Computer Labs: Lab5
VBE function 0x01: Return VBE Mode Information

2º MIEIC

Pedro F. Souto (pfs@fe.up.pt)

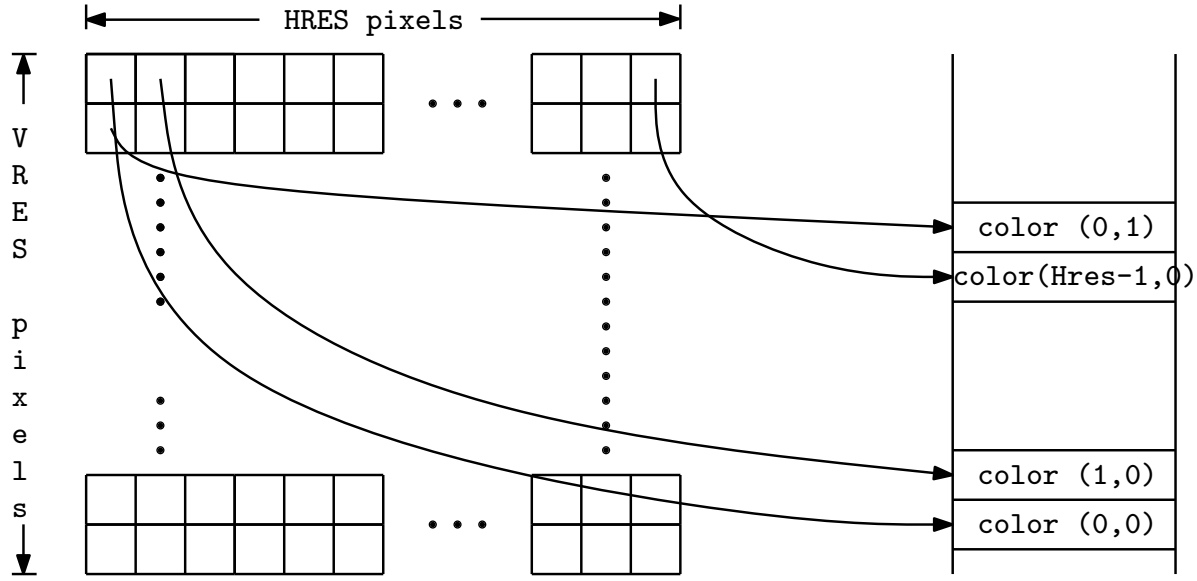
November 14, 2019

Contents

VBE Function `0x01`

Mapping the Linear Frame Buffer

- Before you can write to the frame buffer.



1. Obtain the physical memory address
   - 1.1 Using LCF's `vbe_get_mode_info()`, first;
     - Essencially, uses Function `0x01` Return VBE Mode Information
   - 1.2 Later you can use your own implementation of `vbe_get_mode_info()`
2. Map the physical memory region into the process' (virtual) address space

Finding the Physical Memory Address with VBE (1/5)

Problem How do you find the physical memory address of the video frame buffer automatically (i.e. by programming)?

Answer VBE Function 01h - Return VBE Mode Information

	AX	= 4F01h	Return VBE Mode Information
Input	CX	=	Mode number
	ES:DI	=	Pointer to ModeInfoBlock structure

Ouput AX = VBE return status

- The ModeInfoBlock includes among other information:
  1. The mode attributes, which comprise a set of bits that describe some general characteristics of the mode, including whether:
     - it is supported by the adapter
     - the linear frame buffer is available
  2. The screen resolution of the mode
  3. The color model
  4. The number of bits per pixel
  5. The physical address of the linear frame buffer

Finding the Physical Memory Address with VBE (2/5)

Problem

- The ModeInfoBlock structure must be accessible both in protected mode and in real mode
  - VBE Function 01h is a real mode function
  - Real mode addresses are only 20-bit long (must be in the lower 1MiB).

Solution

- Use `liblm.a` library
  - Provides a simple interface for applications:

    `lm_alloc()`
    `lm_free()`
  - Hides some non-documented functions provided by Minix 3
  - No need to invoke `lm_init()` anymore.
- The `mmap_t` includes both:
  - The physical address, for use by VBE
  - The virtual address, for use in Minix 3

`lm_alloc()`: Parenthesis

Problem

- `lm_alloc()` fails to allocate the desired memory sometimes
  - In Minix 3.1.8 this did not use to happen
  - But in Minix 3.4.0rc6, this appears to be quite frequent

Solution

Retry this often works;

Reboot Minix sometimes this appears to be the quickest solution

Finding the Physical Memory Address with VBE (3/5)

```
phys_bytes buf;
struct reg86u r;

[...]                    /* use liblm.a to initialize buf */

r.u.w.ax = 0x4F01;              /* VBE get mode info */
/* translate the buffer linear address to a far pointer */
r.u.w.es = PB2BASE(buf);        /* set a segment base */
r.u.w.di = PB2OFF(buf);         /* set the offset accordingly */
r.u.w.cx = mode;
r.u.b.intno = 0x10;
if( sys_int86(&r) != OK ) {  /* call BIOS  */
```

`PB2BASE` Is a macro for computing the base of a segment, a 16-bit value, given a 32-bit linear address;

`PB2OFF` Is a macro for computing the offset with respect to the base of a segment, a 16-bit value, given a 32-bit linear address;

Finding the Physical Memory Address with VBE (4/5)

Problem The parameters contained in the buffer returned by VBE function 0x01 are layed out sequentially, with no holes between them

- Simply defining a C struct with one member per parameter with an appropriate type, is not enough
- C compilers layout the members of a struct in order and place them in memory positions whose address is aligned according to their type

Solution Use `#pragma pack`

- Must also reset to the default by adding after the structure:

```
#pragma options align=reset
```

- Alternatively you can also use GCC's `__attribute__((packed))` , which is also supported by clang

Finding the Physical Memory Address with VBE (5/5)

```
#pragma pack(1)

typedef struct {
    uint16_t ModeAttributes;
    [...]
    uint16_t XResolution;
    uint16_t YResolution;
    [...]
    uint8_t BitsPerPixel;
    [...]
    uint8_t RedMaskSize;
    uint8_t RedFieldPosition;
    [...]
    uint8_t RsvdMaskSize;
    uint8_t RsvdFieldPosition;
    [...]
    uint32_t  PhysBasePtr;
    [...]
}  vbe_mode_info_t;

#pragma options align = reset
```

Implementation Notes

- You can call `vbe_get_mode_info()` only once and store its information somewhere
  - This way, you avoid the issues with `lm_alloc()`
- As suggested in the handout, you can use static global variables:

```
static uint16_t hres; /* XResolution */
static uint16_t vres; /* YResolution */
```

- Although you should avoid global variables, this is use akin to the use of static member variables in C++
- You can define `get()` methods, if you want to access these variables from outside of the file where they are declared.
  - For example, `get_hres()`

video_test_rectangle()

- Draw a rectangle on the screen in the desired mode

```
int video_test_rectangle(uint16_t mode, uint16_t x, uint16_t y,
                         uint16_t width, uint16_t height, uint32_t color)
```

- Unlike in previous years, we will test your code for different graphical modes, i.e. different:

  Resolution both horizontal and vertical

  Bits per pixel And color models

  Indexed color modes also called packed-pixel by VBE, appear to have only 8 bits per pixel

  Direct color modes May use a different number of bits per pixel

  - And sometimes, the number of bits per component may be different, even if the number of bits per pixel is the same.

- These affect the offset, with respect to the frame-buffer base address, of the memory location with the color value of a pixel, or of one of its RGB components.
- The goal is that your code be parameterizable so that it can easily handle these differences