CHARACTER DEVICE DRIVER

PROGRAMACION DE SISTEMAS LINUX EMBEBIDOS

Yonnier Alexander Muñoz Salazar

Overview:

This project implements a Linux character device driver to control a GPIO-connected LED on the Lichee RV Dock. A user-space application is also provided to control and monitor the LED.

Directory structure:

```
gpio_led_driver /
|----- driver /
|        |----- gpio_led.c
|        └---- Makefile
|----- user_app /
|        |----- led_control.c
|        └---- Makefile
|----- docs /
         └---- testing_log.md
```

Files description (summary):

> gpio_led.c

Main Functions:

1. **led_open()** – Called when the device is opened (logs: [LED-DRV] Device opened).
2. **led_read()** – Returns the current LED state (0 or 1) to userspace (logs: [LED-DRV] Reading state: X).
3. **led_write()** – Controls the LED:
   - '1' → Turns **ON** (gpio_set_value(1)).
   - '0' → Turns **OFF** (gpio_set_value(0)).
   - Logs: [LED-DRV] Writing value: X.
4. **led_release()** – Called on device close (logs: [LED-DRV] Device closed).

Initialization & Exit:

- **led_init()** – Sets up the char device, GPIO, and logs:
  - [LED-DRV] Initializing LED driver

    - [LED-DRV] Driver initialized successfully (or error logs if fails).
- **led_exit()** – Cleans up resources and logs:
    - [LED-DRV] Exiting driver
    - [LED-DRV] Driver unloaded.

Expected dmesg Logs:

```
[LED-DRV] Initializing LED driver
[LED-DRV] Driver initialized successfully
[LED-DRV] Device opened
[LED-DRV] Writing value: 1
[LED-DRV] Reading state: 1
[LED-DRV] Device closed
[LED-DRV] Exiting driver
[LED-DRV] Driver unloaded
```

Error logs (if any) will appear for GPIO failures, invalid inputs, or device setup issues.

> led_control.c

Main Functionality:

A command-line tool to interact with the **/dev/led_driver** device:

- **Turn LED ON** (on)
- **Turn LED OFF** (off)
- **Check LED status** (status)

Key Functions:

1. **usage()** – Displays help text if incorrect arguments are given.
2. **main()** – Handles device interaction:
    - Opens /dev/led_driver (fails with perror if device unavailable).
    - Writes '1' (ON) or '0' (OFF) based on user input.
    - Reads and displays the current LED state (status).

Expected Usage:

```
./led_control on      # Turns LED ON
./led_control off     # Turns LED OFF
./led_control status  # Prints "LED is currently: ON/OFF"
```

Error Handling:

- Checks for **correct argument count** (exits with usage() if wrong).
- Reports **device open errors** (perror if /dev/led_driver missing).
- **Ignores invalid commands** (shows usage()).

Output Examples:

- **Success:**

```
LED is currently: ON
```

- **Errors:**

```
Failed to open device: Permission denied
Usage: ./led_control [on|off|status]
```

Build instructions

Summary of Commands

# Build the kernel module

```
> cd /home/root/gpio_led_driver/driver
> make
```

# Build the userspace app

```
> cd /home/root/gpio_led_driver/user_app
> make
```

Running the Driver

Load the Module

```
> cd /home/root
> insmod gpio_led.ko
```

Check Logs

```
> dmesg | tail -20
```

Verify Device Registered

```
> cat /proc/devices | grep led_driver
```

Create Device Node

```
> mknod /dev/led_driver c <major> 0
> chmod 666 /dev/led_driver
```

Test the Driver

```
> ./led_control on
> ./led_control off
> ./led_control status
```

Check Kernel Logs

```
> dmesg | tail
```

Unload Module

```
> rmmod gpio_led
> rm /dev/led_driver
```

References

- Based on tutorials by Johannes4Linux