# Librerie specifiche generali

«

File «build.h» ..... 4463
Libreria «io.h» ..... 4463
Libreria «multiboot.h» ..... 4465
File «os.h» ..... 4468
Libreria «vga.h» ..... 4473

build.h 4463 inb.s 4463 io.h 4463 mboot_info.c 4465 mboot_show.c 4465 multiboot.h 4465 os.h 4468 outb.s 4463 vga.h 4473 vga_clear.c 4473 vga_new_line.c 4473 vga_printf.c 4473 vga_putc.c 4473 vga_puts.c 4473 vga_set.c 4473 vga_vprintf.c 4473

Dopo le librerie standard vanno predisposte anche altre librerie specifiche per il proprio sistema. Quelle descritte nelle sezioni successive sono quelle di uso generale.

## File «build.h»

Il file ‘05/include/kernel/build.h’ viene prodotto dallo script ‘05/makeit’, allo scopo di generare la macro-variabile ***BUILD_DATE*** contenente il momento esatto della compilazione. Durante gli esperimenti per la realizzazione del sistema è importante rendersi conto se ciò che si sta osservando corrisponde effettivamente al risultato dell’ultima compilazione oppure no. Il contenuto del file ha un aspetto simile a quello seguente:

```
#define BUILD_DATE "20070817191030"
```

«

# Libreria «io.h»

«

La libreria rappresentata dal file di intestazione ‘`io.h`’ contiene la dichiarazione di funzioni necessarie alla comunicazione con le componenti hardware. In questo caso si utilizzano solo funzioni per riprodurre le istruzioni ‘**`INB`**’ e ‘**`OUTB`**’ del linguaggio assemblatore, ma potrebbe essere estesa anche con altre funzioni per istruzioni analoghe, per la comunicazione con dati di dimensione maggiore del byte.

Listato u167.2. ‘`./05/include/kernel/io.h`’

```
#ifndef _IO_H
#define _IO_H   1

void         outb (unsigned int port, unsigned int data);
unsigned int inb  (unsigned int port);

#endif
```

Naturalmente è necessario realizzare entrambe le funzioni. È il caso di ricordare che il valore restituito dalle funzioni scritte in linguaggio assemblatore è quello contenuto nel registro ***EAX***.

Listato u167.3. ‘`./05/lib/io/inb.s`’

```
.globl  inb
#
inb:
    enter $4, $0
    pusha
    .equ inb_port,  8           # Primo parametro.
    .equ inb_data, -4           # Variabile locale.
    mov  inb_port(%ebp), %edx   # Successivamente si usa
                                # solo DX.
```

```
    inb %dx, %al
    mov  %eax, inb_data(%ebp)    # Salva EAX nella variabile
                                 # locale.
    popa
    mov  inb_data(%ebp), %eax    # Recupera EAX e termina.
    leave
    ret
```

Listato u167.4. ‘./05/lib/io/outb.s’

```
.globl  outb
#
outb:
    enter $0, $0
    pusha
    .equ outb_port,  8           # Primo parametro.
    .equ outb_data, 12           # Secondo parametro.
    mov  outb_port(%ebp), %edx   # Successivamente si usa
                                 # solo DX.
    mov  outb_data(%ebp), %eax   # Successivamente si usa
                                 # solo AL.
    outb %al, %dx
    popa
    leave
    ret
```

## Libreria «multiboot.h»

«

La libreria rappresentata dal file di intestazione ‘multiboot.h’ contiene semplicemente una struttura per facilitare la lettura delle informazioni più importanti che offre un sistema di avvio aderente alle specifiche *multiboot*; inoltre dichiara due funzioni: una per la raccolta delle informazioni e l’altra per la loro visualizzazione.

Listato u167.5. ‘./05/include/kernel/multiboot.h’

```
#ifndef _MULTIBOOT_H
#define _MULTIBOOT_H    1

#include <inttypes.h>

typedef struct {
    uint32_t  flags;
    uint32_t  mem_lower;
    uint32_t  mem_upper;
    uint32_t  boot_device;
        char *cmdline;
} multiboot_t;

void mboot_info (multiboot_t *info);
void mboot_show (void);

#endif
```

La funzione ***mboot_info()*** deve raccogliere e salvare le informazioni *multiboot*, all’interno della variabile strutturata ***os.multiboot*** (la variabile ***os*** complessiva è descritta nel file ‘os.h’).

Listato u167.6. ‘./05/lib/multiboot/mboot_info.c’

```
#include <kernel/multiboot.h>
#include <string.h>
#include <stdio.h>
void
mboot_info (multiboot_t *info)
{
    os.multiboot.flags = info->flags;
    //
    if ((info->flags & 1) > 0)
      {
        os.multiboot.mem_lower = info->mem_lower;
        os.multiboot.mem_upper = info->mem_upper;
```

```
      }
    if ((info->flags & 2) > 0)
      {
        os.multiboot.boot_device = info->boot_device;
      }
    if ((info->flags & 4) > 0)
      {
        strncpy (os.multiboot.cmdline, info->cmdline, 1024);
      }
}
```

La funzione ***mboot_show()*** deve visualizzare direttamente le informazioni *multiboot*, salvate in precedenza, pertanto si avvale della funzione ***printf()*** che deve essere ancora descritta.

Listato u167.7. ‘`./05/lib/multiboot/mboot_show.c`’

```
#include <kernel/multiboot.h>
#include <stdio.h>
void
mboot_show (void)
{
    printf ("[%s] flags: %032b ", __func__,
            os.multiboot.flags);
    //
    if ((os.multiboot.flags & 1) > 0)
      {
        printf ("mlow: %04X mhigh: %08X",
                os.multiboot.mem_lower,
                os.multiboot.mem_upper);
      }
    printf ("\n");
    printf ("[%s] ", __func__);
    if ((os.multiboot.flags & 2) > 0)
      {
        printf ("bootdev: %08X ", os.multiboot.boot_device);
      }
```

```
    if ((os.multiboot.flags & 4) > 0)
      {
        printf ("cmdline: \"%s\"", os.multiboot.cmdline);
      }
    printf ("\n");
}
```

## File «os.h»

«

Il file di intestazione ‘`os.h`’ serve esclusivamente per definire una struttura, con la quale si crea la variabile strutturata ***os***, accessibile a ogni parte del sistema. In questa superstruttura vengono annotate tutte le informazioni che devono essere condivise. Il senso delle varie componenti della variabile ***os*** si chiarisce successivamente; a ogni modo è importante osservare che nel sistema non vengono usate altre variabili pubbliche.

Listato u167.8. ‘`./05/include/kernel/os.h`’

```
#ifndef _OS_H
#define _OS_H   1

#include <stdint.h>
#include <kernel/multiboot.h>
#include <stdbool.h>
#include <time.h>

typedef struct {
    //
    // Multiboot.
    //
    struct {
        uint32_t  flags;
        uint32_t  mem_lower;
```

```
        uint32_t  mem_upper;
        uint32_t  boot_device;
        char      cmdline[1024];
    } multiboot;
    //
    // Stato dello schermo VGA.
    //
    struct {
        unsigned short *video;
        unsigned short  columns;
        unsigned short  rows;
        unsigned int    position;
        unsigned char   attribute;
    } vga;
    //
    // «os.mem_ph»    Mappa della memoria fisica.
    //
    struct {
        uintptr_t  total_s;     // «..._s» = start
        uintptr_t  total_e;     // «..._e» = end.
           size_t  total_l;     // «..._l» = limit.
        uintptr_t  k_text_s;    // «k_...» = kernel.
        uintptr_t  k_text_e;    //
        uintptr_t  k_rodata_s;  //
        uintptr_t  k_rodata_e;  //
        uintptr_t  k_data_s;    //
        uintptr_t  k_data_e;    //
        uintptr_t  k_bss_s;     //
        uintptr_t  k_bss_e;     //
        uintptr_t  available_s; //
        uintptr_t  available_e; //
    } mem_ph;
    //
    // «os.gtd»              Tabella GTD.
```

```
    //
    union {
        struct {
            uint32_t  limit_a              : 16,
                      base_a               : 16;
            uint32_t  base_b               : 8,
                      accessed             : 1,
                      write_execute        : 1,
                      expansion_conforming : 1,
                      code_or_data         : 1,
                      code_data_or_system  : 1,
                      dpl                  : 2,
                      present              : 1,
                      limit_b              : 4,
                      available            : 1,
                      reserved             : 1,
                      big                  : 1,
                      granularity          : 1,
                      base_c               : 8;
        } cd;
        struct {
            uint32_t  limit_a              : 16,
                      base_a               : 16;
            uint32_t  base_b               : 8,
                      type                 : 4,
                      code_data_or_system  : 1,
                      dpl                  : 2,
                      present              : 1,
                      limit_b              : 4,
                      reserved             : 3,
                      granularity          : 1,
                      base_c               : 8;
        } system;
    } gdt[3];
```

```
    //
    // «os.gtdr»      Registro GTDR.
    //
    // È necessario che la struttura sia compattata, in modo
    // da usare complessivamente 48 bit; pertanto si usa
    // l'attributo «packed» del compilatore GNU C.
    //
    struct {
        uint16_t  limit;
        uint32_t  base;
    } __attribute__ ((packed)) gdtr;
    //
    // «os.idt»               Tabella IDT.
    //
    struct {
        uint32_t  offset_a : 16,
                  selector : 16;
        uint32_t  filler   : 8,
                  type     : 4,
                  system   : 1,
                  dpl      : 2,
                  present  : 1,
                  offset_b : 16;
    } idt[129];
    //
    // «os.idtr»      Registro IDTR.
    //
    // È necessario che la struttura sia compattata, in modo
    // da usare complessivamente 48 bit; pertanto si usa
    // l'attributo «packed» del compilatore GNU C.
    //
    struct {
        uint16_t  limit;
        uint32_t  base;
```

```
    } __attribute__ ((packed)) idtr;
    //
    // PIT: programmable interval timer.
    //
    struct {
        clock_t freq;
        clock_t clocks;
    } timer;
    //
    // Stato della tastiera.
    //
    struct {
        bool shift;
        bool shift_lock;
        bool ctrl;
        bool alt;
        bool echo;
        char key;
        char map1[128];
        char map2[128];
    } kbd;
    //
} os_t;
//
// Struttura pubblica con tutte le informazioni sul sistema.
//
os_t os;

#endif
```

# Libreria «vga.h»

La libreria che fa capo al file di intestazione ‘`vga.h`’ è responsabile della visualizzazione del testo attraverso lo schermo.

Listato u167.9. ‘`./05/include/kernel/vga.h`’

```
#ifndef _VGA_H
#define _VGA_H 1

#include <restrict.h>
#include <kernel/io.h>
#include <kernel/os.h>
#include <stddef.h>
#include <stdarg.h>
#include <stdint.h>
#include <stdio.h>

#define vga_char(c, attrib) \
    ((int16_t) c | ((((int16_t) attrib) << 8) & 0xFF00))

void vga_init     (void);
void vga_set      (unsigned short *video, int columns,
                   int rows, int x, int y, int position,
                   int attribute);
int  vga_clear    (void);
void vga_new_line (void);
void vga_putc     (int c);
void vga_puts     (char *string, size_t n);
int  vga_vprintf  (const char *restrict format,
                   va_list arg);
int  vga_printf   (const char *restrict format, ...);

#define clear()   (vga_clear ())
#define echo()    (os.kbd.echo = 1)
#define noecho()  (os.kbd.echo = 0)
```

```
#endif
```

Alcune macroistruzioni definite nel file ‘vga.h’ si limitano a scrivere un valore all’interno di ***os.kbd.echo***, la quale, se attiva, rappresenta la richiesta di visualizzare sullo schermo il testo che viene digitato. La macroistruzione ***vga_char()*** assembla due valori in modo da ottenere un valore a 16 bit adatto alla visualizzazione sullo schermo di un carattere (l’unione dell’attributo di visualizzazione e del carattere stesso).

La funzione ‘vga_init()’ va usata prima di fare qualunque cosa con lo schermo VGA, per attribuire dei valori iniziali corretti alla struttura ***os.vga***, la quale serve a tenere memoria della posizione corrente del cursore di scrittura e dell’attributo corrente da usare per i colori dei caratteri da scrivere.

Listato u167.10. ‘./05/lib/vga/vga_init.c’

```
#include <kernel/vga.h>
void
vga_init (void)
{
    os.vga.video     = (unsigned short *) 0xB8000;
    os.vga.columns   = 80;
    os.vga.rows      = 25;
    os.vga.position  = 0;
    os.vga.attribute = 0x07;
}
```

La funzione ‘vga_set()’ che appare nel listato successivo ha lo scopo di spostare e di tenere traccia della posizione corrente del cursore di scrittura, in base alle informazioni che gli vengono fornite, determinando il resto in modo predefinito. Va osservato che, quando

si tratta di valori interi, per dire alla funzione ***vga_set()*** di utilizzare i dati predefiniti si trasmette un valore negativo.

Listato u167.11. ‘`./05/lib/vga/vga_set.c`’

```
#include <kernel/vga.h>
void
vga_set (unsigned short *video,
         int columns, int rows,
         int x, int y,
         int position,
         int attribute)
{
    unsigned short int current_y = os.vga.position / os.vga.columns;
    unsigned short int current_x = os.vga.position - current_y * os.vga.columns;
    unsigned int screen_size = os.vga.columns * os.vga.rows;
    char position_high;
    char position_low;
    //
    if (video != NULL)  os.vga.video   = video;
    if (columns >= 0)   os.vga.columns = columns;
    if (rows >= 0)      os.vga.rows    = rows;
    if (columns >= 0 || rows >= 0)
                        screen_size    = os.vga.columns * os.vga.rows;
    if (x >= 0)         current_x      = x;
    if (y >= 0)         current_y      = y;
    if (x >= 0 || y >= 0)
      {
        os.vga.position   = current_y * os.vga.columns + current_x;
        os.vga.position   = os.vga.position % screen_size;
      }
    if (position >= 0)
      {
        //
        // Ricalcola la posizione anche se è già stata determinata
        // con i parametri "x" and "y".
        //
        os.vga.position   = position % screen_size;
      }
    if (x >= 0 || y >= 0 || position >= 0)
      {
        //
        // Deve riposizionare il cursore.
        //
```

```
        position_high = (unsigned char) (os.vga.position >> 8);
        position_low  = (unsigned char) os.vga.position;
        //
        outb (0x3D4, 0x0E);
        outb (0x3D5, position_high);
        outb (0x3D4, 0x0F);
        outb (0x3D5, position_low);
      }
    if (attribute >= 0) os.vga.attribute = attribute;
}
```

Listato u167.12. ‘./05/lib/vga/vga_clear.c’

```
#include <kernel/vga.h>
int
vga_clear (void)
{
    unsigned short blank = vga_char (' ', os.vga.attribute);
    unsigned int   i;
    unsigned int   screen_size = os.vga.columns * os.vga.rows;

    for (i = 0; i < screen_size ; i++)
      {
        *(os.vga.video + i) = blank;
      }
    return 0;   // Per essere compatibile, in qualche modo, con «clear()».
}
```

Listato u167.13. ‘./05/lib/vga/vga_new_line.c’

```
#include <kernel/vga.h>
void
vga_new_line (void)
{
    unsigned short int current_y = os.vga.position / os.vga.columns;
    unsigned short int current_x = os.vga.position - current_y * os.vga.columns;
    unsigned short blank       = vga_char (' ', os.vga.attribute);
    unsigned int   screen_size = os.vga.columns * os.vga.rows;
    int i;
    int j;

    current_x = 0;
    current_y++;
```

```
    if (current_y >= os.vga.rows)
      {
        //
        // Copia il testo in su di una riga.
        //
        for (i = 0, j = os.vga.columns; j < screen_size; i++, j++)
          {
            *(os.vga.video + i) = *(os.vga.video + j);
          }
        //
        // Ripulisce l'ultima riga di testo.
        //
        for (i = screen_size - os.vga.columns; i < screen_size; i++)
          {
            *(os.vga.video + i) = blank;
          }
        current_y--;
      }
    vga_set (NULL, -1, -1, current_x, current_y, -1, -1);
}
```

Listato u167.14. `./05/lib/vga/vga_putc.c`

```
#include <kernel/vga.h>
void
vga_putc (int c)
{
    unsigned short int current_y = os.vga.position / os.vga.columns;
    unsigned short int current_x = os.vga.position - current_y * os.vga.columns;
    unsigned short int cell;

    if (c == '\n' || c == '\r')
      {
        vga_new_line ();
      }
    else
      {
        cell = vga_char (c, os.vga.attribute);

        *(os.vga.video + os.vga.position) = cell;

        if (current_x == os.vga.columns)
          {
            vga_new_line ();
```

```
          }
        else
          {
            vga_set (NULL, -1, -1, -1, -1, os.vga.position + 1, -1);
          }
      }
}
```

Listato u167.15. ‘./05/lib/vga/vga_puts.c’

```
#include <kernel/vga.h>
void
vga_puts (char *string, size_t n)
{
    size_t i;
    for (i = 0; i < n ; i++)
      {
        if (string[i] == 0) break;
        if (string[i] != 0) vga_putc (string[i]);
      }
    // Non aggiunge "\n"!
}
```

Listato u167.16. ‘./05/lib/vga/vga_vprintf.c’

```
#include <kernel/vga.h>
int
vga_vprintf (const char *restrict format, va_list arg)
{
    const size_t dim = 2000;  // Dimensione massima dello schermo: 25x80.
    char  string[dim];
    int   ret;
    string[0] = 0;
    ret = vsprintf(string, format, arg);
    vga_puts (string, dim);
    return ret;
}
```

Listato u167.17. ‘./05/lib/vga/vga_printf.c’

```
#include <kernel/vga.h>
int
vga_printf  (const char *restrict format, ...)
{
    va_list ap;
    va_start (ap, format);
    return vga_vprintf (format, ap);
}
```