

## Tabella GDT

Struttura .....	1943	
Libreria «gdt.h» .....	1943	
Modifiche da apportare a «kernel_main.c» .....	1946	
gdt.c 1943	gdt.h 1943	gdt_desc_seg.c 1943
gdt_load.s 1943	gdt_print.c 1943	

La preparazione di una tabella GDT è indispensabile per poter accedere alla memoria in modalità protetta. Nel sistema in corso di realizzazione si intende usare la memoria in modo lineare, senza segmentazioni e senza pagine, pertanto si compila la tabella GDT con il minimo indispensabile, avendo cura di indicare in modo preciso la memoria esistente effettivamente.

### Struttura

Nel file di intestazione 'os.h' è già stata predisposta la struttura che facilita la compilazione e l'interpretazione dei descrittori della tabella GDT. In particolare viene usata un'unione, con due suddivisioni alternative: una per i descrittori di segmento codice o dati e l'altra per i descrittori di sistema. Per il lavoro in corso di realizzazione, i descrittori di sistema non vengono utilizzati, pertanto è sufficiente concentrarsi sulla struttura 'os.gdt[n].cd':

```
...
union {
    struct {
        uint32_t limit_a      : 16,
        uint32_t 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 {
        ...
        ...
    } system;
} gdt[3];
...
```

### Libreria «gdt.h»

Il file di intestazione 'gdt.h' contiene la dichiarazione delle funzioni che riguardano la gestione della tabella GDT.

Listato u169.2. './05/include/kernel/gdt.h'

```
#ifndef _GDT_H
#define _GDT_H 1

#include <inttypes.h>
#include <stdbool.h>
#include <kernel/os.h>

void gdt_desc_seg (int          descriptor,
                  uint32_t     base,
                  uint32_t     limit,
                  bool          present,
                  bool          granularity,
                  bool          code,
                  bool          write_execute,
                  bool          expand_down_non_conforming,
                  unsigned char dpl);

void gdt_print (void *gptr);
void gdt_load (void *gptr);
```

```
void gdt      (void);

#endif
```

La funzione `gdt_desc_seg()` serve a facilitare la compilazione di un descrittore della tabella; la funzione `gdt_print()` consente di visualizzare il contenuto della tabella, partendo dal contenuto del registro `GDTR`, indipendentemente da altre informazioni; la funzione `gdt_load()` fa in modo che il microprocessore utilizzi il contenuto della tabella GDT; la funzione `gdt()`, avvalendosi delle altre funzioni già citate, crea la tabella minima richiesta, ne mostra il contenuto e la attiva.

Listato u169.3. './05/lib/gdt/gdt\_desc\_seg.c'

```
#include <kernel/gdt.h>
#include <stdio.h>
void
gdt_desc_seg (int      desc,
              uint32_t  base,
              uint32_t  limit,
              bool      present,
              bool      granularity,
              bool      code,
              bool      write_execute,
              bool      expand_down_non_conforming,
              unsigned char dpl)
{
    //
    // Verifica di non eccedere la dimensione dell'array.
    //
    int max = ((sizeof (os.gdt)) / 8) - 1;
    if (desc > max)
    {
        printf ("%s] ERROR: selected descriptor %i when max is %i!\n",
                __func__, desc, max);
        return;
    }
    //
    // Limite.
    //
    os.gdt[desc].cd.limit_a = (limit & 0x0000FFFF);
    os.gdt[desc].cd.limit_b = limit / 0x10000;
    //
    // Indirizzo base.
    //
    os.gdt[desc].cd.base_a = (base & 0x0000FFFF);
    os.gdt[desc].cd.base_b = ((base / 0x10000) & 0x000000FF);
    os.gdt[desc].cd.base_c = (base / 0x1000000);
    //
    // Attributi.
    //
    os.gdt[desc].cd.accessed      = 0;
    os.gdt[desc].cd.write_execute = write_execute;
    os.gdt[desc].cd.expansion_conforming = expand_down_non_conforming;
    os.gdt[desc].cd.code_or_data  = code;
    os.gdt[desc].cd.code_data_or_system = 1;
    os.gdt[desc].cd.dpl          = dpl;
    os.gdt[desc].cd.present      = present;
    os.gdt[desc].cd.available    = 0;
    os.gdt[desc].cd.reserved     = 0;
    os.gdt[desc].cd.big          = 1;
    os.gdt[desc].cd.granularity  = granularity;
}
```

Listato u169.4. './05/lib/gdt/gdt\_print.c'

```
#include <kernel/gdt.h>
#include <stdio.h>
//
// Mostra il contenuto di una tabella GDT, a partire dal puntatore al
// registro GDTR in memoria. Pertanto non si avvale, volutamente, della
// struttura già predisposta con il linguaggio C, mentre «gdt_t» viene
// creato qui solo provvisoriamente, per uso interno. Ciò serve ad
// assicurare che questa funzione compia il proprio lavoro in modo
// indipendente, garantendo la visualizzazione di dati reali.
//
typedef struct {
    uint16_t limit;
    uint32_t base;
} __attribute__((packed)) local_gdtr_t;
//
void
gdt_print (void *gdtr)
{
    local_gdtr_t *g = gdtr;
    uint32_t *p = (uint32_t *) g->base;

    int max = (g->limit + 1) / (sizeof (uint32_t));
    int i;
```

```
printf ("%s] base: 0x%08" PRIx32 " limit: 0x%04" PRIx32 "\n",
        __func__, g->base, g->limit);

for (i = 0; i < max; i+=2)
{
    printf ("%s] %" PRIx32 " %032" PRIb32 " %032" PRIb32 "\n",
            __func__, i/2, p[i], p[i+1]);
}
}
```

Listato u169.5. './05/lib/gdt/gdt\_load.s'

```
.globl gdt_load
#
gdt_load:
    enter $0, $0
    .equ gdtr_pointer, 8          # Primo argomento.
    mov  gdtr_pointer(%ebp), %eax # Copia il puntatore
                                    # in EAX.

    leave
    #
    lgdt (%eax) # Carica il registro GDTR dall'indirizzo
                                    # in EAX.

    #
    # 2 dati per il kernel, DPL 0, comprendente tutta la
    # memoria disponibile: selettore 0x10+0.
    #
    mov  $0x10, %ax
    mov  %ax, %ds
    mov  %ax, %es
    mov  %ax, %fs
    mov  %ax, %gs
    mov  %ax, %ss
    #
    # 1 codice per il kernel, DPL 0, comprendente tutta
    # la memoria disponibile: selettore 0x08+0.
    #
    jmp  $0x08, $flush
flush:
    ret
```

Listato u169.6. './05/lib/gdt/gdt.c'

```
#include <kernel/gdt.h>
void
gdt (void)
{
    //
    // Imposta i dati necessari al registro GDTR.
    //
    os.gdtr.limit = (sizeof (os.gdt) - 1);
    os.gdtr.base = (uint32_t) &os.gdt[0];
    //
    // Azzerare le voci previste dell'array «os.gdt[]».
    // La prima di queste voci (0) rimane azzerata e non
    // deve essere utilizzata.
    //
    int i;
    for (i = 0; i < ((sizeof (os.gdt)) / 8); i++)
    {
        gdt_desc_seg (i, 0, 0, 0, 0, 0, 0, 0, 0);
    }
    //
    // 1 codice per il kernel, DPL 0, comprendente tutta la
    // memoria disponibile: selettore 0x08+0.
    //
    gdt_desc_seg (1, 0,
                 os.mem_ph.total_l, 1, 1, 1, 0, 0);
    //
    // 2 dati per il kernel, DPL 0, comprendente tutta la
    // memoria disponibile: selettore 0x10+0.
    //
    gdt_desc_seg (2, 0,
                 os.mem_ph.total_l, 1, 1, 0, 1, 0, 0);
    //
    // Mostra la tabella GDT e poi la carica.
    //
    gdt_print (&os.gdtr);
    gdt_load (&os.gdtr);
}
```

## Modifiche da apportare a «kernel\_main.c»

« Nel file 'kernel\_main.c' va aggiunta l'incorporazione del file 'gdt.h' e la chiamata alla funzione *gdt()*:

```
#include <kernel/kernel.h>
#include <kernel/build.h>
#include <stdio.h>
#include <kernel/gdt.h>
...
//
// Raccoglie i dati sulla memoria fisica.
//
kernel_memory (info);
//
// Predisporre la tabella GDT.
//
gdt ();
...
```

Una volta ricompilato il lavoro e avviato con Bochs, si deve ottenere una schermata simile a quella seguente:

```
05 20070819115151
[mboot_show] flags: 00000000000000000000001111100111 mlow: 027F mhigh: 00007BC0
[mboot_show] bootdev: 00FFFFFF cmdline: "(fd0)/kernel"
[kernel_memory_show] kernel 00100000..0010BF7C avail. 0010BF7C..01EF0000
[kernel_memory_show] text 00100000..00103418 rodata 00103418..001035FC
[kernel_memory_show] data 001035FC..001035FC bss 00103600..0010BF7C
[kernel_memory_show] limit 00001EF0
[gdt_print] base: 0x0010B648 limit: 0x0017
[gdt_print] 0 00000000000000000000000000000000 000000001000000000100000000000
[gdt_print] 1 0000000000000000000000111011110000 00000000110000001001101000000000
[gdt_print] 2 0000000000000000000000111011110000 00000000110000001001001000000000
[kernel_main] system halted
```