Linux 设备驱动程序设计与实现

📌 **实验报告**

- **实验课题作业 6：Linux设备驱动程序设计与实现**
- **姓名**：梁嘉嘉
- **学号**：23125240
- **课程**：高级操作系统
- **提交日期**：2024/11/04

https://www.bilibili.com/video/BV1m24y1A7Fi/?spm_id_from=333.337.search-card.all.click&vd_source=57ecc3132bb0c3…

设备驱动程序与文件系统 (Linux 设备驱动；目录管理 API) [南京大学2023操作系统-P27] (蒋炎岩)_哔哩哔哩_bilibili

2023 南京大学《操作系统:设计与实现》课程主页(含讲义):https://jyywiki.cn/OS/2023/, 视频播放量 25540、弹幕量 77、点赞…

https://jyywiki.cn/OS/2023/build/lect27.ipynb.html

27. 设备驱动程序与文件系统 ¶

操作系统通过设备驱动程序，将硬件设备五花八门的 “寄存器协议” 抽象成了统一的接口，并且进而成为目录树的一部分。UNIX “一切皆文件” 的设计使应用程序可以自由访问各类设备，但带来的代价则是文件系统部分巨大的历史包袱和移植性问题:操作…

本实验参考了**《设备驱动程序与文件系统 (Linux 设备驱动；目录管理 API) [南京大学2023操作系统-P27] (蒋炎岩)》**中的相关内容，对我在设备驱动程序设计与实现过程中的理解和实现提供了重要的指导，在此表示感谢。

1. 实验目的

本次实验旨在通过编写简单的 Linux 设备驱动程序，加深对设备驱动结构和内核模块机制的理解。通过实现字符设备的读写操作，我掌握了内核与用户空间的交互方法，并学会了如何加载和卸载内核模块。此次实验不仅提高了我的编程和调试技能，也为后续更复杂的驱动开发奠定了基础。

2. 实验环境

硬件环境：

- **虚拟机平台**：VMware Workstation Pro 17
- **处理器**：AMD Ryzen 7 8845H with Radeon 780M Graphics

- **架构**：x86_64
- **CPU 核心数**：2 个核心
- **每个核心的线程数**：4
- **L1 缓存**：64 KiB (2 实例)
- **L2 缓存**：2 MiB (2 实例)
- **L3 缓存**：32 MiB (2 实例)

- **内存**：4 GiB
  - **可用内存**：2.6 GiB
  - **交换分区**：3.7 GiB
- **存储**：
  - **硬盘大小**：80 GB
  - **分区**：单一分区 `/` 挂载在 `sda2`

软件环境：

- **操作系统**：Ubuntu 24.04.1 LTS (noble)
- **内核版本**：6.8.0-45-generic
- **编译工具**：
  - **GCC 版本**：13.2.0 (Ubuntu)
  - **Make 版本**：4.3 (GNU Make)
  - **Git 版本**：2.43.0

3. 实验步骤

3.1 编写代码

- `driver.c`：实现字符设备驱动的功能。

```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include <linux/fs.h>
#include <linux/uaccess.h>

#define MAX_DEV 2

```

```c
static int dev_major = 0;
static struct class *ljj_class = NULL;

static ssize_t ljj_read(struct file *, char __user *, size_t, loff_t *);
static ssize_t ljj_write(struct file *, const char __user *, size_t, loff_t *);

static struct file_operations fops = {
    .owner = THIS_MODULE,
    .read = ljj_read,
    .write = ljj_write,
};

struct nuke {
    struct cdev cdev;
} devs[MAX_DEV];

static int __init ljj_init(void) {
    dev_t dev;
    int i;

    // allocate device range
    alloc_chrdev_region(&dev, 0, MAX_DEV, "nuke");

    // create device major number
    dev_major = MAJOR(dev);

    // create class
    ljj_class = class_create("nuke");

    for (i = 0; i < MAX_DEV; i++) {
        // register device
        cdev_init(&devs[i].cdev, &fops);
        devs[i].cdev.owner = THIS_MODULE;
        cdev_add(&devs[i].cdev, MKDEV(dev_major, i), 1);
        device_create(ljj_class, NULL, MKDEV(dev_major, i), NULL, "nuke%d", i);
    }
    return 0;
}

static void __exit ljj_exit(void) {
    int i;
    for (i = 0; i < MAX_DEV; i++) {
        device_destroy(ljj_class, MKDEV(dev_major, i));
        cdev_del(&devs[i].cdev);
    }
    class_destroy(ljj_class);
    unregister_chrdev_region(MKDEV(dev_major, 0), MAX_DEV);
```

```
}

static ssize_t ljj_read(struct file *file, char __user *buf, size_t count,
loff_t *offset) {
    if (*offset != 0) {
        return 0;
    } else {
        const char *data = "This is dangerous!\n";
        size_t datalen = strlen(data);
        if (count > datalen) {
            count = datalen;
        }
        if (copy_to_user(buf, data, count)) {
            return -EFAULT;
        }
        *offset += count;
        return count;
    }
}

static ssize_t ljj_write(struct file *file, const char __user *buf, size_t
count, loff_t *offset) {
    char databuf[4] = "\0\0\0\0";
    if (count > 4) {
        count = 4;
    }

    if (copy_from_user(databuf, buf, count)) {
        return -EFAULT;  // Handle the error
    }
    if (strncmp(databuf, "\x01\x14\x05\x14", 4) == 0) {
        const char *EXPLODE[] = {
            // Explosion ASCII Art
            "                                    ",
            "                                    ",
            "                                    ",
            "                                    ",
            "                                    ",
            "                                    ",
            "                                    ",
            "                                    ",
            "                                    ",
            "                                    ",
            "                                    ",
            "                                   ",
            "                                   ",
            "                                  ",
            "                                  ",
```

```
            "                                   ",
        };
        int i;

        for (i = 0; i < sizeof(EXPLODE) / sizeof(EXPLODE[0]); i++) {
            printk("\033[01;31m%s\033[0m\n", EXPLODE[i]);
        }
    } else {
        printk("nuke: incorrect secret, cannot launch.\n");
    }
    return count;
}

module_init(ljj_init);
module_exit(ljj_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("ljj");
```

- `user.c`：测试驱动的读写操作。

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

#define SECRET "\x01\x14\x05\x14"

int main()
{
  int fd = open("/dev/nuke0", O_WRONLY);
  if (fd > 0)
  {
    write(fd, SECRET, sizeof(SECRET) - 1);
    close(fd);
  }
  else
  {
    perror("launcher");
  }
}
```

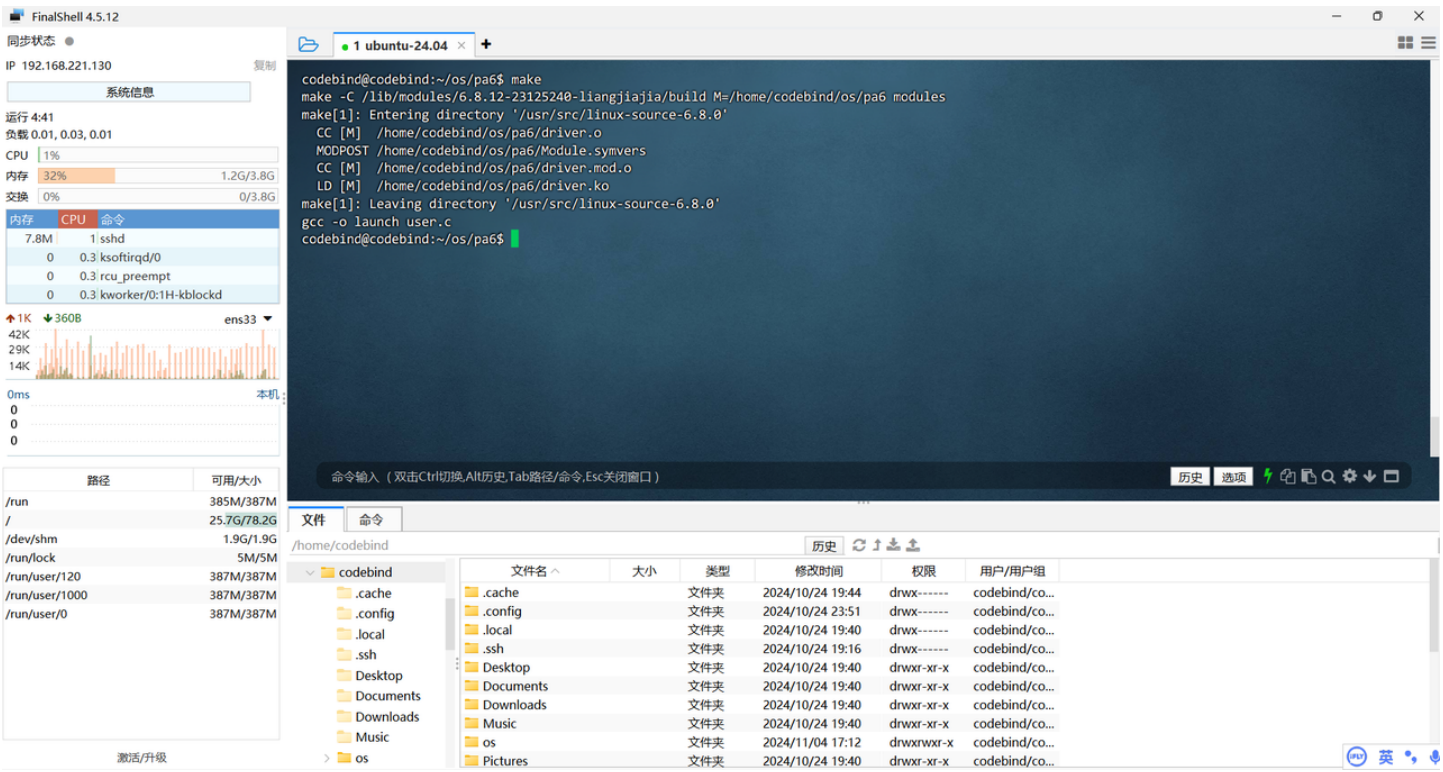
- `Makefile`：编译驱动和测试程序。

```
# Set the name of the kernel module
```

```
obj-m := driver.o

# Set the path to the kernel source directory
KERNEL_SRC := /lib/modules/$(shell uname -r)/build

# Set the current working directory
PWD := $(shell pwd)

# Default target to build the kernel module
all:
    $(MAKE) -C $(KERNEL_SRC) M=$(PWD) modules
    gcc -o launch user.c

# Target to clean the build artifacts
clean:
    $(MAKE) -C $(KERNEL_SRC) M=$(PWD) clean

.PHONY: all clean
```

3.2 编译代码

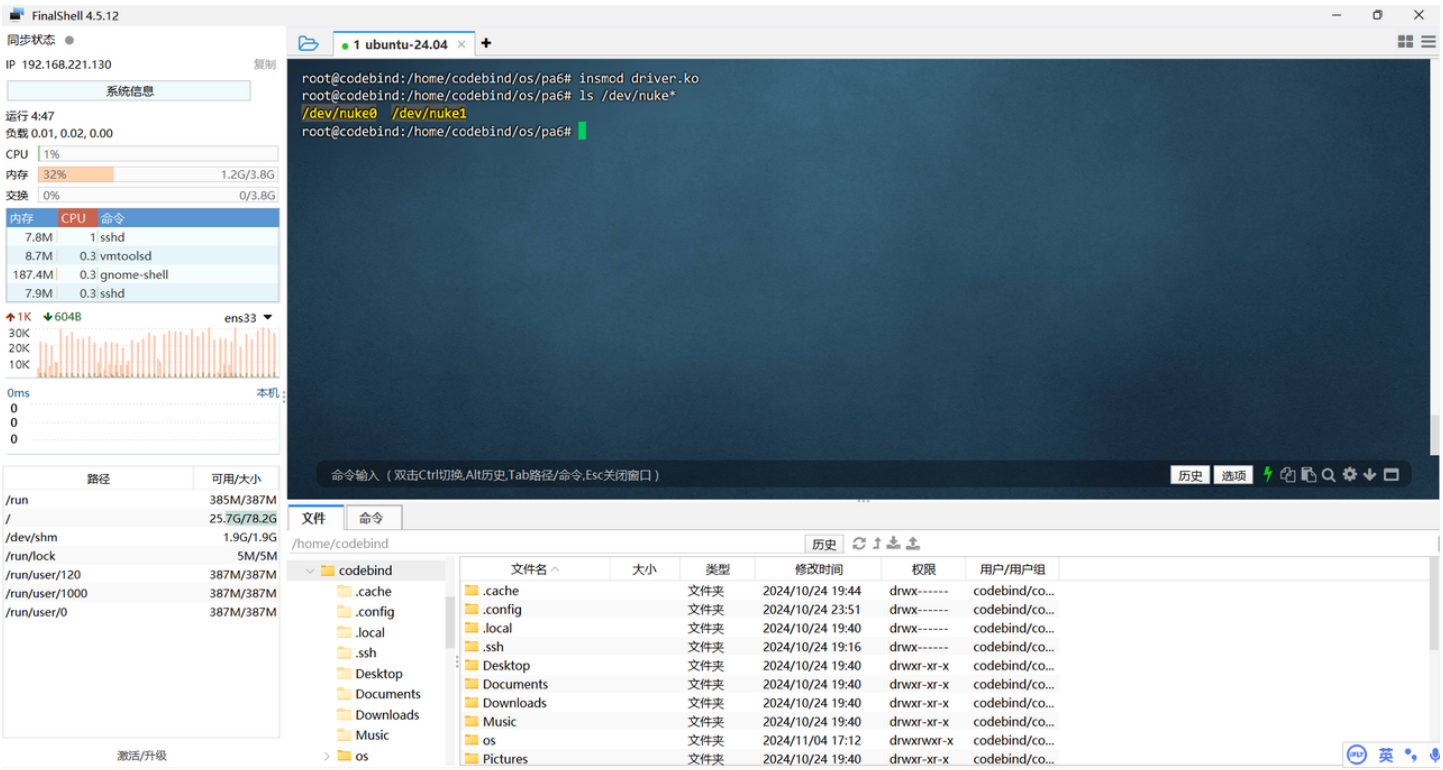
```
$ make
```



3.3 加载驱动

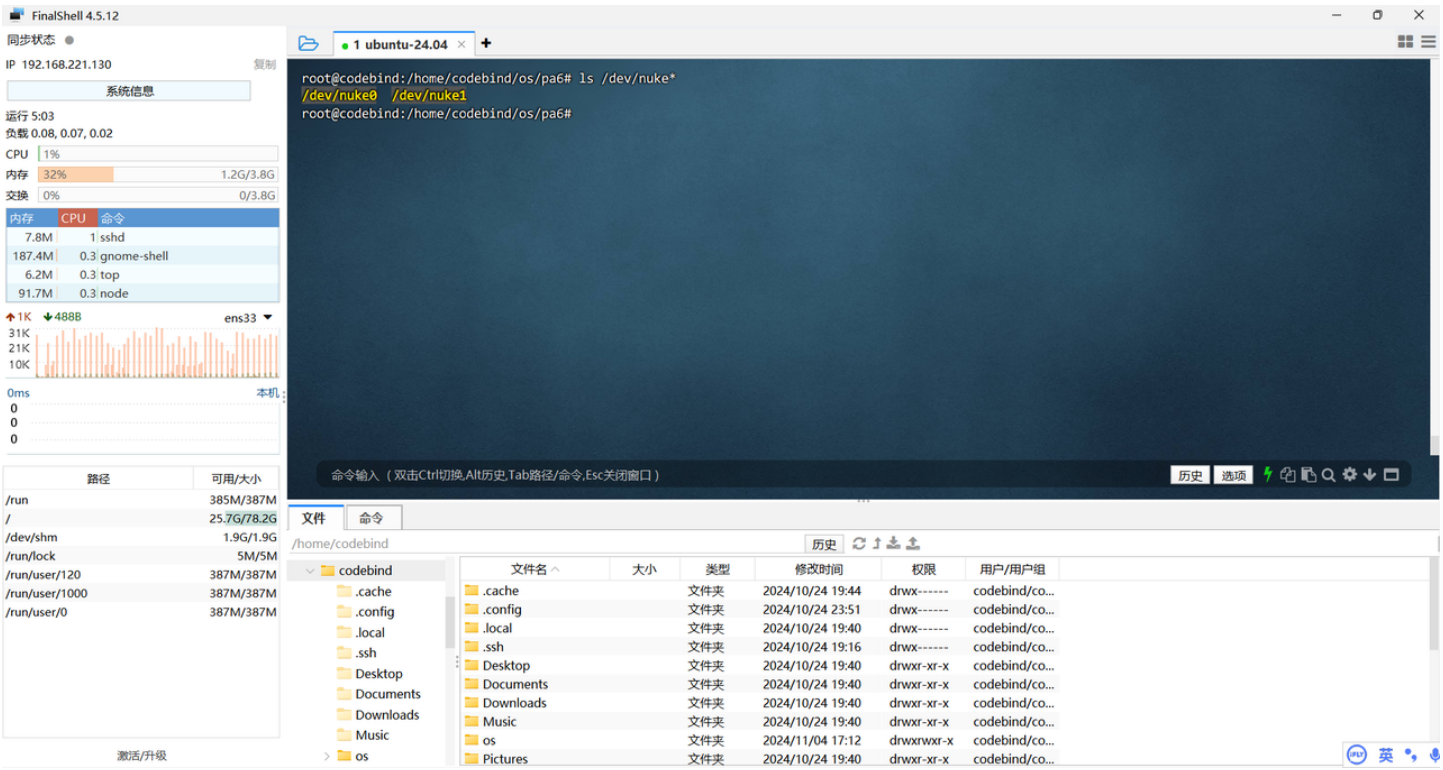
切换到 root 用户：

```
$ insmod driver.ko
```



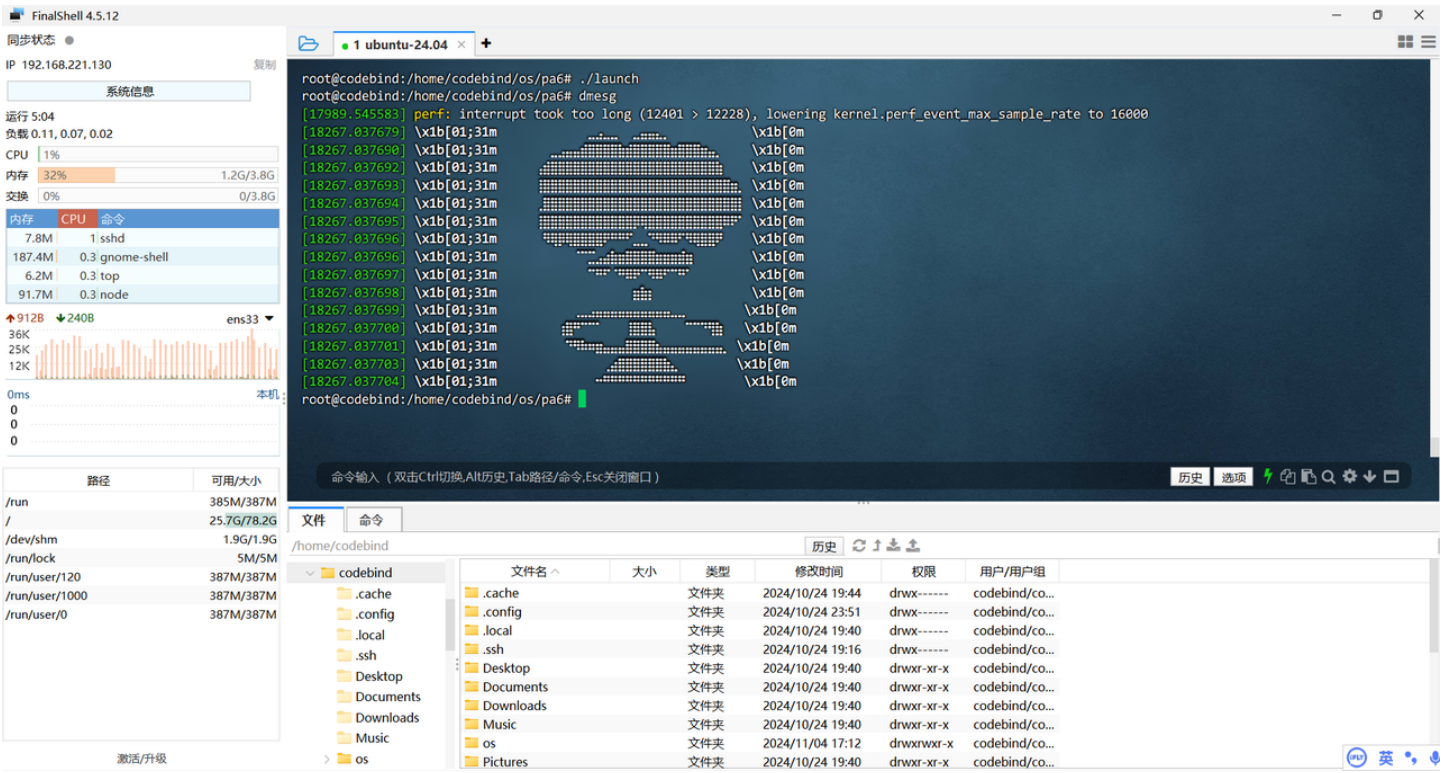
3.4 查看设备

```
$ ls /dev/nuke*
```



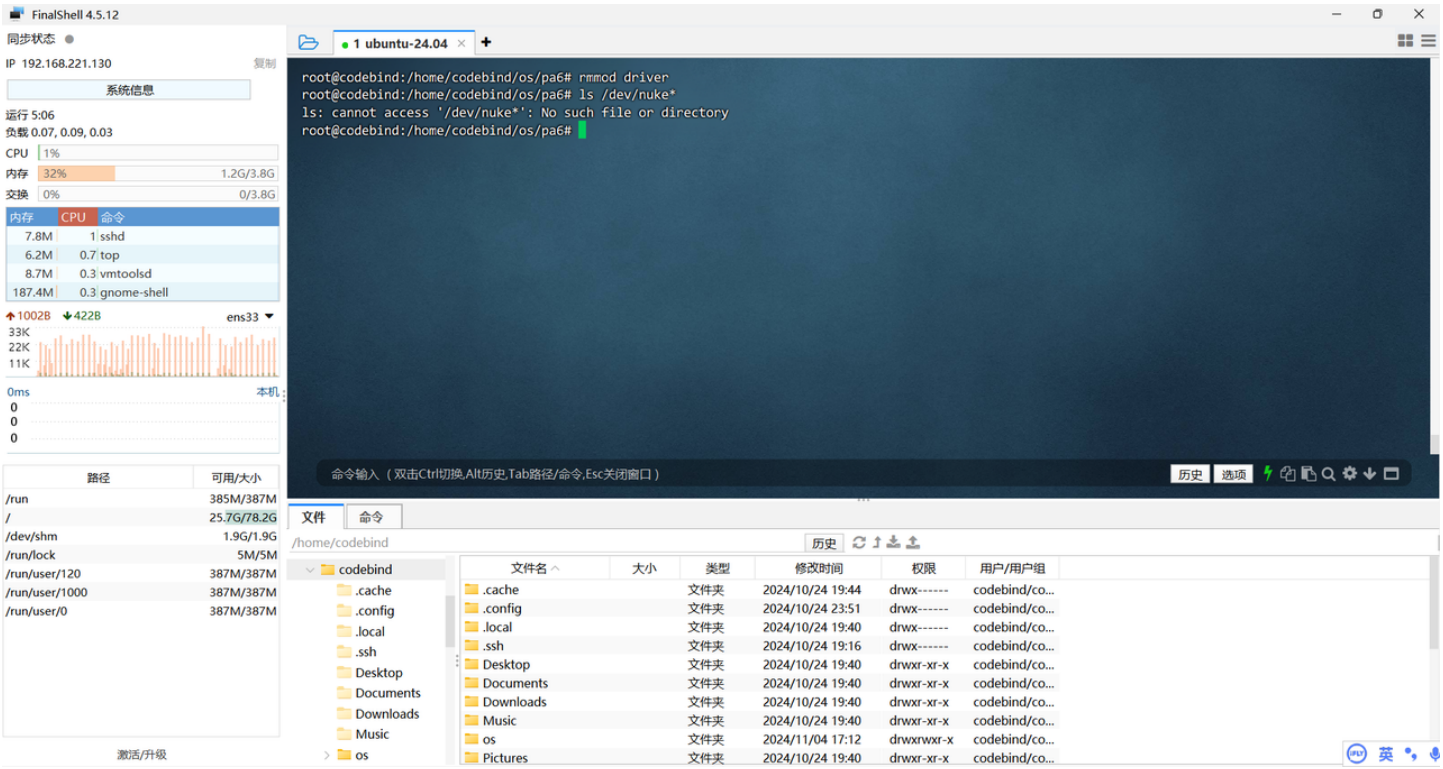
3.5 测试程序

```
$ ./launch
$ dmesg
```



3.6 卸载驱动

```
$ rmmod driver
$ ls /dev/nuke*
```



4. 疑难解惑与经验教训

无

5. 结论与体会

通过本次实验，我深入理解了 Linux 设备驱动的基本原理和开发流程，掌握了如何编写、加载和卸载内核模块的技能，并体会到调试能力在驱动开发中的重要性。实验中遇到的问题让我认识到不断学习和解决问题的必要性，这对我后续的驱动开发有很大帮助。