Computer Labs: Lab 5 & VBE Function 0x0

2º MIEIC

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

November 18, 2019

Lab5: Video Card in Graphics Mode - 2nd Lab Class

- Write a set of functions:

```
int video_test_xpm(const char *xpm, uint16_t xi, uint16_t yi)
int video_test_move(const char *xpm, uint_t xi,
                    uint_t yi, ...)
int video_test_controller()
```

- Develop your own implementation of `vbe_get_mode_info()`, which must call VBE function 0x01, Return VBE Mode Information.

Lab 5: `video_test_xpm()`

```
int video_test_xpm(const char *xpm, uint16_t xi,  uint16_t yi)
```

What Display the XPM provided in the `xpm` array at the screen coordinates `(xi,yi)`

- Use VBE mode `0x105`

Pixmaps and XPM

pixmap is a short term for “pixel map”, the representation of a digital image as an array of pixel color values

- I.e. it is a map of screen coordinates to color values
- **bitmap** is a pixmap that uses a single bit to denote the color of each pixel

XPM X Pixmap is an image format that allows to represent a pixmap in a textual form, by representing each color value by a different character

- An XPM for a given pixmap can be stored either in a text file, or in a data structure of a C program
- In LCOM we use a simplified version of the XPM format: the XPM format allows to use more than one character to encode a color

Example: Using C Arrays to Store XPMs ("legacy")

```
static char *pic1[] = {
"32 13 4", /* number of columns and rows, in pixels, and color
". 0", /* '.' denotes color value 0 */
"x 2", /* 'x' denotes color value 2 */
"o 14", /* .. and so on */
"+ 4",
"................................", /* the map  */
"............xxx.............",
"..........xxxxxxx...........",
".........xxxxxx+xxxxxx.........",
"......xxxxxxx++++xxxxxxx......",
"....xxxxxxx++++++++xxxxxxx....",
"....xxxxxxx++++++++xxxxxxx....",
"......xxxxxxx++++xxxxxxx......",
".........xxxxxx+xxxxxx.........",
"..........ooxxxxxxxoo..........",
".......ooo..........ooo........",
".....ooo..............ooo......",
"...ooo..................ooo...."
};
```

Question How many elements does an XPM array have?

Example: Using C Arrays to Store XPMs

```
static xpm_row_t const minix3_xpm[] = {
    "196 196 950 2",
    "      c None",
    ".     c #C1C1C1",
    "+     c #323232",
    "@     c #090909",
    "#     c #010101",
    "$     c #161616",
    "%     c #9C9C9C",
    [...]
    "                                                  + @ # $ %
    "                                                & * = - ;
     , ' )          ! ~ { ] ^ / ( _ : <
    "                                              [ } | |
    | | | 1 2 3 4 5      6 7 8 9
     0 a / | | b c d e f ] g h @ g i i i j k
    [...]
}
```

Lab 5: `video_test_xpm()`

```
int video_test_xpm(char *xpm, uint16_t xi, uint16_t yi)
```

What Display the XPM provided in the `xpm` array at the screen coordinates `(xi,yi)`

- Use VBE mode `0x105`

Issue How to convert the XPM to a pixmap?

Answer Use the `xpm_load()` function

Reading a Pixmap from its XPM: `xpm_load()`

```
#define TRANSPARENCY_COLOR_1_5_5_5 0x8000
#define TRANSPARENCY_COLOR_8_8_8_8 0xFF000000
#define CHROMA_KEY_GREEN_888 0x00b140
#define CHROMA_KEY_GREEN_565 0x0588
enum xpm_image_type {
  XPM_INDEXED,
  XPM_1_5_5_5,
  XPM_5_6_5,
  XPM_8_8_8,
  XPM_8_8_8_8,
  INVALID_XPM
};
typedef struct {
  enum xpm_image_type type;
  uint16_t width;
  uint16_t height;
  size_t size;
  uint8_t *bytes;
} xpm_image_t;
uint8_t *(xpm_load)(xpm_map_t map, enum xpm_image_type type,
                    xpm_image_t *img);
```

Reading a Pixmap from its XPM: `xpm_load()`

```
xpm_map_t xmap;
xpm_image_t img;
uint8_t *map;
// get the pixmap from the XPM
map =  xpm_load(xmap, XPM_INDEXED, &img);
// copy it to graphics memory
```

```
uint8_t *xpm_load(xpm_map_t xmap,
                  enum xpm_image_type type,
                  xpm_image_t *img);
```

reads an XPM pixmap `xmap`, and returns the pixmap as a two-dimensional `uint8_t` array. The color encoding of the (output) pixmap is the one specified in `type`. It assumes that the XPM uses:

- Either one char per color and one byte per color – this is enough if it uses few colors and is OK for mode `0x105`
- Or 3 bytes per color, with no restriction on the number of chars per color (this is the format generated by GIMP)

Lab 5: `video_test_move()`

```
int video_test_move( char *xpm[],,
                     uint16_t xi, uint16_t yi,
                     uint16_t xf, uint16_t yf
                     int8_t speed, uint8_t frame_rate)
```

What? Move a sprite on the screen (only along the x or y axes)

- `xpm` XPM for the sprite
- `(xi,yi)` initial coordinates (of ULC)
- `(xf,yf)` final coordinates (of ULC)
- `speed` speed
  - **If non-negative** number of pixels between consecutive frames
  - **If negative** number of frames required for a 1 pixel movement
- `frame_rate` number of frames per second

video_test_controller()

What? Display some fields of VBE 2.0 `VbeInfoBlock struc`

```
VbeInfoBlock struc
VbeSignature            db      'VESA' ; VBE Signature
VbeVersion              dw      0200h           ; VBE Version
OemStringPtr            dd      ?               ; Pointer to OEM String
Capabilities            db      4 dup (?)       ; Capabilities of graphics controller
VideoModePtr    dd      ?               ; Pointer to VideoModeList
TotalMemory             dw      ?               ; Number of 64kb memory blocks
                                                ; Added for VBE 2.0
OemSoftwareRev          dw      ?               ; VBE implementation Software revision
OemVendorNamePtr        dd      ?               ; Pointer to Vendor Name String
OemProductNamePtr       dd      ?               ; Pointer to Product Name String
OemProductRevPtr        dd      ?               ; Pointer to Product Revision String
Reserved                db      222 dup (?)     ; Reserved for VBE implementation scratch
                                                ;   area
OemData                 db      256 dup (?)     ; Data Area for OEM Strings
VbeInfoBlock ends
```

IMP. The *Ptr fields are far-pointers, i.e.: their 2 MSBytes are the base address of a segment (right shifted by 4 bits), whereas their 2 LSBytes are the offset in that segment

video_test_controller()

How?

1. Call VBE function 0x00, Return VBE Controller Info
2. Copy selected fields from the VbeInfoBlock struct to a structure of type `vg_vbe_contr_info_t`

```
typedef struct {
    char VBESignature[4]; /**< @brief VBE Signature */
    BCD VBEVersion[2]; /**< @brief VBE Version,  */
    char *OEMString; /**< @brief OEM String */
    uint16_t *VideoModeList; /**< @brief Address of first
                element in an array of VBE modes supported.
                Last element must have value 0xFF.*/
    uint32_t TotalMemory; /**< @brief Total Memory in KB */
    char *OEMVendorNamePtr; /**< @brief OEM Vendor Name str
    char *OEMProductNamePtr; /**< @brief OEM Product Name s
    char *OEMProductRevPtr; /**< @brief OEM Product Revisio
} vg_vbe_contr_info_t;
```

3. Call

```
vg_display_vbe_contr_info(vg_vbe_contr_info_t *)
```

Call VBE function 0x00, Return VBE Controller Info

- This is similar to other VBE functions, especially function 0x01
  - Must allocate a buffer in low memory; use `lm_alloc()`
    - You must free it ass soon as possible using `lm_free()`
  - But, must initialize field `VBESignature` with value "VBE2"
    - So that the information returned is compatible with VBE 2.0

Copying the VBEInfoBlock to `vg_vbe_contr_info_t`

Issue Apparently VirtualBox is not fully conformant to VBE 2.0.

> *"VBE 2.0 BIOS implementations must place this string [OemStringPtr] in the OemData area within the VbeInfoBlock if 'VBE2' is preset in the VbeSignature field on entry to Function 00h."*

But this is not what VirtualBox's VBE implementation does.

Instead This field points to some memory location in the low memory, i.e. the 1st MiByte.

- This is also true for the other pointer fields

Issue How to convert those far-pointers to virtual addresses?

1. Convert the far-pointers to linear physical addresses
2. Convert a linear physical address to a virtual address

How to convert those far-pointers to virtual addresses?

Far-pointer to linear physical address

- Far-pointers are 4 byte addresses
  - 2 MS bytes specify a base address
    - Must be shifted left by 4 bits
  - 2 LS bytes specify an offset
- The linear address is obtained by adding the offset to the base address (after shifting)

Linear physical address to a virtual address

- `liblm` maps the 1st MiByte of physical memory on the address space of the process
- Use the information in the `mmap_t` structure filled by `lm_alloc(size_t size, mmap_t)`

```
typedef struct {
   phys_bytes phys; // physical address of allocate region
   void *virt;      // mapped virtual address
   size_t size;     // size of allocated region in bytes
 } mmap_t;
```