**Low Level Design Document**

**Simulated Character Device Driver**

# Table of Contents

[Table of Contents i](#_Toc43168)

[Revision History i](#_Toc43169)

[1. Introduction 1](#_Toc43170)

[1.1 Purpose 1](#_Toc43171)

[1.2 Document Conventions 1](#_Toc43172)

[1.3 Intended Audience and Reading Suggestions 1](#_Toc43173)

[1.4 References 2](#_Toc43174)

[2. System Use Cases 2](#_Toc43175)

[3. Detailed System Design 2](#_Toc43176)

[3.1 Character device drivers 7](#_Toc43177)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| GROUP A | 06/14/2024 | Initial Draft | 1.0 |

# 1. Introduction

## 1.1 Purpose:

## The purpose of this project is to provide a comprehensive learning experience in developing a character device driver for a simulated hardware device. It covers essential aspects of kernel module programming, device registration, file operations, IOCTL handling, testing, and documentation. Through this project, participants gain proficiency in Linux kernel development and device driver architecture.

## 

## 1.2 Document Conventions

NA

## 1.3 Intended Audience and Reading Suggestions

Document is primarily intended for members of GROUP A team which consists of graduate students working under the guidance of SHWEATANK

## 1.4 References

* Project Proposal Document
* System Specification for character device driver
* System Architecture and High Level Design for character device driver

# 2. System Use Cases

|  |  |  |  |
| --- | --- | --- | --- |
| **Name:** | CHARACTER DEVICE DRIVERS | | |
| **Number:** | 1.0 | | |
| **Author:** | GROUP A | **Date:** | JUNE 14, 2024 |
| **Description:** | The objective of this project is to design, implement, test, and document a character device driver for a simulated hardware device within the Linux kernel environment. This project aims to provide hands-on experience and understanding of kernel module programming, device registration, file operations, IOCTL commands, and interaction between kernel modules and user space applications. | | |
| **Actors:** | USER SPACE, KERNEL SPACE , SYSTEM CALL, MAKEFILE | | |
| **Pre-conditions:** | None | | |
| **Course of**  **Events:** | Day 1: Designing Character Device Features and Planning Module Structure  Day 2: Implementing Module Initialization and Cleanup Routines  .  Day 3: Adding File Operations (open, close, read, write)  Day 4: Implementing IOCTL Command for Device Control  Day 5: Testing the Driver with User-Space Application and Documenting the | | |
| **Postconditions:** | NA | | |
| **Alternatives/ Exceptions:** | NA | | |
| **Notes:** | NA | | |

# 3. Detailed System Design

## 3.1 Character device drivers

### 3.1.1 Design Description:

Objective:

The objective of this project is to design, implement, test, and document a character device driver for a simulated hardware device within the Linux kernel environment. This project aims to provide hands-on experience and understanding of kernel module programming, device registration, file operations, IOCTL commands, and interaction between kernel modules and user space applications.

Project Outline and Tasks:

Day 1: Designing Character Device Features and Planning Module Structure

Character Device Features: Define the characteristics and functionalities of the simulated hardware device that the driver will emulate. This includes deciding on the type of data it handles (text, binary, etc.), potential operations (read, write), and any specific IOCTL commands for device control.

Module Structure Planning: Plan the structure of the kernel module. Identify necessary data structures, initialization routines, cleanup routines, and placeholders for implementing file operations and IOCTL commands.

Day 2: Implementing Module Initialization and Cleanup Routines

Module Initialization: Implement the module initialization routine (init\_module) to register the character device with the kernel. This involves allocating resources, setting up device identifiers, and linking file operations.

Module Cleanup: Implement the module cleanup routine (cleanup\_module) to unregister the character device and release any allocated resources. Ensure proper error handling and resource deallocation.

Day 3: Adding File Operations (open, close, read, write)

Open Operation: Implement the open operation to handle device file opening requests from user space applications. Initialize any necessary data structures or state variables.

Close Operation: Implement the release operation to handle device file closing requests. Clean up resources associated with the device file.

Read Operation: Implement the read operation to allow reading data from the simulated hardware device. Manage data buffers and handle read requests from user space.

Write Operation: Implement the write operation to allow writing data to the simulated hardware device. Manage data buffers and handle write requests from user space.

Day 4: Implementing IOCTL Command for Device Control

IOCTL Command Definition: Define custom IOCTL commands that will be used for device control and configuration. Choose appropriate IOCTL numbers and specify command structures if needed.

IOCTL Handler Implementation: Implement handlers for each IOCTL command defined. Handle IOCTL commands to perform specific actions or retrieve device information based on user requests.

Day 5: Testing the Driver with User-Space Application and Documenting the Development Process

Testing: Develop a user-space application or utilize existing tools to test the functionality of the character device driver. Verify correct operation of open, close, read, write, and IOCTL commands. Test edge cases and error handling scenarios.

Documentation: Document the development process comprehensively. Include sections on design decisions, implementation details, challenges faced, solutions implemented, and lessons learned. Provide clear instructions on how to compile and load the module, interact with the device from user space, and interpret debugging outputs if any.

Conclusion:

This project provides a structured approach to learning kernel module programming and device driver development for a character device in Linux. It emphasizes practical implementation of kernel interfaces, understanding of device management within the kernel, and integration with user space applications. By completing this project, participants gain valuable skills in low-level system programming and a deeper understanding of operating system internals.

### 3.1.2 Design and Implementation Constraints

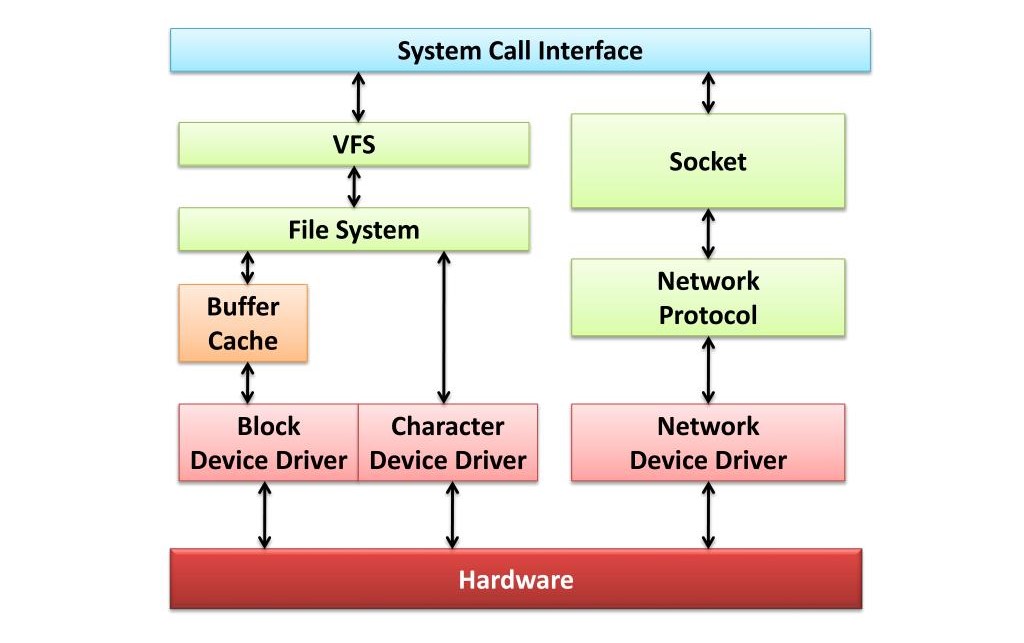
### 3.1.3 Class Diagram:

### 

**Description:**

|  |  |
| --- | --- |
| **Component Name** | **Description** |
| **Character device driver** | **Character devices are typically used for devices that require direct communication with user programs or other devices. Examples of character devices include serial ports, parallel ports, keyboards, mice, and various other input/output devices.** |

**3.1.4 Sequence Diagram**



**Component in the device driver space architecture:**

1. System Call Interface: Provides a means for user applications to request services from the operating system kernel.

2. Virtual File System (VFS): Abstract layer that provides a common interface for different file systems, allowing for uniform file operations.

3. File System: Manages how data is stored and retrieved on storage devices, organizing files and directories.

4. Socket: Endpoint for communication between two machines over a network, supporting network protocols.

5. Buffer Cache: Memory area that stores frequently accessed disk data to improve read/write performance.

6. Network Protocol: Defines rules and conventions for data exchange over a network, enabling communication between devices.

7. Block Device Driver: Manages devices that read/write data in fixed-size blocks, such as hard drives and SSDs.

8. Character Device Driver: Manages devices that transmit data as a stream of bytes, like keyboards and serial ports.

9. Network Device Driver: Handles the interface between the operating system and network hardware, such as NICs (Network Interface Cards).

10. Hardware: Physical components of a computer system, including processors, memory, storage devices, and network interfaces.

**3.1.5 Activity Diagram**

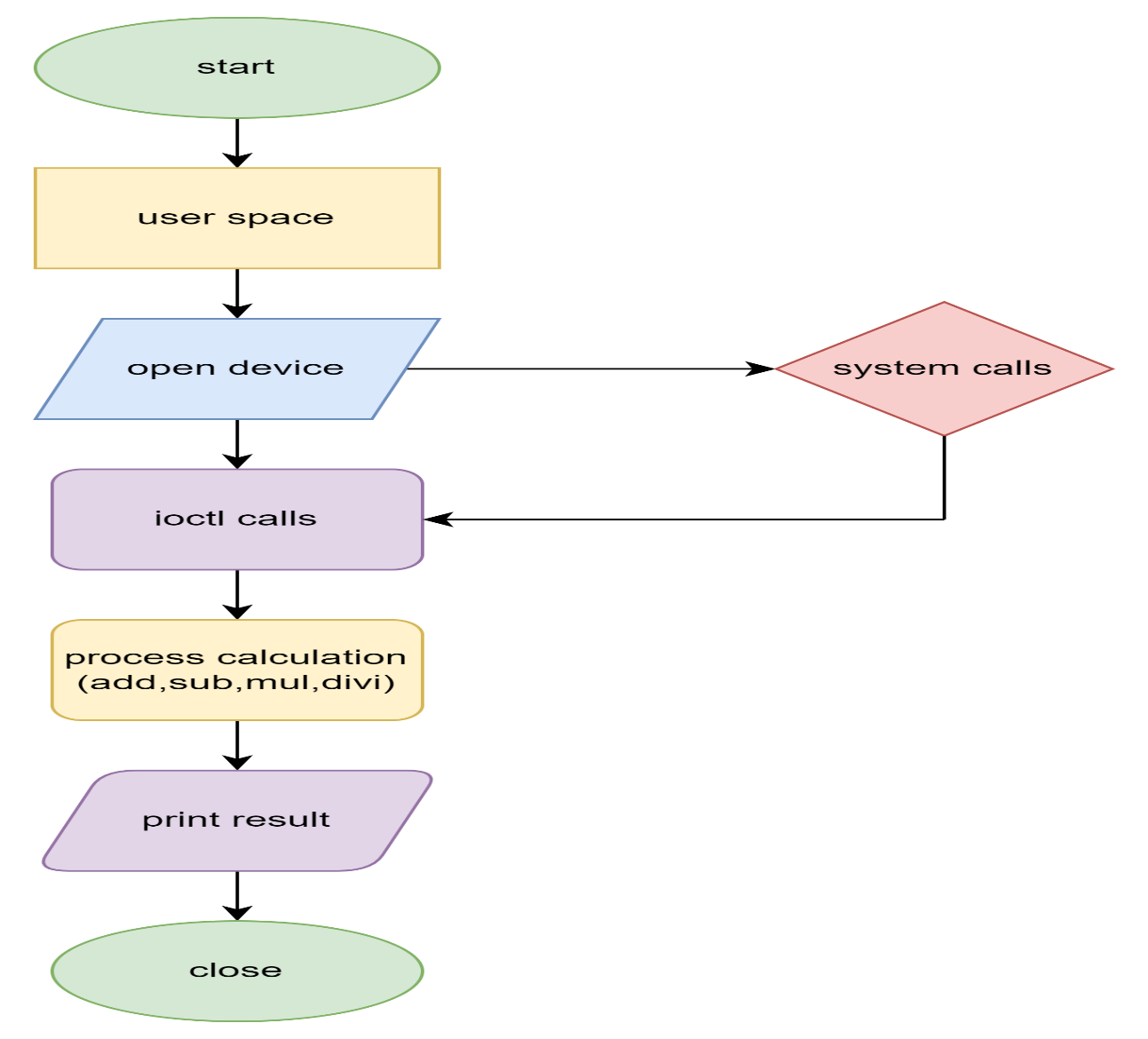
TBD

### 3.1.6 External Interface Requirements

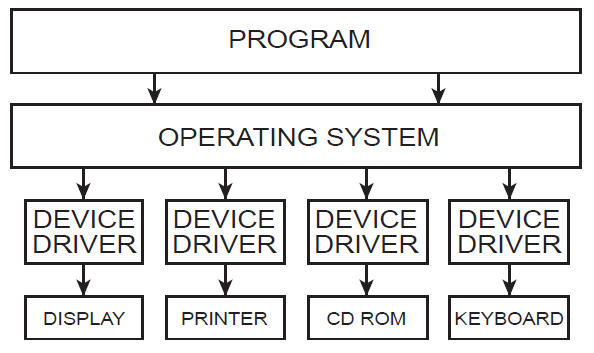
TBD

**3.1.6.1 User Interfaces**

**User space flow diagram:**

****

**3.1.6.2 Hardware Interfaces**



1. PROGRAM: User-level applications that request services from the operating system to perform various tasks, such as reading a file or printing a document.

2. OPERATING SYSTEM: Manages hardware resources and provides services to programs through system calls. It acts as an intermediary between user programs and hardware devices.

3. DEVICE DRIVER (General): Software that allows the operating system and programs to communicate with hardware devices. Each type of hardware device typically has a corresponding driver.

4. DEVICE DRIVER (Display): Manages communication between the operating system and the display hardware, handling tasks like rendering graphics and updating the screen.

5. DEVICE DRIVER (Printer): Manages communication between the operating system and the printer hardware, translating print jobs into a format the printer can understand.

6. DEVICE DRIVER (CD ROM): Manages communication between the operating system and the CD-ROM drive, handling tasks like reading data from CDs.

7. DEVICE DRIVER (Keyboard): Manages communication between the operating system and the keyboard hardware, handling tasks like interpreting keystrokes and sending them to the operating system.

8. Ima: This seems to be a typo or an unclear abbreviation. If it's intended to represent a component, please provide more context.

9. DISPLAY: The hardware device that presents visual output from the computer, such as a monitor or screen.

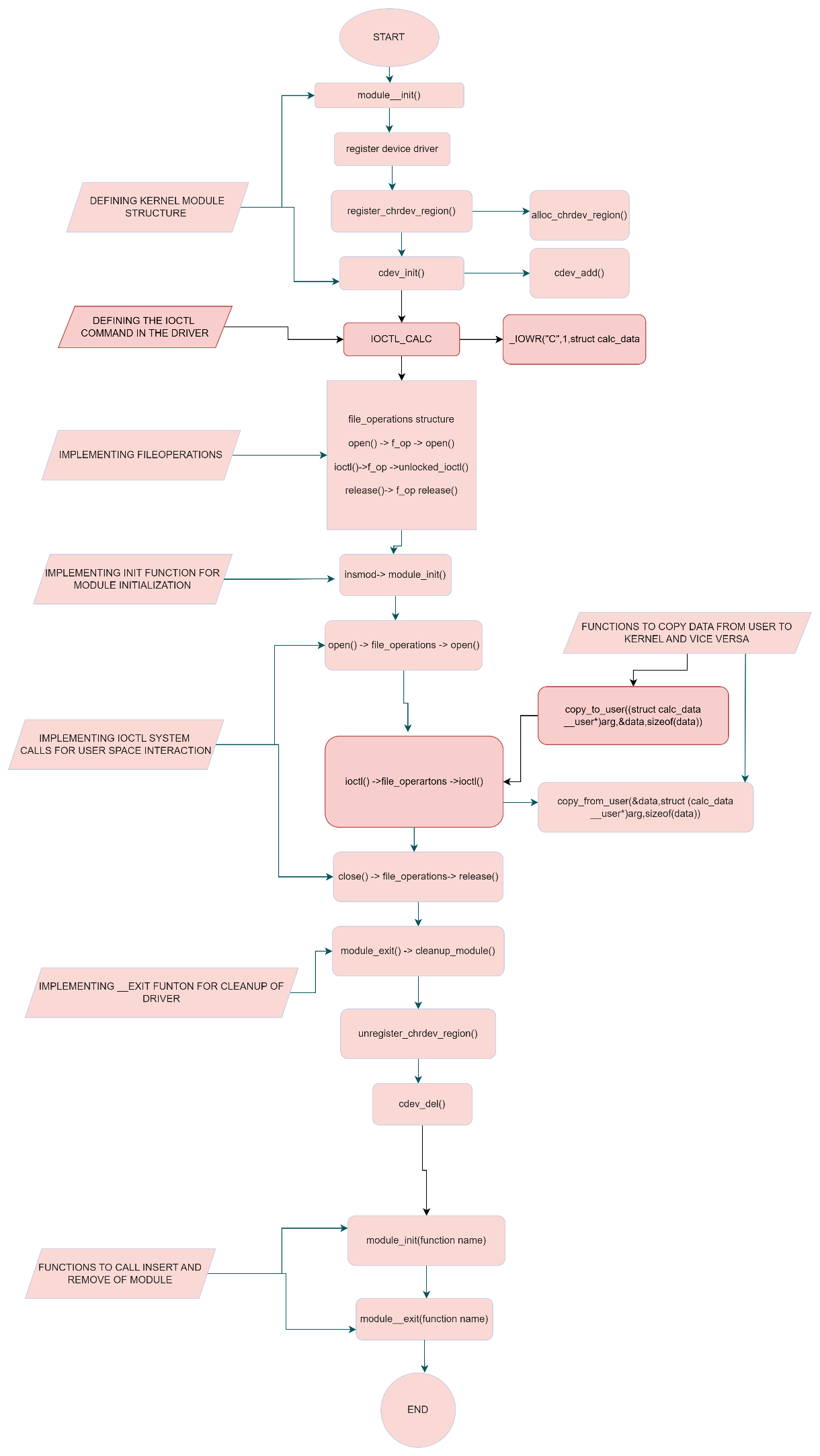
10. PRINTER: The hardware device that produces physical copies of documents and images on paper.

11. CD ROM: The hardware drive that reads data from CDs (Compact Discs), used for installing software, media playback, and data storage.

12. KEYBOARD: The hardware device used for inputting text and commands into the computer, typically consisting of keys for letters, numbers, and functions.

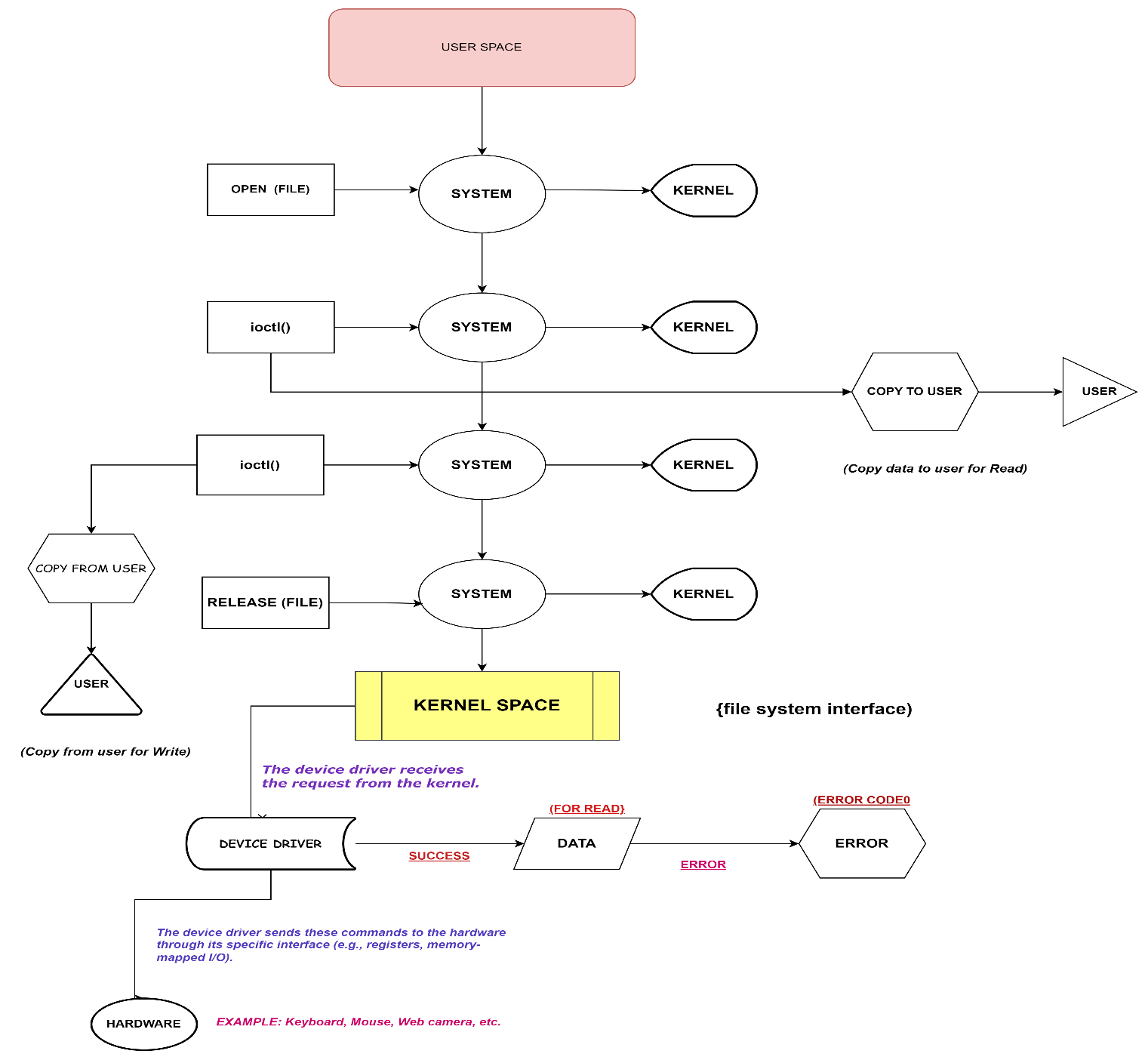
**3.1.6.3 Software Interfaces**

**Kernel space flow diagram:**



**3.1.6.4 Communications Interfaces:**

**User and kernel communication:**

****