实验三：

内核驱动模块的创建与加载

创建动态模块源码

修改Makefile文件生成编译规则

编译创建的模块源码，生成驱动模块

安装驱动模块

查看是否安装成功

使用驱动模块

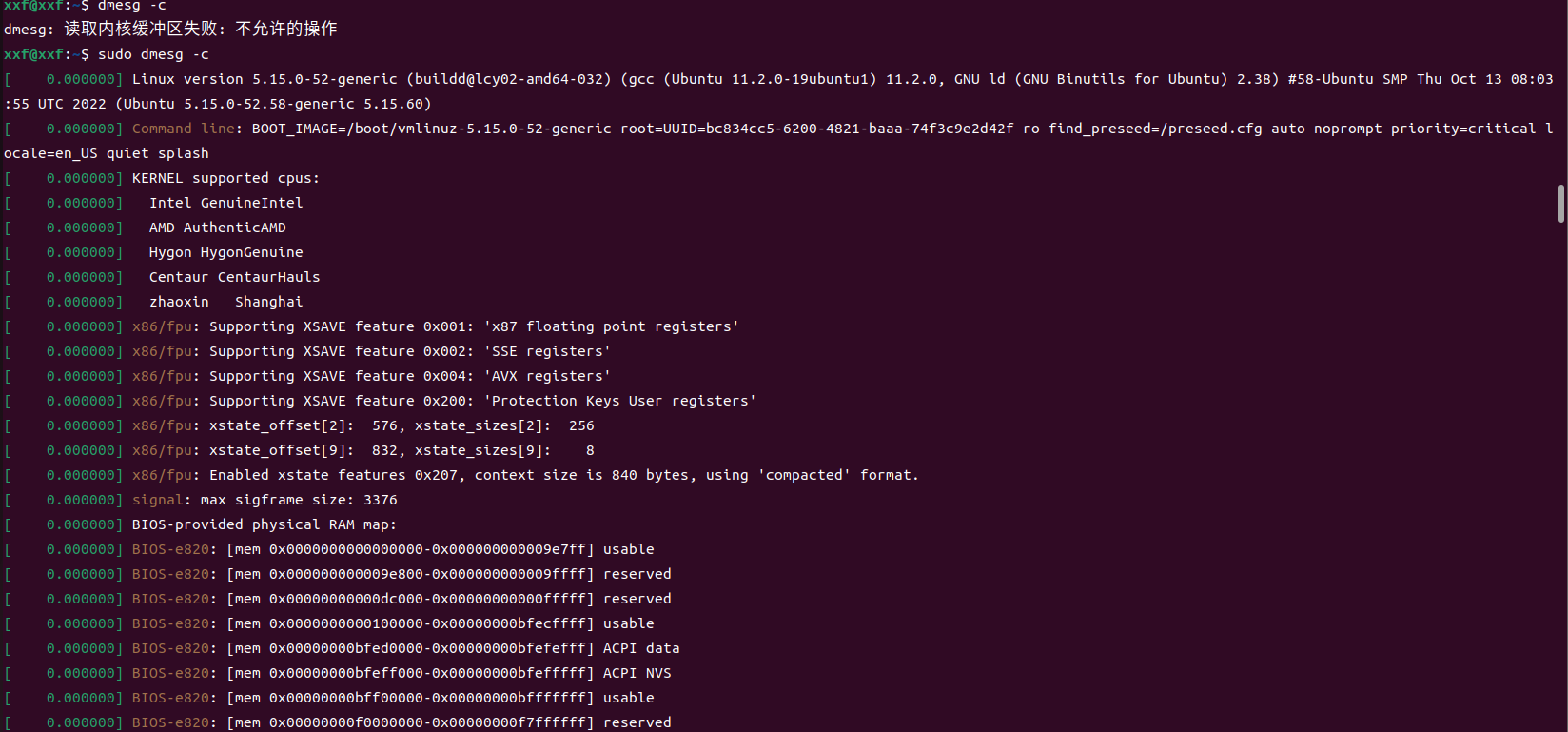
卸载驱动模块

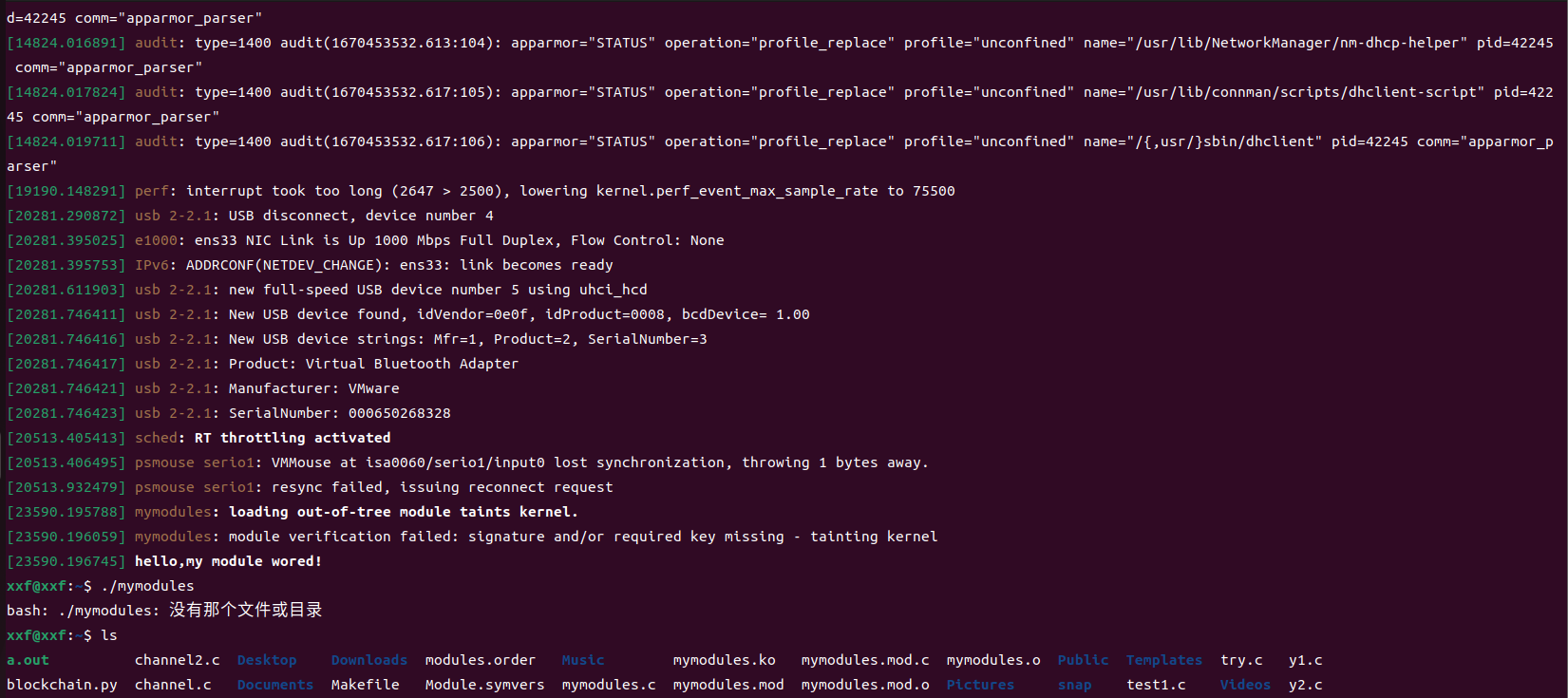


Make时报错

成功make并加载模板insmod

读取内核缓冲区：刚开始没用sudo，导致权限不够

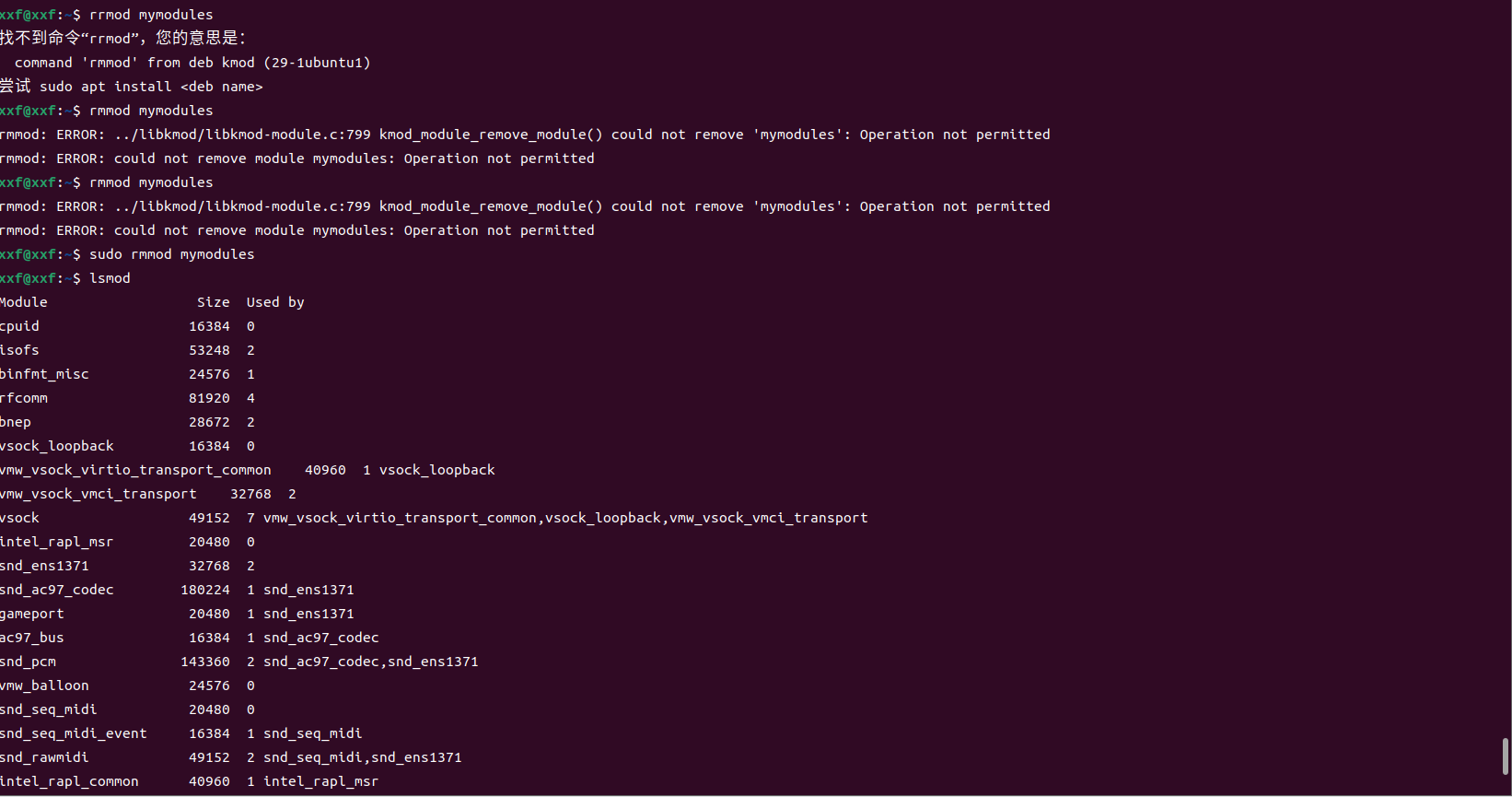


可以看到内核日志信息

Lsmod观察



Rrmod卸载模板并lsmod观察：已经没有了



内核日志信息：已卸载



#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/unistd.h>

#include <linux/time.h>

#include <asm/uaccess.h>

#include <linux/sched.h>

#include <linux/kallsyms.h>

//using syscall 335

#define \_\_NR\_syscall 335

unsigned long \*sys\_call\_table;

unsigned int clear\_cr0(void);

void setback\_cr0(unsigned int val);

static int sys\_mycall(int a, int b, int c);

//to save the original value of the register cr0

unsigned long orig\_cr0;

unsigned long \*sys\_call\_table = 0;

//to save the original syscall func

unsigned long old\_sys\_call\_func;

//set 0 to the 17th bit (WP) in reg cr0

unsigned int clear\_cr0(void)

{

unsigned int cr0 = 0;

unsigned int ret;

//move the value in reg cr0 to reg rax

//movl moves a 32-bits operand

//movq moves a 64-bits operand

//rax is a 64-bits register

//an assembly language code

//asm volatile ("movl %%cr0, %%eax" : "=a"(cr0));//32-bits

asm volatile ("movq %%cr0, %%rax" : "=a"(cr0)); //64-bits

ret = cr0;

//var cr0 is rax

cr0 &= 0xfffeffff; //set 0 to the 17th bit

//asm volatile ("movl %%eax, %%cr0" :: "a"(cr0));//32-bits

//note that cr0 above is a variable while cr0 below is a reg.

asm volatile ("movq %%rax, %%cr0" :: "a"(cr0));

return ret;

}

//recover the value of WP

void setback\_cr0(unsigned int val)

{

//asm volatile ("movl %%eax, %%cr0" :: "a"(val));//32-bits

asm volatile ("movq %%rax, %%cr0" :: "a"(val));//64-bits

}

//my syscall function

static int sys\_mycall(int a,int b,int c)

{

printk("Change syscall successfully!\nReturn a+b+c\nBy xhw-class 92-No 2160506148\n");

return a+b+c;

}

static int \_\_init init\_addsyscall(void)

{

printk("Begin changing syscall...\n");

//Automatically get sys\_call\_table address

sys\_call\_table = (unsigned long \*)kallsyms\_lookup\_name("sys\_call\_table");

//print sys\_call\_table address

printk("sys\_call\_table: 0x%p\n", sys\_call\_table);

//save original syscall func

old\_sys\_call\_func = (int(\*)(void))(sys\_call\_table[\_\_NR\_syscall]);

//modify the value of WP in CR0

orig\_cr0 = clear\_cr0();

//change the syscall address

sys\_call\_table[\_\_NR\_syscall] = (unsigned long)&sys\_mycall;

//setback the value of WP in CR0

//to read only

setback\_cr0(orig\_cr0);

return 0;

}

static void \_\_exit exit\_addsyscall(void)

{

//modify the value of WP in CR0

orig\_cr0 = clear\_cr0();

//change the syscall address

sys\_call\_table[\_\_NR\_syscall] =old\_sys\_call\_func;

//setback the value of WP in CR0

//to read only

setback\_cr0(orig\_cr0);

printk("Recovering syscall...\n");

}

module\_init(init\_addsyscall);

module\_exit(exit\_addsyscall);

MODULE\_LICENSE("GPL");

建立makefile：

ifneq ($(KERNELRELEASE),)

obj-m := modify\_syscall.o

else

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

PWD := $(shell pwd)

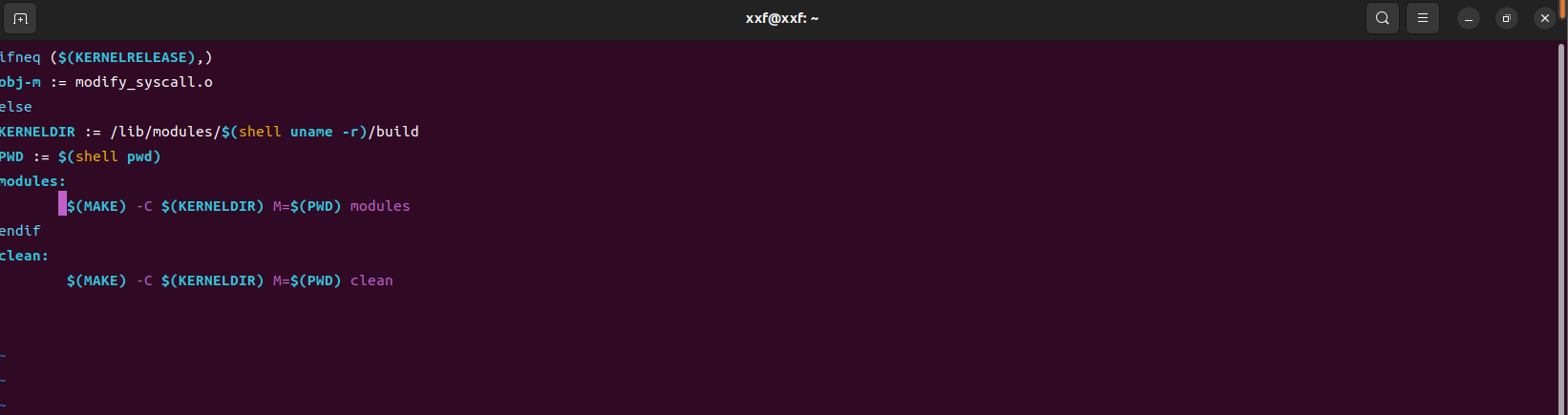
modules:

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

endif

clean:

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



编写test.c测试：

#include<stdio.h>

#include<unistd.h>

#include<sys/syscall.h>

#include<time.h>

int main()

{

printf("Set parameters 78,10,20\nReturn:");

int ret=syscall(321,78,10,20);

if(ret == -1)

printf("syscall failed!\n");

else

printf("%d\n",ret);

return 0;

}

