Faculdade de Engenharia da Universidade do Porto

Media Player

T08G2

Joana Maia up202108835

Tomás Moreira up202108858

João Fernandes up202108044

Contents

1.User’s Instructions

The goal of our project was to create a video player (similar to VLC) that allows the user to open video files and perform basic actions such as pause/play and navigate forward or backward on the timeline. It was also intended to display the dates on wich the files were created. We use ffmpeg libraries so it is necessary to set them up for the program to run.

1.1 Main menu

In the main menu there is a test file section where it is possible to test if a file is working. After that the user must follow the instruction that appears on the menu wich is “press space to play file” and he/she can also see how long ago was the file added.

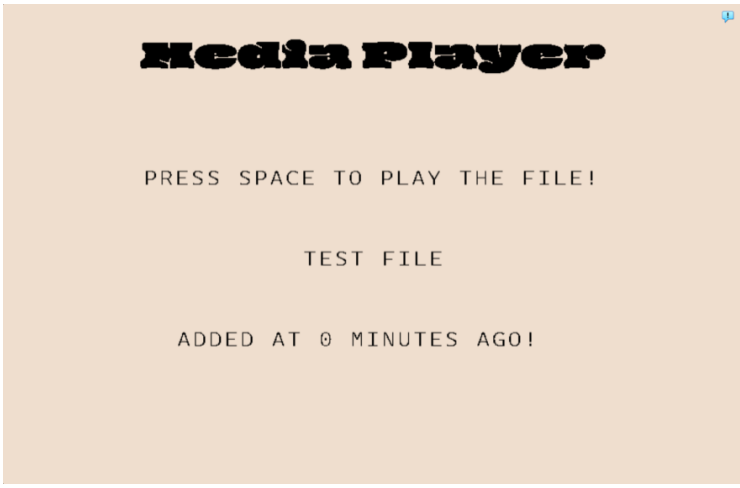
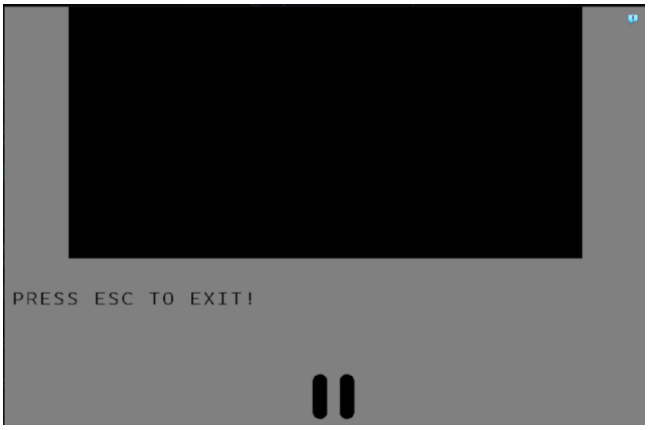


Figure 1- Main menu

1.2 Video menu

In the video menu it is possible to see the respective file running in the format of a video (mp4) and there is a pause/play button so that everyone can pause or play the video whenever they want by pressing the space key on the keyboard.



2. Project Status

Device	What for	Interrupt/polling
Timer	Controlling frame rate	Interrupt
KBD	Controlling the app and the menu	Interrupt
Video Card	Display the video, the menu and characters	Polling
RTC	Know how long ago the files were added	Interrupt

We haven't implemented the functionality to navigate forward or backward on the timeline and the functionality to see the dates on which files were created. We chose to dedicate our time on making the most important functionalities work.

The RTC is there but it's not implemented because we didn't have time to implement the mouse, and as we know, they are related. However, all the functions that we thought were necessary for implementing the RTC are there.

2.1 Graphics card

Video mode 115 was used, which features an 800x600 resolution and a direct color mode with 3 different colors: beige and black were used on the function init_draw() and grey and black were used on the function draw_video_menu().

We also implemented a single buffer but no moving objects were needed.

Our xpmap has all the letters and fonts were also used as well as vbe functions like graphic_set_VBE_mode(uint16_t mode).

2.2 Keyboard

The keyboard was used to control some menu operations like changing the main menu to the video menu using the space key(kbd_scan_space()).

It was also used to pause the video, also with the space key (kbd_scan_space()) and to leave the app with the esc key (kbd_scan_esc()).

2.3 RTC

As it was said before, we didn't implement the RTC but it was supposed to register how long ago the files, that we want to see, were added.

We have the functions that we though were the most important for this implementation, such as rtc_wait_valid(void), rtc_enable(), rtc_disable() and rtc_insert_instruction(uint8_t arrpos, uint8_t *ans).

3. Code organization/structure

3.1 RTC.c

Provides functions that keep and give information about the time that a filed was added.

The function rtc_wait_valid(void) waits until the RTC is ready to be accessed.

The function rtc_enable() activates the RTC.

The function rtc_disable() disables the RTC.

The function rtc_insert_instruction(uint8_t arrpos, uint8_t *ans) reads the value from a specific RTC register.

Weight: 1%

Contributors: João Fernandes

3.2 graphic.c

Provides functions that are related to the graphics manipulation in VBE mode.

The function graphic_set_VBE_mode(uint16_t mode) is responsible for setting the specified VBE mode.

The function changePixelColor(uint16_t x, uint16_t y, uint32_t color) changes the color of a pixel at position (x, y) in the framebuffer.

The function xmp_draw(xpm_image_t img, xpm_map_t xpm, uint16_t x, uint16_t y) draws an XPM image in the framebuffer.

The function xmp_draw_char(xpm_image_t img, xpm_map_t xpm, uint16_t x, uint16_t y, char a) draws a specific character from an XPM font in the framebuffer.

The function vg_draw_hline(uint16_t x,uint16_t y,uint16_t len,uint32_t color) draws a horizontal line in the framebuffer.

The function graphic_draw_rectangle(uint16_t x,uint16_t y,uint16_t width,uint16_t height,uint32_t color) draws a rectangle in the framebuffer.

The functions Red, Green and Blue are auxiliary functions to calculate the red, green and blue color components respectively, based on a start value, a step and a position.

Weight: 40%

Contributors: João Fernandes, Joana Maia

3.3 keyboard.c

Provides functions that control different functionalities.

The kbc_subscribe_int(uint8_t *bit_no) function is responsible for subscribing to keyboard interrupts.

The kbc_unsubscribe_int() function is responsible for unsubscribing from keyboard interrupts.

The kbc_ih() function is the keyboard interrupt service routine.

The function kbd_enable() is responsible for enabling the keyboard.

The function kbd_scan_esc() is responsible for scanning the keyboard until the "ESC" key is pressed.

The function kbd_scan_space() is responsible for scanning the keyboard until the space key is pressed.

Weight: 5%

Contributors: Joana Maia

3.4 menu.c

Provides functions that display the menu and interact with the keyboard.

The function ih_enable() is responsible for initializing the GUI and subscribing to keyboard interrupts.

The function init_draw() is responsible for drawing the initial menu.

The draw_video_menu() function is responsible for drawing the video menu.

The function read_frame() is responsible for passing the frames that come from the codec.

Weight: 5%

Contributors: João Fernanes, Tomás Moreira

3.5 proj.c

Uses the LCF library to handle project initialization, execution, and cleanup.

The main function is the program's entry point and is responsible for setting up the LCF environment.

The function proj_main_loop(int argc, char *argv[]) is the function that will be called by the LCF during the main loop of the program.

Weight: 1%

Contributors: João Fernandes

3.6 timer.c

Provides functions that set and control the system timer, allowing you to adjust the frequency and get information about the current setting.

The timer_set_frequency(uint8_t timer, uint32_t freq) function is responsible for setting the frequency of a specific timer.

The timer_subscribe_int(uint8_t *bit_no) function is used to subscribe to timer interrupts.

The timer_unsubscribe_int() function unsubscribes the timer interrupts.

The timer_int_handler() function is the timer's interrupt handling routine.

The timer_get_conf(uint8_t timer, uint8_t *st) function is used to get the current setting of a timer.

The function timer_display_conf (uint8_t timer, uint8_t st, enum timer_status_field field) is used to display the configuration of a timer in a readable format.

Weight: 5%

Contributors: João Fernandes

3.7 utils.c

Provides some utility functions for bit manipulation operations and I/O port access.

The util_get_LSB(uint16_t val, uint8_t *lsb) function takes a 16-bit value (val) and stores the least significant byte of this value in the memory address pointed to by lsb.

The util_get_MSB(uint16_t val, uint8_t *msb) function takes a 16-bit value (val) and stores the most significant byte of that value (after shifting 8 bits to the right) at the memory address pointed to by msb.

The util_sys_inb(int port, uint8_t *value) function reads a byte from the I/O port specified by port and stores this value in the memory address pointed to by value.

Weight: 1%

Contributors: João Fernandes

3.8 videoCodec.c

Provides functions that belong to the ffmpeg library like avcodec_find_decoder() and avformat_find_stream_info().

These functions are responsible for going to the codec library that ffmpeg has, then choose one to process the video, generate the frames, resize the frame with the function av_image_fill_arrays(), and send the frame to the screen.

Weight: 42%

Contributors: Tomás Moreira

4. Implementation Details

In our program, a layered architecture was implemented to organize and modularize the code. There are separate layers for the user interface (menu), graphics, keyboard input, timer handling and other functionality. Each layer can have well-defined interfaces and responsibilities, promoting code reuse and maintenance.

The program uses an event-driven architecture, handling user interrupts such as keyboard presses and timer interrupts. This approach allows the program to respond to external events in a timely manner, providing a more interactive and responsive user experience.

We use state machines to manage the different states of the program. For example, define states for the menu screen and video screen.Transitions between states can be triggered by specific events or user inputs, and the program can perform actions or change behavior based on the current state.

5. Conclusion

We had some problems with the codec implementation during the project. It was very complex to include all the ffmpeg libraries on Minix, especially on the makefile wich was big stepback of this topic. Furthermore, the inability to read files from the system, being them video files or just test frames was the major issue and the main reason we coldnt achieve the main goal as it limmited debugging a lot. Integrating the codec on the code that we already had on the minix proved to also be tough at start but with some time it was easilly ovecome. It was also a chalenge to output characters, but we manage to do it with success.

We would like to add the functionalities to navigate forward or backward on the timeline and to display the dates on wich the files were created if we had mores time, which would include using the mouse and therefore implement the RTC.

Our main goal was to play a video despite not being able to fully achieve it, we learned that it is not easy to create a video player and it takes a lot of time.

In conclusion despite the many difficulties, we manage to overcome most of them and deliver product that is very close to being able to fully exert all the intended capabillities.