操作系统实验报告

班号：10012209　　姓名：梁桐　　学号：2023370018

实验日期：2024.6.14　　实验名称：设备管理

一、实验目的

任务一：

参考给出的内核源码中的 drivers/usb/usb_detect.c 文件，编写一个 USB 探测驱动程序，能够实现以下基本功能：

（1）在插入 U 盘时能够探测到；

（2）在拔出 U 盘时能够探测到；

加载、卸载模块并查看模块打印信息。

任务二

编写内核模块测试硬盘的读写速率，加载、卸载模块并查看模块打印信息。

使用用户态下 iozone 工具测试硬盘的读写速率，注意：测试范围需包含与内核模块读写相同的文件大小和块大小(例如都将写入文件大小设置为 512M,每次写入数据块大小为 1kb)。

对比用户态和内核态下测试的读写速率，并作分析。

二、实验要求

任务一：

正确编写满足功能的源文件，正确编译。

正常加载、卸载内核模块；且内核模块功能满足任务所述。

提交相关源码与运行截图。

任务二：

正确编写满足功能的源文件，正确编译、加载、卸载内核模块；且内核模块功能满足任务所述。

正确安装使用 iozone 工具测试硬盘读写速率。（可能会比较耗时）

提交相关源码、运行截图与最终分析。

三、实验过程及结果

任务一：

插入 U 盘前 lsusb

```
[metap@localhost ~]$ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 004: ID 0e0f:0008 VMware, Inc. Virtual Bluetooth Adapter
Bus 002 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub
Bus 002 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

插入 U 盘后 lsusb

```
[metap@localhost ~]$ lsusb
Bus 001 Device 003: ID ffff:5678 USB Disk 2.0
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 004: ID 0e0f:0008 VMware, Inc. Virtual Bluetooth Adapter
Bus 002 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub
Bus 002 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

修改源码

```
#include <linux/usb.h>
#include <linux/mutex.h>


/* Define these values to match your devices */
#define USB_DETECT_VENDOR_ID    0xffff
#define USB_DETECT_PRODUCT_ID   0x5678

/* table of devices that work with this driver */
static const struct usb_device_id usbdetect_table[] = {
    { USB_DEVICE(USB_DETECT_VENDOR_ID, USB_DETECT_PRODUCT_ID) },
```

源码：

usb_detect.c:

/*

 * USB Detect driver

 *

 * This driver is based on the 2.6.3 version of drivers/usb/usb-skeleton.c

 */

#include <linux/kernel.h>

#include <linux/errno.h>

#include <linux/slab.h>

```
#include <linux/module.h>
#include <linux/kref.h>
#include <linux/uaccess.h>
#include <linux/usb.h>
#include <linux/mutex.h>


/* Define these values to match your devices */
#define USB_DETECT_VENDOR_ID 0xffff
#define USB_DETECT_PRODUCT_ID       0x5678

/* table of devices that work with this driver */
static const struct usb_device_id usbdetect_table[] = {
        { USB_DEVICE(USB_DETECT_VENDOR_ID, USB_DETECT_PRODUCT_ID) },
        { }                             /* Terminating entry */
};
MODULE_DEVICE_TABLE(usb, usbdetect_table);


/* Get a minor range for your devices from the usb maintainer */
#define USB_DETECT_MINOR_BASE       192

#define WRITES_IN_FLIGHT      8
/* arbitrarily chosen */

/* Structure to hold all of our device specific stuff */
struct usb_detect {
        struct usb_device     *udev;                /* the usb device for this device */
        struct usb_interface *interface;            /* the interface for this device */
        struct semaphore      limit_sem;            /* limiting the number of writes in progress */
```

```
	struct usb_anchor	submitted;		/* in case we need to retract our submissions */
	struct urb		*bulk_in_urb;		/* the urb to read data with */
	unsigned char		*bulk_in_buffer;	/* the buffer to receive data */
	size_t			bulk_in_size;		/* the size of the receive buffer */
	size_t			bulk_in_filled;		/* number of bytes in the buffer */
	size_t			bulk_in_copied;		/* already copied to user space */
	__u8			bulk_in_endpointAddr;	/* the address of the bulk in endpoint */
	__u8			bulk_out_endpointAddr;	/* the address of the bulk out endpoint */
	int			errors;			/* the last request tanked */
	bool			ongoing_read;		/* a read is going on */
	spinlock_t		err_lock;		/* lock for errors */
	struct kref		kref;
	struct mutex		io_mutex;		/* synchronize I/O with disconnect */
	unsigned long		disconnected:1;
	wait_queue_head_t	bulk_in_wait;		/* to wait for an ongoing read */
};
#define to_detect_dev(d) container_of(d, struct usb_detect, kref)

static struct usb_driver usbdetect_driver;

static void usbdetect_delete(struct kref *kref)
{
	struct usb_detect *dev = to_detect_dev(kref);

	usb_free_urb(dev->bulk_in_urb);
	usb_put_intf(dev->interface);
	usb_put_dev(dev->udev);
	kfree(dev->bulk_in_buffer);
	kfree(dev);
}
```

```
static const struct file_operations usbdetect_fops = {};

/*
  * usb class driver info in order to get a minor number from the usb core,
  * and to have the device registered with the driver core
  */
static struct usb_class_driver usbdetect_class = {
        .name =         "usbdetect%d",
        .fops =         &usbdetect_fops,
        .minor_base = USB_DETECT_MINOR_BASE,
};

static int usbdetect_probe(struct usb_interface *interface,
                      const struct usb_device_id *id)
{
        struct usb_detect *dev;
        struct usb_endpoint_descriptor *bulk_in, *bulk_out;
        int retval;

        /* allocate memory for our device state and initialize it */
        dev = kzalloc(sizeof(*dev), GFP_KERNEL);
        if (!dev)
                return -ENOMEM;

        kref_init(&dev->kref);
        sema_init(&dev->limit_sem, WRITES_IN_FLIGHT);
        mutex_init(&dev->io_mutex);
        spin_lock_init(&dev->err_lock);
        init_usb_anchor(&dev->submitted);
```

```
init_waitqueue_head(&dev->bulk_in_wait);

dev->udev = usb_get_dev(interface_to_usbdev(interface));
dev->interface = usb_get_intf(interface);

/* set up the endpoint information */
/* use only the first bulk-in and bulk-out endpoints */
retval = usb_find_common_endpoints(interface->cur_altsetting,
        &bulk_in, &bulk_out, NULL, NULL);
if (retval) {
    dev_err(&interface->dev,
        "Could not find both bulk-in and bulk-out endpoints\n");
    goto error;
}

dev->bulk_in_size = usb_endpoint_maxp(bulk_in);
dev->bulk_in_endpointAddr = bulk_in->bEndpointAddress;
dev->bulk_in_buffer = kmalloc(dev->bulk_in_size, GFP_KERNEL);
if (!dev->bulk_in_buffer) {
    retval = -ENOMEM;
    goto error;
}
dev->bulk_in_urb = usb_alloc_urb(0, GFP_KERNEL);
if (!dev->bulk_in_urb) {
    retval = -ENOMEM;
    goto error;
}

dev->bulk_out_endpointAddr = bulk_out->bEndpointAddress;
```

```
        /* save our data pointer in this interface device */
        usb_set_intfdata(interface, dev);

        /* we can register the device now, as it is ready */
        retval = usb_register_dev(interface, &usbdetect_class);
        if (retval) {
                /* something prevented us from registering this driver */
                dev_err(&interface->dev,
                        "Not able to get a minor for this device.\n");
                usb_set_intfdata(interface, NULL);
                goto error;
        }

        /* let the user know what node this device is now attached to */
        dev_info(&interface->dev,
                 "USB detect device now attached to USBdetect-%d\n",
                 interface->minor);

        /* Print Vendor ID, Product ID, Manufacturer, and Product */
        dev_info(&interface->dev,
                 "Vendor ID: 0x%04X, Product ID: 0x%04X\n",
                 le16_to_cpu(dev->udev->descriptor.idVendor),
                 le16_to_cpu(dev->udev->descriptor.idProduct));
        dev_info(&interface->dev,
                 "Manufacturer: %s, Product: %s\n",
                 dev->udev->manufacturer, dev->udev->product);

        return 0;

error:
```

```
	/* this frees allocated memory */
	kref_put(&dev->kref, usbdetect_delete);

	return retval;
}

static void usbdetect_disconnect(struct usb_interface *interface)
{
	struct usb_detect *dev;

	dev = usb_get_intfdata(interface);

	dev_info(&interface->dev,
		 "USB detect device removed from USBdetect-%d\n",
		 interface->minor);

	dev_info(&interface->dev,
		 "Vendor ID: 0x%04X, Product ID: 0x%04X\n",
		 le16_to_cpu(dev->udev->descriptor.idVendor),
		 le16_to_cpu(dev->udev->descriptor.idProduct));

	/* Clean up */
	usb_set_intfdata(interface, NULL);
	usb_deregister_dev(interface, &usbdetect_class);

	mutex_lock(&dev->io_mutex);
	dev->disconnected = 1;
	mutex_unlock(&dev->io_mutex);

	usb_kill_anchored_urbs(&dev->submitted);
```

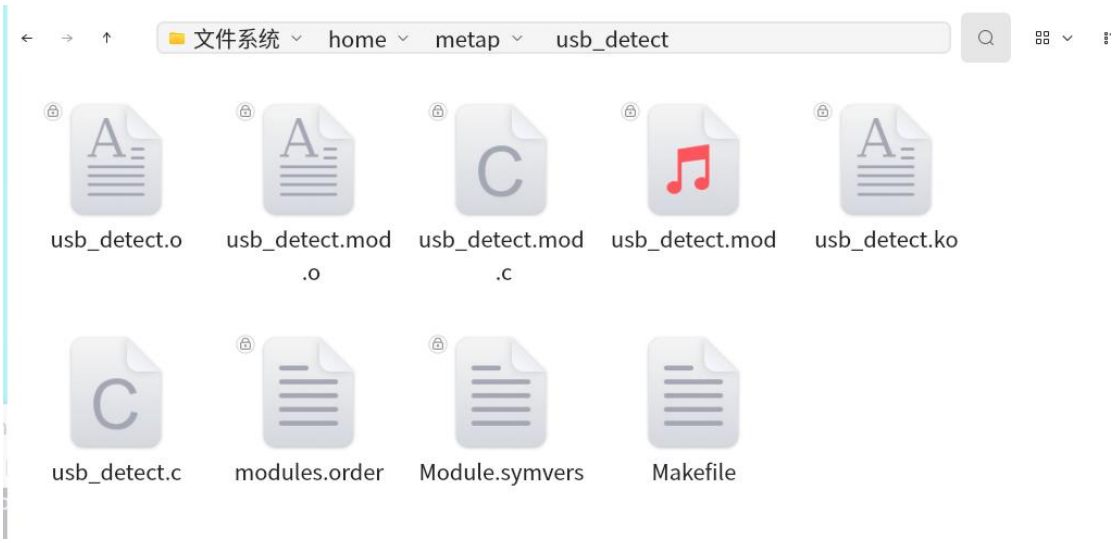
```
	kref_put(&dev->kref, usbdetect_delete);
}



static struct usb_driver usbdetect_driver = {
	.name =		"usbdetect",	//
	.probe =	usbdetect_probe,
	.disconnect =	usbdetect_disconnect,
	.id_table =	usbdetect_table,
	.supports_autosuspend = 1,
};

MODULE_LICENSE("GPL v2");

static int __init usb_detect_init(void)
{
	int result;
	printk("Start usb_detect module...\n");
	/* register this driver with the USB subsystem */
	result = usb_register(&usbdetect_driver);
	if (result < 0) {
		printk("usb_register failed. Error number %d\n", result);
		return -1;
	}
	return 0;
}

static void __exit usb_detect_exit(void)
{
	printk("Exit usb_detect module...\n");
```

```
    /* deregister this driver with the USB subsystem */
    usb_deregister(&usbdetect_driver);
}

module_init(usb_detect_init);
module_exit(usb_detect_exit);
```

Makefile 文件

```
obj-m += usbdetect.o

KERNELDIR ?= /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

default:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules

clean:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) clean
```

编译



卸载自带的模块

sudo rmmod uas

sudo rmmod usb_storage

```
[metap@localhost usb_detect]$ sudo rmmod uas
[sudo] metap 的密码：
[metap@localhost usb_detect]$ sudo rmmod usb_storage
```

加载新创建的模块

```
[metap@localhost usb_detect]$ sudo insmod usb_detect.ko
[metap@localhost usb_detect]$
[metap@localhost usb_detect]$ lsmod | grep usb_detect

usb_detect             16384  0
[metap@localhost usb_detect]$
[metap@localhost usb_detect]$
```

查看加载的内容——dmesg

U 盘插入检测结果：

```
[metap@localhost usb_detect]$
[metap@localhost usb_detect]$ sudo dmesg | tail

[  483.498041] sd 33:0:0:0: [sdb] No Caching mode page found
[  483.498049] sd 33:0:0:0: [sdb] Assuming drive cache: write through
[  483.524383]  sdb: sdb1
[  483.526898] sd 33:0:0:0: [sdb] Attached SCSI removable disk
[ 1038.839134] hrtimer: interrupt took 2592357 ns
[ 1595.141713] usbcore: deregistering interface driver uas
[ 1672.492897] usb_detect: loading out-of-tree module taints kernel.
[ 1672.492956] usb_detect: module verification failed: signature and/or required key missing - tainting kernel
[ 1672.494372] Start usb_detect module...
[ 1672.494471] usbcore: registered new interface driver usbdetect
[metap@localhost usb_detect]$
[metap@localhost usb_detect]$ lsusb
Bus 001 Device 003: ID ffff:5678 USB Disk 2.0
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 004: ID 0e0f:0008 VMware, Inc. Virtual Bluetooth Adapter
Bus 002 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub
Bus 002 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

拔出 U 盘后

sudo dmesg | tail

```
[metap@localhost usb_detect]$ sudo dmesg | tail
[sudo] metap 的密码：
抱歉，请重试。
[sudo] metap 的密码：
[  483.498049] sd 33:0:0:0: [sdb] Assuming drive cache: write through
[  483.524383]  sdb: sdb1
[  483.526898] sd 33:0:0:0: [sdb] Attached SCSI removable disk
[ 1038.839134] hrtimer: interrupt took 2592357 ns
[ 1595.141713] usbcore: deregistering interface driver uas
[ 1672.492897] usb_detect: loading out-of-tree module taints kernel.
[ 1672.492956] usb_detect: module verification failed: signature and/or required key missing - tainting kernel
[ 1672.494372] Start usb_detect module...
[ 1672.494471] usbcore: registered new interface driver usbdetect
[ 2415.190226] usb 1-1: USB disconnect, device number 3
```

lsusb

```
[metap@localhost usb_detect]$ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 004: ID 0e0f:0008 VMware, Inc. Virtual Bluetooth Adapter
Bus 002 Device 003: ID 0e0f:0002 VMware, Inc. Virtual USB Hub
Bus 002 Device 002: ID 0e0f:0003 VMware, Inc. Virtual Mouse
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
[metap@localhost usb_detect]$
```

卸载模块

sudo dmesg | tail

```
[metap@localhost usb_detect]$ sudo rmmod usb_detect
[metap@localhost usb_detect]$
```

任务二：

read_speed_test.c 文件源码：

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

#include <linux/kernel.h>

#include <linux/fs.h>

#include <linux/buffer_head.h>

#include <linux/slab.h>

#include <linux/time.h>


#define MY_BLOCK_SIZE 4096 //  自定义块大小


static int __init read_speed_init(void) {

    struct file *file;

    char *buffer;

    loff_t pos = 0;

    size_t bytes_read;

    struct timespec64 start, end;

    unsigned long long duration;

    int i;


    buffer = kmalloc(MY_BLOCK_SIZE, GFP_KERNEL);
```

```
    if (!buffer) {
        printk(KERN_ERR "Failed to allocate memory\n");
        return -ENOMEM;
    }

    file = filp_open("/dev/sda", O_RDONLY, 0);
    if (IS_ERR(file)) {
        printk(KERN_ERR "Failed to open device\n");
        kfree(buffer);
        return PTR_ERR(file);
    }

    ktime_get_real_ts64(&start);

    for (i = 0; i < 1024; i++) { // 读取 1024 个块
        bytes_read = kernel_read(file, buffer, MY_BLOCK_SIZE, &pos);
        if (bytes_read < 0) {
            printk(KERN_ERR "Failed to read from device\n");
            filp_close(file, NULL);
            kfree(buffer);
            return bytes_read;
        }
    }

    ktime_get_real_ts64(&end);

    duration = (end.tv_sec - start.tv_sec) * 1000000000ULL + (end.tv_nsec - start.tv_nsec);
    printk(KERN_INFO "Read speed: %llu MB/s\n", (1024ULL * MY_BLOCK_SIZE * 1000000000ULL) / (duration * 1024ULL * 1024ULL));
```

```
    filp_close(file, NULL);
    kfree(buffer);

    return 0;
}

static void __exit read_speed_exit(void) {
    printk(KERN_INFO "Read speed test module unloaded\n");
}

module_init(read_speed_init);
module_exit(read_speed_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Author");
MODULE_DESCRIPTION("A module to test read speed");
```

Makefile 文件

```
obj-m += read_speed_test.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
```

write_speed_test.c 文件源码：

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

```
#include <linux/buffer_head.h>
#include <linux/slab.h>
#include <linux/time.h>

#define MY_BLOCK_SIZE 4096

static int __init write_speed_init(void) {
    struct file *file;
    char *buffer;
    loff_t pos = 0;
    size_t bytes_written;
    struct timespec64 start, end;
    unsigned long long duration;
    int i;

    buffer = kmalloc(MY_BLOCK_SIZE, GFP_KERNEL);
    if (!buffer) {
        printk(KERN_ERR "Failed to allocate memory\n");
        return -ENOMEM;
    }

    memset(buffer, 0, MY_BLOCK_SIZE);

    file = filp_open("/dev/sda", O_WRONLY, 0);
    if (IS_ERR(file)) {
        printk(KERN_ERR "Failed to open device\n");
        kfree(buffer);
        return PTR_ERR(file);
    }
```

```
    ktime_get_real_ts64(&start);

    for (i = 0; i < 1024; i++) {
        bytes_written = kernel_write(file, buffer, MY_BLOCK_SIZE, &pos);
        if (bytes_written < 0) {
            printk(KERN_ERR "Failed to write to device\n");
            filp_close(file, NULL);
            kfree(buffer);
            return bytes_written;
        }
    }

    ktime_get_real_ts64(&end);

    duration = (end.tv_sec - start.tv_sec) * 1000000000ULL + (end.tv_nsec - start.tv_nsec);
    printk(KERN_INFO "Write speed: %llu MB/s\n", (1024ULL * MY_BLOCK_SIZE * 1000000000ULL) / (duration * 1024ULL * 1024ULL));

    filp_close(file, NULL);
    kfree(buffer);

    return 0;
}

static void __exit write_speed_exit(void) {
    printk(KERN_INFO "Write speed test module unloaded\n");
}

module_init(write_speed_init);
module_exit(write_speed_exit);
```

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Author");
MODULE_DESCRIPTION("A module to test write speed");
```

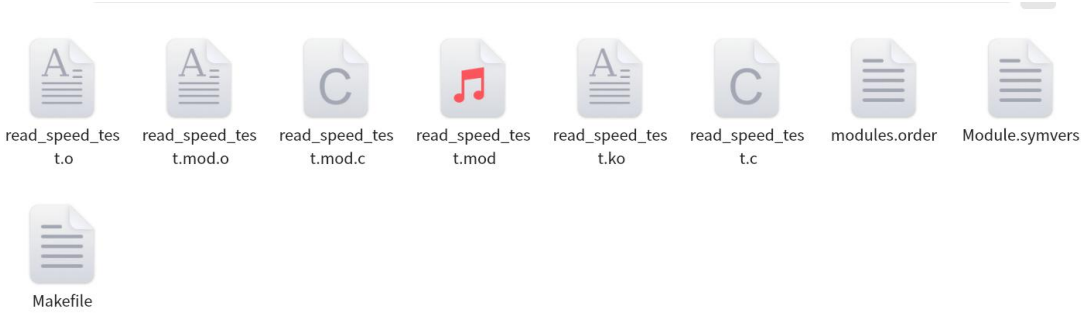
Makefile 文件

```
obj-m := write_speed_test.o

KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

all:
	$(MAKE) -C $(KDIR) M=$(PWD) modules

clean:
	$(MAKE) -C $(KDIR) M=$(PWD) clean
```

操作步骤与结果：

read_speed_test 装载：

sudo make 编译



sudo insmod read_speed_test.ko 装载

sudo dmesg | grep 查看日志

```
[metap@localhost read_speed_test]$ make
make -C /lib/modules/5.10.0-182.0.0.95.oe2203sp3.x86_64/build M=/home/metap/read_speed_test modules
make[1]: 进入目录“/usr/src/kernels/5.10.0-182.0.0.95.oe2203sp3.x86_64”
  CC [M]  /home/metap/read_speed_test/read_speed_test.o
  MODPOST /home/metap/read_speed_test/Module.symvers
  CC [M]  /home/metap/read_speed_test/read_speed_test.mod.o
  LD [M]  /home/metap/read_speed_test/read_speed_test.ko
make[1]: 离开目录“/usr/src/kernels/5.10.0-182.0.0.95.oe2203sp3.x86_64”
[metap@localhost read_speed_test]$ sudo insmod read_speed_test.ko
[metap@localhost read_speed_test]$ dmesg | grep "Read speed"

dmesg: 读取内核缓冲区失败：Operation not permitted
[metap@localhost read_speed_test]$
[metap@localhost read_speed_test]$ sudo dmesg | grep "Read speed"

[ 2261.862131] Read speed: 298 MB/s
[metap@localhost read_speed_test]$
```

sudo rmmod read_speed_test

卸载模块

```
metap@localhost read_speed_test]$ sudo dmesg | grep "Read speed"

 2261.862131] Read speed: 298 MB/s
metap@localhost read_speed_test]$
metap@localhost read_speed_test]$ sudo rmmod read_speed_test
sudo] metap 的密码：
metap@localhost read_speed_test]$ cd
metap@localhost ~]$
```

write_speed_test 装载：

sudo make 编译

```
[metap@localhost write_speed_test]$ sudo make
make -C /lib/modules/5.10.0-182.0.0.95.oe2203sp3.x86_64/build M=/home/metap/writ
e_speed_test modules
make[1]: 进入目录“/usr/src/kernels/5.10.0-182.0.0.95.oe2203sp3.x86_64”
  CC [M]  /home/metap/write_speed_test/write_speed_test.o
mak  MODPOST /home/metap/write_speed_test/Module.symvers
  CC [M]  /home/metap/write_speed_test/write_speed_test.mod.o
e  LD [M]  /home/metap/write_speed_test/write_speed_test.ko
make[1]: 离开目录“/usr/src/kernels/5.10.0-182.0.0.95.oe2203sp3.x86_64”
[metap@localhost write_speed_test]$ sudo insmod write_speed_test.ko
```

write_speed_test 装载

sudo insmod write_speed_test.ko

```
[metap@localhost write_speed_test]$ sudo insmod write_speed_test.ko
[metap@localhost write_speed_test]$
```

查看日志

dmesg | grep "Write speed"

```
[metap@localhost write_speed_test]$ sudo dmesg | grep "Write speed"
[ 3437.394399] Write speed: 924 MB/s
[metap@localhost write_speed_test]$
```

卸载

```
[metap@localhost write_speed_test]$ sudo rmmod write_speed_test
[metap@localhost write_speed_test]$ cd
[metap@localhost ~]$
```

下载和解压 iozone

wget http://www.iozone.org/src/current/iozone3_489.tar

tar -xvf iozone3_489.tar

```
[metap@localhost write_speed_test]$ cd
[metap@localhost ~]$ wget http://www.iozone.org/src/current/iozone3_489.tar
--2024-06-21 20:15:30--  http://www.iozone.org/src/current/iozone3_489.tar
正在解析主机 www.iozone.org (www.iozone.org)... 38.146.202.4
正在连接 www.iozone.org (www.iozone.org)|38.146.202.4|:80... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度： 4136960 (3.9M) [application/x-tar]
正在保存至：“iozone3_489.tar”

iozone3_489.tar     100%[===================>]   3.95M  7.61KB/s  用时 6m 39s

2024-06-21 20:22:11 (10.1 KB/s) - 已保存 “iozone3_489.tar” [4136960/4136960])

[metap@localhost ~]$
```

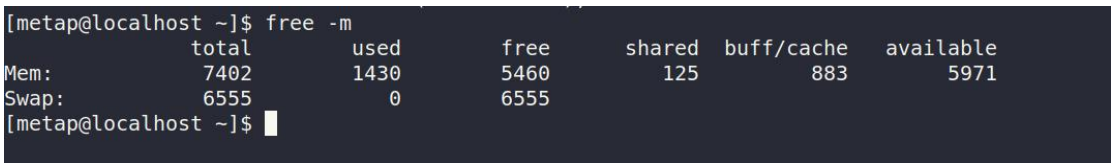
```
[metap@localhost ~]$ tar -xvf iozone3_489.tar
/
/iozone3_489/
/iozone3_489/index.html
/iozone3_489/gifs/
/iozone3_489/src/
/iozone3_489/src/current/
/iozone3_489/src/current/makefile
/iozone3_489/src/current/write_telemetry
/iozone3_489/src/current/Gnuplot.txt
/iozone3_489/src/current/spec.in
/iozone3_489/src/current/fileop.c
/iozone3_489/src/current/iozone.1
/iozone3_489/src/current/client_list
/iozone3_489/src/current/libasync.c
/iozone3_489/src/current/Android.mk
/iozone3_489/src/current/gnuplotps.dem
/iozone3_489/src/current/read_telemetry
/iozone3_489/src/current/Changes.txt
```

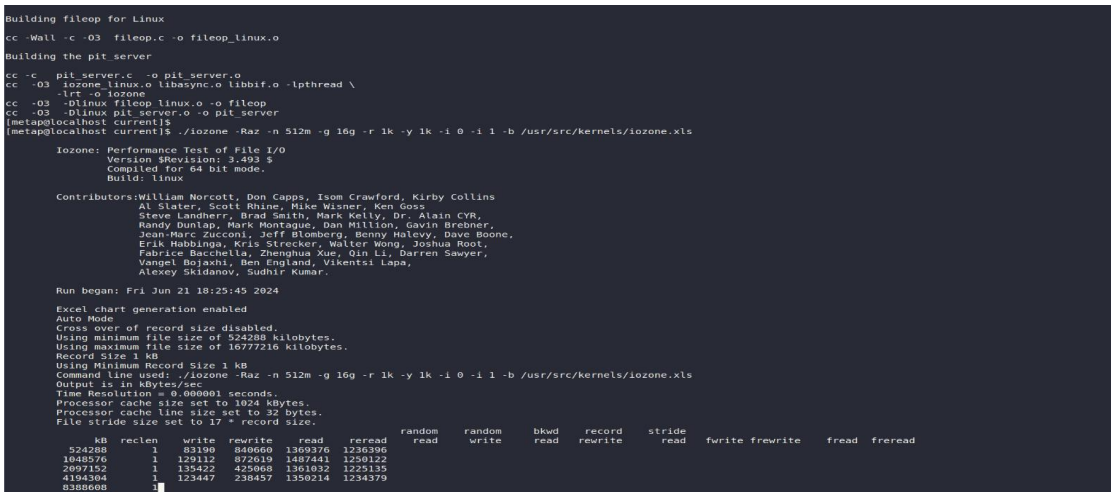
cd iozone3_489/src/current

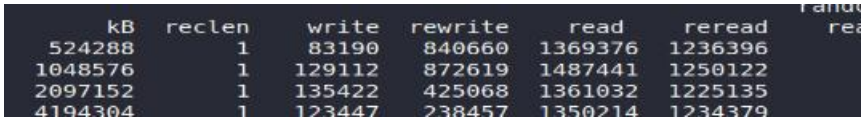
编译 iozone 工具

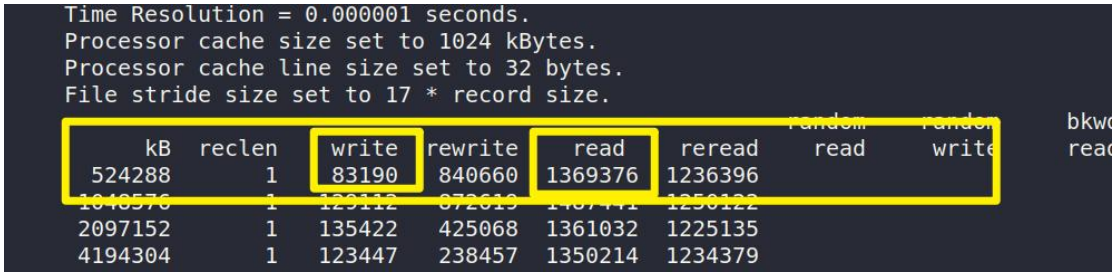
make linux

free -m









由最后一张图数据可知:

对于内核模块程序中，写入的文件大小是 512M，读取的文件是之前写入的大小为 512M 的文件；且读写的记录块都是 1024byte；对应于图中的数据，iozone 测试的文件大小为 512M、记录块为 1024byte 的写速率是 83190kb/s（即 81.24M/s），读速率是 1369376kb/s（即 1337.28M/s）。

比较和分析

iozone 测试结果与模块测试结果存在较大差异，这可能由以下几个原因导致：

IO 方式不同：iozone 可能使用了更高级的 IO 方式，如直接 IO（Direct IO）或者异步 IO，这些方式可以绕过操作系统的缓存和同步 IO 的开销，从而获得更高的性能。

缓存策略：操作系统和文件系统的缓存策略可能不同，iozone 测试可能会优化缓存管理以提升 IO 性能。

测试环境：iozone 可能在测试中使用了更多的优化策略和并发操作，而我的模块测试可能

仅限于简单的顺序读写操作。

四、实验分析

任务一

在进行 USB Detect 驱动程序的修改过程中，通过添加和调整代码，成功实现了对特定 USB 设备的识别和驱动加载。通过这个实验，我学会了如何利用 Linux 内核编程接口编写 USB 驱动程序，理解了驱动注册、设备探测和断开处理的过程。这不仅加深了对 Linux 内核编程的理解，还提升了对 USB 子系统的认识和应用能力。

任务二

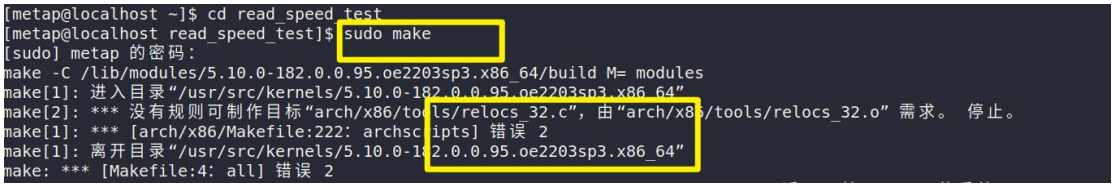
通过本次实验，学到了如何利用 Linux 内核模块进行磁盘读写速度测试，并且了解到不同工具（如 iozone）对性能测试的不同方法和计量标准。实验过程中遇到的挑战包括内核版本和依赖项的管理，以及测试结果与预期之间的差异分析。这些经验增强了对 Linux 内核 IO 操作和性能优化的理解，为未来系统调优提供了宝贵的经验和方法。

五、所遇问题及解决方法

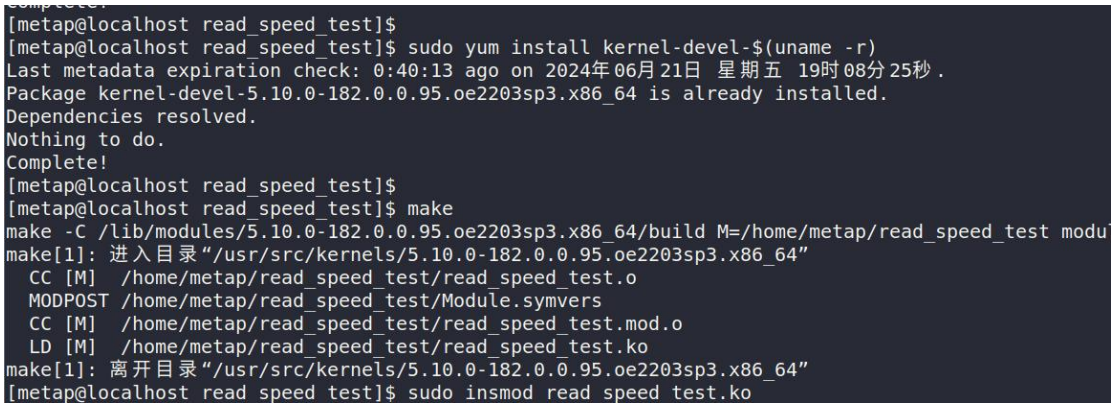
1.编译出错，检查后是 Makefile 文件出错，更改后编译成功

```
obj-m += usb_detect.o

KERNELDIR ?= /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

default:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules

clean:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) clean
```

```
[sudo] metap 的密码：
make -C /lib/modules/5.10.0-182.0.0.95.oe2203sp3.x86_64/build M=/home/metap/usb_detect modules
make[1]: 进入目录“/usr/src/kernels/5.10.0-182.0.0.95.oe2203sp3.x86_64”
make[2]: *** 没有规则可制作目标“/home/metap/usb_detect/usbdetect.o”，由“/home/metap/usb_detect/usbdetect.mod”需求。 停止。
make[1]: *** [Makefile:1850: /home/metap/usb_detect] 错误 2
make[1]: 离开目录“/usr/src/kernels/5.10.0-182.0.0.95.oe2203sp3.x86_64”
make: *** [Makefile:9: default] 错误 2
[metap@localhost usb_detect]$ sudo make
make -C /lib/modules/5.10.0-182.0.0.95.oe2203sp3.x86_64/build M=/home/metap/usb_detect modules
make[1]: 进入目录“/usr/src/kernels/5.10.0-182.0.0.95.oe2203sp3.x86_64”
  CC [M]  /home/metap/usb_detect/usb_detect.o
  MODPOST /home/metap/usb_detect/Module.symvers
  CC [M]  /home/metap/usb_detect/usb_detect.mod.o
  LD [M]  /home/metap/usb_detect/usb_detect.ko
make[1]: 离开目录“/usr/src/kernels/5.10.0-182.0.0.95.oe2203sp3.x86_64”
[metap@localhost usb_detect]$
```



2.编译失败



sudo yum install kernel-devel-$(uname -r)

安装工具包后成功