DEVICE DRIVER: High Level Document

|  |  |
| --- | --- |
| 1. Document Information | |
| Document Name | Device Driver for a simulated character |
| Service | NA |
| Author | GROUP - A |
| Contributors | Dhriti – KERNEL SPACE CODE  Susmitha - BLOCK DIAGRAM  Sravani – USER SPACE CODE  Srinivas - BLOCK DIAGRAM  Badrinath - DOCUMENTATION |
| Issue Date | 14/06/2024 |

|  |  |  |  |
| --- | --- | --- | --- |
| 1. Document History | | |  |
| **Version** | **Date** | **Summary of change** | **Reference ID** |
| 1.0 | 14/06/2024 | Creating summary |  |
| 1.0 | 14/06/2024 | Implementing code |  |
| 1.0 | 14/06/2024 | Assumptions |  |
| 1.0 | 14/06/2024 | Block diagrams |  |
| 1.0 | 14/06/2024 | Drafting the document |  |

Contents

[1. Document Information 1](#_Toc398883969)

[2. Document History 1](#_Toc398883970)

[1 Design ownership 4](#_Toc398883971)

[1.1 Ownership Matrix 4](#_Toc398883972)

[2 Context 5](#_Toc398883973)

[2.1 Summary 5](#_Toc398883974)

[2.2 Service categorisation 5](#_Toc398883975)

[3 Links to documentation 5](#_Toc398883976)

[4 Drivers and objectives 6](#_Toc398883977)

[4.1 Drivers 6](#_Toc398883978)

[4.2 Requirements 6](#_Toc398883979)

[4.3 Objectives 6](#_Toc398883980)

[4.4 Impact of no action 7](#_Toc398883981)

[5 Assumptions 7](#_Toc398883982)

[6 Constraints 7](#_Toc398883983)

[7 Current state 8](#_Toc398883984)

[7.1 Logical summary 8](#_Toc398883985)

[7.2 Logical diagrams 8](#_Toc398883986)

[7.3 Physical summary 8](#_Toc398883987)

[7.4 Physical diagrams 8](#_Toc398883988)

[7.5 Interfaces 8](#_Toc398883989)

[8 Current limitations 9](#_Toc398883990)

[9 Target state logical 9](#_Toc398883991)

[9.1 Brief option/alternatives 9](#_Toc398883992)

[9.2 Changes to Logical design 9](#_Toc398883993)

[9.3 Known limitations 9](#_Toc398883994)

[9.4 Residual risk and limitations 10](#_Toc398883995)

[10 Design schematics 10](#_Toc398883996)

[10.1 Physical solution summary 10](#_Toc398883997)

[10.2 Physical solution diagrams 10](#_Toc398883998)

[10.3 Interfaces 10](#_Toc398883999)

[11 Service continuity capability 10](#_Toc398884000)

[11.1 Criticality 10](#_Toc398884001)

[11.2 Backup requirements and solution 11](#_Toc398884002)

[11.3 Anticipated failure states 11](#_Toc398884003)

[11.4 Single points of failure 11](#_Toc398884004)

[11.5 Symmetry 11](#_Toc398884005)

[12 Update methodology, frequency, cost 12](#_Toc398884006)

[13 Benefits of the solutions 12](#_Toc398884007)

[14 Residual risks/mitigations of the solution 12](#_Toc398884008)

[15 High level costs of solution (excluding project costs) 12](#_Toc398884009)

[15.1 Capital expenditure 12](#_Toc398884010)

[15.2 Capital generated operating expenditure 13](#_Toc398884011)

[15.3 Operating expenditure 13](#_Toc398884012)

[16 High level support structure, capabilities and responsibilities 13](#_Toc398884013)

[16.1 Service governance 13](#_Toc398884014)

[16.2 Operational support 13](#_Toc398884015)

[17 Decommissioning targets 14](#_Toc398884016)

[18 Block scheduling, lifecycle 14](#_Toc398884017)

[19 Potential future improvements 14](#_Toc398884018)

[20 Design review log and schedule 14](#_Toc398884019)

21 Kernel space program……………………………………………………………...19

21.1 Key components of kernel space program……………………………………24

22 User space program………………………………………………………………24

22.1 Key Components Of User Space Program……………………………………26

23 Make file…………………………………………………………………………….27

24 Switching to user……………………………………………………………………27

25 Initializing and Cleaning modules ………………………………………………..27

26 User-space make………………………………………………………………….. 27

27 Output……………………………………………………………………………….29

28 Clean up inserted modules ……………………………………………………….29

29 Output Reference ………………………………………………………………….30

**Prologue**

Device drivers are essential for enabling communication between the operating system and hardware components. These drivers serve as the bridge that allows software to interact with hardware, ensuring that devices function correctly and efficiently. The development and testing of device drivers are critical tasks, requiring careful design and robust testing environments to ensure reliability and performance.

This project involves creating a device driver for a simulated character device. A character device handles data as a stream of bytes, making it a fundamental component in many systems. By using a simulated device, we can thoroughly test the driver in a controlled environment without needing physical hardware.

The following sections will delve into the specifics of the design, including the assumptions, dependencies, and detailed descriptions of each module. Our aim is to create a reliable, high-performance driver that can be easily integrated into the operating system, facilitating seamless interaction between user applications and the simulated device.

# Design ownership

## Ownership Matrix

The following table tracks ownership and accountability of the design document and is used to assist with future sign off. Your design may need additional roles and they can be added.

|  |  |  |
| --- | --- | --- |
| **Role** | **Summary of content provided or responsibility** | **Named individual** |
| Design owner  (Essential) | Owns design document, versioning, content, syndication and handover to delivery, operations and governance teams. | NA |
| Project manager |  | NA |
| Service owner or proposed SO section  (Essential) | Monitors service change or service introduction. | NA |
| Service operations manager or proposed SOM section  (Essential) | Monitors operational change or operational introduction. | NA |
| Design mentor  (Essential) | Member of Design Advisory Group assigned to this design | NA |
| Business | Ensure business architecture/requirements are monitored, maintained and matched to the solution. | NA |
| Applications and data | Ensures applications and data architecture | NA |
| Technology | Ensure technology required for solution is in line with architectural principles, standard technologies | NA |

# Context:

This character device driver project is part of a initiative to enhance our system's capability to interact with hardware devices through custom drivers. It fits into the context of operating system development and embedded systems, impacting areas such as device communication, system stability, and performance.

**Nature of the Project**:

This project involves creating a new character device driver from scratch.

**Expected Outcomes:**

**The expected outcomes include:**

- A fully functional character device driver that supports basic file operations (open, read, write, close) and ioctl commands.

- Improved system functionality and flexibility in hardware device interaction.

- Comprehensive documentation detailing the development process, usage instructions, and testing results.

**High-Level Purpose:**

The primary purpose of developing this driver is to provide a learning experience in kernel module programming and device driver development. It also aims to enhance the system's capability to manage and interact with hardware devices efficiently.

**High-Level Implementation Approach:**

The project will be carried out in several steps:

1. Designing the device features and planning the module structure.

2. Implementing module initialization and clean up routines.

3. Adding file operations for interacting with the device.

4. Implementing ioctl commands for device control.

5. Testing the driver with a user-space application and documenting the entire process.

# Links to documentation

NA

# Drivers and objectives

## Drivers

The design helps everyone to understand how the driver will help in planning and using resources effectively.

## Requirements

To achieve the above objectives, the following requirements must be met:

**Functional Requirements:**

**Device Initialization and Cleanup:**

Implement init\_module to initialize the device and register the driver.

Implement cleanup\_module to unregister the driver and clean up resources.

**Basic Operations:**

Implement open and close operations to manage access to the device.

Implement read and write operations to handle data transfer between user applications and the device.

**IOCTL Support:**

Implement ioctl operations to support additional control commands

**Operating System:**

The driver must be compatible with the target operating system (e.g., Linux).

**Development Tools:**

Use appropriate development tools such as GCC, Make, and kernel headers.

**Testing Requirements:**

Develop a comprehensive test plan to validate the driver's functionality, performance, and reliability.

Use simulation tools to test the driver in various scenarios and load conditions.

## Objectives

The primary objective of this project is to develop a device driver for a simulated character device, ensuring efficient and reliable communication between the operating system and the simulated hardware. This driver will facilitate seamless read and write operations for user applications. The key objectives are as follows:

**Functionality:**

Develop a device driver that can handle basic operations (open, close, read, write) for a simulated character device.

Ensure the driver correctly interprets and processes IOCTL (Input/Output Control) commands.

**Performance:**

Optimize the driver for minimal latency in read/write operations.

Ensure efficient use of system resources, avoiding excessive CPU and memory usage.

**Security:**

Restrict access to the simulated character device to authorized user applications.

Protect data integrity during transmission between user applications and the device.

**Maintainability:**

Write clean, modular, and well-documented code to facilitate future maintenance and updates.

Ensure the driver conforms to coding standards and best practices for kernel development.

# Assumptions:

**Design assumptions**:

These assumptions provide a framework for designing and implementing the character device driver, guiding decisions related to functionality, robustness, and compatibility with the target environment.

|  |  |
| --- | --- |
| **Assumption** | **Assessment or impact on this design** |
| UCL will grow year by year at a rate of 10% | NA |
| ISD will shrink year by year at a rate of 1% in real terms | NA |
| UCL will be operating in a split data centre model with 1 half at a range of 30km | NA |
| **Device Characteristics:** | The simulated character device will mimic the behaviour of a typical hardware device, including characteristics such as input/output operations, buffer sizes, and control commands. |
| **Kernel Version Compatibility:** | The driver will be developed and tested for a specific range of kernel versions to ensure compatibility and stability. |
| **User-space Interaction:** | The driver will interact with user-space applications through standard file operations and potentially custom IOCTL commands for device control. |
| **Concurrency and Synchronization:** | The driver must handle concurrent access from multiple processes or threads safely, requiring appropriate locking mechanisms and synchronization primitives. |
| **Error Handling:** | The driver should implement robust error handling mechanisms to handle unexpected situations gracefully and provide meaningful error messages to user-space applications. |
| **Documentation and Testing:** | Comprehensive documentation will be provided to explain the usage, interfaces, and implementation details of the driver. Additionally, thorough testing will be conducted to ensure the driver's correctness and reliability. |
| **Performance Considerations:** | While efficiency is essential, initial development efforts will prioritize correctness and functionality. Performance optimizations may be considered in later iterations if necessary. |
| **Modularity and Extensibility:** | The driver design should be modular and extensible to facilitate future enhancements, such as adding support for additional features or integrating with other kernel subsystems. |
| **Resource Management:** | Proper management of system resources, such as memory allocation and device registration, will be implemented to avoid leaks and conflicts. |

# Constraints:

**Interface Compatibility:** The driver must follow instructions with the interface specifications established for character devices in the operating system kernel. This contains operations such as open, read, write, ioctl, and close.

**Resource Management:** Effective management of system resources, such as memory allocation and deallocation, is critical for avoiding memory leaks and system instability.

**Concurrency Control:** It involves creating systems to manage simultaneous access to a device by multiple users or processes. It ensures that data remains consistent and it prevents conflicts known as race conditions, where different actions interfere with each other's outcomes.

**Performance:** Efficient data flow between user and kernel spaces to reduce the time it takes for data to travel and to improve the rate at which data can be processed.

**Security:** Implementing appropriate security measures to prevent unauthorised access malicious usage of the device.

**Documentation:** Giving clear and complete information about the driver's usage, configuration, and internal workings to developers and users.

**Testing:** Thorough testing under a variety of scenarios is required to guarantee that the driver performs as intended and meets performance and reliability standards.

# Dependency and Management

# The driver may depend on other kernel modules or libraries, and appropriate measures will be taken to manage these dependencies and ensure smooth integration.

## Those dependent on this design

NA

# Current state

In some rare case there is no current state and this section can be skipped.

## Logical/architectural summary

NA

## Logical/architectural diagrams

NA

### Internal interfaces

NA

## Technology summary

A summary of the current hardware, software and or systems and processes. In most cases this will form the scope of physical items being replaces or removed by the design.

## Technology diagrams

NA

## External Interfaces

NA

# Current limitations:

Considering the provided scenario, here are some potential limitations that may impact the design and implementation of the character device driver:

|  |  |  |
| --- | --- | --- |
| **ID** | **Limitation name** | **Limitation summary and impact** |
| 1 | Hardware Abstraction | Simulated devices might not act exactly like real hardware, which could mess up testing and need extra checks. |
| 2 | Kernel version dependency | The driver might only work with certain kernel versions, which could make it hard for users with different versions to use it, needing constant updates for new kernels. |
| 3 | User-space environment | The driver's performance could change depending on what else is happening on the computer, like how busy it is or what other programs are running, which might make it less reliable or predictable. |
| 4 | Testing environment | Testing with simulated hardware might not show exactly how things will work in real life, so it's crucial to test in different situations to catch any problems that might come up. |
| 5 | Documentation and Support | Not enough help or instructions might make it tough for users to understand or fix problems with the driver, causing confusion and errors in how they use or maintain it. |
| 6 | Resource Constraints | The driver might use a lot of memory or processing power, which could slow it down or make it less scalable, especially when the system is busy or other parts of the system need those resources too. |
| 7 | Security Considerations | The driver could have security weaknesses, like being hacked or abused by malicious software, so it needs careful checks and protections to prevent problems like data breaches or unauthorized access. |
| 8 | Interoperability | Making sure the driver works well with other parts of the system might be tricky, as it might not fit in smoothly with existing software setups, needing careful attention to how it talks to other parts and shares information. |

# Target state logical

## Brief option/alternatives

NA

## Changes to Logical design

NA

## Known limitations

NA

Detail anything that will not be available in the target state logical model

- 1 year view

- 5 year view

SW: scale due to change of institute (link to strategy of UCL)

## Residual risk and limitations

NA

## Design schematics

NA

## Physical solution summary

NA

A summary of the target physical hardware, virtual machines, database instances, software and or systems and processes. It most cases this will form the scope of the new physical items being built, installed or purchased by the design.

You should refer to reference designs here from suppliers or SMEs.

## Physical solution diagrams

NA

Provide diagrammatic view of the new hardware or software. This may take the following forms.

1. Network, server and storage topologies.
2. Reference technical, application or data architectures from suppliers.
3. Database data structures.

## Interfaces

Placing your system at the centre draw all the interfaces the new system has with systems outside the scope of this design. Examples include;

- Management information systems (myFinance, myView, etc)

- Server infrastructure systems (AD, DNS, email, logging, monitoring)

- ETLs from other databases or information systems

- Storage

For each interfaces please detail the design of those interfaces here.

|  |  |  |  |
| --- | --- | --- | --- |
| **Interface summary** | **Summary of work required due to this change** | **Effort** | **Owner** |
| NA | NA | NA | NA |

# Service continuity capability

## Criticality

NA

What criticality level is this service?

- Impact of a days outage to the service

- Impact of a total loss of the service

## Backup requirements and solution

NA

Explain what elements of the solution described in the physical summary and diagrams require backup.

- How will this backup be performed

- How often with this backup occur

- How many versions will be retained

- How will data be recovered in the event of a full loss, partial loss

- Is a business continuity plan required or available for this service

## Anticipated failure states

Please detail how your solution and service is expected to function and is anticipated to be effected by the following failure states

|  |  |  |
| --- | --- | --- |
| **Failure mode** | **Required function** | **High level process to achieve requirement** |
| Loss of single site due to planned maintenance | NA | NA |
| Loss of single site due to unplanned maintenance | NA | NA |
| Loss of data either partial or complete (malicious act or corruptions) | NA | NA |
| During component updates (please add for each component) | NA | NA |

## Single points of failure

NA

Please detail any known single points of failure in the system.

## Symmetry

NA

Please detail any asymmetry in the system between primary and secondary sites if applicable.

# Update methodology, frequency, cost

NA

For each major element of the systems please detail the high level process for keeping the system up to date.

- Frequency

- Type

- Responsibility

- Anticipated costs

-

# Benefits (high level)

* Improved Device Interaction : The new driver will enable the system to interact with the simulated hardware device through custom file operations and control commands, enhancing overall system functionality.
* Increased Flexibility : Adding support for a new device type increases the system's flexibility in handling various hardware devices, making it more adaptable to different use cases and requirements.
* Efficient Data Handling : Character device drivers handle data as a continuous stream of bytes, making them ideal for devices like serial ports, keyboards, and mice, which transmit data in a sequential manner.
* Simplified Device Management : They provide a straightforward way to manage character-oriented devices, facilitating easier reading and writing of data without the need for buffering or block management.
* Direct Access to Hardware : Character device drivers often allow direct access to hardware devices, enabling precise control over device operations and configurations.

# Residual risks

The following risks will remain after the successful delivery of the design

|  |  |  |
| --- | --- | --- |
| Risk | Type | Summary |
| 1 | Concurrency issue | The program might face concurrency issues, such as race conditions, when handling multiple IOCTL requests simultaneously |
| 2 | Wrong calculations | The program might produce wrong calculations because of coding mistakes. |
| 3 | Data corruption | Data might get corrupted during ioctl operations due to unforeseen issues. |

# High level costs of solution (excluding project costs)

NA

Please use the logical and physical summary to help produce a full hardware/software/facilities costs

## Capital expenditure

NA

## Capital generated operating expenditure

NA

## Operating expenditure

NA

# High level support structure, capabilities and responsibilities

Developing an IOCTL (Input/Output Control) calculation program involves several key components:

Device Driver

IOCTL Interface

User Application

Kernel Module

## Service governance

NA

## Operational support

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Element** | **Capabilities required** | **Anticipated load** | **Current capability and capacity** | **Training or additional resource required** |
| NA | NA | NA | NA | NA |

# Decommissioning targets

- Services - NA

- Infrastructure -NA

From the current state logical and physical statement please detail which technologies will be decommissioned due to this solution.

**Examples of elements that will be decommissioned:**

Physical and virtual servers - NA

Software and hardware licensing and maintenance agreements

Processes - NA

Webpages - NA

|  |  |  |  |
| --- | --- | --- | --- |
| **Element** | **Replaced by** | **Type of decommission activity** | **Anticipated decommission timescale (measured from point of this designs delivery)** |
| NA | NA | NA | NA |

# Block scheduling, lifecycle

NA

# Potential future improvements

NA

# Design review log and schedule

NA

# 21 KERNEL SPACE PROGRAM :

#include <linux/init.h> // Required for module initialization and exit macros

#include <linux/module.h> // Required for module-related macros and functions

#include <linux/fs.h> // Required for file operations structure

#include <linux/uaccess.h> // Required for copy\_to\_user and copy\_from\_user functions

#include <linux/cdev.h> // Required for character device registration

#define DEVICE\_NAME "my\_calculator\_device" // Define the name of the device

#define IOCTL\_MAGIC 'c' // Define a magic number for ioctl

#define IOCTL\_CALC \_IOWR(IOCTL\_MAGIC, 1, struct calc\_data) // Define ioctl command with magic number and command number

// Define a structure to hold the calculation data

struct calc\_data {

int num1; // First operand

int num2; // Second operand

char operation; // Operation to perform (+, -, \*, /)

int result; // Result of the operation

};

// Function to handle device open

static int device\_open(struct inode \*inode, struct file \*file) {

printk(KERN\_INFO "Calculator device opened\n"); // Log message

return 0; // Return success

}

// Function to handle device close

static int device\_release(struct inode \*inode, struct file \*file) {

printk(KERN\_INFO "Calculator device closed\n"); // Log message

return 0; // Return success

}

// Function to handle ioctl calls

static long device\_ioctl(struct file \*file, unsigned int cmd, unsigned long arg) {

struct calc\_data data; // Declare a structure to hold calculation data

switch (cmd) { // Check which ioctl command was received

case IOCTL\_CALC: // If the command is IOCTL\_CALC

if (copy\_from\_user(&data, (struct calc\_data \_\_user \*)arg, sizeof(data))) { // Copy data from user space

return -EFAULT; // Return error if copy fails

}

// Perform the requested operation

switch (data.operation) {

case '+':

data.result = data.num1 + data.num2; // Addition

break;

case '-':

data.result = data.num1 - data.num2; // Subtraction

break;

case '\*':

data.result = data.num1 \* data.num2; // Multiplication

break;

case '/':

if (data.num2 == 0) { // Check for division by zero

return -EINVAL; // Return invalid argument error

}

data.result = data.num1 / data.num2; // Division

break;

default:

return -EINVAL; // Return invalid argument error for unknown operations

}

if (copy\_to\_user((struct calc\_data \_\_user \*)arg, &data, sizeof(data))) { // Copy result back to user space

return -EFAULT; // Return error if copy fails

}

// Log the operation performed

printk(KERN\_INFO "Operation %c: %d %c %d = %d\n", data.operation, data.num1, data.operation, data.num2, data.result);

break;

default:

return -ENOTTY; // Return "inappropriate ioctl" error for unknown commands

}

return 0; // Return success

}

// Define the file operations structure

static struct file\_operations fops = {

.owner = THIS\_MODULE, // Set the owner of the module

.open = device\_open, // Set the open function

.release = device\_release, // Set the release function

.unlocked\_ioctl = device\_ioctl, // Set the ioctl function

};

static int major\_number; // Declare a variable to hold the major number

static struct cdev my\_cdev; // Declare a character device structure

// Module initialization function

static int \_\_init my\_calculator\_init(void) {

major\_number = register\_chrdev(0, DEVICE\_NAME, &fops); // Register the character device and get the major number

if (major\_number < 0) { // Check if registration failed

printk(KERN\_ALERT "Registering char device failed with %d\n", major\_number); // Log error message

return major\_number; // Return the error code

}

cdev\_init(&my\_cdev, &fops); // Initialize the character device with file operations

my\_cdev.owner = THIS\_MODULE; // Set the owner of the character device

if (cdev\_add(&my\_cdev, MKDEV(major\_number, 0), 1)) { // Add the character device to the system

unregister\_chrdev(major\_number, DEVICE\_NAME); // Unregister the character device if adding fails

printk(KERN\_ALERT "Adding cdev failed\n"); // Log error message

return -1; // Return error code

}

printk(KERN\_INFO "Calculator device registered with major number %d\n", major\_number); // Log success message

return 0; // Return success

}

// Module exit function

static void \_\_exit my\_calculator\_exit(void) {

cdev\_del(&my\_cdev); // Delete the character device

unregister\_chrdev(major\_number, DEVICE\_NAME); // Unregister the character device

printk(KERN\_INFO "Calculator device unregistered\n"); // Log success message

}

module\_init(my\_calculator\_init); // Set the initialization function

module\_exit(my\_calculator\_exit); // Set the exit function

MODULE\_LICENSE("GPL"); // Set the license of the module

MODULE\_AUTHOR("Group A"); // Set the author of the module

MODULE\_DESCRIPTION("A simple calculator char driver with ioctl"); // Set the description of the module

MODULE\_VERSION("0.1"); // Set the version of the module

21.1 Key Components Of Kernel Space Program :

**Includes and Macros:**

* #include <linux/init.h>, #include <linux/module.h>, etc. include the necessary Linux kernel headers.
* #define DEVICE\_NAME "my\_calculator\_device" sets the name for the device.
* #define IOCTL\_MAGIC 'c' and #define IOCTL\_CALC \_IOWR(IOCTL\_MAGIC, 1, struct calc\_data) define the ioctl command for calculation.
* Data Structure:
* struct calc\_data contains the fields needed for performing a calculation (num1, num2, operation, and result).
* File Operations:
* device\_open and device\_release handle opening and closing the device.
* device\_ioctl handles the ioctl calls, performing calculations based on the received command and data.

22 User-Space program:

#include <stdio.h> // For standard input/output functions

#include <stdlib.h> // For standard library functions, including EXIT\_FAILURE and EXIT\_SUCCESS

#include <fcntl.h> // For file control options (open function)

#include <unistd.h> // For close function

#include <sys/ioctl.h> // For ioctl function

#define DEVICE\_PATH "/dev/my\_calculator\_device" // Path to the calculator device

#define IOCTL\_MAGIC 'c' // Magic number for ioctl

#define IOCTL\_CALC \_IOWR(IOCTL\_MAGIC, 1, struct calc\_data) // ioctl command definition

// Define a structure to hold the calculation data

struct calc\_data {

int num1; // First operand

int num2; // Second operand

char operation; // Operation to perform (+, -, \*, /)

int result; // Result of the operation

};

int main() {

int fd; // File descriptor for the device

struct calc\_data data; // Structure to hold user input and result

// Open the device for reading and writing

fd = open(DEVICE\_PATH, O\_RDWR);

if (fd < 0) { // Check if the device failed to open

perror("Failed to open the device"); // Print error message

return EXIT\_FAILURE; // Return failure status

}

// Prompt user for input and read the values

printf("Enter first number: ");

scanf("%d", &data.num1); // Read first number

printf("Enter second number: ");

scanf("%d", &data.num2); // Read second number

printf("Enter operation (+, -, \*, /): ");

scanf(" %c", &data.operation); // Read operation (note the space to handle newline)

// Perform ioctl to send data to the kernel module and get the result

if (ioctl(fd, IOCTL\_CALC, &data) < 0) { // Check if ioctl failed

perror("Failed to execute ioctl"); // Print error message

close(fd); // Close the device

return EXIT\_FAILURE; // Return failure status

}

// Print the result of the calculation

printf("Result: %d %c %d = %d\n", data.num1, data.operation, data.num2, data.result);

close(fd); // Close the device

return EXIT\_SUCCESS; // Return success status

}

22.1 Key Components Of User Space Program :

**Includes headers and Defines:**

* + The headers included provide functions and macros necessary for standard input/output, file operations, and ioctl handling.
  + DEVICE\_PATH defines the path to the calculator device.
  + IOCTL\_MAGIC and IOCTL\_CALC define the magic number and ioctl command, matching those in the kernel module.

**Structure Definition:**

* + struct calc\_data is defined to hold the operands, operation, and result of the calculation. This structure matches the one defined in the kernel module.

**Main Function:**

* + Opens the device using open. If the device fails to open, it prints an error message and exits with failure status.
  + Prompts the user to enter the first number, second number, and operation. The scanf function is used to read these values.
  + The ioctl function is called to send the calculation request to the kernel module and receive the result. If the ioctl call fails, it prints an error message, closes the device, and exits with failure status.
  + If the ioctl call succeeds, it prints the result of the calculation.
  + Finally, it closes the device and exits with success status.

23 Make file :

* + obj-m += device\_ioctl.o
  + all:
    - make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
  + clean:
    - make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
* This obj-m += device\_ioctl.o line indicates that the kernel module to be built is named device\_ioctl.o. The obj-m variable is used by the kernel build system to determine which object files to build.
* This rule, named all, specifies how to build the module. It invokes make with the -C option to change directory to the kernel source directory (specified by /lib/modules/$(shell uname - r)/build) and passes the M parameter to that Makefile. This instructs the kernel build system to use the current directory ($(PWD)) to find the source files for the module. The modules target tells make to build all kernel modules specified in the Makefile.
* This rule, named clean, specifies how to clean up the build artifacts. Similar to the all rule, it uses make to change directory to the kernel source directory and then executes the clean target, which removes all built files and temporary build artifacts.
* After make command we create string\_device.ko which is kernel object file that is used to build kernel understandable code.
* These .ko module file is inserted into kernel by usind insmod command.

24 Switching to user :

* + By using sudo su command we able to switch into kernel space and we need to enter password to switch user space to kernel.



25 Initializing and Cleaning modules :

**Inserting module into kernel :**

* + Sudo insmod device\_ioctl.ko // inserts module into kernel
  + Sudo dmesg | tail //”display messagae” it is used to print the message from kernel log.

**Creating special file using mknod :**

* + The character special file is used by user-space with kernel space to get information from hardware.
  + Sudo mknod /dev/my\_calculator\_device c 241 0
  + sudo: This command is used to execute the subsequent command with superuser privileges.
  + mknod: This is a command used to create device nodes (special files) in Unix-like operating systems.
  + /dev/my\_calculator\_device: This specifies the path and name of the device node to be created. In this case, it creates a device node named " my\_calculator\_device " in the /dev directory.
  + c: This flag indicates that the device node to be created is a character device
  + 241: This is the major number assigned to the device. Major numbers are used by the kernel to identify device drivers.
  + 0: This is the minor number assigned to the device. Minor numbers are used to identify specific devices managed by the same driver.
  + After executing this command, the device node

/dev/ my\_calculator\_device will be created,

allowing user space programs to interact with the corresponding kernel module.

26 User-space make :

* By using make ioctluser command we create object file for user-space that makes understand to operating system.
* By using ./ioctluser command the user and kernel work with each other by transfering data. Here ./ioctluser is executable file.

27 OUTPUT :

* In the output we will be ask to enter the
* Enter first number : 3
* Enter second number : 5
* Enter operation (+ , - , \*,/) : +
* Result : 3+5 = 8

28 Clean up inserted modules :

➢ By using rmmod command we can exit or cleans the inserted module which makes kernel environment free of conflicts while working with other files.

29 Output Reference :

