Screensaver Kernel Module

Zhaohan Song(jimmy95@bu.edu)
Geye Lin(gylin@bu.edu)
https://github.com/JimmySong95/EC535_project

***Abstract*—*This report presents the design and implementation of a custom Linux kernel module for a framebuffer-based screensaver. The screensaver displays the letters "BU" on the screen and moves them horizontally, bouncing off the screen edges. The module also sets up an input event handler to reset the screensaver's 15-second timer when an input event occurs. The project demonstrates the use of various Linux kernel APIs, such as framebuffer operations, input event handling, timers, and character device registration. The implemented screensaver operates efficiently, being activated after 15 seconds of inactivity and resetting upon any input event detection.*

1. Introduction

The project topic is the development and implementation of a custom Linux kernel module that creates a framebuffer-based screensaver. The screensaver displays the letters "BU" on the screen and moves them horizontally, bouncing off the screen edges. The kernel module also sets up an input event handler to reset the screensaver's 15-second timer when an input event occurs.

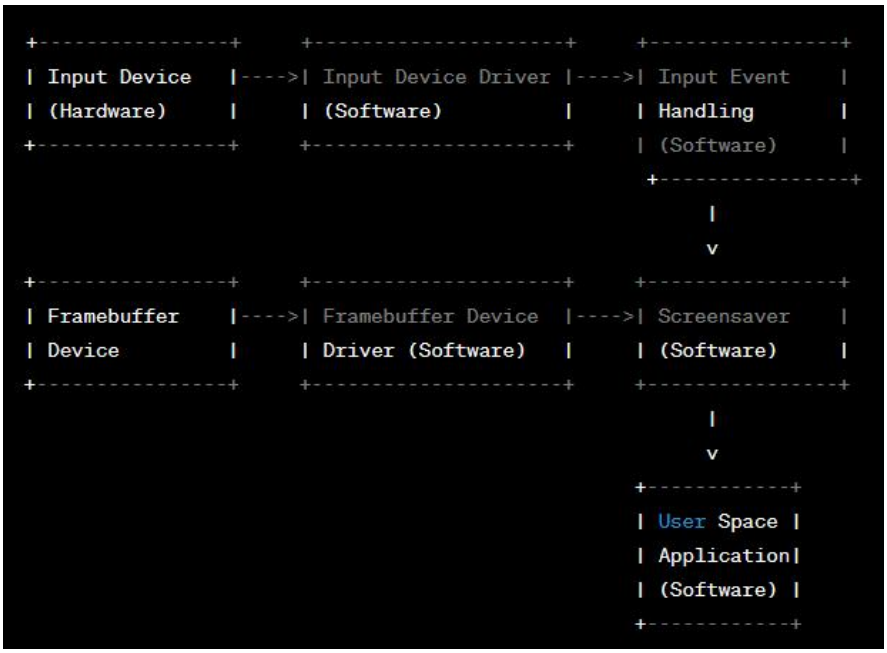
The motivation behind this project is to explore the intricacies of Linux kernel programming, focusing on framebuffer operations, input event handling, and timer management. This problem is important because it demonstrates the process of developing and integrating custom kernel modules into the Linux operating system. Additionally, it emphasizes the significance of efficient and low-level system programming in modern computing systems.

The "big picture" overview of the project components includes:
Framebuffer operations: responsible for drawing and updating the screensaver on the screen.
Input event handling: monitors user input and resets the screensaver timer upon any input event.
Timers: manages the 15-second timer for screensaver activation and resetting.
Throughout the course of the project, we managed to implement a screensaver that operates efficiently, being activated after 15 seconds of inactivity and resetting when an input event is detected. The custom kernel module also demonstrated seamless integration with the Linux operating system, exhibiting the potential for further development and optimization.

2. Design Flow

The project includes the following components and tasks:

- Framebuffer operations: responsible for drawing and updating the screensaver on the screen (implemented by Jimmy95).
- Input event handling: monitors user input and resets the screensaver timer upon any input event (implemented by Gylin).
- Timers: manages the 15-second timer for screensaver activation and resetting (implemented by Gylin).



Overall contributions to the project:
jimmy95: 50% gylin: 50%

3. Project Details

a. Framebuffer Operations(See myscreensaver.c for the source code)[1]

The implementation of framebuffer operations in the provided code can be broken down into several steps:

- Initialization and data structures: The code initializes various data structures to handle framebuffer operations, such as struct fb_info *info, struct fb_fillrect *blank, struct fb_fillrect *Rblank, and struct fb_fillrect *bulogo[9]. These structures store necessary information about the framebuffer and the graphical elements to be drawn, such as the dimensions, positions, and colors of the elements.
- Obtaining framebuffer information: The get_fb_info(unsigned int idx) function retrieves framebuffer information for a given index (in this case, 0) and increments the reference count to the framebuffer information structure.

- Building graphical elements: The build_arrays() function initializes the Rblank and bulogo data structures with the appropriate values for the dimensions, positions, and colors of the background (red) and the "BU" logo (white).

Figure 1. background (red) and the "BU" logo (white)

- Drawing on the framebuffer: The screensaver_handler() function is responsible for the main loop of the screensaver. It locks the framebuffer using lock_fb_info(info) and then uses the sys_fillrect() function to draw the red background and the "BU" logo elements on the framebuffer. The logo's horizontal movement is controlled by updating the dx values of the bulogo elements, and the edge collision is handled by changing the direction of movement (xdir).When implementing this step, we first want to try to display real time on screen, but every time we implement current time as a string, the screensaver can't close after touch the screen. We think this may be because in the screensaver_handler() function we realize all graphs drawing by a while loop, and at the end of the loop we detect the screen event. The handler might always be drawing text and never reach to the step detecting screen events. Eventually we had to give up on this idea.

- Clearing the framebuffer: When the screensaver is deactivated due to user input, the screensaver_handler() function clears the framebuffer by drawing a black background using the blank structure and the sys_fillrect() function.

- Unlocking the framebuffer: After the drawing operations are complete, the unlock_fb_info(info) function is called to release the lock on the framebuffer, allowing other processes to access it.

- In summary, the implementation of framebuffer operations in the provided code involves initializing the necessary data structures, obtaining framebuffer information, building graphical elements, drawing on the framebuffer, clearing the framebuffer, and managing locks to ensure proper access to the framebuffer.

b. Input Event Handling(See myscreensaver.c for the source code)[2][3]

The implementation of input event handling in the provided code can be described as follows:

- Initializing input device: The code starts by initializing the input device using the input_allocate_device() function, which allocates memory for a new input device structure. The input device is then configured by setting its properties, such as its name, ID, and supported event types (e.g., EV_KEY for key events).

- Registering input device: After configuring the input device, it is registered using the input_register_device() function. This makes the device available to the input subsystem and allows other parts of the system to interact with it.

- Handling input events: The main input event handling logic is implemented in the input_event_handler() function. This function is responsible for processing input events that are generated by the input subsystem. It is designed to run as a separate thread, ensuring that input events are processed in parallel with other tasks.

- Reading input events: The input_event_handler() function uses the input_event() function to read input events from the input device. This function blocks until an input event is available, ensuring that the event handling thread does not consume excessive CPU resources when there are no input events to process.

- Processing input events: Once an input event has been read, the input_event_handler() function processes it by checking its type, code, and value. In the provided code, the function is specifically interested in key events with a value of 1 (i.e., screen touch events). When such an event is detected, the screensaver_deactivate() function is called to deactivate the screensaver and clear the framebuffer.

- Cleaning up: When the input event handling thread is no longer needed (e.g., when the program is terminating), the input device is unregistered using the input_unregister_device() function, and the memory allocated for the input device structure is released using the input_free_device() function.

- In summary, the implementation of input event handling in the provided code involves initializing and registering an input device, handling input events in a separate thread, reading input events from the device, processing the events to detect relevant actions (such as screen touch), and cleaning up the input device when it is no longer needed.

c. Timers (See myscreensaver.c for the source code)

- The timers module manages the 15-second timer for screensaver activation and resetting. The timer is set up using the timer_setup function and is modified using the mod_timer function. When the timer expires, the mytimer_callback function is called, activating the screensaver.

4. Summary

In this project, we implemented a Linux kernel module for a framebuffer device, creating a simple screensaver that displays the letters "BU" and moves them horizontally across the screen. The screensaver is activated after 15 seconds of inactivity and resets upon detecting any input event. The project demonstrates the use of various Linux kernel APIs, such as framebuffer operations, input event handling, timers, and character device registration.

Our accomplishments include the successful implementation of the kernel module, efficient screensaver operation, and seamless integration with the Linux operating system. Remaining challenges include improving the screensaver's visual appeal, adding more features(such as real time display), and optimizing its performance.

References

[1] the source code for hellofb.c under $EC535/examples/other/hellofb.c.

[2] the source code for evdev.c - drivers/input/evdev.c - Linux source code (v6.3) - Bootlinhttps://elixir.bootlin.com/linux/latest/source/drivers/input/evdev.c

[3] “Linux Graphics Drivers: an Introduction”: https://people.freedesktop.org/~marcheu/linuxgraphicsdrivers.pdf

[4] the source code for fbmem.c - drivers/video/fbdev/core/fbmem.c - Linux source code (v4.19.82) - Bootlin

[5] input.h - include/linux/input.h - Linux source code (v4.19.82) - Bootlin