F1 produce un listato dei processi

```
--nr-name---- -prior-quant- -user---sys- -text---data---size- -rts flags-
(-4)  IDLE    15/15 06/08 451301   771     4K    24K    60K ------
[-3]  CLOCK   00/00 00/00     23     0     4K    24K    60K --R--- ANY
[-2]  SYSTEM  00/00 00/00    687     0     4K    24K    60K ------
[-1]  KERNEL  00/00 00/00      0     0     4K    24K    60K M-----
  0   pm      03/03 26/32     38     0  1024K  1044K    92K --R--- ANY
  1   fs      04/04 16/32     80     0  1116K  1160K  4928K --R--- ANY
  2   rs      03/03 03/04      2     0  6044K  6052K   160K --R--- ANY
  3   memory  02/02 04/04      0     0  6420K  6428K    16K --R--- ANY
  4   log     02/02 04/04     96     0  6436K  6444K    76K --R--- ANY
  5   tty     01/01 04/04    144     0  6340K  6368K    80K --R--- ANY
  6   driver  02/02 01/04     55     0  6512K  6536K    56K --R--- ANY
  7   ds      03/03 04/04      0     0  6204K  6208K   136K --R--- ANY
  8   init    07/07 01/08      1     9  6568K  6576K    16K --R--- pm
 10   sh      07/07 08/08      4    28  8384K  8448K   180K --R--- fs
 11   floppy  03/03 02/04      0     0   252K   264K    24K --R--- ANY
 12   is      03/03 02/04     11     0  6736K  6752K   256K --R--- SYSTEM
 13   cmos    03/03 02/04     16     0   276K   284K    16K --R--- ANY
 15   random  03/03 03/04     86     0   292K   312K    48K ------
 16   dp8390  03/03 04/04      1     0   340K   376K    60K --R--- ANY
 17   inet    03/03 03/04      3     0  7136K  7252K   896K --R--- ANY
 18   printer 03/03 01/04      0     0  8032K  8036K   136K --R--- ANY
 19   usyslog 07/07 05/08     13    72   536K   552K    44K --R--- fs
 20   cron    07/07 02/08      0    10  6992K  7016K    64K --R--- pm
,_more--
```

Durante send/receive il processo è bloccato

kernel/proc.h

```
struct proc {
  ...
  char p_rts_flags; /* SENDING, RECEIVING, etc. */
  ...
};

/* Bits for the runtime flags.
 * A process is runnable iff p_rts_flags == 0.
 */
...
#define SENDING 0x04 /* process blocked trying to SEND */
#define RECEIVING 0x08 /* process blocked trying to RECEIVE */
...
```

Implementazione del rendezvouz

kernel/proc.h

```
struct proc {
  ...
  struct proc *p_caller_q; /* head of list of procs wishing to send
  struct proc *p_q_link; /* link to next proc wishing to send */
  message *p_messbuf; /* pointer to passed message buffer */
  proc_nr_t p_getfrom; /* from whom does process want to receive? *
  proc_nr_t p_sendto; /* to whom does process want to send? */
  ...
};
```

Scheduling priorities

kernel/proc.h

```
/* Scheduling priorities for p_priority. Values must start at zero (highest
 * priority) and increment.  Priorities of the processes in the boot image
 * can be set in table.c. IDLE must have a queue for itself, to prevent low
 * priority user processes to run round-robin with IDLE.
 */
#define NR_SCHED_QUEUES   16    /* MUST equal minimum priority + 1 */
#define TASK_Q             0    /* highest, used for kernel tasks */
#define MAX_USER_Q         0    /* highest priority for user processes */
#define USER_Q             7    /* default (should correspond to nice 0) */
#define MIN_USER_Q        14    /* minimum priority for user processes */
#define IDLE_Q            15    /* lowest, only IDLE process goes here */
```

La process table vera e propria

kernel/proc.h

```
/* The process table and pointers to process table slots. The pointers allow
 * faster access because now a process entry can be found by indexing the
 * pproc_addr array, while accessing an element i requires a multiplication
 * with sizeof(struct proc) to determine the address.
 */
EXTERN struct proc proc[NR_TASKS + NR_PROCS]; /* process table */
EXTERN struct proc *pproc_addr[NR_TASKS + NR_PROCS];
EXTERN struct proc *rdy_head[NR_SCHED_QUEUES]; /* ptrs to ready list headers
EXTERN struct proc *rdy_tail[NR_SCHED_QUEUES]; /* ptrs to ready list tails */
```

Privilegi

kernel/proc.h

```
struct proc {
  ...
  struct priv *p_priv; /* system privileges structure */
  ...
};
```

I processi di sistema ne hanno una ciascuno

Tutti i processi utente ne condividono una

La struct priv

kernel/priv.h

```
struct priv {
  proc_nr_t s_proc_nr;          /* number of associated process */
  sys_id_t s_id;                /* index of this system structure */
  short s_flags;                /* PREEMTIBLE, BILLABLE, etc. */

  short s_trap_mask;            /* allowed system call traps */
  sys_map_t s_ipc_from;         /* allowed callers to receive from */
  sys_map_t s_ipc_to;           /* allowed destination processes */
  long s_call_mask;             /* allowed kernel calls */
  ...
};
```

F4 produce una lista dei privilegi dei processi

```
--nr-id-name---- -flags- -traps- -ipc_to mask------------------------
(-4) (01) IDLE    P-BS-    -----    00000000 00000111 11111000 00000000
[-3] (02) CLOCK   ---S-    --R--    00000000 00000111 11111000 00000000
[-2] (03) SYSTEM  ---S-    --R--    00000000 00000111 11111000 00000000
[-1] (04) KERNEL  ---S-    -----    00000000 00000111 11111000 00000000
  0  (05) pm      P--S-    ESRBN    11111111 11111111 11111000 00000000
  1  (06) fs      P--S-    ESRBN    11111111 11111111 11111000 00000000
  2  (07) rs      P--S-    ESRBN    11111111 11111111 11111000 00000000
  3  (10) memory  P--S-    ESRBN    11111111 11111111 11111000 00000000
  4  (11) log     P--S-    ESRBN    11111111 11111111 11111000 00000000
  5  (09) tty     P--S-    ESRBN    11111111 11111111 11111000 00000000
  6  (12) driver  P--S-    ESRBN    11111111 11111111 11111000 00000000
  7  (08) ds      P--S-    ESRBN    11111111 11111111 11111000 00000000
  8  (00) init    P-B--    E--B-    00010111 00000000 00000000 00000000
 10  (00) sh      P-B--    E--B-    00010111 00000000 00000000 00000000
 11  (13) floppy  P--S-    ESRBN    01111111 11111111 11111111 11111111
 12  (14) is      P--S-    ESRBN    01111111 11111111 11111111 11111111
 13  (15) cmos    P--S-    ESRBN    01111111 11111111 11111111 11111111
 15  (16) random  P--S-    ESRBN    01111111 11111111 11111111 11111111
 16  (17) dp8390  P--S-    ESRBN    01111111 11111111 11111111 11111111
 17  (18) inet    P--S-    ESRBN    01111111 11111111 11111111 11111111
 18  (19) printer P--S-    ESRBN    01111111 11111111 11111111 11111111
 19  (00) usyslog P-B--    E--B-    00010111 00000000 00000000 00000000
 20  (00) cron    P-B--    E--B-    00010111 00000000 00000000 00000000
--more--
```

La tabella delle struct priv

kernel/priv.h

```
/* The system structures table and pointers to individual table slots
 * pointers allow faster access because now a process entry can be fo
 * indexing the psys_addr array, while accessing an element i require
 * multiplication with sizeof(struct sys) to determine the address.
 */
EXTERN struct priv priv[NR_SYS_PROCS]; /* system properties table */
EXTERN struct priv *ppriv_addr[NR_SYS_PROCS]; /* direct slot pointers
```

La tabella della boot image in kernel/table.c

```
PUBLIC struct boot_image image[] = {
/* process nr,   pc, flags, qs,  queue, stack, traps, ipcto, call,  name */
 { IDLE,  idle_task, IDL_F,  8, IDLE_Q, IDL_S,     0,     0,     0, "IDLE"  }
 { CLOCK,clock_task, TSK_F,  0, TASK_Q, TSK_S, TSK_T,     0,     0, "CLOCK" }
 { SYSTEM, sys_task, TSK_F,  0, TASK_Q, TSK_S, TSK_T,     0,     0, "SYSTEM"}
 { HARDWARE,      0, TSK_F,  0, TASK_Q, HRD_S,     0,     0,     0, "KERNEL"}
 { PM_PROC_NR,    0, SRV_F, 32,      3, 0,     SRV_T, SRV_M,  PM_C, "pm"    }
 { FS_PROC_NR,    0, SRV_F, 32,      4, 0,     SRV_T, SRV_M,  FS_C, "fs"    }
 { RS_PROC_NR,    0, SRV_F,  4,      3, 0,     SRV_T, SYS_M,  RS_C, "rs"    }
 { DS_PROC_NR,    0, SRV_F,  4,      3, 0,     SRV_T, SYS_M,  DS_C, "ds"    }
 { TTY_PROC_NR,   0, SRV_F,  4,      1, 0,     SRV_T, SYS_M, TTY_C, "tty"   }
 { MEM_PROC_NR,   0, SRV_F,  4,      2, 0,     SRV_T, SYS_M, MEM_C, "memory"}
 { LOG_PROC_NR,   0, SRV_F,  4,      2, 0,     SRV_T, SYS_M, DRV_C, "log"   }
 { DRVR_PROC_NR,  0, SRV_F,  4,      2, 0,     SRV_T, SYS_M, DRV_C, "driver"}
 { INIT_PROC_NR,  0, USR_F,  8, USER_Q, 0,     USR_T, USR_M,     0, "init"  }
};
```

Inizializzazione: kernel/main.c

Invocato dal programma di boot dopo un preambolo in assembler

- inizializza la tabella dei processi
- stampa un banner
- fa partire lo scheduling

Inizializzazione: kernel/main.c I

```
PUBLIC void main()
{
  ...
  /* Initialize the interrupt controller. */
  intr_init(1);
```

Inizializzazione: kernel/main.c II

```
/* Clear the process table. Anounce each slot as empty and set up mappings
 * for proc_addr() and proc_nr() macros. Do the same for the table with
 * privilege structures for the system processes.
 */
for (rp = BEG_PROC_ADDR, i = -NR_TASKS; rp < END_PROC_ADDR; ++rp, ++i) {
      rp->p_rts_flags = SLOT_FREE;            /* initialize free slot */
      rp->p_nr = i;                           /* proc number from ptr */
      (pproc_addr + NR_TASKS)[i] = rp;        /* proc ptr from number */
}
for (sp = BEG_PRIV_ADDR, i = 0; sp < END_PRIV_ADDR; ++sp, ++i) {
      sp->s_proc_nr = NONE;                   /* initialize as free */
      sp->s_id = i;                           /* priv structure index */
      ppriv_addr[i] = sp;                     /* priv ptr from number */
}
```

Inizializzazione: kernel/main.c III

```
/* Set up proc table entries for processes in boot image.    */

for (i=0; i < NR_BOOT_PROCS; ++i) {
  struct boot_image *ip;        /* boot image pointer */
  register struct proc *rp;     /* process pointer */
  register struct priv *sp;     /* privilege structure pointer */

  ip = &image[i];                            /* process' attributes */
  rp = proc_addr(ip->proc_nr);               /* get process pointer */
  rp->p_max_priority = ip->priority;         /* max scheduling priority */
  rp->p_priority = ip->priority;             /* current priority */
  rp->p_quantum_size = ip->quantum;          /* quantum size in ticks */
  rp->p_ticks_left = ip->quantum;            /* current credit */
  strncpy(rp->p_name, ip->proc_name, P_NAME_LEN); /* set process name */
  (void) get_priv(rp, (ip->flags & SYS_PROC));    /* assign structure */
  priv(rp)->s_flags = ip->flags;                  /* process flags */
  priv(rp)->s_trap_mask = ip->trap_mask;          /* allowed traps */
  priv(rp)->s_call_mask = ip->call_mask;          /* kernel call mask */
  priv(rp)->s_ipc_to.chunk[0] = ip->ipc_to;       /* restrict targets */
```

Inizializzazione: kernel/main.c IV

```
  ...
  /* Set initial register values.  The processor status word for tasks
   * is different from that of other processes because tasks can
   * access I/O; this is not allowed to less-privileged processes
   */
  rp->p_reg.pc = (reg_t) ip->initial_pc;
  rp->p_reg.psw = (iskernelp(rp)) ? INIT_TASK_PSW : INIT_PSW;

  ...
  /* Set ready. The HARDWARE task is never ready. */
  if (rp->p_nr != HARDWARE) {
    rp->p_rts_flags = 0;            /* runnable if no flags */
    lock_enqueue(rp);               /* add to scheduling queues */
  } else {
    rp->p_rts_flags = NO_MAP;       /* prevent from running */
  }
  ...
}
```

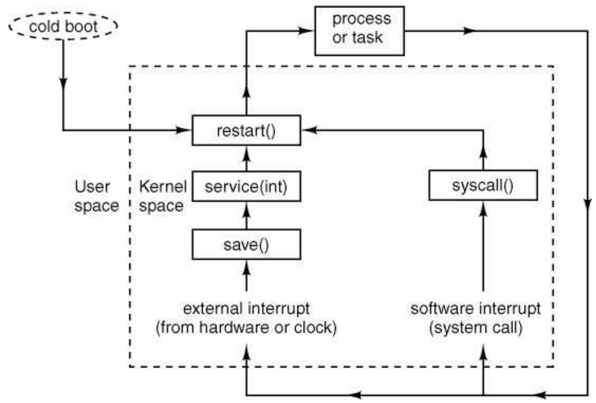
Inizializzazione: kernel/main.c V

```
  /* MINIX is now ready. All boot image processes are on the ready queue.
   * Return to the assembly code to start running the current process.
   */
  bill_ptr = proc_addr(IDLE);           /* it has to point somewhere */
  announce();                           /* print MINIX startup banner */
  restart();
}
```

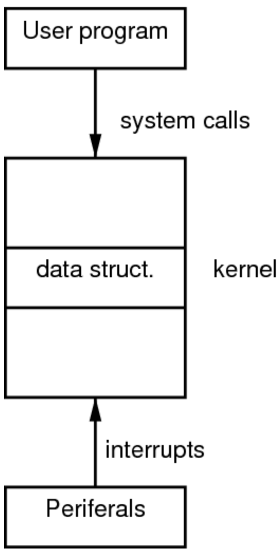
Inizializzazione: kernel/main.c VI

```
PRIVATE void announce(void)
{
  /* Display the MINIX startup banner. */
  kprintf("\nMINIX %s.%s. "
      "Copyright 2006, Vrije Universiteit, Amsterdam, The Netherlands\n",
      OS_RELEASE, OS_VERSION);
#if (CHIP == INTEL)
  /* Real mode, or 16/32-bit protected mode? */
  kprintf("Executing in %s mode.\n\n",
      machine.protected ? "32-bit protected" : "real");
#endif
}
```

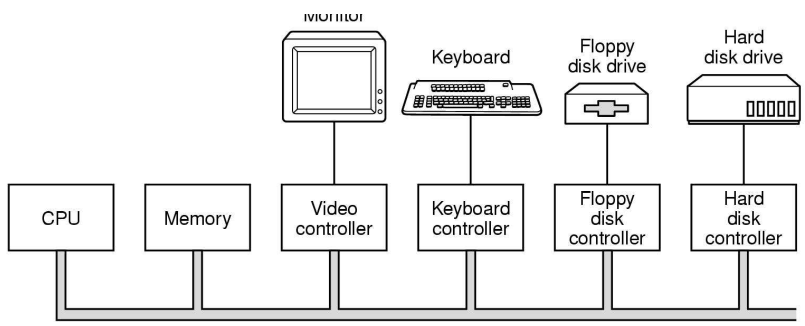
Restart



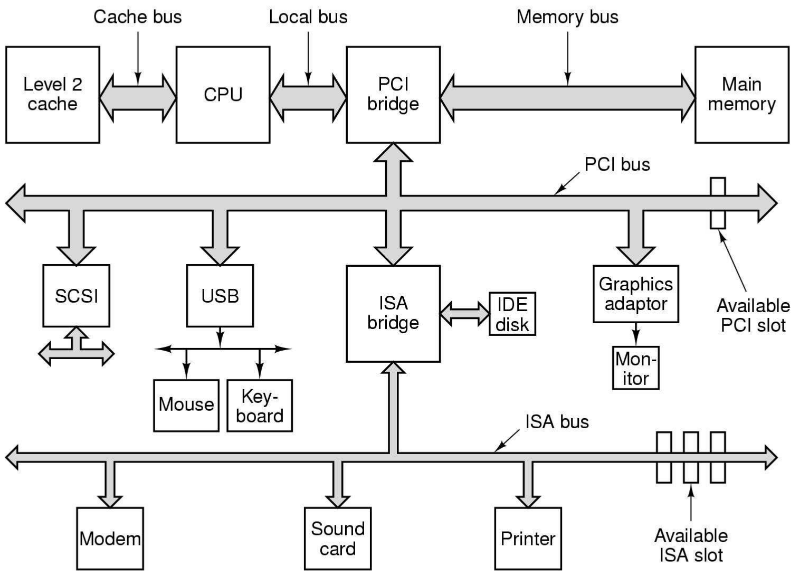
Struttura di un kernel



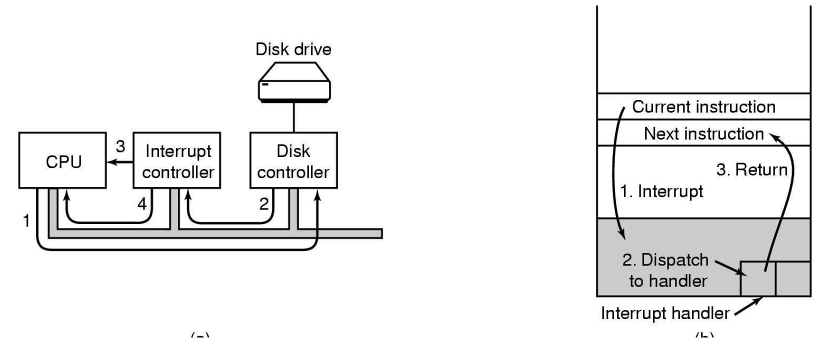
Architettura di un computer, semplificata



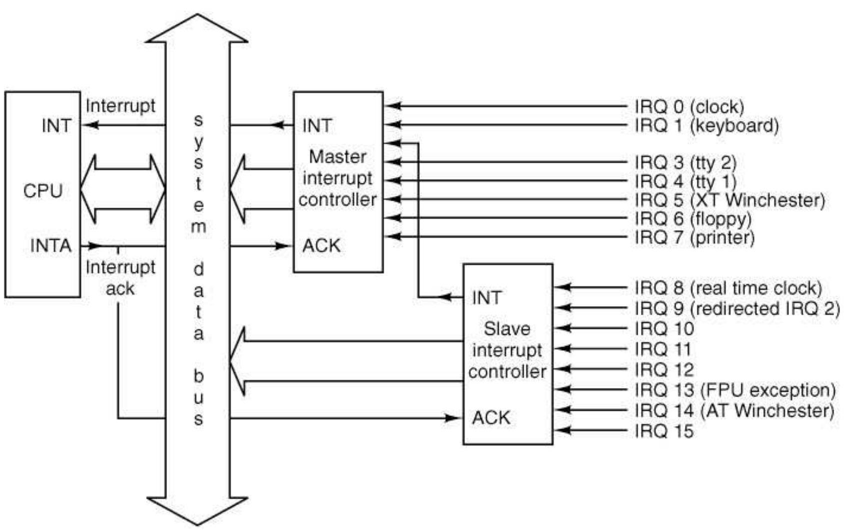
Architettura di un computer, dettagliata



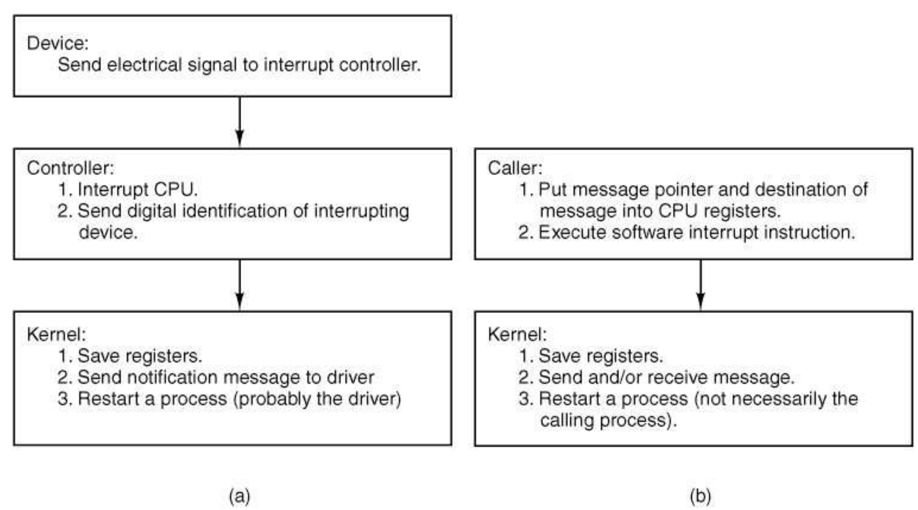
Gestione di interrupt



Gestione di interrupt



Confronto: hw interrupt e chiamata di sistema



Gestione interrupt

HW interrupt provoca

- disabilita interrupt
- nuovo stack dipendente dal Task State Segment (TSS)
  ⇒ inizio della struct proc per il proc corr
- push di alcuni registri su questo stack
- salto all'interrupt handler

L'interrupt handler

- usa lo stack del kernel
- ... gestisce interrupt
- punta lo stack a una struct proc
  (non necessariam ql del proc originale)
- esegue `iretd`

Interrupt handlers in kernel/mpx386.s I

```
_hwint00:               ! Interrupt routine for irq 0 (the clock).
        hwint_master(0)

        .align  16
_hwint01:               ! Interrupt routine for irq 1 (keyboard)
        hwint_master(1)
...
```

Interrupt handlers in kernel/mpx386.s II

```
#define hwint_master(irq)       \
      call    save                 /* save interrupted process state */;\
      push    (_irq_handlers+4*irq) /* irq_handlers[irq]              */;\
      call    _intr_handle         /* intr_handle(irq_handlers[irq]) */;\
      pop     ecx                                                      ;\
      cmp     (_irq_actids+4*irq), 0 /* interrupt still active?       */;\
      jz      0f                                                       ;\
      inb     INT_CTLMASK          /* get current mask */              ;\
      orb     al, [1<<irq]         /* mask irq */                      ;\
      outb    INT_CTLMASK          /* disable the irq                */;\
0:    movb    al, END_OF_INT                                           ;\
      outb    INT_CTL              /* reenable master 8259           */;\
      ret                          /* restart (another) process      */
```

Subroutine chiamate da hwint_master

save

- salva i registri
- passa al kernel stack
- spinge sullo stack l'indirizzo di _restart

Intr_handle

- Scandisce una lista di funzioni da chiamare in risposta a un interrupt

Restart in kernel/mpx386.s

```
_restart:
! Restart the current process or the next process if it is set.
        cmp     (_next_ptr), 0          ! see if another process is schedulec
        jz      0f
        mov     eax, (_next_ptr)
        mov     (_proc_ptr), eax        ! schedule new process
        mov     (_next_ptr), 0
0:      mov     esp, (_proc_ptr)        ! will assume P_STACKBASE == 0
        lldt    P_LDT_SEL(esp)          ! enable process' segment descriptors
        lea     eax, P_STACKTOP(esp)    ! arrange for next interrupt
        mov     (_tss+TSS3_S_SP0), eax  ! to save state in process table
restart1:
        decb    (_k_reenter)
    o16 pop     gs
    o16 pop     fs
    o16 pop     es
    o16 pop     ds
        popad
        add     esp, 4          ! skip return adr
        iretd                   ! continue process
```