

Interruzioni hardware



Gestione del temporizzatore	4554
Gestione della tastiera	4557
Verifica del funzionamento	4562

`isr_irq.c` [4553](#) `keyboard.c` [4557](#) `keyboard.h` [4557](#)
`keyboard_load.c` [4557](#) `timer.c` [4554](#) `timer.h` [4554](#)
`timer_freq.c` [4554](#)

Le interruzioni hardware che vengono gestite in questo sistema sono solo IRQ 0 (temporizzatore o *timer*) e IRQ 1 (tastiera). Il file ‘`isr_irq.c`’ che in precedenza è stato ridotto per sospendere il problema delle interruzioni hardware ha la forma finale del listato successivo.

Listato u173.1. ‘./05/lib/int/isr_irq.c’

```
#include <kernel/int.h>
#include <kernel/io.h>
#include <kernel/timer.h>
#include <kernel/keyboard.h>
void
isr_irq (uint32_t eax, uint32_t ecx, uint32_t edx, uint32_t ebx,
         uint32_t ebp, uint32_t esi, uint32_t edi, uint32_t ds,
         uint32_t es,  uint32_t fs,  uint32_t gs,  uint32_t interrupt)
{
    int irq = interrupt - 32;
    //
    //
    //
    switch (irq)
    {
        case 0: timer (); break;
        case 1: keyboard (); break;
    }
}
```

```

//
// Finito il compito della funzione che deve reagire all'interruzione
// IRQ, occorre informare i PIC (programmable interrupt controller).
//
// Se il numero IRQ è tra 8 e 15, manda un messaggio «EOI»
// (End of IRQ) al PIC 2.
//
if (irq >= 8)
{
    outb (0xA0, 0x20);
}
//
// Poi manda un messaggio «EOI» al PIC 1.
//
outb (0x20, 0x20);
}

```

Gestione del temporizzatore

«

La gestione del temporizzatore è raccolta dalla libreria che fa capo al file di intestazione 'timer.h' come appare nel listato successivo.

Listato u173.2. './05/include/kernel/timer.h'

```

#ifndef _TIMER_H
#define _TIMER_H        1

#include <time.h>
#include <kernel/os.h>

void timer      (void);
void timer_freq (clock_t freq);

#endif

```

Il temporizzatore genera impulsi a una frequenza data e a ogni impulso produce un'interruzione. Per regolare tale frequenza occorre

comunicare con le porte 43_{16} e 40_{16} , inviando il divisore da applicare alla frequenza di riferimento che è 1,193181 MHz. La funzione *timer_freq()* stabilisce la frequenza da generare, calcolando il divisore da applicare.¹

Listato u173.3. './05/lib/timer/timer_freq.c'

```
#include <kernel/timer.h>
#include <stdint.h>
#include <stdio.h>
void
timer_freq (clock_t freq)
{
    int input_freq = 1193180;
    //
    // La frequenza di riferimento è 1,19318 MHz, la quale va
    // divisa per la frequenza che si intende avere effettivamente.
    //
    int divisor = input_freq / freq;
    //
    // Il risultato deve essere un valore intero maggiore di zero
    // e inferiore di UINT16_MAX, altrimenti è stata chiesta una
    // frequenza troppo elevata o troppo bassa.
    //
    if (divisor == 0 || divisor > UINT16_MAX)
    {
        printf ("%s] ERROR: IRQ 0 frequency wrong: %i Hz!\n"
                "[%s]          The min allowed frequency is 18.22 Hz.\n",
                "[%s]          The max allowed frequency is 1.19 MHz.\n",
                __func__, freq, __func__, __func__);
        return;
    }
    //
    // Il valore che si ottiene, ovvero il «divisore», va
    // comunicato al PIT (programmable interval timer),
    // spezzandolo in due parti.
    //
    outb (0x43, 0x36);
    outb (0x40, divisor & 0x0F);          // Byte inferiore del numero.
    outb (0x40, divisor / 0x10);         // Byte superiore del numero.
    //
    // Annota la frequenza attuale degli impulsi provenienti dal
    // PIT (programmable interval timer).
```

```
//  
os.timer.freq = freq;  
}
```

La funzione *timer()* è quella che viene eseguita automaticamente, ogni volta che si presenta un'interruzione che deriva da un IRQ 0. Di norma lo scopo di una funzione di questo tipo è controllare la gestione corretta dei processi, ma in mancanza di questi, si potrebbero avviare delle funzioni che assicurano un'esecuzione brevissima, salvo il verificarsi di eventi specifici. Nel listato successivo si presenta una funzione *timer()* praticamente vuota e i file di intestazione incorporati sono ipotetici.

Listato u173.4. './05/lib/timer/timer.c'

```
#include <kernel/timer.h>  
#include <kernel/int.h>  
#include <time.h>  
void  
timer (void)  
{  
    //  
    // Conta le interruzioni.  
    //  
    os.timer.clocks++;  
    //  
    // Dovrebbe lanciare lo «scheduler», ma qui non c'è;  
    // pertanto, lancia direttamente delle applicazioni molto  
    // brevi (devono garantire di terminare rapidamente).  
    //  
    ;  
    ;  
    ;  
}
```

L'incremento della variabile *os.timer.clocks* consentirebbe di compiere delle azioni quando risulta trascorso un certo intervallo di

tempo. Un'ipotesi di utilizzo potrebbe essere quella seguente, dove, ammesso che la frequenza del temporizzatore sia pari a 'CLOCKS_PER_SEC', al trascorrere di ogni secondo fa qualcosa:

```
void
timer (void)
{
    os.timer.clocks++;
    if ((os.timer.clocks % CLOCKS_PER_SEC) == 0)
    {
        fa_qualcosa
    }
}
```

Gestione della tastiera

La gestione della tastiera è raccolta dalla libreria che fa capo al file di intestazione 'keyboard.h' come appare nel listato successivo. <<

Listato u173.6. './05/include/kernel/keyboard.h'

```
#ifndef _KEYBOARD_H
#define _KEYBOARD_H    1

#include <kernel/os.h>

void keyboard (void);
void keyboard_load (void);

#endif
```

La funzione *keyboard_load()* definisce una mappa della tastiera, memorizzata negli array *os.kbd.map1[]* e *os.kbd.map2[]*. Le due mappe riguardano i due livelli di scrittura: quello normale e quello che solitamente produce principalmente le maiuscole. L'indice degli array corrisponde al codice grezzo generato dalla tastiera (*scan-code*). Il listato successivo riguarda una funzione *keyboard_load()* adatta alla disposizione italiana dei simboli, tenendo conto però che non si possono generare lettere accentate.

Listato u173.7. './05/lib/keyboard/keyboard_load.c'

```
#include <kernel/keyboard.h>
void
keyboard_load (void)
{
    int i;
    for (i = 0; i <= 127; i++)
        {
            os.kbd.map1[i] = 0;
            os.kbd.map2[i] = 0;
        }
    //
    //
    //
    os.kbd.map1[1]   = 27;      os.kbd.map2[1]   = 27;
    os.kbd.map1[2]   = '1';    os.kbd.map2[2]   = '!';
    os.kbd.map1[3]   = '2';    os.kbd.map2[3]   = '"';
    os.kbd.map1[4]   = '3';    os.kbd.map2[4]   = 'L';      // 3, £
    os.kbd.map1[5]   = '4';    os.kbd.map2[5]   = '$';
    os.kbd.map1[6]   = '5';    os.kbd.map2[6]   = '%';
    os.kbd.map1[7]   = '6';    os.kbd.map2[7]   = '&';
    os.kbd.map1[8]   = '7';    os.kbd.map2[8]   = '/';
    os.kbd.map1[9]   = '8';    os.kbd.map2[9]   = '(';
    os.kbd.map1[10]  = '9';    os.kbd.map2[10]  = ')';
    os.kbd.map1[11]  = '0';    os.kbd.map2[11]  = '=';
    os.kbd.map1[12]  = '\';    os.kbd.map2[12]  = '?';
    os.kbd.map1[13]  = 'i';    os.kbd.map2[13]  = '^';      // ì, ^
    os.kbd.map1[14]  = '\b';   os.kbd.map2[14]  = '\b';    // Backspace
    os.kbd.map1[15]  = '\t';   os.kbd.map2[15]  = '\t';
    os.kbd.map1[16]  = 'q';    os.kbd.map2[16]  = 'Q';
    os.kbd.map1[17]  = 'w';    os.kbd.map2[17]  = 'W';
    os.kbd.map1[18]  = 'e';    os.kbd.map2[18]  = 'E';
    os.kbd.map1[19]  = 'r';    os.kbd.map2[19]  = 'R';
    os.kbd.map1[20]  = 't';    os.kbd.map2[20]  = 'T';
    os.kbd.map1[21]  = 'y';    os.kbd.map2[21]  = 'Y';
    os.kbd.map1[22]  = 'u';    os.kbd.map2[22]  = 'U';
    os.kbd.map1[23]  = 'i';    os.kbd.map2[23]  = 'I';
    os.kbd.map1[24]  = 'o';    os.kbd.map2[24]  = 'O';
    os.kbd.map1[25]  = 'p';    os.kbd.map2[25]  = 'P';
    os.kbd.map1[26]  = '[';    os.kbd.map2[26]  = '{';      // è, é
    os.kbd.map1[27]  = ']';    os.kbd.map2[27]  = '}';      // +, *
    os.kbd.map1[28]  = '\n';   os.kbd.map2[28]  = '\n';    // Invio
    os.kbd.map1[30]  = 'a';    os.kbd.map2[30]  = 'A';
    os.kbd.map1[31]  = 's';    os.kbd.map2[31]  = 'S';
```

```

os.kbd.map1[32] = 'd';      os.kbd.map2[32] = 'D';
os.kbd.map1[33] = 'f';      os.kbd.map2[33] = 'F';
os.kbd.map1[34] = 'g';      os.kbd.map2[34] = 'G';
os.kbd.map1[35] = 'h';      os.kbd.map2[35] = 'H';
os.kbd.map1[36] = 'j';      os.kbd.map2[36] = 'J';
os.kbd.map1[37] = 'k';      os.kbd.map2[37] = 'K';
os.kbd.map1[38] = 'l';      os.kbd.map2[38] = 'L';
os.kbd.map1[39] = '@';      os.kbd.map2[39] = '@';      // ò, ç
os.kbd.map1[40] = '#';      os.kbd.map2[40] = '#';      // à, °
os.kbd.map1[41] = '\\';     os.kbd.map2[41] = '|';
os.kbd.map1[43] = 'u';      os.kbd.map2[43] = 'U';      // ù, §
os.kbd.map1[44] = 'z';      os.kbd.map2[44] = 'Z';
os.kbd.map1[45] = 'x';      os.kbd.map2[45] = 'X';
os.kbd.map1[46] = 'c';      os.kbd.map2[46] = 'C';
os.kbd.map1[47] = 'v';      os.kbd.map2[47] = 'V';
os.kbd.map1[48] = 'b';      os.kbd.map2[48] = 'B';
os.kbd.map1[49] = 'n';      os.kbd.map2[49] = 'N';
os.kbd.map1[50] = 'm';      os.kbd.map2[50] = 'M';
os.kbd.map1[51] = ',';      os.kbd.map2[51] = ';';
os.kbd.map1[52] = '.';      os.kbd.map2[52] = ':';
os.kbd.map1[53] = '-';      os.kbd.map2[53] = '_';
os.kbd.map1[56] = '<';      os.kbd.map2[56] = '>';
os.kbd.map1[57] = ' ';      os.kbd.map2[57] = ' ';
//
os.kbd.map1[55] = '*';      os.kbd.map2[55] = '*';
os.kbd.map1[71] = '7';      os.kbd.map2[71] = '7';
os.kbd.map1[72] = '8';      os.kbd.map2[72] = '8';
os.kbd.map1[73] = '9';      os.kbd.map2[73] = '9';
os.kbd.map1[74] = '-';      os.kbd.map2[74] = '-';
os.kbd.map1[75] = '4';      os.kbd.map2[75] = '4';
os.kbd.map1[76] = '5';      os.kbd.map2[76] = '5';
os.kbd.map1[77] = '6';      os.kbd.map2[77] = '6';
os.kbd.map1[78] = '+';      os.kbd.map2[78] = '+';
os.kbd.map1[79] = '1';      os.kbd.map2[79] = '1';
os.kbd.map1[80] = '2';      os.kbd.map2[80] = '2';
os.kbd.map1[81] = '3';      os.kbd.map2[81] = '3';
os.kbd.map1[82] = '0';      os.kbd.map2[82] = '0';
os.kbd.map1[83] = '.';      os.kbd.map2[83] = '.';
os.kbd.map1[92] = '/';      os.kbd.map2[92] = '/';
os.kbd.map1[96] = '\\n';    os.kbd.map2[96] = '\\n';    // Invio
}

```

La funzione *keyboard()*, avviata ogni volta che si preme un tasto o lo

si rilascia (attraverso l'impulso dato da IRQ 2), interpreta il codice grezzo proveniente dalla tastiera e aggiorna la variabile strutturata *os.kbd*. Per esempio tiene traccia della pressione dei tasti [*Ctrl*], [*Alt*] e della selezione delle maiuscole. Quando si tratta di un tasto che deve produrre un carattere, questo viene annotato nella variabile *os.kbd.key*, ma solo se questa è vuota. In pratica ci deve essere un programma che «consuma» questa informazione, azzerando di conseguenza la variabile. Si osservi che la variabile *os.kbd.echo*, se contiene un valore diverso da zero, indica la richiesta di visualizzare sullo schermo ciò che si preme ed è controllata dalla macroistruzione *echo()* (da 'vga.h').

Listato u173.8. './05/lib/keyboard/keyboard.c'

```
#include <kernel/keyboard.h>
#include <kernel/io.h>
#include <stdio.h>
void
keyboard (void)
{
    unsigned char scancode = inb (0x60);

    //
    // Shift, Shift-Lock, Ctrl, Alt
    //
    switch (scancode)
    {
        case 0x2A: os.kbd.shift = 1; break;
        case 0x36: os.kbd.shift = 1; break;
        case 0xAA: os.kbd.shift = 0; break;
        case 0xB6: os.kbd.shift = 0; break;
        case 0x1D: os.kbd.ctrl = 1; break;
        case 0x9D: os.kbd.ctrl = 0; break;
        case 0x38: os.kbd.alt = 1; break;
        case 0xB8: os.kbd.alt = 0; break;
        case 0x3A: os.kbd.shift_lock = ! os.kbd.shift_lock; break;
    }
    //
    // Ctrl+
```



```

//
if (scancode <= 127 && os.kbd.ctrl && os.kbd.key == 0)
{
    switch (os.kbd.map1[scancode])
    {
        case 'a': os.kbd.key = 0x01; break; // SOH
        case 'b': os.kbd.key = 0x02; break; // STX
        case 'c': os.kbd.key = 0x03; break; // ETX
        case 'd': os.kbd.key = 0x04; break; // EOT
        case 'e': os.kbd.key = 0x05; break; // ENQ
        case 'f': os.kbd.key = 0x06; break; // ACK
        case 'g': os.kbd.key = 0x07; break; // BEL
        case 'h': os.kbd.key = 0x08; break; // BS
        case 'i': os.kbd.key = 0x09; break; // HT
        case 'j': os.kbd.key = 0x0A; break; // LF
        case 'k': os.kbd.key = 0x0B; break; // VT
        case 'l': os.kbd.key = 0x0C; break; // FF
        case 'm': os.kbd.key = 0x0D; break; // CR
        case 'n': os.kbd.key = 0x0E; break; // SO
        case 'o': os.kbd.key = 0x0F; break; // SI
        case 'p': os.kbd.key = 0x10; break; // DLE
        case 'q': os.kbd.key = 0x11; break; // DC1
        case 'r': os.kbd.key = 0x12; break; // DC2
        case 's': os.kbd.key = 0x13; break; // DC3
        case 't': os.kbd.key = 0x14; break; // DC4
        case 'u': os.kbd.key = 0x15; break; // NAK
        case 'v': os.kbd.key = 0x16; break; // SYN
        case 'w': os.kbd.key = 0x17; break; // ETB
        case 'x': os.kbd.key = 0x18; break; // CAN
        case 'y': os.kbd.key = 0x19; break; // EM
        case 'z': os.kbd.key = 0x1A; break; // SUB
        case '[': os.kbd.key = 0x1B; break; // ESC
        case '\\': os.kbd.key = 0x1C; break; // FS
        case ']': os.kbd.key = 0x1D; break; // GS
        case '^': os.kbd.key = 0x1E; break; // RS
        case '_': os.kbd.key = 0x1F; break; // US
    }
//
if (os.kbd.echo && os.kbd.key)
{
    (void) putchar (os.kbd.key);
}

```

```

    }
else if (scancode <= 127 && os.kbd.key == 0 && os.kbd.map1[scancode] != 0)
    {
        if (os.kbd.shift || os.kbd.shift_lock)
            {
                os.kbd.key = os.kbd.map2[scancode];
            }
        else
            {
                os.kbd.key = os.kbd.map1[scancode];
            }
        //
        if (os.kbd.echo && os.kbd.key)
            {
                (void) putchar (os.kbd.key);
            }
    }
}

```

Verifica del funzionamento

«

Per verificare il funzionamento delle chiamate di sistema, si può modificare il file ‘kernel_main.c’ nel modo seguente. Ciò che si ottiene è di poter visualizzare sullo schermo il primo tasto che si preme (ciò avviene subito dopo la dichiarazione che il sistema è arrestato), in quanto non si possono inserire altri caratteri fino a quando «qualcuno» non svuota *os.kbd.key*.

Figura u173.9. Modifiche da apportare al file ‘./05/kernel/kernel_main.c’

```

#include <kernel/kernel.h>
#include <kernel/build.h>
#include <stdio.h>
#include <kernel/gdt.h>
#include <kernel/mm.h>
#include <stdlib.h>
#include <kernel/int.h>

```

```

#include <sys/syscall.h>
#include <kernel/timer.h>
#include <kernel/keyboard.h>
...
    //
    // Predispone la memoria libera per l'utilizzo.
    //
mm_init ();
    //
    // Predispone il timer.
    //
timer_freq (CLOCKS_PER_SEC);
    //
    // Predispone la tastiera.
    //
keyboard_load ();
echo ();
    //
    // Predispone la tabella IDT.
    //
idt ();
...

```

Dopo avere ricompilato, riavviando la simulazione si deve ottenere una schermata simile a quella seguente, dove dopo l'arresto dichiarato del sistema si può premere un tasto che viene visualizzato:

