flag.
- const natq [BIT_TF](#) = 1UL << 8
Trap Flag.

6.8.1 Descrizione dettagliata

Dispensa: <https://calcolatori.iet.unipi.it/resources/interruzioni.pdf>

6.8.2 Documentazione delle funzioni

6.8.2.1 apic_send_EOI()

```
void apic_send_EOI ( )
```

Invia l'End Of Interrupt.

Funzione analoga a [apic::send_EOI](#), ma invocabile dall'assembly più comodamente

Definizione alla linea 13 del file `send_EOI.cpp`.

6.8.3 Documentazione delle variabili

6.8.3.1 BIT_IF

```
const natq BIT_IF = 1UL << 9
```

Interrupt Flag.

Flag del registro RFLAGS. Se attivo abilita il processore ad accettare le richieste di interruzione esterne mascherabili.

Definizione alla linea 973 del file `libce.h`.

6.8.3.2 BIT_TF

```
const natq BIT_TF = 1UL << 8
```

Trap Flag.

Flag del registro RFLAGS. Se attivo, il processore solleva una eccezione di tipo 1 (debug) dopo l'esecuzione di ogni istruzione.

Definizione alla linea 979 del file `libce.h`.

6.9 Funzioni per la manipolazione della tabella IDT.

Funzioni

- void [gate_init](#) ([natb](#) num, void routine(), bool trap=false, int liv=[LIV_UTENTE](#))
Inizializza un gate della IDT.
- bool [gate_present](#) ([natb](#) num)
Controlla che un gate non sia già occupato.

6.9.1 Descrizione dettagliata

Queste funzioni sono definite nella directory `util`.

6.9.2 Documentazione delle funzioni

6.9.2.1 `gate_init()`

```
void gate_init (
    natb num,
    void routine(),
    bool trap = false,
    int liv = LIV_UTENTE )
```

Inizializza un gate della IDT.

Il gate viene inizializzato in ogni caso per portare il processore a livello sistema.

Parametri

<i>num</i>	numero del gate (0-255)
<i>routine</i>	funzione da associare al gate
<i>trap</i>	se true, crea un gate di tipo trap
<i>liv</i>	DPL del gate (LIV_UTENTE o LIV_SISTEMA)

Definizione alla linea 4 del file `gate_init.cpp`.

6.9.2.2 `gate_present()`

```
bool gate_present (
    natb num )
```

Controlla che un gate non sia già occupato.

Parametri

<i>num</i>	numero del gate
------------	-----------------

Restituisce

true se il gate è occupato (`P==1`), false altrimenti

Definizione alla linea 3 del file `gate_present.cpp`.

6.10 Eccezioni.

Funzioni

- void `log_exception` (int tipo, natq errore, vaddr rip)
Decodifica delle eccezioni.

Eccezione di Page Fault (14)

Il microprogramma di gestione delle eccezioni di page fault lascia in cima alla pila (oltre ai valori consueti) una parola quadrupla i cui 4 bit meno significativi specificano più precisamente il motivo per cui si è verificato un page fault.

- static const `natq PF_PROT` = 1U << 0
Page fault causato da errore di protezione.
- static const `natq PF_WRITE` = 1U << 1
Page fault con accesso in scrittura.
- static const `natq PF_USER` = 1U << 2
Page fault con accesso da livello utente.
- static const `natq PF_RES` = 1U << 3
Page fault con bit riservati non validi.

Eccezioni con codice di errore.

Alcune eccezioni, per lo più legate al meccanismo della segmentazione, lasciano in pila un codice di errore che è l'offset all'interno di una delle tre tabelle IDT, GDT o LDT (quest'ultima non usata da noi) e di tre bit che specificano meglio il tipo di errore.

- static const `natq SE_EXT` = 1U << 0
Eccezione causata da evento esterno.
- static const `natq SE_IDT` = 1U << 1
Eccezione durante accesso alla IDT.
- static const `natq SE_TI` = 1U << 2
Table Indicator dell'eccezione.

6.10.1 Descrizione dettagliata

Dispensa: <https://calcolatori.iet.unipi.it/resources/eccezioni.pdf>

6.10.2 Documentazione delle funzioni

6.10.2.1 `log_exception()`

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

Decodifica delle eccezioni.

Invia sul log alcuni messaggi che mostrano i dettagli associati ad una eccezione.

Parametri

<i>tipo</i>	tipo dell'eccezione
<i>errore</i>	eventuale codice di errore salvato in pila dal processore (0 se assente)
<i>rip</i>	instruction pointer salvato in pila dal processore

Definizione alla linea 9 del file log_exception.cpp.

6.10.3 Documentazione delle variabili

6.10.3.1 PF_PROT

```
const natq PF_PROT = 1U << 0 [static]
```

Page fault causato da errore di protezione.

Se questo bit vale 1 la traduzione era presente, ma c'è stato un errore diverso, per es. il processore si trovava a livello utente e la pagina era di livello sistema (bit US = 0 in una qualunque delle tabelle dell'albero che porta al descrittore della pagina). Se invece il bit PF_PROT è zero, la pagina o una delle tabelle erano assenti (bit P = 0)

Definizione alla linea 1090 del file libce.h.

6.10.3.2 PF_RES

```
const natq PF_RES = 1U << 3 [static]
```

Page fault con bit riservati non validi.

Uno dei bit riservati nel descrittore di pagina o di tabella non avevano il valore richiesto (il bit D deve essere 0 per i descrittori di tabella e il bit PS deve essere 0 per i descrittori di pagina).

Definizione alla linea 1116 del file libce.h.

6.10.3.3 PF_USER

```
const natq PF_USER = 1U << 2 [static]
```

Page fault con accesso da livello utente.

L'accesso che ha causato il fault è avvenuto mentre il processore si trovava a livello utente.

Nota

Questo non implica che la pagina fosse invece di livello sistema: il page fault potrebbe essere stato causato da altro (per es. traduzione assente, o accesso in scrittura su pagina di sola lettura).

Definizione alla linea 1109 del file libce.h.

6.10.3.4 PF_WRITE

```
const natq PF_WRITE = 1U << 1 [static]
```

Page fault con accesso in scrittura.

L'accesso che ha causato il page fault era in scrittura.

Nota

Questo non implica che la pagina non fosse scrivibile: il page fault potrebbe essere stato causato da altro (per es. traduzione assente o accesso da livello utente a pagina di livello sistema).

Definizione alla linea 1099 del file libce.h.

6.10.3.5 SE_EXT

```
const natq SE_EXT = 1U << 0 [static]
```

Eccezione causata da evento esterno.

L'eccezione è stata sollevata mentre il processore cercava di gestire un evento esterno al programma (per esempio, una richiesta di interruzione esterna).

Definizione alla linea 1132 del file libce.h.

6.10.3.6 SE_IDT

```
const natq SE_IDT = 1U << 1 [static]
```

Eccezione durante accesso alla IDT.

L'eccezione è stata sollevata durante un accesso a un gate della IDT. In questo caso il resto del codice di errore è l'offset del gate all'interno della IDT.

Definizione alla linea 1139 del file libce.h.

6.10.3.7 SE_TI

```
const natq SE_TI = 1U << 2 [static]
```

Table Indicator dell'eccezione.

Se [SE_IDT](#) è 0, il bit [SE_IDT](#) indica se l'eccezione è stata sollevata durante un accesso a un descrittore della LDT ([SE_TI](#)==1) o alla GDT ([SE_TI](#)==0). In questo caso il resto del codice di errore è l'offset del descrittore all'interno della tabella interessata.

Definizione alla linea 1147 del file libce.h.

6.11 Funzioni di uso meno generale, usate dal nucleo.

Funzioni

- int `vsnprintf` (char *buf, `size_t` size, const char *fmt, va_list ap)
Come `snprintf`, ma usa una `va_list` esplicita invece di essere variadica.
- bool `find_eh_frame` (vaddr elf, vaddr &eh_frame, natq &eh_frame_len)
Trova l'exception header all'interno di un file ELF caricato in memoria.
- void `reboot` ()
Riavvia il sistema.
- void `do_log` (log_sev sev, const char *buf, natl quanti)
Funzione di basso livello per la scrittura sul log.

6.11.1 Descrizione dettagliata

Questa sezione raggruppa alcune funzioni che sono realizzate dalla libce per comodità, ma è improbabile che sia necessario usarle al di fuori degli usi specifici all'interno del nucleo.

6.11.2 Documentazione delle funzioni

6.11.2.1 do_log()

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

Funzione di basso livello per la scrittura sul log.

`flog()` formatta il messaggio e poi chiama `do_log()` per inviarlo effettivamente (i programmi 'bare' e il nucleo possono usare la `do_log` fornita dalla libce, che scrive direttamente sulla porta seriale, ma nei moduli I/O e utente `do_log` è ridefinita in modo da invocare una primitiva di sistema)

Parametri

<i>sev</i>	severità del messaggio
<i>buf</i>	buffer contenente il messaggio
<i>quanti</i>	dimensione in byte del messaggio

Definizione alla linea 7 del file `do_log.cpp`.

6.11.2.2 find_eh_frame()

```
bool find_eh_frame (
    vaddr elf,
```

```
vaddr & eh_frame,
natq & eh_frame_len )
```

Trova l'exception header all'interno di un file ELF caricato in memoria.

L'exception header è generato dal compilatore e serve a recuperare le informazioni per lo stack-unwinding. Lo usiamo per implementare il backtrace in caso di errori.

Parametri

	<i>elf</i>	indirizzo virtuale dell'header ELF caricato in memoria
out	<i>eh_frame</i>	indirizzo virtuale dell'exception header
out	<i>eh_frame_len</i>	lunghezza dell'exception header

Restituisce

true se trovata, false altrimenti

Definizione alla linea 6 del file `find_eh_frame.cpp`.

6.11.2.3 reboot()

```
void reboot ( )
```

Riavvia il sistema.

Nella configurazione standard QEMU è stato impostato per fare lo shutdown invece di riavviare, quindi questa funzione ha l'effetto di spegnere la macchina virtuale.

Definizione alla linea 13 del file `reboot.cpp`.

6.11.2.4 vsnprintf()

```
int vsnprintf (
    char * buf,
    size_t size,
    const char * fmt,
    va_list ap )
```

Come `snprintf`, ma usa una `va_list` esplicita invece di essere variadica.

Questa funzione contiene l'implementazione vera e propria del parser delle stringhe di formato ed è chiamata sia da `printf` che da `snprintf`.

Parametri

<i>buf</i>	buffer di destinazione
<i>size</i>	dimensione di <i>buf</i>
<i>fmt</i>	stringa di formato
<i>ap</i>	lista degli argomenti per <i>fmt</i>

Definizione alla linea 62 del file vsnprintf.cpp.

6.12 Costanti usate da boot64.

Definizioni

- #define `DIM_GDT_ENTRY` 8
dimensione in byte di una entrata della GDT
- #define `NUM_GDT_ENTRIES` 6
numero di righe della GDT
- #define `DIM_TSS` 104
dimensione in byte del segmento TSS

Indici all'interno della GDT

- #define `ID_CODE_SYS` 1
segmento codice/sistema
- #define `ID_CODE_USR` 2
segmento codice/utente
- #define `ID_DATA_USR` 3
segmento dati/utente
- #define `ID_SYS_TSS` 4
segmento TSS

Selettori di segmento

- #define `SEL_NULLO` 0
selettore nullo
- #define `SEL_CODICE_SISTEMA` ((`ID_CODE_SYS` << 3) | `LIV_SISTEMA`)
selettore del segmento codice di livello sistema
- #define `SEL_CODICE_UTENTE` ((`ID_CODE_USR` << 3) | `LIV_UTENTE`)
selettore del segmento codice di livello utente
- #define `SEL_DATI_UTENTE` ((`ID_DATA_USR` << 3) | `LIV_UTENTE`)
selettore del segmento dati scrivibili di livello utente
- #define `SEL_SYS_TSS` ((`ID_SYS_TSS` << 3) | `LIV_SISTEMA`)
selettore del segmento TSS

6.12.1 Descrizione dettagliata

Alcune informazioni su queste definizioni e sul bootstrap in generale si trovano nei commenti dei file [boot64/boot.s](#) e [boot64/boot.cpp](#)

6.12.2 Documentazione delle definizioni

6.12.2.1 NUM_GDT_ENTRIES

```
#define NUM_GDT_ENTRIES 6
```

numero di righe della GDT

5 entrate compresa la 0 che deve essere nulla, ma l'entrata del TSS occupa due righe.

Definizione alla linea 1276 del file libce.h.

6.13 Funzioni per la memoria virtuale.

Strutture dati

- class `tab_iter`
Iteratore per la visita di un TRIE.

Ridefinizioni di tipo (typedef)

- using `tab_entry` = `natq`
descrittore di pagina o tabella

Funzioni

- static constexpr `vaddr norm` (`vaddr a`)
Normalizza un indirizzo.
- static constexpr `natq dim_region` (`int liv`)
Calcola la dimensione di una regione del TRIE dato il livello.
- static constexpr `vaddr base` (`vaddr v`, `int liv`)
Calcola la base della regione di un dato livello a cui appartiene un dato indirizzo virtuale.
- static constexpr `vaddr limit` (`vaddr v`, `int liv`)
Calcola la base della prima regione di un dato livello che giace a destra di un dato indirizzo virtuale.
- static constexpr `paddr extr_IND_FISICO` (`tab_entry e`)
Estrae l'indirizzo fisico da un descrittore di pagina o tabella.
- static void `set_IND_FISICO` (`tab_entry &e`, `paddr f`)
Imposta l'indirizzo fisico in un descrittore di pagina o tabella.
- static constexpr `int i_tab` (`vaddr v`, `int liv`)
Indice nelle tabelle.
- static `tab_entry & get_entry` (`paddr tab`, `natl i`)
Accesso ad un'entrata di una tabella.
- void `set_entry` (`paddr tab`, `natl j`, `tab_entry se`)
Scriva una entrata di una tabella.
- void `copy_des` (`paddr src`, `paddr dst`, `natl i`, `natl n`)
Copia descrittori da una tabella ad un'altra.
- void `set_des` (`paddr dst`, `natl i`, `natl n`, `tab_entry e`)
Inizializza (parte dei) descrittori di una tabella.
- `paddr trasforma` (`paddr root_tab`, `vaddr v`)
Traduzione di un indirizzo virtuale in fisico.

- void `loadCR3` (`paddr` dir)
Carica un nuovo valore in cr3.
- `paddr` `readCR3` ()
Lettura di cr3.
- void `invalida_TLB` ()
Invalida tutto il TLB.
- void `invalida_entrata_TLB` (`vaddr` v)
Invalida una entrata del TLB.
- `vaddr` `readCR2` ()
Lettura di cr2.
- template<typename T >
`vaddr` `map` (`paddr` tab, `vaddr` begin, `vaddr` end, `natl` flags, T &getpaddr, int ps_lvl=1)
Crea tutto il sottoalbero necessario a tradurre tutti gli indirizzi di un intervallo.
- template<typename T >
`vaddr` `map` (`paddr` tab, `vaddr` begin, `vaddr` end, `natl` flags, const T &getpaddr, int ps_lvl=1)
Overloading di `map()` per il caso in cui `getpaddr` non sia un lvalue.
- template<typename T >
void `unmap` (`paddr` tab, `vaddr` begin, `vaddr` end, T &putpaddr)
Elimina tutte le traduzioni di un intervallo.
- template<typename T >
void `unmap` (`paddr` tab, `vaddr` begin, `vaddr` end, const T &putpaddr)
Overloading di `unmap()` per il caso in cui `putpaddr` non sia un lvalue.

Variabili

- static const int `MAX_LIV` = 4
numero di livelli del TRIE (4 o 5)
- static const int `MAX_PS_LVL` = 2
massimo livello supportato per le pagine di grandi dimensioni
- static const `natq` `VADDR_MSBIT` = (1ULL << (12 + 9 * `MAX_LIV` - 1))
ultimo bit significativo di un indirizzo virtuale
- static const `natq` `VADDR_MASK` = `VADDR_MSBIT` - 1
maschera per selezionare i bit significativi di un indirizzo virtuale

Bit del byte di accesso

- const `natq` `BIT_P` = 1U << 0
bit di presenza
- const `natq` `BIT_RW` = 1U << 1
bit di lettura/scrittura
- const `natq` `BIT_US` = 1U << 2
bit utente/sistema
- const `natq` `BIT_PWT` = 1U << 3
bit Page Wright Through
- const `natq` `BIT_PCD` = 1U << 4
bit Page Cache Disable
- const `natq` `BIT_A` = 1U << 5
bit di accesso
- const `natq` `BIT_D` = 1U << 6
bit "dirty"

- const `natq BIT_PS` = 1U << 7
bit "page size"
- const `natq ACCB_MASK` = 0x00000000000000FF
maschera per il byte di accesso
- const `natq ADDR_MASK` = 0x7FFFFFFFFFFFFFFF000
maschera per l'indirizzo

Funzioni usate da `map()` e `unmap()`.

Queste funzioni devono essere definite per poter usare, in particolare, `map()` e `unmap()`. La libce fornisce una versione semplificata che può essere usata dai programmi 'bare'. La versione semplificata alloca le tabelle sullo heap e non le rilascia mai. Dal momento che non le rilascia mai, non gestisce nemmeno un contatore delle entrate valide delle tabelle. Il nucleo fornisce le proprie versioni di queste funzioni, che gestiscono correttamente i contatori delle entrate valide e permettono di deallocare le tabelle.

- `paddr alloca_tab ()`
invocata quando serve una nuova tabella
- void `rilascia_tab (paddr)`
invocata quando una tabella non serve più
- void `inc_ref (paddr)`
invocata quando una tabella acquisisce una nuova entrata valida
- void `dec_ref (paddr)`
invocata quando una tabella perde una entrata precedentemente valida
- `natl get_ref (paddr)`
invocata per leggere il contatore delle entrate valide di una tabella

6.13.1 Descrizione dettagliata

Dispensa: <https://calcolatori.iet.unipi.it/resources/paginazione-libreria.pdf> ↩

6.13.2 Documentazione delle funzioni

6.13.2.1 `base()`

```
static constexpr vaddr base (
    vaddr v,
    int liv ) [inline], [static], [constexpr]
```

Calcola la base della regione di un dato livello a cui appartiene un dato indirizzo virtuale.

Parametri

<code>v</code>	indirizzo virtuale
<code>liv</code>	livello della regione (0 - <code>MAX_LIV</code>)

Restituisce

base della regione

Definizione alla linea 57 del file vm.h.

6.13.2.2 copy_des()

```
void copy_des (
    paddr src,
    paddr dst,
    natl i,
    natl n )
```

Copia descrittori da una tabella ad un'altra.

La funzione usa `inc_ref()` o `dec_ref()` per aggiornare opportunamente il contatore delle entrate valide della tabella di destinazione, nel caso in cui qualche bit di presenza cambi per effetto della copia.

Parametri

<i>src</i>	indirizzo fisico della tabella sorgente
<i>dst</i>	indirizzo fisico della tabella di destinazione
<i>i</i>	indice del primo descrittore da copiare (0 - 511)
<i>n</i>	numero di descrittori da copiare (0 - 512)

Definizione alla linea 3 del file copy_des.cpp.

6.13.2.3 dim_region()

```
static constexpr natq dim_region (
    int liv ) [inline], [static], [constexpr]
```

Calcola la dimensione di una regione del TRIE dato il livello.

Parametri

<i>liv</i>	livello del TRIE (0 - <code>MAX_LIV</code>)
------------	--

Restituisce

dimensione in byte della regione

Definizione alla linea 44 del file vm.h.

6.13.2.4 extr_IND_FISICO()

```
static constexpr paddr extr_IND_FISICO (
    tab_entry e ) [inline], [static], [constexpr]
```

Estrae l'indirizzo fisico da un descrittore di pagina o tabella.

Parametri

<i>e</i>	descrittore di pagina o tabella
----------	---------------------------------

Restituisce

indirizzo fisico contenuto in *e*

Definizione alla linea 110 del file vm.h.

6.13.2.5 get_entry()

```
static tab_entry& get_entry (
    paddr tab,
    natl i ) [inline], [static]
```

Accesso ad un'entrata di una tabella.

La funzione restituisce un riferimento tramite il quale è possibile modificare il descrittore. Se viene modificato il bit di presenza, è compito del chiamante di aggiornare anche il contatore delle entrate valide nella tabella.

Parametri

<i>tab</i>	indirizzo fisico della tabella
<i>i</i>	indice in <i>tab</i> (0 - 511)

Restituisce

riferimento al descrittore *i*-esimo di *tab*

Definizione alla linea 151 del file vm.h.

6.13.2.6 i_tab()

```
static constexpr int i_tab (
    vaddr v,
    int liv ) [inline], [static], [constexpr]
```

Indice nelle tabelle.

Parametri

<i>v</i>	indirizzo virtuale
<i>liv</i>	livello della tabella (1 - MAX_LIV)

Restituisce

indice di *v* nelle tabelle di livello *liv*

Definizione alla linea 133 del file vm.h.

6.13.2.7 invalida_entrata_TLB()

```
void invalida_entrata_TLB (
    vaddr v )
```

Invalida una entrata del TLB.

Parametri

<i>v</i>	indirizzo virtuale di cui invalidare la traduzione
----------	--

6.13.2.8 limit()

```
static constexpr vaddr limit (
    vaddr v,
    int liv ) [inline], [static], [constexpr]
```

Calcola la base della prima regione di un dato livello che giace a destra di un dato indirizzo virtuale.

Parametri

<i>v</i>	indirizzo virtuale
<i>liv</i>	livello della regione (0 - MAX_LIV)

Restituisce

base della regione

Definizione alla linea 70 del file vm.h.

6.13.2.9 loadCR3()

```
void loadCR3 (
    paddr dir )
```

Carica un nuovo valore in cr3.

Parametri

<i>dir</i>	indirizzo fisico della tabella radice del TRIE
------------	--

6.13.2.10 map()

```
template<typename T >
vaddr map (
    paddr tab,
    vaddr begin,
    vaddr end,
    natl flags,
    T & getpaddr,
    int ps_lvl = 1 )
```

Crea tutto il sottoalbero necessario a tradurre tutti gli indirizzi di un intervallo.

L'intero intervallo non deve contenere traduzioni pre-esistenti.

I bit RW e US che sono a 1 nel parametro *flags* saranno settati anche in tutti i descrittori rilevanti del sottoalbero. Se *flags* contiene i bit PCD e/o PWT, questi saranno settati solo sui descrittori foglia.

Il parametro *getpaddr* deve poter essere invocato come *getpaddr(v)*, dove *v* è un indirizzo virtuale. L'invocazione deve restituire l'indirizzo fisico che si vuole far corrispondere a *v*, o zero in caso di errore.

La funzione, per default, crea traduzioni con pagine di livello 1. Se si vogliono usare pagine di livello superiore (da 2 a [MAX_PS_LVL](#)) occorre passare anche il parametro *ps_lvl*.

Parametri dei template

<i>T</i>	tipo di <i>getpaddr</i>
----------	-------------------------

Parametri

<i>tab</i>	indirizzo fisico della radice del TRIE
<i>begin</i>	base dell'intervallo
<i>end</i>	limite dell'intervallo
<i>flags</i>	flag da settare
<i>getpaddr</i>	funzione che deve restituire l'indirizzo fisico da associare ad ogni indirizzo virtuale.
<i>ps_lvl</i>	livello delle traduzioni da creare

Restituisce

il primo indirizzo non mappato (*end* in caso di successo)

Definizione alla linea 453 del file vm.h.

6.13.2.11 norm()

```
static constexpr vaddr norm (  
    vaddr a ) [inline], [static], [constexpr]
```

Normalizza un indirizzo.

Se `MAX_LIV == 4` (TRIE a 4 livelli) pone i 16 bit più significativi uguali al bit 47; Se `MAX_LIV == 5` (TRIE a 5 livelli) pone i 7 bit più significativi uguali al bit 56.

Parametri

<i>a</i>	indirizzo virtuale da normalizzare
----------	------------------------------------

Restituisce

indirizzo virtuale normalizzato

Definizione alla linea 32 del file vm.h.

6.13.2.12 readCR2()

```
vaddr readCR2 ( )
```

Lettura di cr2.

Restituisce

contenuto di cr2

6.13.2.13 readCR3()

```
paddr readCR3 ( )
```

Lettura di cr3.

Restituisce

contenuto di cr3

6.13.2.14 set_des()

```
void set_des (
    paddr dst,
    natl i,
    natl n,
    tab_entry e )
```

Inizializza (parte dei) descrittori di una tabella.

La funzione inizializza uno o più descrittori della tabella con lo stesso valore. È utile principalmente quando questo valore è zero, per resettare alcune entrate di una tabella.

La funzione usa `inc_ref()` o `dec_ref()` per aggiornare opportunamente il contatore delle entrate valide della tabella, nel caso in cui qualche bit di presenza cambi per effetto della scrittura.

Parametri

<i>dst</i>	indirizzo fisico della tabella
<i>i</i>	indice della prima entrata da sovrascrivere (0 - 511)
<i>n</i>	numero di entrate da sovrascrivere (0 - 512)
<i>e</i>	nuovo valore delle entrate (uguale per tutte)

Definizione alla linea 3 del file `set_des.cpp`.

6.13.2.15 set_entry()

```
void set_entry (
    paddr tab,
    natl j,
    tab_entry se )
```

Scrive una entrata di una tabella.

La funzione usa `inc_ref()` o `dec_ref()` per aggiornare opportunamente il contatore delle entrate valide della tabella, nel caso in cui il bit di presenza cambi per effetto della scrittura.

Parametri

<i>tab</i>	indirizzo fisico della tabella
<i>j</i>	indice in <i>tab</i> (0 - 511)
<i>se</i>	nuovo contenuto del descrittore <i>j</i> -esimo di <i>tab</i>

Definizione alla linea 3 del file `set_entry.cpp`.

6.13.2.16 set_IND_FISICO()

```
static void set_IND_FISICO (
    tab_entry & e,
    paddr f ) [inline], [static]
```

Imposta l'indirizzo fisico in un descrittore di pagina o tabella.

Parametri

<i>e</i>	riferimento a un descrittore di pagina o tabella
<i>f</i>	indirizzo fisico da impostare in <i>e</i>

Definizione alla linea 121 del file vm.h.

6.13.2.17 trasforma()

```
paddr trasforma (
    paddr root_tab,
    vaddr v )
```

Traduzione di un indirizzo virtuale in fisico.

La funzione percorre un TRIE in modo analogo a quanto la MMU fa in hardware, e restituisce la traduzione di un dato indirizzo virtuale (o zero se la traduzione è assente).

Parametri

<i>root_tab</i>	indirizzo fisico della tabella radice del TRIE
<i>v</i>	indirizzo virtuale da tradurre

Restituisce

indirizzo fisico corrispondente, o zero se *v* non è mappato

Definizione alla linea 3 del file trasforma.cpp.

6.13.2.18 unmap()

```
template<typename T >
void unmap (
    paddr tab,
    vaddr begin,
    vaddr end,
    T & putpaddr )
```

Elimina tutte le traduzioni di un intervallo.

Rilascia automaticamente tutte le sottotabelle che diventano vuote dopo aver eliminato le traduzioni. Per liberare le pagine vere e proprie, invece, chiama la funzione *putpaddr()* passandole l'indirizzo virtuale, l'indirizzo fisico e il livello della pagina da eliminare.

Parametri dei template

<i>T</i>	tipo di <i>putpaddr</i>
----------	-------------------------

Parametri

<i>tab</i>	indirizzo fisico della radice del TRIE
<i>begin</i>	base dell'intervallo
<i>end</i>	limite dell'intervallo
<i>putpaddr</i>	funzione che verrà invocata per ogni traduzione eliminata

Definizione alla linea 626 del file vm.h.

6.13.3 Documentazione delle variabili

6.13.3.1 VADDR_MSBIT

```
const natq VADDR_MSBIT = (1ULL << (12 + 9 * MAX_LIV - 1)) [static]
```

ultimo bit significativo di un indirizzo virtuale

bit 47 se `MAX_LIV == 4`, bit 56 se `MAX_LIV == 5`.

Definizione alla linea 18 del file vm.h.

Capitolo 7

Documentazione dei namespace

7.1 Riferimenti per il namespace apic

Funzioni per interagire con l'APIC.

Funzioni

- bool `init` ()
Inizializza l'APIC.
- void `send_EOI` ()
Invia l'End Of Interrupt.
- void `set_MIRQ` (natl irq, bool enable)
Maschera/smaschera una sorgente di richieste di interruzione.
- void `set_TRGM` (natl irq, bool enable)
Setta la modalità di riconoscimento di una sorgente di richieste. di interruzione.
- void `set_VECT` (natl irq, natb vec)
Imposta il vettore di interruzione associato ad una sorgente di richieste di interruzione.

Variabili

- static const int `MAX_IRQ` = 24
Massimo numero di IRQ supportati dall'APIC.

7.1.1 Descrizione dettagliata

Funzioni per interagire con l'APIC.

Dispensa: <https://calcolatori.iet.unipi.it/resources/interruzioni.pdf>

esempiO: interrupt-1, interrupt-2, interrupt-3

7.1.2 Documentazione delle funzioni

7.1.2.1 init()

```
bool apic::init ( )
```

Inizializza l'APIC.

Abilita l'APIC, quindi maschera tutti gli IRQ e azzeri gli altri campi.

Restituisce

false in caso di errore, true altrimenti

Definizione alla linea 5 del file init.cpp.

7.1.2.2 set_MIRQ()

```
void apic::set_MIRQ (
    natl irq,
    bool enable )
```

Maschera/smaschera una sorgente di richieste di interruzione.

Parametri

<i>irq</i>	irq da mascherare/smascherare
<i>enable</i>	maschera se true, smaschera se false

Definizione alla linea 5 del file set_MIRQ.cpp.

7.1.2.3 set_TRGM()

```
void apic::set_TRGM (
    natl irq,
    bool enable )
```

Setta la modalità di riconoscimento di una sorgente di richieste. di interruzione.

Parametri

<i>irq</i>	irq su cui agire
<i>enable</i>	false -> riconoscimento sul livello true -> riconoscimento sul fronte

Definizione alla linea 5 del file set_TRGM.cpp.

7.1.2.4 set_VECT()

```
void apic::set_VECT (
    natl irq,
    natb vec )
```

Imposta il vettore di interruzione associato ad una sorgente di richieste di interruzione.

Parametri

<i>irq</i>	irq a cui associare il vettore
<i>vec</i>	vettore di interruzione

Definizione alla linea 5 del file set_VECT.cpp.

7.2 Riferimenti per il namespace bm

Funzioni per il PCI BUS Mastering ATA.

Funzioni

- void **ack** ()
azione di risposta alle richieste di interruzione del bus master
- bool **find** (natb &bus, natb &dev, natb &fun)
cerca il prossimo bus master ATA.
- void **init** (natb bus, natb dev, natb fun)
inizializza un bus master.
- void **prepare** (paddr prd, bool write)
prepara una operazione di bus mastering.
- void **start** ()
avvia l'operazione di bus mastering precedentemente preparata.

7.2.1 Descrizione dettagliata

Funzioni per il PCI BUS Mastering ATA.

Dispensa: <https://calcolatori.iet.unipi.it/resources/dma.pdf>

Specifiche: <https://calcolatori.iet.unipi.it/deep/idems100.pdf>

Esempio: bm-hard-disk

7.2.2 Documentazione delle funzioni

7.2.2.1 find()

```
bool bm::find (
    natb & bus,
    natb & dev,
    natb & fun )
```

cerca il prossimo bus master ATA.

In ingresso *bus* / *dev* / *fun* devono contenere le coordinate da cui iniziare la ricerca. Al ritorno, se la ricerca ha avuto successo, contengono le coordinate del bus master.

Parametri

<i>in, out</i>	<i>bus</i>	numero del bus
<i>in, out</i>	<i>dev</i>	numero del dispositivo
<i>in, out</i>	<i>fun</i>	numero di funzione

Restituisce

true se trovato, false altrimenti

Definizione alla linea 5 del file find.cpp.

7.2.2.2 init()

```
void bm::init (
    natb bus,
    natb dev,
    natb fun )
```

inizializza un bus master.

Parametri

<i>bus</i>	numero di bus del bus master
<i>dev</i>	numero di dispositivo del bus master
<i>fun</i>	numero di funzione del bus master

Definizione alla linea 5 del file init.cpp.

7.2.2.3 prepare()

```
void bm::prepare (
    paddr prd,
    bool write )
```

prepara una operazione di bus mastering.

Parametri

<i>prd</i>	indirizzo fisico dell'array di descrittori
<i>write</i>	operazione di scrittura (true) o lettura (false)

Definizione alla linea 5 del file prepare.cpp.

7.3 Riferimenti per il namespace hd

Funzioni per interagire con l'hard disk.

Tipi enumerati (enum)

- enum `cmd` { `WRITE_SECT` = 0x30 , `READ_SECT` = 0x20 , `WRITE_DMA` = 0xCA , `READ_DMA` = 0xC8 }

Possibili valori da scrivere nel registro di comando dell'hard disk.

Funzioni

- void `ack` ()
Azione di risposta alla richiesta di interruzione dell'interfaccia dell'hard disk.
- void `disable_intr` ()
Disabilita l'interfaccia dell'hard disk a generare richieste di interruzione.
- void `enable_intr` ()
Abilita l'interfaccia dell'hard disk a generare richieste di interruzione.
- void `start_cmd` (`natl lba`, `natb quanti`, `cmd cmd`)
Avvia una operazione sull'hard disk.
- void `output_sect` (`natb *buf`)
Scrive un settore nel buffer interno dell'interfaccia dell'hard disk.
- void `input_sect` (`natb *buf`)
Legge un settore dal buffer interno dell'interfaccia dell'hard disk.

7.3.1 Descrizione dettagliata

Funzioni per interagire con l'hard disk.

Dispensa: <https://calcolatori.iet.unipi.it/resources/periferiche.pdf>

EsempioO: hard-disk-1, hard-disk-2

7.3.2 Documentazione dei tipi enumerati

7.3.2.1 cmd

enum `hd::cmd`

Possibili valori da scrivere nel registro di comando dell'hard disk.

Valori del tipo enumerato

WRITE_SECT	scrittura di settori
READ_SECT	lettura di settori
WRITE_DMA	scrittura di settori in DMA
READ_DMA	lettura di settori in DMA

Definizione alla linea 685 del file libce.h.

7.3.3 Documentazione delle funzioni

7.3.3.1 input_sect()

```
void hd::input_sect (
    natb * buf )
```

Legge un settore dal buffer interno dell'interfaccia dell'hard disk.

Parametri

<i>buf</i>	buffer che dovrà ricevere il settore
------------	--------------------------------------

7.3.3.2 output_sect()

```
void hd::output_sect (
    natb * buf )
```

Scrive un settore nel buffer interno dell'interfaccia dell'hard disk.

Parametri

<i>buf</i>	buffer contenente il settore da scrivere
------------	--

7.3.3.3 start_cmd()

```
void hd::start_cmd (
    natl lba,
    natb quanti,
    cmd cmd )
```

Avvia una operazione sull'hard disk.

Parametri

<i>lba</i>	logical block address del primo settore
<i>quanti</i>	numero di settori
<i>cmd</i>	codice del comando

Definizione alla linea 5 del file start_cmd.cpp.

7.4 Riferimenti per il namespace kbd

Funzioni per interagire con la tastiera.

Funzioni

- `natb get_code ()`
Restituisce l'ultimo codice di scansione ricevuto dalla tastiera (esegue una attesa attiva fino a quando il codice non è disponibile).
- `char conv (natb c)`
Converte un codice di scansione nel corrispondente codice ASCII.
- `char char_read ()`
Restituisce il codice ASCII dell'ultimo carattere letto dalla tastiera (esegue una attesa attiva fino a quando il carattere non è disponibile).
- `char char_read_intr ()`
Restituisce il codice ASCII corrispondente al codice di scansione contenuto in RBR (non esegue una attesa attiva).
- `void enable_intr ()`
Abilita l'interfaccia della tastiera a generare richieste di interruzione.
- `void disable_intr ()`
Disabilita l'interfaccia della tastiera a generare richieste di interruzione.
- `void drain ()`
Svuota il buffer della tastiera.

7.4.1 Descrizione dettagliata

Funzioni per interagire con la tastiera.

Dispensa: <https://calcolatori.iet.unipi.it/resources/periferiche.pdf>

Esempio: tastiera-1, tastiera-2

7.4.2 Documentazione delle funzioni

7.4.2.1 char_read()

```
char kbd::char_read ( )
```

Restituisce il codice ASCII dell'ultimo carattere letto dalla tastiera (esegue una attesa attiva fino a quando il carattere non è disponibile).

Restituisce

carattere ricevuto; 0 se è stato ricevuto un codice sconosciuto

Definizione alla linea 5 del file char_read.cpp.

7.4.2.2 char_read_intr()

```
char kbd::char_read_intr ( )
```

Restituisce il codice ASCII corrispondente al codice di scansione contenuto in RBR (non esegue una attesa attiva).

La funzione assume che sia già noto che RBR contiene un nuovo valore, per esempio perché è stata ricevuta una richiesta di interruzione dalla tastiera.

Restituisce

carattere letto; 0 se RBR conteneva un codice sconosciuto

Definizione alla linea 5 del file char_read_intr.cpp.

7.4.2.3 conv()

```
char kbd::conv (
    natb c )
```

Converte un codice di scansione nel corrispondente codice ASCII.

Parametri

<i>c</i>	codice di scansione da convertire
----------	-----------------------------------

Restituisce

codice ASCII corrispondente; 0 se il codice è sconosciuto

Definizione alla linea 5 del file conv.cpp.

7.4.2.4 get_code()

```
natb kbd::get_code ( )
```

Restituisce l'ultimo codice di scansione ricevuto dalla tastiera (esegue una attesa attiva fino a quando il codice non è disponibile).

Restituisce

codice di scansione

Definizione alla linea 5 del file get_code.cpp.

7.5 Riferimenti per il namespace pci

Funzioni per il bus PCI.

Funzioni

- `natb read_confb` (`natb bus`, `natb dev`, `natb fun`, `natb regn`)
Legge un byte dallo spazio di configurazione PCI.
- `natw read_confw` (`natb bus`, `natb dev`, `natb fun`, `natb regn`)
Legge una word (2 byte) dallo spazio di configurazione PCI.
- `natl read_confl` (`natb bus`, `natb dev`, `natb fun`, `natb regn`)
Legge un long (4 byte) dallo spazio di configurazione PCI.
- `void write_confb` (`natb bus`, `natb dev`, `natb fun`, `natb regn`, `natb data`)
Scrive un byte nello spazio di configurazione PCI.
- `void write_confw` (`natb bus`, `natb dev`, `natb fun`, `natb regn`, `natw data`)
Scrive una word (2 byte) nello spazio di configurazione PCI.
- `void write_confl` (`natb bus`, `natb dev`, `natb fun`, `natb regn`, `natl data`)
Scrive un long (4 byte) nello spazio di configurazione PCI.
- `bool find_dev` (`natb &bus`, `natb &dev`, `natb &fun`, `natw vendorID`, `natw deviceID`)
Trova una funzione PCI dato il vendor ID e il device ID.
- `bool find_class` (`natb &bus`, `natb &dev`, `natb &fun`, `natb code[]`)
Trova una funzione PCI dato il Class Code.
- `bool next` (`natb &bus`, `natb &dev`, `natb &fun`)
Calcola le prossime coordinate nel bus PCI.
- `const char * decode_class` (`natb class_code`)
Decodifica il primo byte di un Class Code.

7.5.1 Descrizione dettagliata

Funzioni per il bus PCI.

Dispensa: <https://calcolatori.iet.unipi.it/resources/pci.pdf>

Esempio: pci

7.5.2 Documentazione delle funzioni

7.5.2.1 decode_class()

```
const char * pci::decode_class (
    natb class_code )
```

Decodifica il primo byte di un Class Code.

Parametri

<i>class_code</i>	byte meno significativo di un Class Code
-------------------	--

Restituisce

stringa che descrive il Class Code

Definizione alla linea 27 del file decode_class.cpp.

7.5.2.2 find_class()

```
bool pci::find_class (
    natb & bus,
    natb & dev,
    natb & fun,
    natb code[ ] )
```

Trova una funzione PCI dato il Class Code.

In ingresso, *bus* / *dev* / *fun* devono contenere le coordinate da cui iniziare la ricerca. Al ritorno, se la ricerca ha avuto successo, contengono le coordinate della funzione.

code deve essere un array di 3 byte. In ingresso, uno o più di questi byte possono contenere il valore 0xFF che vale come "jolly". In uscita, se la ricerca ha avuto successo, i valori jolly saranno sostituiti dai byte effettivamente contenuti nella funzione trovata.

Parametri

in, out	<i>bus</i>	numero di bus
in, out	<i>dev</i>	numero di dispositivo
in, out	<i>fun</i>	numero di funzione
in, out	<i>code</i>	il class code

Restituisce

true se trovato, false altrimenti

Definizione alla linea 5 del file `find_class.cpp`.

7.5.2.3 find_dev()

```
bool pci::find_dev (
    natb & bus,
    natb & dev,
    natb & fun,
    natw vendorID,
    natw deviceID )
```

Trova una funzione PCI dato il vendor ID e il device ID.

In ingresso, *bus* / *dev* / *fun* devono contenere le coordinate da cui iniziare la ricerca. Al ritorno, se la ricerca ha avuto successo, contengono le coordinate della funzione.

Parametri

in, out	<i>bus</i>	numero di bus
in, out	<i>dev</i>	numero di dispositivo
in, out	<i>fun</i>	numero di funzione
	<i>vendorID</i>	vendor ID della funzione cercata
	<i>deviceID</i>	device ID della funzione cercata

Restituisce

true se trovato, false altrimenti

Definizione alla linea 5 del file `find_dev.cpp`.

7.5.2.4 next()

```
bool pci::next (
    natb & bus,
    natb & dev,
    natb & fun )
```

Calcola le prossime coordinate nel bus PCI.

Assumiamo che le coordinate bus/dev/fun siano ordinate nel modo seguente:

```
0/0/0, 0/0/1, 0/0/2, ..., 0/0/7,
0/1/0, 0/1/1, 0/0/2, ..., 0/1/7,
0/2/0, ...,                0/2/7,
...,
0/31/0, ...,                0/31,7,
1/0/0,
...,
255/0/0, ...,              255/31/7.
```

In ingresso *bus* / *dev* / *fun* devono contenere delle coordinate valide. Al ritorno, se le coordinate in ingresso non erano le ultime, *bus* / *dev* / *fun* conterranno le coordinate successive.

Parametri

<i>in, out</i>	<i>bus</i>	numero di bus
<i>in, out</i>	<i>dev</i>	numero di dispositivo
<i>in, out</i>	<i>fun</i>	numero di funzione

Restituisce

false se la coordinata in ingresso era l'ultima, true altrimenti

Definizione alla linea 5 del file next.cpp.

7.5.2.5 read_confb()

```
natb pci::read_confb (
    natb bus,
    natb dev,
    natb fun,
    natb regn )
```

Legge un byte dallo spazio di configurazione PCI.

Parametri

<i>bus</i>	numero di bus
<i>dev</i>	numero di dispositivo
<i>fun</i>	numero di funzione
<i>regn</i>	offset del byte

Restituisce

byte letto

Definizione alla linea 5 del file read_confb.cpp.

7.5.2.6 read_confl()

```
natl pci::read_confl (
    natb bus,
    natb dev,
    natb fun,
    natb regn )
```

Legge un long (4 byte) dallo spazio di configurazione PCI.

Parametri

<i>bus</i>	numero di bus
<i>dev</i>	numero di dispositivo
<i>fun</i>	numero di funzione
<i>regn</i>	offset del long

Restituisce

long letto

Definizione alla linea 5 del file read_conf1.cpp.

7.5.2.7 read_confw()

```
natw pci::read_confw (
    natb bus,
    natb dev,
    natb fun,
    natb regn )
```

Legge una word (2 byte) dallo spazio di configurazione PCI.

Parametri

<i>bus</i>	numero di bus
<i>dev</i>	numero di dispositivo
<i>fun</i>	numero di funzione
<i>regn</i>	offset della word

Restituisce

word letta

Definizione alla linea 5 del file read_confw.cpp.

7.5.2.8 write_confb()

```
void pci::write_confb (
    natb bus,
    natb dev,
    natb fun,
    natb regn,
    natb data )
```

Scrive un byte nello spazio di configurazione PCI.

Parametri

<i>bus</i>	numero di bus
<i>dev</i>	numero di dispositivo
<i>fun</i>	numero di funzione
<i>regn</i>	offset del byte su cui scrivere
<i>data</i>	valore da scrivere

Definizione alla linea 5 del file write_confb.cpp.

7.5.2.9 write_conf1()

```
void pci::write_conf1 (
    natb bus,
    natb dev,
    natb fun,
    natb regn,
    natl data )
```

Scrive un long (4 byte) nello spazio di configurazione PCI.

Parametri

<i>bus</i>	numero di bus
<i>dev</i>	numero di dispositivo
<i>fun</i>	numero di funzione
<i>regn</i>	offset del long su cui scrivere
<i>data</i>	valore da scrivere

Definizione alla linea 5 del file write_conf1.cpp.

7.5.2.10 write_confw()

```
void pci::write_confw (
    natb bus,
    natb dev,
    natb fun,
    natb regn,
    natw data )
```

Scrive una word (2 byte) nello spazio di configurazione PCI.

Parametri

<i>bus</i>	numero di bus
<i>dev</i>	numero di dispositivo
<i>fun</i>	numero di funzione
<i>regn</i>	offset della word su cui scrivere
<i>data</i>	valore da scrivere

Definizione alla linea 5 del file write_confw.cpp.

7.6 Riferimenti per il namespace serial

Funzioni per interagire con le due porte seriali.

Funzioni

- void `init1` ()
inizializza la porta seriale 1.
- void `out1` (`natb` c)
invia un byte sulla porta seriale 1.
- `natb in1` ()
riceve un byte dalla porta seriale 1.
- void `init2` ()
inizializza la porta seriale 2.
- void `out2` (`natb` c)
invia un byte sulla porta seriale 2.
- `natb in2` ()
ricevi un byte dalla porta seriale 2.

7.6.1 Descrizione dettagliata

Funzioni per interagire con le due porte seriali.

Nota

Le porte seriali non sono più trattate a lezione, ma sono usate per implementare il log. Ci sono comunque due esempi in esempiIO: seriale-1 e seriale-2.

7.6.2 Documentazione delle funzioni

7.6.2.1 `in1()`

```
natb serial::in1 ( )
```

riceve un byte dalla porta seriale 1.

Restituisce

byte ricevuto

Definizione alla linea 5 del file in1.cpp.

7.6.2.2 in2()

```
natb serial::in2 ( )
```

ricevi un byte dalla porta seriale 2.

Restituisce

byte ricevuto

7.6.2.3 out1()

```
void serial::out1 (
    natb c )
```

invia un byte sulla porta seriale 1.

Parametri

c	byte da inviare
---	-----------------

Definizione alla linea 5 del file out1.cpp.

7.6.2.4 out2()

```
void serial::out2 (
    natb c )
```

invia un byte sulla porta seriale 2.

Parametri

c	byte da inviare
---	-----------------

Definizione alla linea 5 del file out2.cpp.

7.7 Riferimenti per il namespace std

Tipo standard per la definizione della new con allineamento.

7.7.1 Descrizione dettagliata

Tipo standard per la definizione della new con allineamento.

7.8 Riferimenti per il namespace svga

Funzioni per interagire con il video in modalità grafica.

Funzioni

- volatile `natb` * `config` (`natw` max_screenx, `natw` max_screeny)

Configura la modalità grafica a 256 colori.

7.8.1 Descrizione dettagliata

Funzioni per interagire con il video in modalità grafica.

Dispensa: <https://calcolatori.iet.unipi.it/resources/periferiche.pdf>

Esempio: svga-1, svga-2, svga-2

7.8.2 Documentazione delle funzioni

7.8.2.1 config()

```
volatile natb * svga::config (  
    natw max_screenx,  
    natw max_screeny )
```

Configura la modalità grafica a 256 colori.

Parametri

<code>max_screenx</code>	pixel orizzontali
<code>max_screeny</code>	pixel verticali

Restituisce

indirizzo del framebuffer

Definizione alla linea 5 del file config.cpp.

7.9 Riferimenti per il namespace timer

Funzioni per interagire con il timer.

Funzioni

- void `start0` (`natw` `N`)
Programma il timer 0 in modo che invii una richiesta di interruzione periodica.
- void `start2` (`natw` `N`)
Programma il timer 2 in modo che produca un'onda quadra.
- void `enable_spk` ()
Abilita lo speaker a ricevere il segnale proveniente dal timer 2.
- void `disable_spk` ()
Disabilita lo speaker a ricevere il segnale proveniente dal timer 2.

7.9.1 Descrizione dettagliata

Funzioni per interagire con il timer.

Dispensa: <https://calcolatori.iet.unipi.it/resources/periferiche.pdf>

EsempioO: timer

7.9.2 Documentazione delle funzioni

7.9.2.1 `start0()`

```
void timer::start0 (
    natw N )
```

Programma il timer 0 in modo che invii una richiesta di interruzione periodica.

Parametri

<code>N</code>	periodo delle richieste di interruzione
----------------	---

Definizione alla linea 5 del file `start0.cpp`.

7.9.2.2 `start2()`

```
void timer::start2 (
    natw N )
```

Programma il timer 2 in modo che produca un'onda quadra.

Parametri

<code>N</code>	periodo dell'onda quadra
----------------	--------------------------

Definizione alla linea 5 del file start2.cpp.

7.10 Riferimenti per il namespace vid

Funzioni per interagire con il video in modalità testo.

Funzioni

- void `clear` (`natb` col)
Ripulisce il video e setta l'attributo colore.
- void `char_write` (`char` c)
Scrivono un carattere nella posizione corrente del cursore.
- void `str_write` (`const char` str[])
Scrivono una stringa (null-terminated) a partire dalla posizione corrente del cursore.
- void `char_put` (`char` c, `natw` x, `natw` y)
Scrivono un carattere in una posizione qualsiasi dello schermo.
- volatile `natw` & `pos` (`natw` x, `natw` y)
Restituisce un riferimento ad una posizione dello schermo, date le coordinate.
- `natl cols` ()
Restituisce il numero di colonne del video (modalità testo).
- `natl rows` ()
Restituisce il numero di righe del video (modalità testo).

7.10.1 Descrizione dettagliata

Funzioni per interagire con il video in modalità testo.

Dispensa: <https://calcolatori.iet.unipi.it/resources/periferiche.pdf>

EsempiO: video-testo-1, video-testo-2

7.10.2 Documentazione delle funzioni

7.10.2.1 `char_put()`

```
void vid::char_put (  
    char c,  
    natw x,  
    natw y )
```

Scrivono un carattere in una posizione qualsiasi dello schermo.

Parametri

<i>c</i>	codice ASCII del carattere da scrivere
<i>x</i>	colonna in cui scrivere
<i>y</i>	riga in cui scrivere

Definizione alla linea 5 del file char_put.cpp.

7.10.2.2 char_write()

```
void vid::char_write (
    char c )
```

Scrive un carattere nella posizione corrente del cursore.

Parametri

<i>c</i>	codice ASCII del carattere da scrivere
----------	--

Definizione alla linea 5 del file char_write.cpp.

7.10.2.3 clear()

```
void vid::clear (
    natb col )
```

Ripulisce il video e setta l'attributo colore.

Parametri

<i>col</i>	attributo colore
------------	------------------

Definizione alla linea 5 del file clear.cpp.

7.10.2.4 cols()

```
natl vid::cols ( )
```

Restituisce il numero di colonne del video (modalità testo).

Restituisce

numero di colonne

Definizione alla linea 5 del file cols.cpp.

7.10.2.5 pos()

```
volatile natw & vid::pos (
    natw x,
    natw y )
```

Restituisce un riferimento ad una posizione dello schermo, date le coordinate.

Parametri

<i>x</i>	colonna della posizione richiesta
<i>y</i>	riga della posizione richiesta

Restituisce

riferimento alla corrispondente word

Definizione alla linea 5 del file pos.cpp.

7.10.2.6 rows()

```
natl vid::rows ( )
```

Restituisce il numero di righe del video (modalità testo).

Restituisce

numero di righe

Definizione alla linea 5 del file rows.cpp.

7.10.2.7 str_write()

```
void vid::str_write (
    const char str[] )
```

Scrive una stringa (null-terminated) a partire dalla posizione corrente del cursore.

Parametri

<i>str</i>	stringa da scrivere
------------	---------------------

Definizione alla linea 5 del file str_write.cpp.

Capitolo 8

Documentazione delle classi

8.1 Riferimenti per la classe `tab_iter`

Iteratore per la visita di un TRIE.

Membri pubblici

- `tab_iter` (`paddr` `tab`, `vaddr` `beg`, `natq` `dim=1`)
inizializza un `tab_iter` per una visita in ordine anticipato.
- `operator bool` () const
visita terminata?
- `int get_l` () const
livello corrente
- `bool is_leaf` () const
fermo su una foglia?
- `vaddr get_v` () const
indirizzo virtuale corrente
- `tab_entry & get_e` () const
descrittore corrente
- `paddr get_tab` () const
tabella corrente
- `bool down` ()
prova a spostare l'iteratore di una posizione in basso nell'albero, se possibile, altrimenti non fa niente.
- `bool up` ()
prova a spostare l'iteratore di una posizione in alto nell'albero, se possibile, altrimenti non fa niente.
- `bool right` ()
prova a spostare l'iteratore di una posizione a destra nell'albero (rimanendo nel livello corrente), se possibile, altrimenti non fa niente.
- `void next` ()
porta l'iteratore alla prossima posizione della visita in ordine anticipato
- `void post` ()
inizia una visita in ordine posticipato
- `void next_post` ()
porta l'iteratore alla prossima posizione della visita in ordine posticipato

Membri pubblici statici

- static bool `valid_interval` (`vaddr` beg, `natq` dim)
controlla che un intervallo sia valido.

8.1.1 Descrizione dettagliata

Iteratore per la visita di un TRIE.

Definizione alla linea 201 del file vm.h.

8.1.2 Documentazione dei costruttori e dei distruttori

8.1.2.1 `tab_iter()`

```
tab_iter::tab_iter (
    paddr tab,
    vaddr beg,
    natq dim = 1 )
```

inizializza un `tab_iter` per una visita in ordine anticipato.

L'iteratore si fermerà su tutte le entrate, di tutti i livelli, coinvolte nella traduzione degli indirizzi in $[beg, beg + dim)$.

Parametri

<i>tab</i>	indirizzo fisico della tabella radice del TRIE
<i>beg</i>	base dell'intervallo
<i>dim</i>	dimensione in byte dell'intervallo

Definizione alla linea 3 del file `tab_iter.cpp`.

8.1.3 Documentazione delle funzioni membro

8.1.3.1 `down()`

```
bool tab_iter::down ( )
```

prova a spostare l'iteratore di una posizione in basso nell'albero, se possibile, altrimenti non fa niente.

Restituisce

true se lo spostamento è avvenuto, false altrimenti

Definizione alla linea 3 del file `down.cpp`.

8.1.3.2 `get_e()`

```
tab_entry& tab_iter::get_e ( ) const [inline]
```

descrittore corrente

Restituisce

riferimento al descrittore (di tabella o di pagina) corrente

Definizione alla linea 312 del file `vm.h`.

8.1.3.3 `get_l()`

```
int tab_iter::get_l ( ) const [inline]
```

livello corrente

Restituisce

livello su cui si trova l'iteratore

Definizione alla linea 287 del file `vm.h`.

8.1.3.4 `get_tab()`

```
paddr tab_iter::get_tab ( ) const [inline]
```

tabella corrente

Restituisce

indirizzo fisico della tabella che contiene il descrittore corrente

Definizione alla linea 320 del file `vm.h`.

8.1.3.5 `get_v()`

```
vaddr tab_iter::get_v ( ) const [inline]
```

indirizzo virtuale corrente

Restituisce

il più piccolo indirizzo virtuale appartenente all'intervallo e la cui traduzione passa dal descrittore corrente

Definizione alla linea 305 del file `vm.h`.

8.1.3.6 is_leaf()

```
bool tab_iter::is_leaf ( ) const [inline]
```

fermo su una foglia?

Restituisce

true se l'iteratore si trova su una foglia, false altrimenti

Definizione alla linea 295 del file vm.h.

8.1.3.7 operator bool()

```
tab_iter::operator bool ( ) const [inline]
```

visita terminata?

Restituisce

true se la visita è terminata, false altrimenti

Definizione alla linea 280 del file vm.h.

8.1.3.8 right()

```
bool tab_iter::right ( )
```

prova a spostare l'iteratore di una posizione a destra nell'albero (restando nel livello corrente), se possibile, altrimenti non fa niente.

Restituisce

true se lo spostamento è avvenuto, false altrimenti

Definizione alla linea 3 del file right.cpp.

8.1.3.9 up()

```
bool tab_iter::up ( )
```

prova a spostare l'iteratore di una posizione in alto nell'albero, se possibile, altrimenti non fa niente.

Restituisce

true se lo spostamento è avvenuto, false altrimenti

Definizione alla linea 3 del file up.cpp.

8.1.3.10 valid_interval()

```
static bool tab_iter::valid_interval (
    vaddr beg,
    natq dim ) [inline], [static]
```

controlla che un intervallo sia valido.

Un intervallo valido se è interamente contenuto nella parte bassa o alta dello spazio di indirizzamento, oppure se è vuoto.

Parametri

<i>beg</i>	base dell'intervallo
<i>dim</i>	dimensione in byte dell'intervallo (≥ 0)

Restituisce

true se l'intervallo è valido, false altrimenti

Definizione alla linea 250 del file `vm.h`.

Capitolo 9

Documentazione dei file

9.1 Riferimenti per il file include/libce.h

File che va incluso sempre per primo.

Namespace

- [std](#)
Tipo standard per la definizione della new con allineamento.
- [kbd](#)
Funzioni per interagire con la tastiera.
- [vid](#)
Funzioni per interagire con il video in modalità testo.
- [svga](#)
Funzioni per interagire con il video in modalità grafica.
- [timer](#)
Funzioni per interagire con il timer.
- [hd](#)
Funzioni per interagire con l'hard disk.
- [pci](#)
Funzioni per il bus PCI.
- [bm](#)
Funzioni per il PCI BUS Mastering ATA.
- [serial](#)
Funzioni per interagire con le due porte seriali.
- [apic](#)
Funzioni per interagire con l'APIC.

Definizioni

- #define **KiB** 1024ULL
kibibyte
- #define **MiB** (1024***KiB**)
mibibyte
- #define **GiB** (1024***MiB**)
gibibyte
- #define **DIM_PAGINA** (4***KiB**)
dimensione in byte di una pagina o di un frame
- #define **DIM_BLOCK** 512ULL
dimensione in byte di un blocco dell'hard disk
- #define **LIV_SISTEMA** 0
DPL del livello sistema.
- #define **LIV_UTENTE** 3
DPL del livello utente.
- #define **DIM_GDT_ENTRY** 8
dimensione in byte di una entrata della GDT
- #define **NUM_GDT_ENTRIES** 6
numero di righe della GDT
- #define **DIM_TSS** 104
dimensione in byte del segmento TSS

Indici all'interno della GDT

- #define **ID_CODE_SYS** 1
segmento codice/sistema
- #define **ID_CODE_USR** 2
segmento codice/utente
- #define **ID_DATA_USR** 3
segmento dati/utente
- #define **ID_SYS_TSS** 4
segmento TSS

Selettori di segmento

- #define **SEL_NULLO** 0
selettore nullo
- #define **SEL_CODICE_SISTEMA** ((**ID_CODE_SYS** << 3) | **LIV_SISTEMA**)
selettore del segmento codice di livello sistema
- #define **SEL_CODICE_UTENTE** ((**ID_CODE_USR** << 3) | **LIV_UTENTE**)
selettore del segmento codice di livello utente
- #define **SEL_DATI_UTENTE** ((**ID_DATA_USR** << 3) | **LIV_UTENTE**)
selettore del segmento dati scrivibili di livello utente
- #define **SEL_SYS_TSS** ((**ID_SYS_TSS** << 3) | **LIV_SISTEMA**)
selettore del segmento TSS

Ridefinizioni di tipo (typedef)

- using `ioaddr` = unsigned short
indirizzo nello spazio di I/O
- using `natb` = unsigned char
naturale su un byte
- using `natw` = unsigned short
naturale su due byte
- using `natl` = unsigned int
naturale su 4 byte
- using `natq` = unsigned long
naturale su 8 byte (64bit)
- using `vaddr` = unsigned long
indirizzo virtuale (64bit)
- using `paddr` = unsigned long
indirizzo fisico (64bit)

Tipi richiesti dallo standard (64bit)

- typedef unsigned long `size_t`
tipo restituito da sizeof
- typedef long `ssize_t`
tipo che può contenere una dimensione o un errore
- typedef long `ptrdiff_t`
tipo che può contenere il risultato della sottrazione tra due puntatori
- typedef long `intmax_t`
tipo intero più capiente supportato dal sistema
- typedef unsigned long `uintmax_t`
tipo intero senza segno più capiente supportato dal sistema
- typedef unsigned long `uintptr_t`
tipo senza segno che può contenere il valore di un puntatore

Tipi enumerati (enum)

- enum `log_sev` {
 `LOG_DEBUG` , `LOG_INFO` , `LOG_WARN` , `LOG_ERR` ,
 `LOG_USR` }
Livello di severità del messaggio inviato al log (usato per colorare i messaggi ove previsto)
- enum `hd::cmd` { `hd::WRITE_SECT` = 0x30 , `hd::READ_SECT` = 0x20 , `hd::WRITE_DMA` = 0xCA ,
 `hd::READ_DMA` = 0xC8 }
Possibili valori da scrivere nel registro di comando dell'hard disk.

Funzioni

- template<typename T >
 T `max` (T a, T b)
Restituisce il massimo tra due valori confrontabili.
- template<typename T >
 T `min` (T a, T b)
Restituisce il minimo tra due valori confrontabili.
- void * `memset` (void *dest, int c, `size_t` n)
Scrive lo stesso valore in tutti i byte di un intervallo.

- void * [memcpy](#) (void *dest, const void *src, [size_t](#) n)
Copia un intervallo di memoria su un altro (i due intervalli non possono sovrapporsi).
- [size_t strlen](#) (const char str[])
Calcola la lunghezza di una stringa.
- int [printf](#) (const char *fmt,...)
Scrittura formattata su schermo.
- int [snprintf](#) (char *buf, [natl](#) n, const char *fmt,...)
Scrittura formattata su un buffer in memoria.
- void [flog](#) ([log_sev](#) sev, const char *fmt,...)
Invio di un messaggio formattato sul log.
- void [pause](#) ()
Stampa un messaggio e attende che venga premuto il tasto ESC.
- void [fpanic](#) (const char *fmt,...)
Invia un messaggio sul log (severità ERR) ed esegue lo shutdown.
- template<typename To , typename From >
static To * [ptr_cast](#) (From v)
Converte da intero a puntatore non void.
- template<typename From >
static void * [voidptr_cast](#) (From v)
Converte da intero a puntatore a void.
- template<typename To , typename From >
static To [int_cast](#) (From *p)
Converte da puntatore a intero.
- static [natq allinea](#) ([natq](#) v, [natq](#) a)
Restituisce il più piccolo multiplo di a maggiore o uguale a v.
- template<typename T >
static T * [allinea_ptr](#) (T *p, [natq](#) a)
Restituisce il più piccolo puntatore a T allineato ad a e maggiore o uguale a p.
- long int [random](#) ()
Restituisce un intero pseudo-causale.
- void [setseed](#) ([natl](#) seed)
Imposta il seme iniziale del generatore di numeri pseudo-casuali.
- void [heap_init](#) (void *start, [size_t](#) size, [natq](#) initmem=0)
Inizializza un intervallo di memoria in modo che possa essere usata con [alloc\(\)](#), [alloc_aligned\(\)](#) e [dealloc\(\)](#).
- void * [alloc](#) ([size_t](#) dim)
Alloca una zona di memoria nello heap.
- void * [alloc_aligned](#) ([size_t](#) dim, std::align_val_t align)
Alloca una zona di memoria nello heap, con vincoli di allineamento.
- void [dealloc](#) (void *p)
Dealloca una zona di memoria, restituendola allo heap.
- [size_t disponibile](#) ()
Memoria libera nello heap.
- [natb inputb](#) ([ioaddr](#) reg)
Legge un byte da una porta di I/O.
- [natw inputw](#) ([ioaddr](#) reg)
Legge una word (2 byte) da una porta di I/O.
- [natl inputl](#) ([ioaddr](#) reg)
Legge un long (4 byte) da una porta di I/O.
- void [inputbw](#) ([ioaddr](#) reg, [natw](#) vetti[], int n)
Legge una successione di word (2 byte) da una porta di I/O.
- void [outputb](#) ([natb](#) a, [ioaddr](#) reg)

- Invia un byte ad una porta di I/O.*
- void `outputw` (`natw` a, `ioaddr` reg)
- Invia una word (2 byte) ad una porta di I/O.*
- void `outputl` (`natl` a, `ioaddr` reg)
- Invia un long (4 byte) ad una porta di I/O.*
- void `outputbw` (`natw` vetto[], int n, `ioaddr` reg)
- Invia una successione di word (2 byte) ad una porta di I/O.*
- `natb kbd::get_code` ()
- Restituisce l'ultimo codice di scansione ricevuto dalla tastiera (esegue una attesa attiva fino a quando il codice non è disponibile).*
- char `kbd::conv` (`natb` c)
- Converte un codice di scansione nel corrispondente codice ASCII.*
- char `kbd::char_read` ()
- Restituisce il codice ASCII dell'ultimo carattere letto dalla tastiera (esegue una attesa attiva fino a quando il carattere non è disponibile).*
- char `kbd::char_read_intr` ()
- Restituisce il codice ASCII corrispondente al codice di scansione contenuto in RBR (non esegue una attesa attiva).*
- void `kbd::enable_intr` ()
- Abilita l'interfaccia della tastiera a generare richieste di interruzione.*
- void `kbd::disable_intr` ()
- Disabilita l'interfaccia della tastiera a generare richieste di interruzione.*
- void `kbd::drain` ()
- Svuota il buffer della tastiera.*
- void `vid::clear` (`natb` col)
- Ripulisce il video e setta l'attributo colore.*
- void `vid::char_write` (char c)
- Scriva un carattere nella posizione corrente del cursore.*
- void `vid::str_write` (const char str[])
- Scriva una stringa (null-terminated) a partire dalla posizione corrente del cursore.*
- void `vid::char_put` (char c, `natw` x, `natw` y)
- Scriva un carattere in una posizione qualsiasi dello schermo.*
- volatile `natw` & `vid::pos` (`natw` x, `natw` y)
- Restituisce un riferimento ad una posizione dello schermo, date le coordinate.*
- `natl vid::cols` ()
- Restituisce il numero di colonne del video (modalità testo).*
- `natl vid::rows` ()
- Restituisce il numero di righe del video (modalità testo).*
- volatile `natb` * `svga::config` (`natw` max_screenx, `natw` max_screeny)
- Configura la modalità grafica a 256 colori.*
- void `timer::start0` (`natw` N)
- Programma il timer 0 in modo che invii una richiesta di interruzione periodica.*
- void `timer::start2` (`natw` N)
- Programma il timer 2 in modo che produca un'onda quadra.*
- void `timer::enable_spk` ()
- Abilita lo speaker a ricevere il segnale proveniente dal timer 2.*
- void `timer::disable_spk` ()
- Disabilita lo speaker a ricevere il segnale proveniente dal timer 2.*
- void `hd::start_cmd` (`natl` lba, `natb` quanti, cmd cmd)
- Avvia una operazione sull'hard disk.*
- void `hd::output_sect` (`natb` *buf)
- Scriva un settore nel buffer interno dell'interfaccia dell'hard disk.*

- void `hd::input_sect` (`natb *buf`)
Legge un settore dal buffer interno dell'interfaccia dell'hard disk.
- void `hd::enable_intr` ()
Abilita l'interfaccia dell'hard disk a generare richieste di interruzione.
- void `hd::disable_intr` ()
Disabilita l'interfaccia dell'hard disk a generare richieste di interruzione.
- void `hd::ack` ()
Azione di risposta alla richiesta di interruzione dell'interfaccia dell'hard disk.
- `natb pci::read_confb` (`natb bus`, `natb dev`, `natb fun`, `natb regn`)
Legge un byte dallo spazio di configurazione PCI.
- `natw pci::read_confw` (`natb bus`, `natb dev`, `natb fun`, `natb regn`)
Legge una word (2 byte) dallo spazio di configurazione PCI.
- `natl pci::read_confl` (`natb bus`, `natb dev`, `natb fun`, `natb regn`)
Legge un long (4 byte) dallo spazio di configurazione PCI.
- void `pci::write_confb` (`natb bus`, `natb dev`, `natb fun`, `natb regn`, `natb data`)
Scriva un byte nello spazio di configurazione PCI.
- void `pci::write_confw` (`natb bus`, `natb dev`, `natb fun`, `natb regn`, `natw data`)
Scriva una word (2 byte) nello spazio di configurazione PCI.
- void `pci::write_confl` (`natb bus`, `natb dev`, `natb fun`, `natb regn`, `natl data`)
Scriva un long (4 byte) nello spazio di configurazione PCI.
- bool `pci::find_dev` (`natb &bus`, `natb &dev`, `natb &fun`, `natw vendorID`, `natw deviceID`)
Trova una funzione PCI dato il vendor ID e il device ID.
- bool `pci::find_class` (`natb &bus`, `natb &dev`, `natb &fun`, `natb code[]`)
Trova una funzione PCI dato il Class Code.
- bool `pci::next` (`natb &bus`, `natb &dev`, `natb &fun`)
Calcola le prossime coordinate nel bus PCI.
- const char * `pci::decode_class` (`natb class_code`)
Decodifica il primo byte di un Class Code.
- bool `bm::find` (`natb &bus`, `natb &dev`, `natb &fun`)
cerca il prossimo bus master ATA.
- void `bm::init` (`natb bus`, `natb dev`, `natb fun`)
inizializza un bus master.
- void `bm::prepare` (`paddr prd`, bool `write`)
prepara una operazione di bus mastering.
- void `bm::start` ()
avvia l'operazione di bus mastering precedentemente preparata.
- void `bm::ack` ()
azione di risposta alle richieste di interruzione del bus master
- void `serial::init1` ()
inizializza la porta seriale 1.
- void `serial::out1` (`natb c`)
invia un byte sulla porta seriale 1.
- `natb serial::in1` ()
riceve un byte dalla porta seriale 1.
- void `serial::init2` ()
inizializza la porta seriale 2.
- void `serial::out2` (`natb c`)
invia un byte sulla porta seriale 2.
- `natb serial::in2` ()
ricevi un byte dalla porta seriale 2.
- void `gate_init` (`natb num`, void `routine()`, bool `trap=false`, int `liv=LIV_UTENTE`)

- Inizializza un gate della IDT.*
- bool `gate_present` (`natb` num)
- Controlla che un gate non sia già occupato.*
- bool `apic::init` ()
- Inizializza l'APIC.*
- void `apic::send_EOI` ()
- Invia l'End Of Interrupt.*
- void `apic::set_MIRQ` (`natl` irq, bool enable)
- Maschera/smaschera una sorgente di richieste di interruzione.*
- void `apic::set_TRGM` (`natl` irq, bool enable)
- Setta la modalità di riconoscimento di una sorgente di richieste di interruzione.*
- void `apic::set_VECT` (`natl` irq, `natb` vec)
- Imposta il vettore di interruzione associato ad una sorgente di richieste di interruzione.*
- void `apic_send_EOI` ()
- Invia l'End Of Interrupt.*
- void `log_exception` (int tipo, `natq` errore, `vaddr` rip)
- Decodifica delle eccezioni.*
- int `vsprintf` (char *buf, `size_t` size, const char *fmt, va_list ap)
- Come `snprintf`, ma usa una `va_list` esplicita invece di essere variadica.*
- bool `find_elf_frame` (`vaddr` elf, `vaddr` &eh_frame, `natq` &eh_frame_len)
- Trova l'exception header all'interno di un file ELF caricato in memoria.*
- void `reboot` ()
- Riavvia il sistema.*
- void `do_log` (`log_sev` sev, const char *buf, `natl` quanti)
- Funzione di basso livello per la scrittura sul log.*

Overload standard.

Overloading degli operatori di default normalmente forniti dalla libreria standard del C++. Si limitano a richiamare in modo appropriato `operator new` e `operator delete`, che devono essere definiti a parte.

- void * `operator new[]` (`size_t` s)
- versione di new per l'allocazione di array*
- void * `operator new[]` (`size_t` s, std::align_val_t a)
- versione di new per l'allocazione di array con allineamento*
- void `operator delete` (void *p, `size_t` s)
- versione di delete con dimensione esplicita*
- void `operator delete` (void *p, `size_t` s, std::align_val_t)
- versione di delete con dimensione esplicita e allineamento*
- void `operator delete[]` (void *p)
- versione di delete per la deallocazione degli array*
- void `operator delete[]` (void *p, `size_t` s)
- versione di delete per la deallocazione degli array con dimensione esplicita*

Variabili

- const `natq` `BIT_IF` = 1UL << 9
- Interrupt Flag.*
- const `natq` `BIT_TF` = 1UL << 8
- Trap Flag.*
- static const int `apic::MAX_IRQ` = 24
- Massimo numero di IRQ supportati dall'APIC.*

Eccezione di Page Fault (14)

Il microprogramma di gestione delle eccezioni di page fault lascia in cima alla pila (oltre ai valori consueti) una parola quadrupla i cui 4 bit meno significativi specificano più precisamente il motivo per cui si è verificato un page fault.

- static const `natq PF_PROT` = 1U << 0
Page fault causato da errore di protezione.
- static const `natq PF_WRITE` = 1U << 1
Page fault con accesso in scrittura.
- static const `natq PF_USER` = 1U << 2
Page fault con accesso da livello utente.
- static const `natq PF_RES` = 1U << 3
Page fault con bit riservati non validi.

Eccezioni con codice di errore.

Alcune eccezioni, per lo più legate al meccanismo della segmentazione, lasciano in pila un codice di errore che è l'offset all'interno di una delle tre tabelle IDT, GDT o LDT (quest'ultima non usata da noi) e di tre bit che specificano meglio il tipo di errore.

- static const `natq SE_EXT` = 1U << 0
Eccezione causata da evento esterno.
- static const `natq SE_IDT` = 1U << 1
Eccezione durante accesso alla IDT.
- static const `natq SE_TI` = 1U << 2
Table Indicator dell'eccezione.

9.1.1 Descrizione dettagliata

File che va incluso sempre per primo.

9.2 Riferimenti per il file `include/vm.h`

file da includere (dopo `libce.h`) per usare le funzioni della memoria virtuale

Strutture dati

- class `tab_iter`
Iteratore per la visita di un TRIE.

Ridefinizioni di tipo (typedef)

- using `tab_entry` = `natq`
descrittore di pagina o tabella

Funzioni

- static constexpr `vaddr norm (vaddr a)`
Normalizza un indirizzo.
- static constexpr `natq dim_region (int liv)`
Calcola la dimensione di una regione del TRIE dato il livello.
- static constexpr `vaddr base (vaddr v, int liv)`
Calcola la base della regione di un dato livello a cui appartiene un dato indirizzo virtuale.
- static constexpr `vaddr limit (vaddr v, int liv)`
Calcola la base della prima regione di un dato livello che giace a destra di un dato indirizzo virtuale.
- static constexpr `paddr extr_IND_FISICO (tab_entry e)`
Estrae l'indirizzo fisico da un descrittore di pagina o tabella.
- static void `set_IND_FISICO (tab_entry &e, paddr f)`
Imposta l'indirizzo fisico in un descrittore di pagina o tabella.
- static constexpr int `i_tab (vaddr v, int liv)`
Indice nelle tabelle.
- static `tab_entry & get_entry (paddr tab, natl i)`
Accesso ad un'entrata di una tabella.
- void `set_entry (paddr tab, natl j, tab_entry se)`
Scriva una entrata di una tabella.
- void `copy_des (paddr src, paddr dst, natl i, natl n)`
Copia descrittori da una tabella ad un'altra.
- void `set_des (paddr dst, natl i, natl n, tab_entry e)`
Inizializza (parte dei) descrittori di una tabella.
- `paddr trasforma (paddr root_tab, vaddr v)`
Traduzione di un indirizzo virtuale in fisico.
- void `loadCR3 (paddr dir)`
Carica un nuovo valore in cr3.
- `paddr readCR3 ()`
Lettura di cr3.
- void `invalida_TLB ()`
Invalida tutto il TLB.
- void `invalida_entrata_TLB (vaddr v)`
Invalida una entrata del TLB.
- `vaddr readCR2 ()`
Lettura di cr2.
- template<typename T >
`vaddr map (paddr tab, vaddr begin, vaddr end, natl flags, T &getpaddr, int ps_lvl=1)`
Crea tutto il sottoalbero necessario a tradurre tutti gli indirizzi di un intervallo.
- template<typename T >
`vaddr map (paddr tab, vaddr begin, vaddr end, natl flags, const T &getpaddr, int ps_lvl=1)`
Overloading di `map()` per il caso in cui `getpaddr` non sia un lvalue.
- template<typename T >
void `unmap (paddr tab, vaddr begin, vaddr end, T &putpaddr)`
Elimina tutte le traduzioni di un intervallo.
- template<typename T >
void `unmap (paddr tab, vaddr begin, vaddr end, const T &putpaddr)`
Overloading di `unmap()` per il caso in cui `putpaddr` non sia un lvalue.

Funzioni usate da map() e unmap().

Queste funzioni devono essere definite per poter usare, in particolare, `map()` e `unmap()`. La libce fornisce una versione semplificata che può essere usata dai programmi 'bare'. La versione semplificata alloca le tabelle sullo heap e non le rilascia mai. Dal momento che non le rilascia mai, non gestisce nemmeno un contatore delle entrate valide delle tabelle. Il nucleo fornisce le proprie versioni di queste funzioni, che gestiscono correttamente i contatori delle entrate valide e permettono di deallocare le tabelle.

- `paddr alloca_tab ()`
invocata quando serve una nuova tabella
- `void rilascia_tab (paddr)`
invocata quando una tabella non serve più
- `void inc_ref (paddr)`
invocata quando una tabella acquisisce una nuova entrata valida
- `void dec_ref (paddr)`
invocata quando una tabella perde una entrata precedentemente valida
- `natl get_ref (paddr)`
invocata per leggere il contatore delle entrate valide di una tabella

Variabili

- `static const int MAX_LIV = 4`
numero di livelli del TRIE (4 o 5)
- `static const int MAX_PS_LVL = 2`
massimo livello supportato per le pagine di grandi dimensioni
- `static const natq VADDR_MSBIT = (1ULL << (12 + 9 * MAX_LIV - 1))`
ultimo bit significativo di un indirizzo virtuale
- `static const natq VADDR_MASK = VADDR_MSBIT - 1`
maschera per selezionare i bit significativi di un indirizzo virtuale

Bit del byte di accesso

- `const natq BIT_P = 1U << 0`
bit di presenza
- `const natq BIT_RW = 1U << 1`
bit di lettura/scrittura
- `const natq BIT_US = 1U << 2`
bit utente/sistema
- `const natq BIT_PWT = 1U << 3`
bit Page Wright Through
- `const natq BIT_PCD = 1U << 4`
bit Page Cache Disable
- `const natq BIT_A = 1U << 5`
bit di accesso
- `const natq BIT_D = 1U << 6`
bit "dirty"
- `const natq BIT_PS = 1U << 7`
bit "page size"
- `const natq ACCB_MASK = 0x00000000000000FF`
maschera per il byte di accesso
- `const natq ADDR_MASK = 0x7FFFFFFFFFFFFFFF000`
maschera per l'indirizzo

9.2.1 Descrizione dettagliata

file da includere (dopo `libce.h`) per usare le funzioni della memoria virtuale

Indice analitico

- alloc
 - Funzioni per la memoria dinamica., [26](#)
- alloc_aligned
 - Funzioni per la memoria dinamica., [26](#)
- apic, [53](#)
 - init, [53](#)
 - set_MIRQ, [54](#)
 - set_TRGM, [54](#)
 - set_VECT, [54](#)
- apic_send_EOI
 - Funzioni legate al meccanismo delle interruzioni., [33](#)
- base
 - Funzioni per la memoria virtuale., [43](#)
- BIT_IF
 - Funzioni legate al meccanismo delle interruzioni., [33](#)
- BIT_TF
 - Funzioni legate al meccanismo delle interruzioni., [33](#)
- bm, [55](#)
 - find, [55](#)
 - init, [56](#)
 - prepare, [56](#)
- char_put
 - vid, [71](#)
- char_read
 - kbd, [59](#)
- char_read_intr
 - kbd, [60](#)
- char_write
 - vid, [72](#)
- clear
 - vid, [72](#)
- cmd
 - hd, [57](#)
- cols
 - vid, [72](#)
- config
 - svga, [69](#)
- conv
 - kbd, [60](#)
- copy_des
 - Funzioni per la memoria virtuale., [44](#)
- Costanti, [15](#)
- Costanti usate da boot64., [40](#)
 - NUM_GDT_ENTRIES, [40](#)
- dealloc
 - Funzioni per la memoria dinamica., [26](#)
- decode_class
 - pci, [62](#)
- dim_region
 - Funzioni per la memoria virtuale., [44](#)
- disponibile
 - Funzioni per la memoria dinamica., [27](#)
- do_log
 - Funzioni di uso meno generale, usate dal nucleo., [38](#)
- down
 - tab_iter, [76](#)
- Eccezioni., [35](#)
 - log_exception, [35](#)
 - PF_PROT, [36](#)
 - PF_RES, [36](#)
 - PF_USER, [36](#)
 - PF_WRITE, [36](#)
 - SE_EXT, [37](#)
 - SE_IDT, [37](#)
 - SE_TI, [37](#)
- extr_IND_FISICO
 - Funzioni per la memoria virtuale., [44](#)
- find
 - bm, [55](#)
- find_class
 - pci, [62](#)
- find_dev
 - pci, [63](#)
- find_eh_frame
 - Funzioni di uso meno generale, usate dal nucleo., [38](#)
- flog
 - Funzioni di utilità generale, [18](#)
- fpanic
 - Funzioni di utilità generale, [19](#)
- Funzioni di uso meno generale, usate dal nucleo., [38](#)
 - do_log, [38](#)
 - find_eh_frame, [38](#)
 - reboot, [39](#)
 - vsnprintf, [39](#)
- Funzioni di utilità generale, [17](#)
 - flog, [18](#)
 - fpanic, [19](#)
 - int_cast, [19](#)
 - LOG_DEBUG, [18](#)
 - LOG_ERR, [18](#)

- LOG_INFO, 18
- log_sev, 18
- LOG_USR, 18
- LOG_WARN, 18
- max, 20
- memcpy, 20
- memset, 21
- min, 21
- printf, 22
- ptr_cast, 22
- setseed, 23
- snprintf, 23
- strlen, 24
- voidptr_cast, 24
- Funzioni legate al meccanismo delle interruzioni., 32
 - apic_send_EOI, 33
 - BIT_IF, 33
 - BIT_TF, 33
- Funzioni per l'interazione con le interfacce di I/O., 28
- Funzioni per la manipolazione della tabella IDT., 33
 - gate_init, 34
 - gate_present, 34
- Funzioni per la memoria dinamica., 25
 - alloc, 26
 - alloc_aligned, 26
 - dealloc, 26
 - disponibile, 27
 - heap_init, 27
- Funzioni per la memoria virtuale., 41
 - base, 43
 - copy_des, 44
 - dim_region, 44
 - extr_IND_FISICO, 44
 - get_entry, 45
 - i_tab, 45
 - invalida_entrata_TLB, 46
 - limit, 46
 - loadCR3, 46
 - map, 47
 - norm, 48
 - readCR2, 48
 - readCR3, 48
 - set_des, 48
 - set_entry, 49
 - set_IND_FISICO, 49
 - trasforma, 50
 - unmap, 50
 - VADDR_MSBIT, 52
- Funzioni per leggere/scrivere nello spazio di I/O., 28
 - inputb, 28
 - inputbw, 29
 - inputl, 29
 - inputw, 29
 - outputb, 31
 - outputbw, 31
 - outputl, 31
 - outputw, 32
- gate_init
 - Funzioni per la manipolazione della tabella IDT., 34
- gate_present
 - Funzioni per la manipolazione della tabella IDT., 34
- get_code
 - kbd, 60
- get_e
 - tab_iter, 76
- get_entry
 - Funzioni per la memoria virtuale., 45
- get_l
 - tab_iter, 77
- get_tab
 - tab_iter, 77
- get_v
 - tab_iter, 77
- hd, 57
 - cmd, 57
 - input_sect, 58
 - output_sect, 58
 - READ_DMA, 58
 - READ_SECT, 58
 - start_cmd, 58
 - WRITE_DMA, 58
 - WRITE_SECT, 58
- heap_init
 - Funzioni per la memoria dinamica., 27
- i_tab
 - Funzioni per la memoria virtuale., 45
- in1
 - serial, 67
- in2
 - serial, 67
- include/libce.h, 81
- include/vm.h, 88
- init
 - apic, 53
 - bm, 56
- input_sect
 - hd, 58
- inputb
 - Funzioni per leggere/scrivere nello spazio di I/O., 28
- inputbw
 - Funzioni per leggere/scrivere nello spazio di I/O., 29
- inputl
 - Funzioni per leggere/scrivere nello spazio di I/O., 29
- inputw
 - Funzioni per leggere/scrivere nello spazio di I/O., 29
- int_cast
 - Funzioni di utilità generale, 19
- invalida_entrata_TLB
 - Funzioni per la memoria virtuale., 46
- is_leaf
 - tab_iter, 77

- kbd, [59](#)
 - char_read, [59](#)
 - char_read_intr, [60](#)
 - conv, [60](#)
 - get_code, [60](#)
- limit
 - Funzioni per la memoria virtuale., [46](#)
- loadCR3
 - Funzioni per la memoria virtuale., [46](#)
- LOG_DEBUG
 - Funzioni di utilità generale, [18](#)
- LOG_ERR
 - Funzioni di utilità generale, [18](#)
- log_exception
 - Eccezioni., [35](#)
- LOG_INFO
 - Funzioni di utilità generale, [18](#)
- log_sev
 - Funzioni di utilità generale, [18](#)
- LOG_USR
 - Funzioni di utilità generale, [18](#)
- LOG_WARN
 - Funzioni di utilità generale, [18](#)
- map
 - Funzioni per la memoria virtuale., [47](#)
- max
 - Funzioni di utilità generale, [20](#)
- memcpy
 - Funzioni di utilità generale, [20](#)
- memset
 - Funzioni di utilità generale, [21](#)
- min
 - Funzioni di utilità generale, [21](#)
- next
 - pci, [63](#)
- norm
 - Funzioni per la memoria virtuale., [48](#)
- NUM_GDT_ENTRIES
 - Costanti usate da boot64., [40](#)
- operator bool
 - tab_iter, [78](#)
- out1
 - serial, [68](#)
- out2
 - serial, [68](#)
- output_sect
 - hd, [58](#)
- outputb
 - Funzioni per leggere/scrivere nello spazio di I/O., [31](#)
- outputbw
 - Funzioni per leggere/scrivere nello spazio di I/O., [31](#)
- outputl
 - Funzioni per leggere/scrivere nello spazio di I/O., [31](#)
- outputw
 - Funzioni per leggere/scrivere nello spazio di I/O., [32](#)
- pci, [61](#)
 - decode_class, [62](#)
 - find_class, [62](#)
 - find_dev, [63](#)
 - next, [63](#)
 - read_confb, [64](#)
 - read_confl, [64](#)
 - read_confw, [65](#)
 - write_confb, [65](#)
 - write_confl, [66](#)
 - write_confw, [66](#)
- PF_PROT
 - Eccezioni., [36](#)
- PF_RES
 - Eccezioni., [36](#)
- PF_USER
 - Eccezioni., [36](#)
- PF_WRITE
 - Eccezioni., [36](#)
- pos
 - vid, [72](#)
- prepare
 - bm, [56](#)
- printf
 - Funzioni di utilità generale, [22](#)
- ptr_cast
 - Funzioni di utilità generale, [22](#)
- read_confb
 - pci, [64](#)
- read_confl
 - pci, [64](#)
- read_confw
 - pci, [65](#)
- READ_DMA
 - hd, [58](#)
- READ_SECT
 - hd, [58](#)
- readCR2
 - Funzioni per la memoria virtuale., [48](#)
- readCR3
 - Funzioni per la memoria virtuale., [48](#)
- reboot
 - Funzioni di uso meno generale, usate dal nucleo., [39](#)
- right
 - tab_iter, [78](#)
- rows
 - vid, [73](#)
- SE_EXT
 - Eccezioni., [37](#)
- SE_IDT

- Eccezioni., 37
- SE_TI
 - Eccezioni., 37
- serial, 67
 - in1, 67
 - in2, 67
 - out1, 68
 - out2, 68
- set_des
 - Funzioni per la memoria virtuale., 48
- set_entry
 - Funzioni per la memoria virtuale., 49
- set_IND_FISICO
 - Funzioni per la memoria virtuale., 49
- set_MIRQ
 - apic, 54
- set_TRGM
 - apic, 54
- set_VECT
 - apic, 54
- setseed
 - Funzioni di utilità generale, 23
- snprintf
 - Funzioni di utilità generale, 23
- start0
 - timer, 70
- start2
 - timer, 70
- start_cmd
 - hd, 58
- std, 68
- str_write
 - vid, 73
- strlen
 - Funzioni di utilità generale, 24
- svga, 69
 - config, 69
- tab_iter, 75
 - down, 76
 - get_e, 76
 - get_l, 77
 - get_tab, 77
 - get_v, 77
 - is_leaf, 77
 - operator bool, 78
 - right, 78
 - tab_iter, 76
 - up, 78
 - valid_interval, 78
- timer, 69
 - start0, 70
 - start2, 70
- Tipi base, 15
- Tipi dipendenti dall'architettura, 16
- trasforma
 - Funzioni per la memoria virtuale., 50
- unmap
 - Funzioni per la memoria virtuale., 50
- up
 - tab_iter, 78
- VADDR_MSBIT
 - Funzioni per la memoria virtuale., 52
- valid_interval
 - tab_iter, 78
- vid, 71
 - char_put, 71
 - char_write, 72
 - clear, 72
 - cols, 72
 - pos, 72
 - rows, 73
 - str_write, 73
- voidptr_cast
 - Funzioni di utilità generale, 24
- vsprintf
 - Funzioni di uso meno generale, usate dal nucleo., 39
- write_confb
 - pci, 65
- write_confl
 - pci, 66
- write_confw
 - pci, 66
- WRITE_DMA
 - hd, 58
- WRITE_SECT
 - hd, 58