Advanced Operating System Architecture

Lab 2: Simple Kernel Module Development

Computer Science Department

Academic Year 2023-2024

Objectives

- Understand Linux kernel module structure
- Learn module initialization and cleanup mechanisms
- Implement a basic "Hello World" kernel module
- Explore module parameter passing
- Practice module loading and unloading techniques
- Implement advanced kernel programming tasks

Theoretical Background

Kernel Modules

Kernel modules are dynamically loaded/unloaded pieces of code that extend kernel functionality without requiring a kernel recompilation. Key characteristics include:

- Compiled separately from the kernel
- Can be inserted and removed at runtime
- Operate in kernel space with full system privileges

1 Practical Implementation

1.1 Prerequisite Setup

Before starting, ensure that your development environment is correctly configured:

Terminal Commands

```
sudo apt-get install linux-headers-$(uname -r)
sudo apt-get install gcc make
```

1.2 Code Example: Hello World Module

Here is a simple "Hello World" kernel module:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

MODULE_LICENSE("GPL");
```

```
MODULE_AUTHOR("Student Name");
MODULE_DESCRIPTION("Simple Hello World Kernel Module");
MODULE_VERSION("0.1");

static int __init hello_init(void) {
    printk(KERN_INFO "Hello, Kernel World!\n");
    return 0;
}

static void __exit hello_exit(void) {
    printk(KERN_INFO "Goodbye, Kernel World!\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

Listing 1: Hello World Kernel Module

1.3 Makefile Configuration

The following Makefile compiles the kernel module:

```
obj-m += hello.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
```

Listing 2: Makefile for Kernel Module

2 Advanced Task: Working with Linked Lists

For this advanced task, your goal is to create a kernel module that dynamically manages a linked list. The module should:

- Create and initialize an empty linked list.
- Dynamically allocate memory for each node.
- Add nodes to the list with integer data values.
- Traverse the list and print the data to the kernel log.
- Properly free all allocated memory during cleanup.

Hints for Implementation

- Use the 'struct list_head' provided by the Linux kernel to define your linked list.
- Explore functions such as 'list_add_tail()' for adding nodes and 'list_for_each_entry_safe()' for traversing and deleting nodes.
- Allocate memory for each node using 'kmalloc()' and free it with 'kfree()'.
- Log messages using 'printk()' to verify that the list is being manipulated correctly.

Key Commands for Testing

Use the following commands to manage your kernel module:

Terminal Commands

```
make
sudo insmod yourmodule.ko
dmesg | tail
sudo rmmod yourmodule
```

3 Assessment Criteria

- Correct module structure
- Successful compilation
- Proper loading and unloading
- Meaningful kernel log messages
- Advanced implementation (linked list manipulation, parameter handling)