PP2 - 2020

1. The LPV-API

In this programming test you will have to develop a Minix privileged program that interfaces with your screen's computer. For that you will have to use a new API, the LCOM Protected Video API, or just LPV-API, which is inspired in the VBE-API, but works in protected mode. Despite their similarities, the two APIs are **different. So, please read carefully its description, to ensure that your implementation is according to its specification in this test.**

Summarizing, the LPV-API supports the following functions:

`int lpv_get_mode_info(uint16_t mode, lpv_mode_info_t * lmi_p)`

Where `lpv_mode_info_t` is:

```
typedef struct {
    uint16_t x_res;          // horizontal resolution
    uint16_t y_res;          // vertical resolution
    uint8_t bpp;             // bits per pixel
    uint8_t r_sz;            // red component size
    uint8_t r_pos;           // red component LSB position
    uint8_t g_sz;            // green component size
    uint8_t g_pos;           // green component LSB position
    uint8_t b_sz;            // blue component size
    uint8_t b_pos;           // blue component LSB position
    phys_addr_t phys_addr;   // video ram base phys address
} lpv_mode_info_t;
```

and `lmi_p`, the second argument, is the address of a previously allocated `lpv_mode_info_t` struct that will be filled with the relevant information for the LPV-mode specified in the first argument, `mode`.

`int lpv_set_mode(uint16_t mode)`

Which allows you to configure the card to operate in the specified LPV-mode in its argument.

Both functions return 0 upon success and non-zero otherwise.

Currently, the LPV-API supports the modes listed in the following table.

Mode	**Screen Resolution**	**Color Model**	**Bits per pixel (R:G:B)**
0	Text Mode (25 lines x 80 chars)	N/A	N/A
1	1024x768	Indexed	8 (N/A)
2	1152x864	Direct color	32 ((8:)8:8:8)
3	1280x1024	Direct color	16 (5:6:5)
4	800x600	Direct color	24 (8:8:8)

Note that in indexed mode, i.e. mode 1, all the members relative to the RGB components of the `lpv_mode_info_t` struct, i.e. their size and the position of their LSB, have value 0.

Therefore, you must use the following function, **already provided by the LCF**, to map physical memory on a process' virtual address space:

```
uint8_t *video_map_phys(uint32_t ph_addr, size_t len);
```

It returns the virtual address of the specified physical memory range.

2. The Problem

2.1. The function to develop

In this test, you must develop the following function:

```
int pp_test_line(uint8_t mode, enum lpv_dir_t dir, uint16_t x,
uint16_t y, uint16_t len, uint32_t color, uint32_t delay);
```

where `enum lpv_dir_t` the following C, enumeration type:

```
enum lpv_dir_t {
    lpv_hor,      // horizontal line
    lpv_vert      // vertical line
};
```

`pp_test_line()` shall configure the video card to operate in the mode specified in its first argument, map its frame-buffer, and then draw a line (either horizontal or vertical) as specified in its second argument, starting at the screen coordinates (x,y) (the pixel on the superior left position of the screen has coordinates (0,0)), with the length specified in the fifth argument, the sixth argument specifies the color to use. Finally the last argument specifies the delay in seconds before switching back to Minix's text mode (you must `sleep()` to measure this).

In the case of modes with direct color, the color is encoded in 3 bytes, such that the LSB encodes the B component, the middle byte encodes the G component, whereas the MSB encodes the R component.

IMP: You should always draw the visible part of the line, even if part of it may not be displayed because it is out of the screen bounds.

IMP: You should develop your code in the `/home/exame/pp/` directory that was created by the setup.sh script. The pp.c file in that directory already includes the `main()` function, which you must not change , and a skeleton for `pp_test_line()`, which you are supposed to complete.

2.2. Building, running and testing

In order to build your program you should use the Makefile that is available in `/home/lcom/labs/pp/`, by executing in that directory:

```
minix$ make depend && make
```

Hint: Remember that you can always login remotely on Minix by running the following command on a terminal (in Linux):

```
$ ssh lcom@localhost -p 2222
```

and use the newly created shell to run (and build) your program.

To learn how to run your program, you should use the command `lcom_run` and specify no program arguments:

```
minix $ lcom_run pp
Usage:
    lcom_run pp "line <mode - one of 1, 2, 3 or 4> <direction - one of
h or v> <x - decimal> <y - decimal> <len - decimal> <color -
hexadecimal> <delay - decimal> -t <test no.>"
    Use 'h' for horizontal, 'v' for vertical
    If mode different from 1, should use 2 hexadecimal symbols per
primary color in RGB order.
```

It will output a usage message that describes the command line arguments that you can use.

IMP: In mode 1, the color pallete is identical to that used by Minix in VBE mode 0x105. I.e. the pallete has only 64 colors, and color 0 is also black.

The `-t` option is **mandatory**. The `<test no.>` parameter is an integer between **1** and **4**. For all test no. 's, the behavior is deterministic, therefore you should get always the same results for the same set of arguments. The following table summarizes each test case:

Test No.	**Test description**	**Guaranteed score**
1	Tests mapping of the video RAM	5%
2	Tests graphics mode setting.	5%
3	Tests resetting to text mode	5%
4	Tests drawn lines.	You can expect about 10% per mode, if your code passes most tests for that mode.

Note that all these tests are parametric. I.e., you can specify arbitrary values for the different arguments.

IMP: To choose other input parameters, you may wish to use the following helper function that displays the LPV-mode information:

```
int pp_display_lpv_mode_info(uint16_t mode);
```

If your program terminates normally, it will print either:

```
Test succeeded.
```

indicating success, or

```
Test FAILED!
```

indicating failure. In case of failure, you should examine carefully the output on the terminal or in the output.txt file and the trace with the calls your program performed in the trace.txt file. (Both text files are in the directory where you executed the `lcom_run` command.)