Faculdade de Engenharia da Universidade do Porto

Laboratório de Computadores

Hugo Gomes – up202004343
João Moreira - up202005035
Sandra Miranda – up202007675
Lia Vieira - up202005042

Final Report
Engenharia Informática e Computação

May 2022

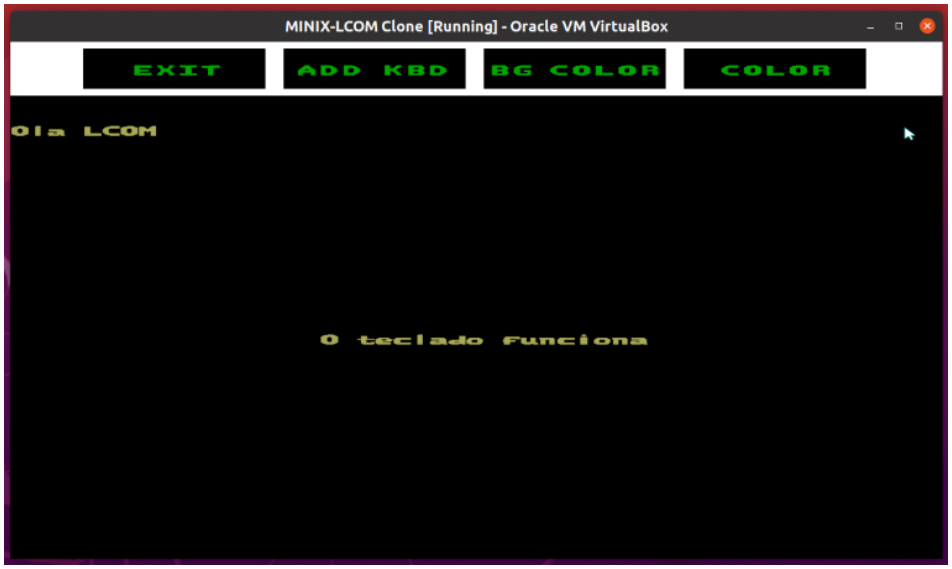
Table of Contents

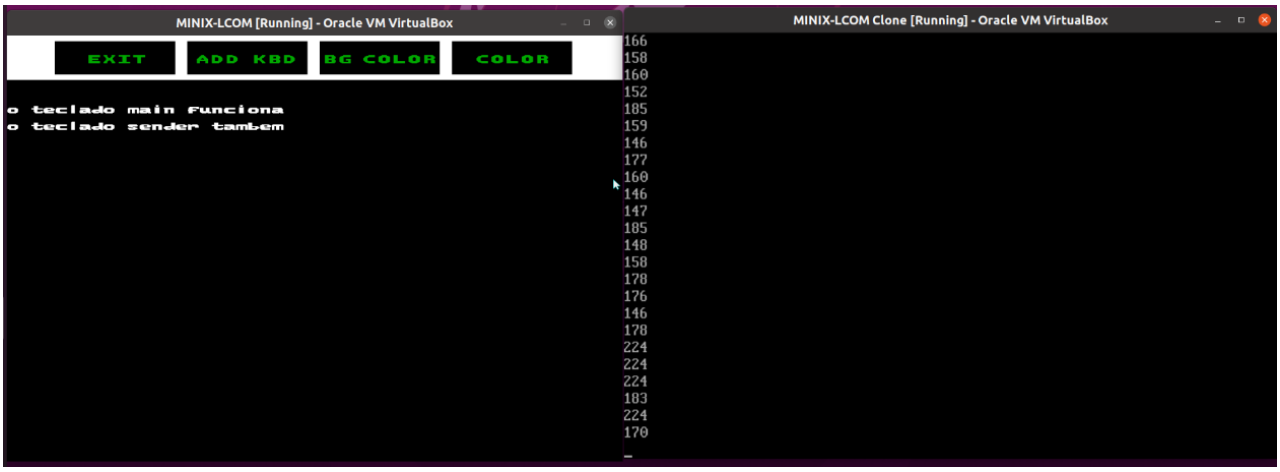
1. User instructions

Our project is a text editor, with which the user will be able to write files using two keyboards. When running the program, the user should add an argument, main or sender, to specify which machine it is. The main one is responsible for running the program, while the sender one oversees the second keyboard. While using the text editor, it's possible to change the color of the letters, the background color and write with two keyboards. At the top, there is a menu with four buttons that allow the user to do the functionalities previously mentioned and exit the program. The menu can be accessed by pressing the buttons with the mouse. The mouse can be also used to click on a position so the user can start writing from there.

When pressing the "COLOR" button, the color of the letters will change in a range of 64 colors. The same number of colors are available for the background that can be changed by pressing the "BG COLOR" button. If the user presses the "ADD KB", he will be able to write to the text editor with two keyboards simultaneously. When the user wants to quit the program, the "EXIT" button should be pressed.


MINIX-LCOM Clone [Running] - Oracle VM VirtualBox
EXIT
ADD KBD
BG COLOR
COLOR
Ola LCOM


MINIX-LCOM Clone [Running] - Oracle VM VirtualBox
EXIT
ADD KBD
BG COLOR
COLOR
Ola LCOM
O teclado funciona


MINIX-LCOM [Running] - Oracle VM VirtualBox
EXIT
ADD KBD
BG COLOR
COLOR
o teclado main funciona
o teclado sender tambem
MINIX-LCOM Clone [Running] - Oracle VM VirtualBox
166
158
160
152
185
159
146
177
160
146
147
185
148
158
178
176
146
178
224
224
224
183
224
170

2. Project status

All the functionalities previously described were fully implemented. The I/O devices used are presented in the following table.

Device	What for	Interrupt/Polling
Timer	Controlling frame rate	Interrupt
Keyboard	User input	Interrupt
Mouse	Buttons selection and text start position selection	Interrupt
Video Card	Application menus and screen display	Interrupt
Serial Port	Use two keyboards	Interrupt

2.1. Timer

The timer is used to generate periodic interrupts at a rate of 60Hz, handling the screen refreshing. The functions responsible for the timer to properly work are **`timer_subscribe_int`**, **`timer_set_frequency`**, **`timer_unsubscribe_int`**, and **`timer_get_conf`**.

2.2. Keyboard

The keyboard controls the user input by handling interrupts on key presses and key releases. The function **`addLetterArray`** is used to insert the character corresponding to the key pressed into an array, so it can be drawn and displayed in the screen. The **`keyboard_ih`**, **`keyboard_subscribe_int`**, **`keyboard_unsubscribe_int`** and **`interruptsKB`** were implemented to handle the interrupts.

2.3. Mouse

The mouse is configured to handle interrupts on movement, selecting buttons on the menu and selecting the position where the user wants to start writing.

The functions **`mouse_subscribe_int`**, **`mouse_enable_info`**, **`handle_byte`**, **`mouse_disable_info`**, **`write_to_mouse`**, **`kbc_ih`** and **`mouse_unsubscribe_int`** were implemented to issue the interrupts and the functions **`interruptsMouse`**, **`getXpos`** and **`getYpos`** are used to handle the position of the mouse.

2.4. Video Card

The video card was configured in graphics mode with indexed color encoding (0x105). Each character was represented using a xpm. The indexed color mode was chosen due to the way that the xpms are loaded. To load them, we created an array with the total number of colors where the index represents the color. Not using the indexed color mode would implicate using a really big array which would not be efficient.

The resolution is 1024x768 pixels and there are 64 different colors available.
We start by using a single buffer, which lead to a constant screen flickering. To handle that issue, double buffering was implemented. Firstly, the auxiliar buffer is updated and at each timer interrupt, the auxiliar buffer is copied to the video card buffer by the function **`draw`**.

The functions **`drawPixMap`**, **`drawPixMapMouse`** and **`drawPixMapChangeColor`** were implemented. The first two were used to draw the letters inside the menus and the mouse, respectively. The last one changes the color of the letters and their background and draws them. The functions **`drawPixel`**, **`vg_draw_hline`**, **`vg_draw_rectangle`** and **`vg_drawMenu`** are responsible for drawing the different structures that compose our text editor. **`clearScreen`** clears the screen by painting everything black.

2.5. Serial Port

The serial port was implemented to enable the use of two distinct keyboards. It is used to send the break codes from the second keyboard to the main machine so it can read and draw the respective characters in the screen. Communication parameters are 9600 bitrates without parity with a stop bit.

The interrupts were handled by the functions **`uart_subscribe_int`** and **`uart_unsubscribe_int`** and the functions responsible for the serial port to properly work are **`uart_subscribe_int`**, **`uart_unsubscribe_int`**, **`uart_read`**, **`uart_write`**, **`uart_send`**, **`uart_receive`** and **`uart_set_conf`**.

3. Code organization/structure

3.1. proj.c

Tests the number of arguments and calls the respective functions whether the argument is main or sender.

3.2. utils.c

Common functions used throughout the different labs.

3.3. videoCard.c

Contains the functions necessary for the graphics to work. The code was developed during lab5 and adapted to the project.

3.4. mouse.c

Contains the functions necessary for the mouse to work. The code was developed during lab4 and adapted to the project.

3.5. timer.c

Contains the functions necessary for the timer to work. The code was developed during lab2 and adapted to the project.

3.6. uart1.c

Contains the functions necessary for the serial port to work.

3.7. keyboard.c

Contains the functions necessary for the keyboard to work. The code was developed during lab3 and adapted to the project.
Provides the struct letter that holds information about each char.

3.8. interrupts.c

Contains the functions responsible for subscribing and unsubscribing all devices.

3.9. loopFuncs.c

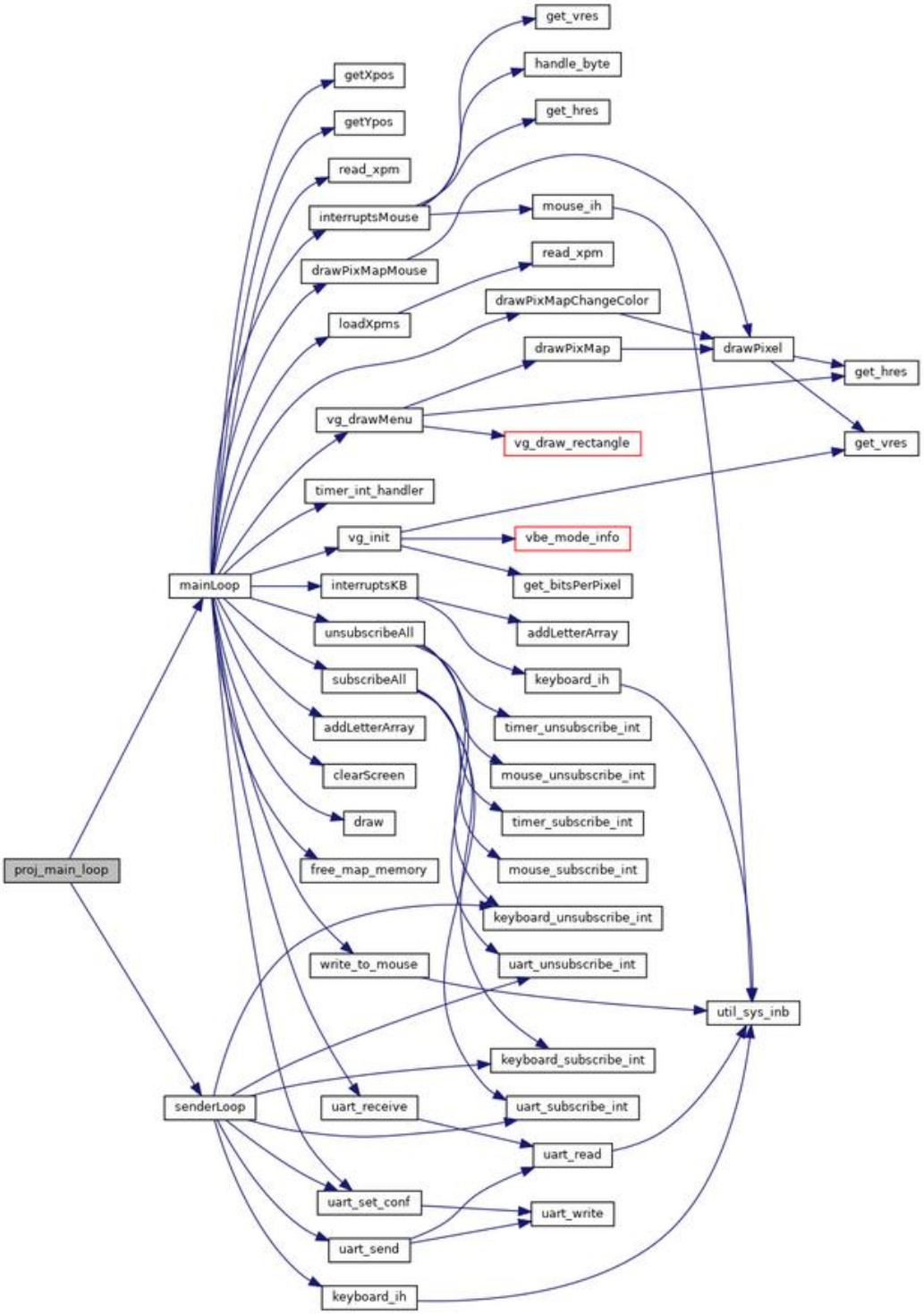
Contains the main loop and the sender loop.

3.10. xpms.c

This file contains all the xpms used in the project and the function responsible for loading them. This function was adapted from a function provided in lab5 by the professor João Cardoso.

Module	Weight	Contribution
proj.c	1%	25% each member
utils.c	1%	25% each member
videoCard.c	20%	25% each member
mouse.c	15%	25% each member
timer.c	5%	25% each member
uart1.c	10%	25% each member
keyboard.c	15%	25% each member
interrupts.c	3%	25% each member
loopFuncs.c	25%	25% each member
xpms.c	5%	25% each member

4. Call graph


get_vres
handle_byte
getXpos
get_hres
getYpos
read_xpm
interruptsMouse
mouse_ih
drawPixMapMouse
read_xpm
drawPixMapChangeColor
loadXpms
drawPixMap
drawPixel
get_hres
vg_drawMenu
vg_draw_rectangle
get_vres
timer_int_handler
vg_init
vbe_mode_info
mainLoop
interruptsKB
get_bitsPerPixel
unsubscribeAll
addLetterArray
keyboard_ih
subscribeAll
addLetterArray
timer_unsubscribe_int
clearScreen
mouse_unsubscribe_int
draw
timer_subscribe_int
proj_main_loop
free_map_memory
mouse_subscribe_int
keyboard_unsubscribe_int
write_to_mouse
uart_unsubscribe_int
util_sys_inb
keyboard_subscribe_int
senderLoop
uart_receive
uart_subscribe_int
uart_read
uart_set_conf
uart_write
uart_send
keyboard_ih

5. Implementation details

5.1. Double buffer

Page flickering was mentioned during theorical classes but because it didn't appear in the labs, some additional information was researched and read. To solve the problem, we implemented a double buffer. At each timer interrupt, the function draw is responsible for copping the auxiliar buffer to the video card buffer.

5.2. Serial Port

Since there was not a lab dedicated to the serial port, there was the need to do some own research. After understanding all the details necessary, the implementation was not hard. The only setback was pre-testing the communication between the two virtual machines with the instructions given.

5.3. `vbe_mode_info`

Since we were told in the laboratory classes that implementing our own version of the **`vbe_get_mode_info`** would be valued, we chose to do so. To accomplish this, we consulted the slides from the previous years that served as the main base for implementing our function.

6. Conclusions

Because we had some difficulties in how to load the xpms there was some delay in the first week, leaving us behind schedule. However, we were able to recover and implemented everything that was planned.

In the future, we would like to implement the use of shift key. In our opinion, it's the only feature that could add something relevant to our project.

As main achievements we would like to highlight drawing and loading the xpms and implementing the serial port, which we thought would be the hardest.