Faculdade de Engenharia da Universidade do Porto

Licenciatura em Engenharia Informática e Computação

Laboratório de Computadores

2022/2023

File Explorer

Team T11G05:

Autores:

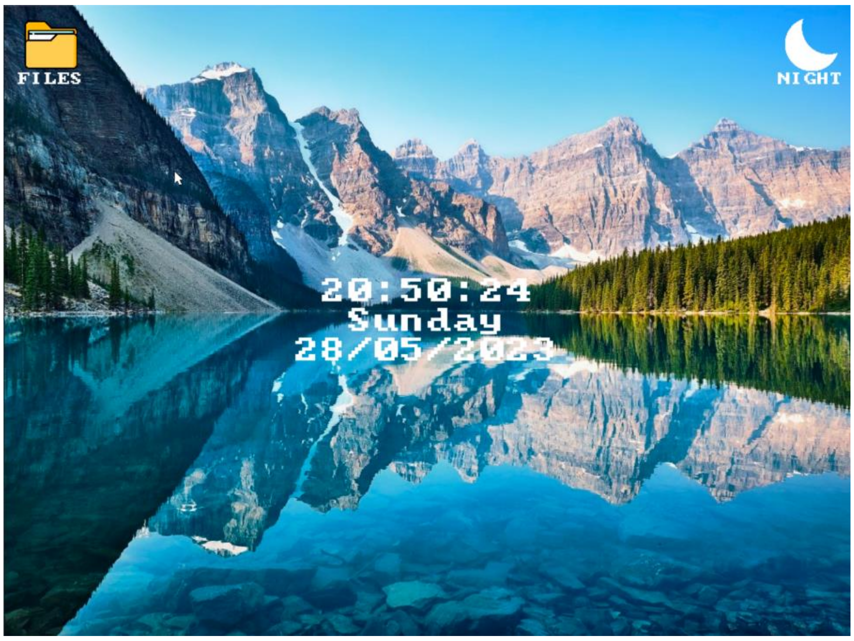
Henrique Silva	up202105647@up.pt
Jose Ribeiro	up202108868@up.pt
Rita Leite	up202105309@up.pt
Tiago Azevedo	up202108699@up.pt

List Of Contents

List Of Figures

User Instructions

1.1. Initial Screen



```
20:50:24
Sunday
28/05/2023
```

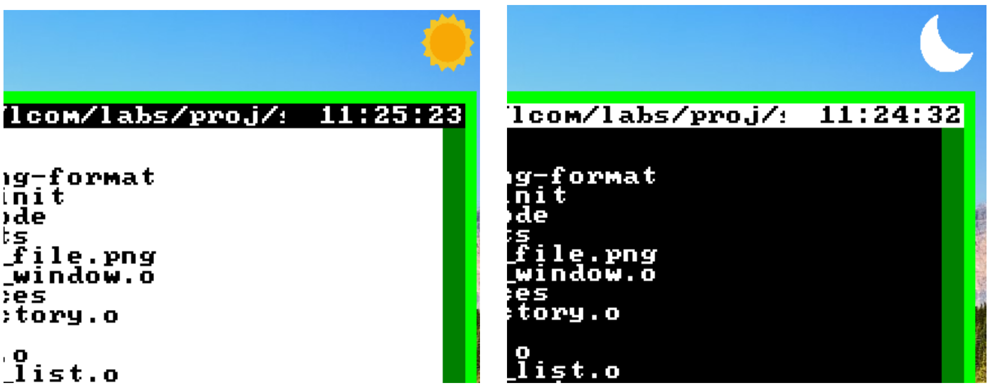
Figure 1- Initial Screen

1.1.1. FILES button

To open this folder, the user has to use the mouse and put it over the icon and press the left button. By doing this, a new window will open, displaying the list of files and/or directories that are inside of the current folder (by default src).

1.1.2. Moon and Sun Icon

At the upper right corner it is always displayed an image of a sun or a moon. Although this feature is mainly for aesthetic purposes, we decided that it made sense to have it because it informs the user what is the mode that the program is (this is, between black or white mood).



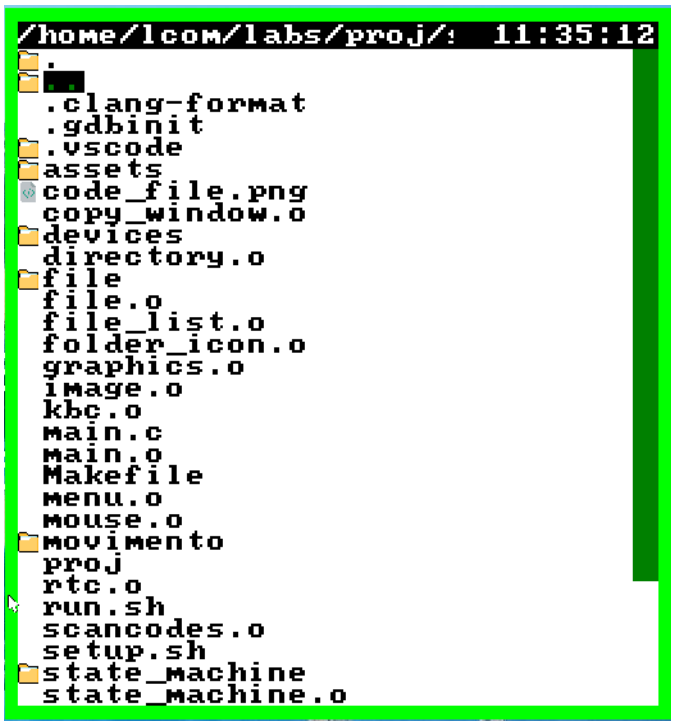
```
lcom/labs/proj/:  11:25:23

ng-format
init
ode
ts
_file.png
_window.o
es
tory.o

.o
_list.o
```

```
lcom/labs/proj/:  11:24:32

g-format
nit
de
s
file.png
window.o
es
tory.o

o
list.o
```

Figure 2- Moon and Sun Image

1.2. List Files Window

1.2.1. Initial Display



```
/home/lcom/labs/proj/:  11:35:12
.
..
.clang-format
.gdbinit
.vscode
assets
code_file.png
copy_window.o
devices
directory.o
file
file.o
file_list.o
folder_icon.o
graphics.o
image.o
kbc.o
main.c
main.o
Makefile
menu.o
mouse.o
movimento
proj
rtc.o
run.sh
scancodes.o
setup.sh
state_machine
state_machine.o
```

Figure 3- List Files Window

This window starts by displaying the list of the names of the files/directories present in the current directory, which in the beginning will be the src directory of our project. (Figure 3).

Folders have a corresponding icon on their left side and images offer a preview of their contents in the same place.

The "." directory represents the directory that we are at the moment (and because of that, if you try to open it, nothing will happen), and the ".." is the previous directory, so it works like a go back button.

The little bar that at the top displays the path of the current directory that we are working with and the time at is at the moment.

To be able to work and operate with the program, we need to use the keyboard or the mouse.

If the user prefers to use the keyboard, then to select a file/directory he has to use the **UP** and **DOWN** keys but if he uses the mouse, he can just click on the content that he wants to select and use the scroll bar to see more.

When clicking on **ENTER** or pressing the left mouse button, if the selected content is a directory, then this will be open and a new list of files will be displayed, but if it is an image then it's content will be displayed in full screen.

1.2.2. Image preview

When opening an image in full screen it will show with a checkered background to indicate transparent areas or areas outside the image (in case the image aspect ratio differs from the screen's). To close this window the user simply has to press the **ESC** key.

1.2.3. Move a File/Directory

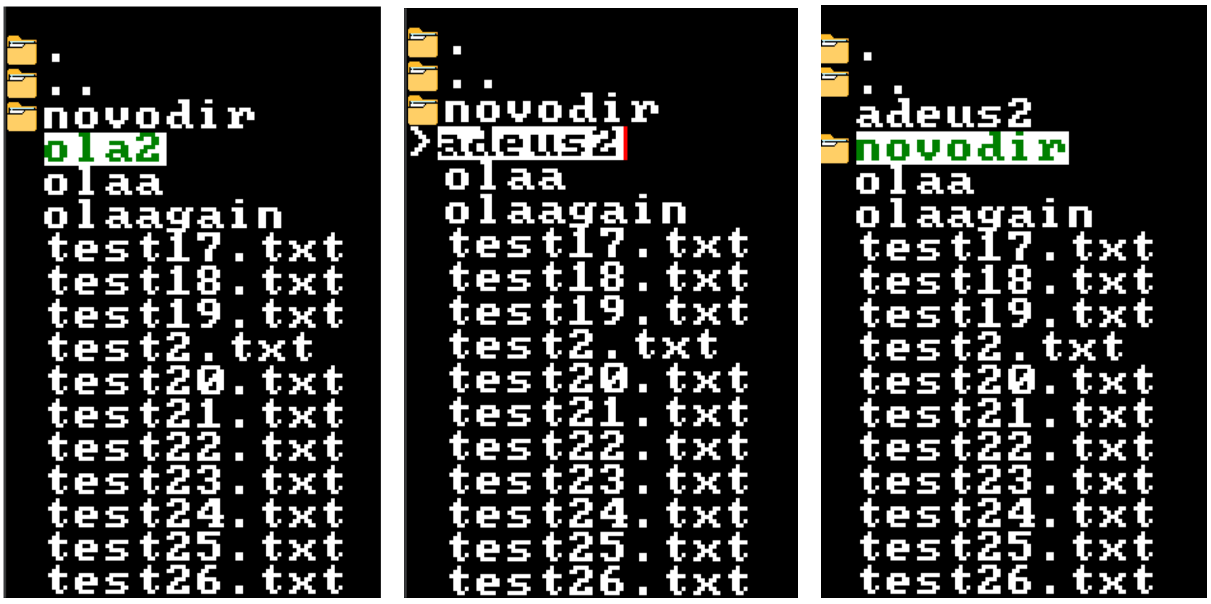
To move a file/directory, the user has to click on **CONTROL** + **X** (this is if the content selected is the desired one) and then click on **CONTROL** + **V** when he goes to the directory that will become the new "parent".

1.2.4. Copy a File/Directory

To copy a file/directory, the user has to click on **CONTROL** + **C** (this is if the content selected is the desired one) and then click on **CONTROL** + **V** to do the operation. It is important to note that if the name of this file already exists in the directory that the user is copying, that it will be added a suffix like "Copy(1)", "Copy(2)", etc.

1.2.5. Rename a File/Directory

To rename a file or directory, the user has to click on **CONTROL** + **R** (this is if the content selected is the desired one) (Figure 4). After doing so, the user must type the new name, and he can use the **BACKSPACE** key to delete the letter behind the cursor and the **RIGHT** and **LEFT** keys to move the cursor itself. After writing the new name, the user has to click on **ENTER** or press the left mouse button inside the window to save the changes, but if in the meantime he has changed his mind and no longer wants to change the name, he can always click on **ESC**, which returns to the normal list mode.



```
.
..
novodir
ola2
olaa
olaagain
test17.txt
test18.txt
test19.txt
test2.txt
test20.txt
test21.txt
test22.txt
test23.txt
test24.txt
test25.txt
test26.txt
```

```
.
..
novodir
>adeus2
olaa
olaagain
test17.txt
test18.txt
test19.txt
test2.txt
test20.txt
test21.txt
test22.txt
test23.txt
test24.txt
test25.txt
test26.txt
```

```
.
..
adeus2
novodir
olaa
olaagain
test17.txt
test18.txt
test19.txt
test2.txt
test20.txt
test21.txt
test22.txt
test23.txt
test24.txt
test25.txt
test26.txt
```

Figure 4- Input Text Box

1.2.6. Create or Delete a File/Directory

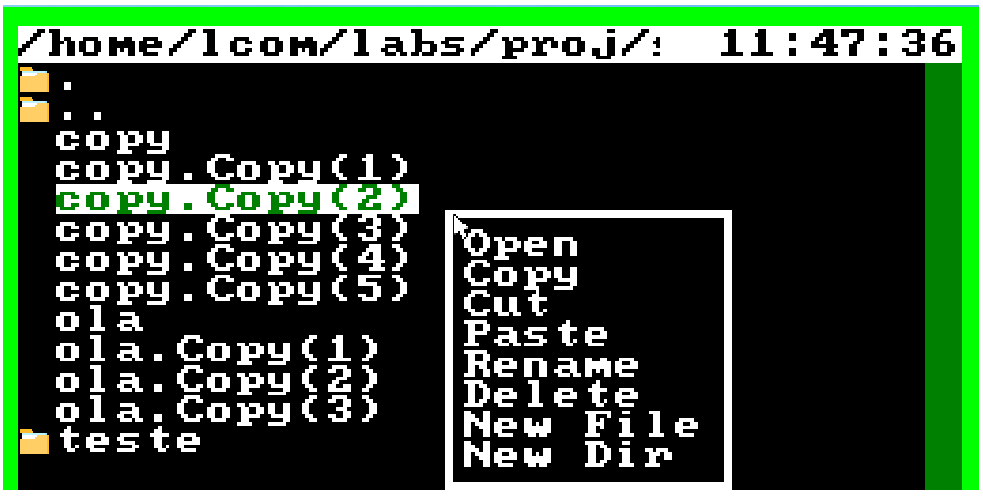
The user has to click on **CONTROL** + **F** or on **CONTROL** + **D** to create a file or a directory, respectively. Next, it will be asked to insert a name and it will work in the same way as renaming, that is, the keys used are the same. To finalize, the key to be pressed is **ENTER** or press the left mouse button inside the window, but if for some reason the user decides to not create anything, all it has to do is press **ESC**.

To delete, it does not matter if it is a file or a directory because in both cases the user just has to press **CONTROL** + **E** (this is if the content selected is the desired one).

1.2.7. Mouse Menu

Although there is a shortcut with the keyboard to perform which operation, the user can just use the mouse and open a menu, by pressing the right button. Here the user just has to press the left button to select the option that he wants to perform, which is much easier.

To close the menu, the user just needs to press a button outside its limits.

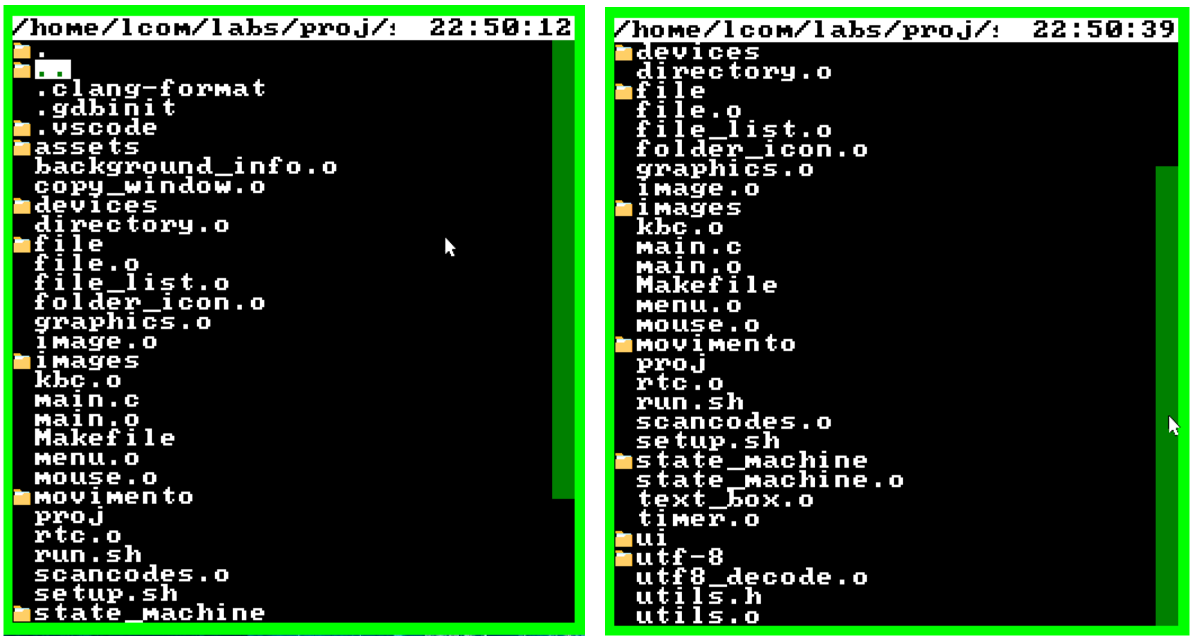


```
/home/lcom/labs/proj/:  11:47:36
.
..
copy
copy.Copy(1)
copy.Copy(2)
copy.Copy(3)
copy.Copy(4)
copy.Copy(5)
ola
ola.Copy(1)
ola.Copy(2)
ola.Copy(3)
teste
```

Figure 5 - Menu

1.2.8. Scroll Bar

When the directory that we are currently seeing is too big, its contents will not be able to fit on the little window, so, because of this, there is a scroll bar that goes up and down. To work with it, the mouse has to hover the bar and to move it is to just keep pressing the left button and do movements up or down. When moving with the mouse the window will scroll automatically to keep the selected option in view.



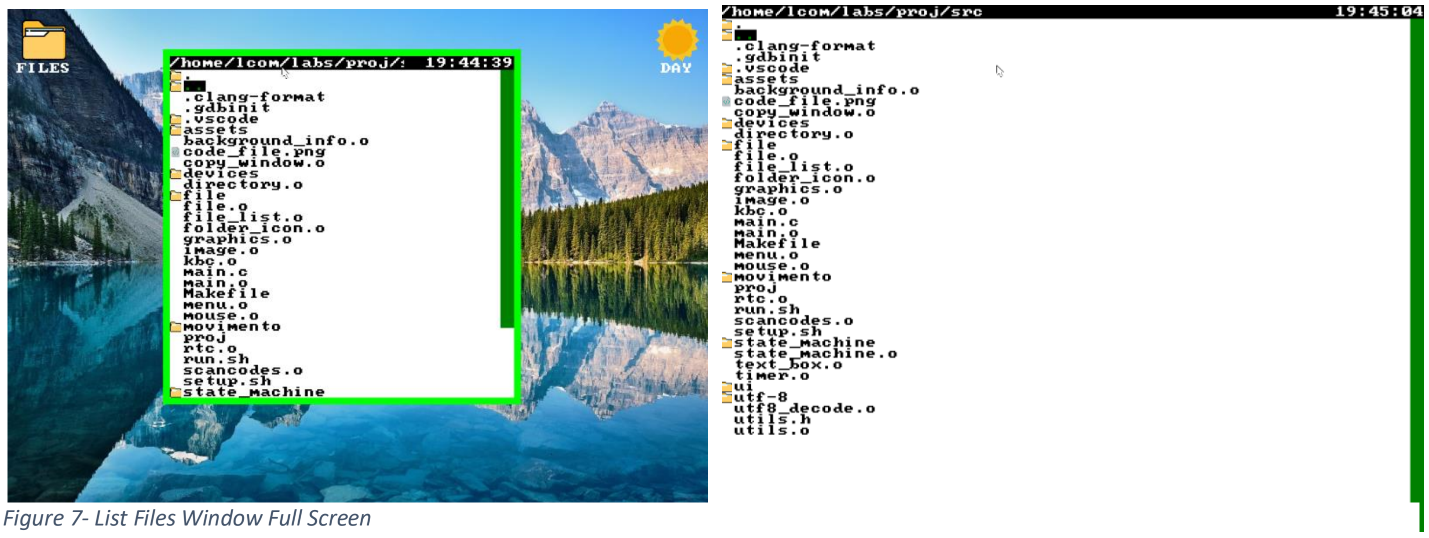
```
/home/lcom/labs/proj/:  22:50:12
.
..
.clang-format
.gdbinit
.vscode
assets
background_info.o
copy_window.o
devices
directory.o
file
file.o
file_list.o
folder_icon.o
graphics.o
image.o
images
kbc.o
main.c
main.o
Makefile
menu.o
mouse.o
movimento
proj
rtc.o
run.sh
scancodes.o
setup.sh
state_machine
```

```
/home/lcom/labs/proj/:  22:50:39
devices
directory.o
file
file.o
file_list.o
folder_icon.o
graphics.o
image.o
images
kbc.o
main.c
main.o
Makefile
menu.o
mouse.o
movimento
proj
rtc.o
run.sh
scancodes.o
setup.sh
state_machine
state_machine.o
text_box.o
timer.o
ui
utf-8
utf8_decode.o
utils.h
utils.o
```

Figure 6- Scroll Bar

1.2.9. Full Screen and Normal Size

When clicking on **CONTROL** + **M** the files list window, if open, switches between full screen and original size. It is important to notice that if this window is in full screen, then the copy status window is displayed on top of the other, and the only way to not see her is by minimizing it.



```
/home/lcom/labs/proj/:  19:44:39
.
..
.clang-format
.gdbinit
.vscode
assets
background_info.o
code_file.png
copy_window.o
devices
directory.o
file
file.o
file_list.o
folder_icon.o
graphics.o
image.o
kbc.o
main.c
main.o
Makefile
menu.o
mouse.o
movimento
proj
rtc.o
run.sh
scancodes.o
setup.sh
state_machine
```

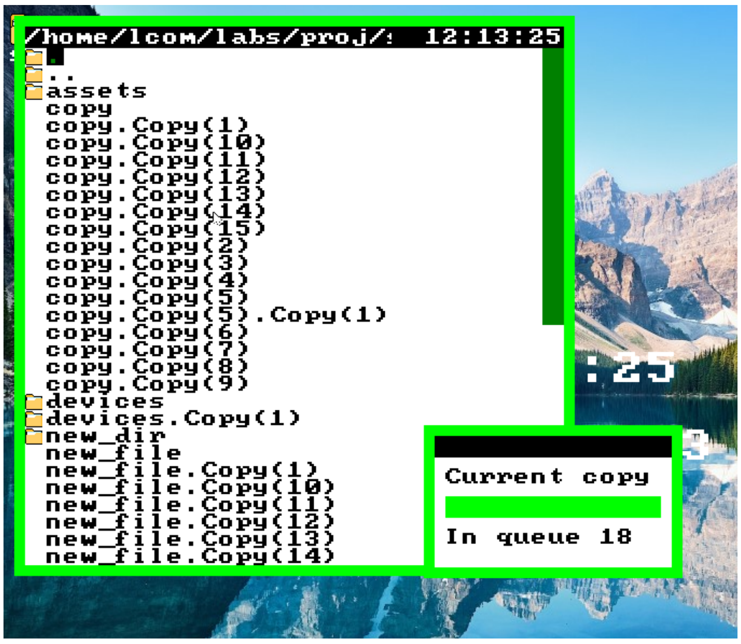
```
/home/lcom/labs/proj/src                                  19:45:04
.
..
.clang-format
.gdbinit
.vscode
assets
background_info.o
code_file.png
copy_window.o
devices
directory.o
file
file.o
file_list.o
folder_icon.o
graphics.o
image.o
kbc.o
main.c
main.o
Makefile
menu.o
mouse.o
movimento
proj
rtc.o
run.sh
scancodes.o
setup.sh
state_machine
state_machine.o
text_box.o
timer.o
ui
utf-8
utf8_decode.o
utils.h
utils.o
```

Figure 7- List Files Window Full Screen

1.3. Window with Copy Status

When the user is copying or moving a file/directory, he can see the progress of the action. This window closes automatically when the copy is completed, so when doing small copies the user will not be able to see it, just with the ones that take a lot of time.

The "In queue" text tells the user how many files are still waiting to be copied and the green bar corresponds to the progress of the copy of the current file.



```
/home/lcom/labs/proj/:  12:13:25
.
..
assets
copy
copy.Copy(1)
copy.Copy(10)
copy.Copy(11)
copy.Copy(12)
copy.Copy(13)
copy.Copy(14)
copy.Copy(15)
copy.Copy(2)
copy.Copy(3)
copy.Copy(4)
copy.Copy(5)
copy.Copy(5).Copy(1)
copy.Copy(6)
copy.Copy(7)
copy.Copy(8)
copy.Copy(9)
devices
devices.Copy(1)
new_dir
new_file
new_file.Copy(1)
new_file.Copy(10)
new_file.Copy(11)
new_file.Copy(12)
new_file.Copy(13)
new_file.Copy(14)
```

```
Current copy

In queue 18
```

Figure 8- Copy Status Window

If both windows are open at the same time the currently active window will show on top, the currently active window is the one that lastly interacted with the mouse, when the copy window shows up it also becomes the active window.

1.4. Drag Windows

The user has to have the mouse hover the top bar of the window (that will be black if it is in light mode and white if in dark mode), keep the left button pressed and move the mouse in the direction of the place where he wants to put it.

1.5. Minimize Windows

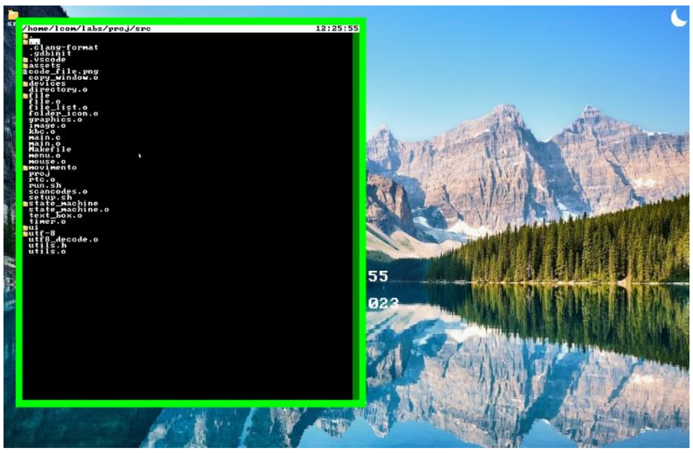
To minimize a window, the user just has to press the left button of the mouse in the background or **CONTROL** + **J** to toggle if the copy window is minimized (it will still only show if there is a copy being performed).

1.6. Change mouse used to keyboard mode

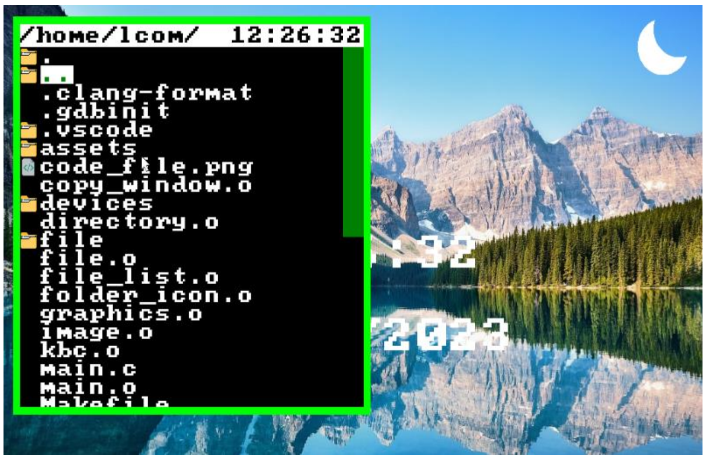
When pressing **TAB** the program switches between using the mouse the normal way or using the arrow keys to move the cursor (it's still possible to move the mouse normally), because sometimes it can be more practical and easy (ex: when copying large files). It is important to notice that if the user is using the mouse in this last mode, then he can't use the arrow keys to go up and down the list of files.

1.7. Zoom In and Out

When clicking on **CONTROL** and **+** or **CONTROL** and **–** the user will be doing zoom in and zoom out respectively.



```
/home/lcom/labs/proj/src                12:25:55
.
..
.clang-format
.gdbinit
.vscode
assets
code_file.png
copy_window.o
devices
directory.o
file
file.o
file_list.o
folder_icon.o
graphics.o
image.o
kbc.o
main.c
main.o
Makefile
menu.o
mouse.o
movimento
proj
rtc.o
run.sh
scancodes.o
setup.sh
state_machine
state_machine.o
text_box.o
timer.o
ui
utf-8
utf8_decode.o
utils.h
utils.o
```



```
/home/lcom/  12:26:32
.
..
.clang-format
.gdbinit
.vscode
assets
code_file.png
copy_window.o
devices
directory.o
file
file.o
file_list.o
folder_icon.o
graphics.o
image.o
kbc.o
main.c
main.o
Makefile
```

Figure 9- Zoom

2. Project Status

DEVICE	WHAT FOR	INTERRUPT
Timer	Controlling frame rate	Y
Keyboard	Navigation and perform actions	Y
Mouse	Navigation and perform actions	Y
Video Card	Screens display	N
Real-Time Clock	Change between light and dark mode and show real-time	Y

2.1 Timer

The timer is used to generate continuous interrupts with a frequency of 60hz in order to refresh the page to ensure that the information is updated.

Because we don't want to refresh the page every time there is a interrupt for which device (because it would affect our project efficiency), we create a variable that is set to true every time something changes, for example when a key that affects the content displayed is pressed or when a mouse packet is completed and ready to be processed.

It is implemented on the folder *src/devices/timer*.

To deal with the subscriptions of the timer interrupts we used the functions: *timer_subscribe_int* and *timer_unsubscribe_int*.

2.2 Keyboard

The keyboard was implemented to perform text input, that is necessary for when the users are naming a file or directory. Besides that, almost every action can be performed with a keyboard shortcut: creating, deleting, renaming, coping, moving and pasting files and directories, zoom in and zoom out and close the program.

It is implemented on the folder *src/devices/keyboard*.

To deal with the subscriptions of the keyboard interrupts we used the functions: *kbc_subscribe_int* and *kbc_unsubscribe_int*. To deal with the interrupt itself we implemented the functions: *kbc_read_scancode* and *kbc_read_byte*.

2.3. Mouse

In the list files window, the mouse will be used to select files, open the menu and select options, and deal with the scroll bar. Besides that, the mouse will be in charge of moving the windows and minimizing them.

It is implemented on the folder *src/devices/mouse*.

To deal with the subscriptions of the mouse interrupts we used the functions: *mouse_subscribe_int* and *mouse_unsubscribe_int*. To deal with the interrupt itself we implemented the functions*: mouse_ih, kbc_read_packet* and *mouse_read_byte*.

2.4. Video-Card

The video graph was configured to work in mode 0x118 which means that the resolution is 1024x768 and the color encoding is direct. The color is in 24 bits which means that each component (green, blue and red in that order) has 8 bits.

We decided to use the page flipping method to reduce the flickering that was happening at the beginning. To do this, we created a buffer twice the size needed for the screen and divided it into a back buffer and a front buffer of equal sizes. While displaying the contents of the front buffer, we draw on the back buffer, and when we finish drawing, we swap the buffers, making the front buffer the back buffer and vice versa. The function that implements this method is called *flip_page*.

It is implemented on the folder *src/devices/graphics*.

The only moving objects are the mouse, scrollbar and the windows themselves and the only collision detection is of the type point vs rectangle involving the mouse and another object.

To draw text we use a library from https://github.com/dhepper/font8x8 that includes a c array of 8x8 bits for several unicode characters where a 1 indicates that the position should be filled with the desired color and a 0 indicating that it should be left empty. In order to draw text in higher sizes we simply turn each bit into a n*n square when drawing it.

The only VBE functions used are to change the mode and to do a page flip.

2.5. Real-Time Clock (RTC)

The Real Time Clock was implemented to change between dark and light modes depending on what time it is. It is also used to display the current time, day of the week and date.

It is implemented on the folder *src/devices/rtc*.

To deal with the subscriptions of the RTC interrupts we used the functions: *rtc_subscribe_int* and *rtc_unsubscribe_int*. To deal with the interrupt itself we implemented the functions: *rtc_ih, rtc_get_real_time* and *background_theme_handler*. To read the current time we implemented the function *rtc_get_real_time* and to set an alarm we implemented the function *rtc_set_alarm*.

3. Code Organization/structure

3.1. devices/utils.c

Developed in lab2 in practical classes.

3.2. devices/timer/

3.2.1. timer.c

Here you will find all the timer specific functions.

Developed in lab2 in practical classes.

3.2.2 i8254.h

Here you will find some constants related to the timer device that will be used constantly throughout the project.

Developed in lab2 in practical classes.

3.3. devices/keyboard/

3.3.1. kbc.c

Here you will find all the keyboard specific functions.

Developed in lab3 in practical classes.

3.3.2. scancodes.c (1%)

Here you will find the mapping made of the scancodes and their meaning, which will be useful to do text input.

Developed by Rita Leite.

3.3.3. i8042.h

Here you will find some constants related to the keyboard device that will be used constantly throughout the project.

Developed in lab3 in practical classes and improved by Rita Leite.

3.4. devices/mouse/

3.4.1. mouse.c (4%)

Here you will find all the mouse specific functions.

Developed in lab4 in practical classes but changed in many aspects by Tiago Azevedo.

3.4.2. i8254_mouse.h

Here you will find some constants related to the mouse that will be used constantly in the functions implemented in this folder.

Developed in lab4 in practical classes and improved by Tiago Azevedo.

3.5. devices/graphics

3.5.1. graphics.c (10%)

Here you will find all the video-card specific functions and the primitive drawing functions, as rectangles, text and images. It also deals with the screen and windows that are displayed.

In this file we use several structures:

- **Bounds**: represents an area given by its top left and bottom right corners (used to make sure there are no overflows when drawing)

- **Window**: represents a screen section and it's offset from the screen

- **WindowStack**: represents a stack of Window objects(used to keep track of the current window and its parents)

Developed by José Ribeiro and Tiago Azevedo.

3.5.2. image.c (3%)

In order to draw these images we had to implement some functions to load them. We created an struct called **Image32** to keep the relevant information about the image loaded, such as the width, height and stride, as well as an array with the pixels used.

In this module is also defined a struct called **Image24** with the same structure used in the previous file's window struct to represent a group of pixels arranged in a rectangular faction.

Developed by José Ribeiro.

3.5.3. pixel.h (2%)

This is where the **pixel24** (used to represent a screen pixel) and the **pixel32** (used on the images) structures are defined.

Developed by José Ribeiro

3.5.4. stb_image.h

To draw images we used a library made by Sean Barret that can be found in https://github.com/nothings/stb/blob/master/stb_image.h.

3.5.5. font

To draw text we used a “font” from https://github.com/dhepper/font8x8.

3.6. devices/rtc

3.6.1. rtc.c (6%)

Here you will find all the real-time clock specific functions as well as some constants related to this device that will be used constantly in the functions implemented in this folder.

We created an struct called **REAL_TIME** that stores the information of the rtc in different components, such as the time, that is divided in seconds, minutes and hours, and the date, divided in days, months, years and weekdays.

Developed by Henrique Silva.

3.7. file

3.7.1. directory.c (3%)

Provides the functions that do the operations asked by the user directories, like renaming, creating, deleting and copying.

Developed by Henrique Silva

3.7.2. file.c (3%)

Provides the functions that do the operations asked by the user in files, like renaming, creating, deleting and copying.

We created two structs called **CopyList** and **FILE_DATA**. CopyList stores the information about the files that are waiting to be copied, while FILE_DATA stores all the information about the file that is being copied.

Developed by Henrique Silva.

3.8. ui

3.8.1. copy_window.c (6%)

Provides the functions that deal with the window that shows the status of the copy of a file/directory that is happening at the moment.

Developed by José Ribeiro.

3.8.2. file_list.c (15%)

Provides the functions that deal with the window that shows the list of files/directories that are in a certain directory.

Developed by José Ribeiro and Rita Leite.

3.8.3. folder_icon.c (3%)

Provides the functions that deal with the folder icon that is at the home screen.

Developed by Rita Leite.

3.8.4. menu.c (7%)

Provides the functions that deal with the menu that is shown when the mouse is being used and the right button is pressed.

In *menu.c*, we use the **Menu** struct which has as members a pointer to an array of **menu_entry** struct, the menu_entry number and the capacity of the array, and the **entry_menu** struct that represent the menu option and has as members a c string with the entry name and a pointer to the function that performs.

Developed by Tiago Azevedo.

3.8.5. text_box.c (6%)

Provides the functions that deal with the input text that is asked when creating or deleting a file or directory.

We also had to create a struct called **String_Builder** that will be responsible for keeping an array with the characters written by the user, that is constantly updated when there is an addition or deletion.

Developed by José Ribeiro and Rita Leite.

3.8.6. background_info.c (0%)

Provides a function to draw the date and time in the center of the desktop.

Developed by Henrique Silva.

3.9. utf-8

We used a library that can be found at https://www.json.org/JSON_checker/ with a function from https://gist.github.com/Miouyouyou/864130e8734afe3f806512b14022226f to handle text conversion between utf8 and utf32 and a wrapper around that same function.

3.10. state_machine/

3.10.1. state_machine.c (25%)

This is where the state machine used is defined. She is updated according to the user actions, so we implemented some functions that deal with these changes and do the respective calls.

To do this state machine we had to create some data structures like: an **enum State** that represents the state that the program is at the moment, an **enum Control_V_Operation** that tells what is the operation that the CONTROL + V keys will perform (if it is moving or copying) and an **enum ActiveWindow** that informs what is the window that is active at the moment, if it is the list of files or the copy status window.

Developed by all members.

3.11. main.c (4%)

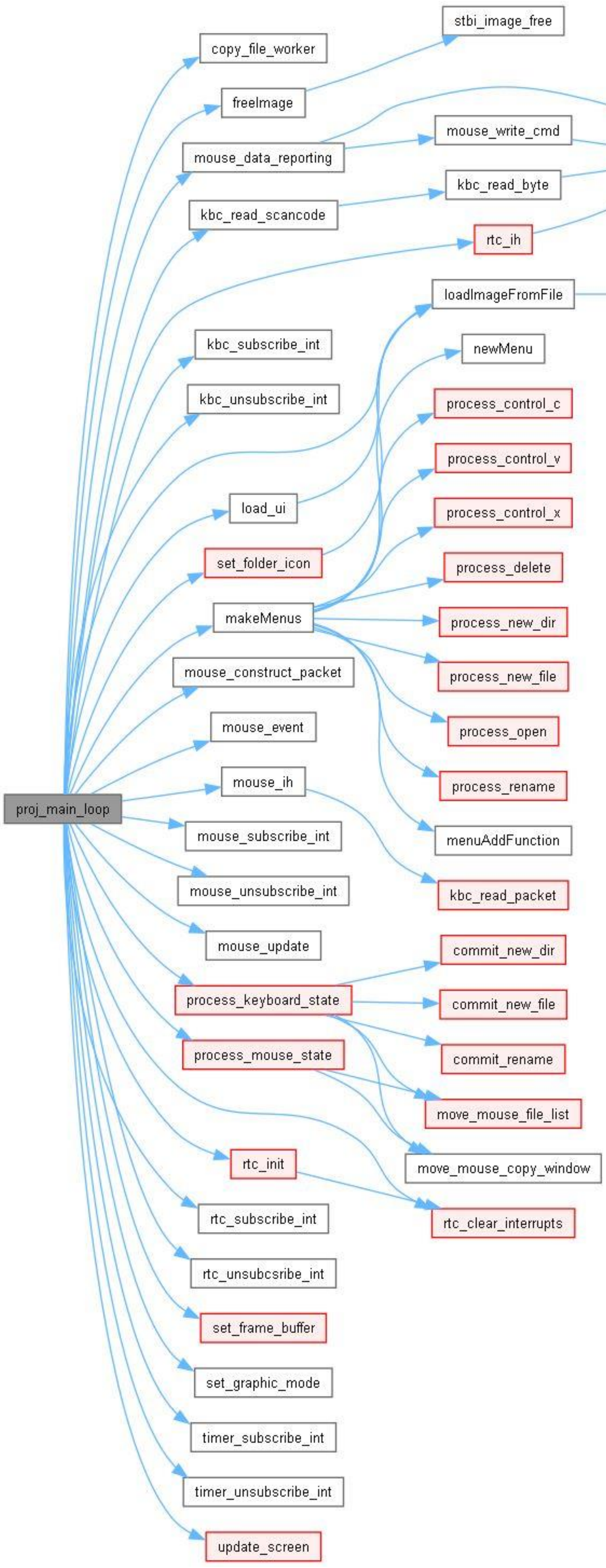
Initializes the program and deals with the interrupts of each devices, calling the right functions.

Developed by all members.

3.12. utils.h (2%)

This is where we can find auxiliary functions/macros. Here, we can find a simple implementation of a queue which was used to store the files that were waiting to be copied. We can also find a simple implementation of a dynamic array, dynamically resizing ring buffer and a deferred return implementation.

Developed by José Ribeiro.



proj_main_loop
copy_file_worker
freeImage
stbi_image_free
mouse_data_reporting
mouse_write_cmd
util_sys_inb
kbc_read_scancode
kbc_read_byte
rtc_ih
loadImageFromFile
stbi_load
kbc_subscribe_int
kbc_unsubscribe_int
newMenu
process_control_c
process_control_v
load_ui
process_control_x
set_folder_icon
process_delete
makeMenus
process_new_dir
process_new_file
mouse_construct_packet
process_open
mouse_event
process_rename
mouse_ih
menuAddFunction
mouse_subscribe_int
mouse_unsubscribe_int
kbc_read_packet
mouse_update
commit_new_dir
process_keyboard_state
commit_new_file
process_mouse_state
commit_rename
move_mouse_file_list
move_mouse_copy_window
rtc_init
rtc_clear_interrupts
rtc_subscribe_int
rtc_unsubcsribe_int
set_frame_buffer
set_graphic_mode
timer_subscribe_int
timer_unsubscribe_int
update_screen

4. Implementation

4.1. State Machine

In our program we had to create a state machine because of the different situations that could happen throughout the program.

One of the reasons was because our mouse wasn’t working properly (too slow) on our computers, which are mainly Windows. To resolve this issue that was affecting our work, especially when testing functions, we decided that we could make the mouse move when clicking on the arrow keys from the keyboard. In summary, we needed to use the arrow keys for different purposes depending on the user's needs, so we created two states, the NORMAL_KBC and NORMAL_MOUSE.

The other reason was because there are different things to draw depending on the action that the user is performing, for example if the user is renaming a file then a text box has to be drawn. To solve this, we created three more states for the actions that needed to deal with text input.

4.2. RTC

In our program we had to find a way to change the background color at a specific time of the day, as well as to display the current time, hour and day of the week. For that, we implemented the Real Time Clock.

The implementation was simple and straightforward. However, we had some trouble in the beginning because the RTC was accepting values in BCD but we were giving the values in binary, so he had to create a function that converts a binary number to BCD. We also had trouble reading the day of the week because it was giving the day before the current day. But, after some digging, we found out that the information in the slides is wrong. It says that the day of the week starts on Sunday and, for that reason, Sunday is equal to 1. But, in reality, Sunday is equal to 0. After that, everything went smoothly. To display the current information of the day, we read the output from the RTC. To change the background color, we set alarms to a specific time of day.

5. Conclusion

One of the main problems we had while doing this project was the mouse implementation because it didn’t work properly, especially when using Windows or Mac. This problem really affected us because we had difficulties when testing features and functions that were depending on this device. However, we were able to fix this issue later on the project by adding “-D __LCOM_OPTIMIZED__” to our make file.

Another problem is that the compiler sometimes randomly crashes and that requires a reboot every time, unfortunately we never could fully discover the reason for this problem since it happened so rarely though it was possible to decrease the chance of it happening by separately compiling the *image.c* file before running make.

6. Appendix

Since we load images from the disk at runtime they must be kept at a directory with path */home/lcom/labs/proj/src/assets* that we used when developing the program or change all the 7 occurrences of */home/lcom/labs/proj/src* in the code.

We included two shell scripts in the src directory setup.sh and run.sh. The **setup.sh** script creates a 1MB file called **upload_test** that can be used to demonstrate the copy progress bar and a folder called test with 50 files inside to demonstrate the scrolling. The **run.sh** script first stops the project if already running then does a make clean to ensure a full compilation of all files then compiles the *image.c* file to minimize the chance of the compiler crashing followed by a make to build the rest of the program and finally it runs the program.

We recommend that the run.sh script be used to run the program to minimize problems and that if using git on windows the user converts the line endings from CRLF to LF since Minix does not seem to understand the previous type.

We also included a folder called images to demonstrate image previewing.