操作系统实验报告

班号：10012209　　姓名：梁桐　　学号：2023370018

实验日期：2024.5.17　　实验名称：《文件系统》

一、实验目的

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。

了解操作系统的内核文件系统管理。

二、实验要求

任务一：

1. 正确使用 setfattr 设置 EA，正确使用 getfattr 获取 EA。
2. 提交命令行的操作过程截图，以及实验分析总结。

任务二：

3. 使用文件系统注册/注销函数，注册一个自定义文件系统类型；
4. 加载模块后，查看系统中是否存在注册的文件系统类型。
5. 加载、卸载模块并查看模块打印信息。

任务三：

6. 正确编写满足功能的源文件，正确编译。
7. 正常加载、卸载内核模块；且内核模块功能满足任务所述。
8. 提交相关源码与运行截图。

任务四：

9. 正确编写满足功能的源文件，正确编译。
10. 正常加载、卸载内核模块；且内核模块功能满足任务所述。
11. 提交相关源码与运行截图。

三、实验过程及结果

1 任务 1：为 Ext4 文件系统添加扩展属性

1、环境准备：为了使用扩展属性，需要安装 libattr

dnf install -y libattr

```
[metap@localhost ~]$ sudo dnf install -y libattr
[sudo] metap 的密码：
对不起，请重试。
[sudo] metap 的密码：
OS                                                                36 kB/s | 3.8 kB     00:00
everything                                                        17 kB/s | 3.8 kB     00:00
EPOL                                                              12 kB/s | 3.0 kB     00:00
debuginfo                                                         22 kB/s | 3.8 kB     00:00
source                                                            21 kB/s | 3.8 kB     00:00
update                                                            22 kB/s | 3.5 kB     00:00
update                                                           1.9 MB/s |  37 MB     00:20
update-source                                                     28 kB/s | 3.5 kB     00:00
update-source                                                    1.7 MB/s | 800 kB     00:00
Package attr-2.5.1-4.oe2203sp3.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[metap@localhost ~]$
```

2、查看当前文件系统类型

df -Th

其中，-T 用于显示文件系统类型；-h 表示以 1024 的幂为单位显示文件系统大小。

```
metap@localhost ~]$ df -Th
文件系统                    类型      容量  已用  可用 已用% 挂载点
devtmpfs                    devtmpfs  4.0M     0  4.0M    0% /dev
tmpfs                       tmpfs     3.7G  8.0K  3.7G    1% /dev/shm
tmpfs                       tmpfs     1.5G  9.3M  1.5G    1% /run
tmpfs                       tmpfs     4.0M     0  4.0M    0% /sys/fs/cgroup
/dev/mapper/openeuler-root  ext4       38G  6.9G   29G   20% /
tmpfs                       tmpfs     3.7G  512K  3.7G    1% /tmp
/dev/sda1                   ext4      974M  174M  733M   20% /boot
/dev/mapper/openeuler-home  ext4       19G  166M   18G    1% /home
tmpfs                       tmpfs     741M   32K  741M    1% /run/user/1000
/dev/sr0                    iso9660   3.5G  3.5G     0  100% /run/media/metap/openEuler-22.03-LTS-SP3-x86_64
[metap@localhost ~]$
```

3、检查当前文件系统是否支持文件扩展属性

在 ext3 和 ext4 文件系统上，可通过命令 tune2fs -l 来检查是否支持扩展属性。

（1）用 fdisk -l 查看硬盘及分区信息

```
[metap@localhost ~]$ sudo fdisk -l
[sudo] metap 的密码：
Disk /dev/sda：64 GiB，68719476736 字节，134217728 个扇区
磁盘型号：VMware Virtual S
单元：扇区 / 1 * 512 = 512 字节
扇区大小(逻辑/物理)：512 字节 / 512 字节
I/O 大小(最小/最佳)：512 字节 / 512 字节
磁盘标签类型：dos
磁盘标识符：0x80fcff84

设备         启动     起点      末尾      扇区 大小 Id 类型
/dev/sda1  *        2048   2099199   2097152   1G 83 Linux
/dev/sda2        2099200 134217727 132118528  63G 8e Linux LVM


Disk /dev/mapper/openeuler-root：38.03 GiB，40831549440 字节，79749120 个扇区
单元：扇区 / 1 * 512 = 512 字节
扇区大小(逻辑/物理)：512 字节 / 512 字节
I/O 大小(最小/最佳)：512 字节 / 512 字节


Disk /dev/mapper/openeuler-swap：6.4 GiB，6874464256 字节，13426688 个扇区
```

（2）用 tune2fs -l 显示设备的详细信息

tune2fs 命令允许系统管理员在 ext2、ext3 或 ext4 文件系统上调整各种可调的文件系统参数。

这些选项的当前值可以使用-l 选项显示。

tune2fs -l /dev/sda1 | grep user_xattr

```
扇区大小(逻辑/物理)：512 字节 / 512 字节
I/O 大小(最小/最佳)：512 字节 / 512 字节
[metap@localhost ~]$ sudo tune2fs -l /dev/sda1 | grep user_xattr
Default mount options:    user_xattr acl
[metap@localhost ~]$
```

通过检查/dev/sda1 文件系统参数中的默认挂载选项“Default mount options”是否有对应内容来确定分区设备的文件系统是否支持扩展属性——上图所示说明，该文件系统支持扩展用户属性 user_xattr、与 ACL 权限

4、创建文件 file.txt，用 setfattr 设置文件系统对象的扩展属性

对于不含转义字符 \ 的纯文本属性值，有无双引号限定效果一样。

touch file.txt

sudo setfattr -n user.attrname -v "value" file.txt

```
[metap@localhost ~]$ touch file.txt
[metap@localhost ~]$ sudo setfattr -n user.attrname -v "value" file.txt
```

getfattr -d file.txt

```
[metap@localhost ~]$ getfattr -d file.txt

# file: file.txt
user.attrname="value"

[metap@localhost ~]$
```

5、设置八进制数属性值“\012”，最终以八进制数的 base64 编码存储。

对于包含转义字符 \ 的文本属性值，无双引号则不对转义符 \ 进行转义；有双引号则对其进行转义。

6、设置十六进制数属性值，所设置的数的位数必须为偶数，即 0x 或 0X 后的数字必须为偶数位，否则出错。 若设置成功，最终以十六进制数的 base64 编码存储。

7、设置 base64 编码属性值，所设置的编码必须符合 base64 编码，即 0s 后的编码字符串必须符合 base64 编码，否则出错。若设置成功，最终以 base64 编码对应的文本信息存储。

tips：可在 https://base64.us/ 中，将需要设置的属性值进行 base64 编码后，再使用 setfattr 命令设置，注意设置时需在 base64 编码前加 0s 前缀。

```
ser.attrname="value"

metap@localhost ~]$
metap@localhost ~]$ setfattr -n user.octal_attr -v "$(echo -n -e '\012')" file.txt
metap@localhost ~]$ setfattr -n user.hex_attr -v "$(echo -n -e '\x0A\x0B')" file.txt
metap@localhost ~]$ setfattr -n user.base64_attr -v "$(echo -n 'Hello World!' | base64)" file.txt
```

8、用 getfattr 编码设置。

保持原编码设置

对属性设置 text 编码

对属性设置 hex 编码

对属性设置 base64 编码

```
[metap@localhost ~]$ setfattr -n user.octal_attr -v "$(echo -n -e '\012')" file.txt
[metap@localhost ~]$ setfattr -n user.hex_attr -v "$(echo -n -e '\x0A\x0B')" file.txt
[metap@localhost ~]$ setfattr -n user.base64_attr -v "$(echo -n 'Hello World!' | base64)" file.txt
[metap@localhost ~]$ # 保持原编码设置
[metap@localhost ~]$ setfattr -n user.original_attr -v "Original Value" file.txt
[metap@localhost ~]$ # 对属性设置 text 编码
[metap@localhost ~]$ setfattr -n user.text_attr -v "$(echo -n 'Text Value' | base64)" file.txt
[metap@localhost ~]$ # 对属性设置 hex 编码
[metap@localhost ~]$ setfattr -n user.hex_attr -v "$(echo -n -e '\x48\x65\x78' | base64)" file.txt
[metap@localhost ~]$ # 对属性设置 base64 编码
[metap@localhost ~]$ setfattr -n user.base64_attr -v "$(echo -n 'Base64 Value' | base64)" file.txt
[metap@localhost ~]$ # 使用 getfattr 命令获取属性值
[metap@localhost ~]$ getfattr -n user.original_attr file.txt
# file: file.txt
user.original_attr="Original Value"

[metap@localhost ~]$ getfattr -n user.text_attr --encoding=text file.txt
# file: file.txt
user.text_attr="VGV4dCBWYWx1ZQ=="

[metap@localhost ~]$ getfattr -n user.hex_attr --encoding=hex file.txt
# file: file.txt
user.hex_attr=0x53475634

[metap@localhost ~]$ getfattr -n user.base64_attr --encoding=base64 file.txt
# file: file.txt
user.base64_attr=0sUW1GelpUWTBJRlpoYkhWbA==

[metap@localhost ~]$
```

user.original_attr: 保持原编码设置，属性值为 "Original Value"。

user.text_attr: 属性值经过 base64 编码后为 "VGV4dCBWYWx1ZQ==", 解码

后为 "Text Value"。

user.hex_attr: 属性值为十六进制数 "0x53475634"，解码后为 "SGVsbG8gV29ybGQh"。

user.base64_attr: 属性值为 base64 编码 "0sUW1GelpUWTBJRlpoYkhWbA==", 解码后为 "Hello World!"。

任务一分析与总结

任务一涉及使用 setfattr 命令为文件设置扩展属性，并使用 getfattr 命令获取扩展属性的过程。

设置扩展属性：

使用 setfattr 命令设置扩展属性。

属性名称通过 -n 参数指定，属性值通过 -v 参数指定。

可以使用不同的编码格式来设置属性值，包括八进制数、十六进制数和 base64 编码。

获取扩展属性：

使用 getfattr 命令获取扩展属性。

可以通过指定 --encoding 参数来指定属性值的编码格式，包括文本编码、十六进制编码和 base64 编码。

示例：

设置八进制数属性值：setfattr -n user.octal_attr -v "$(echo -n -e '\012')" file.txt。

设置十六进制数属性值：setfattr -n user.hex_attr -v "$(echo -n -e '\x0A\x0B')" file.txt。

设置 base64 编码属性值：setfattr -n user.base64_attr -v "$(echo -n 'Hello World!' | base64)" file.txt。

获取属性值并指定编码格式：getfattr -n user.text_attr --encoding=text file.txt。

任务一帮助理解了如何使用 setfattr 和 getfattr 命令来设置和获取文件的扩展属性，以及如何处理不同编码格式的属性值。

2 任务 2：注册一个自定义的文件系统类型

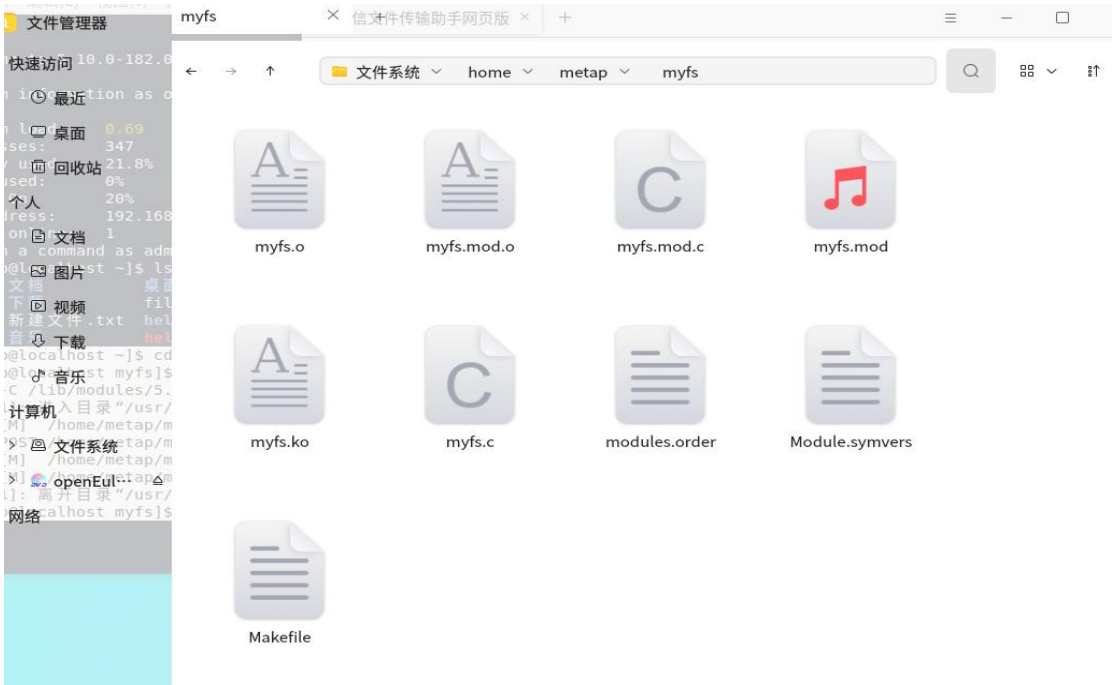
实验步骤

1、查看系统中已经注册的文件系统类型

cat /proc/filesystems

```
[metap@localhost ~]$ cat /proc/filesystems
nodev   sysfs
nodev   tmpfs
nodev   bdev
nodev   proc
nodev   cgroup
nodev   cgroup2
nodev   cpuset
nodev   devtmpfs
nodev   configfs
nodev   debugfs
nodev   tracefs
nodev   securityfs
nodev   sockfs
nodev   bpf
nodev   pipefs
nodev   ramfs
nodev   hugetlbfs
nodev   devpts
nodev   autofs
nodev   mqueue
nodev   selinuxfs
nodev   pstore
        ext3
        ext2
        ext4
        fuseblk
nodev   fuse
nodev   fusectl
        iso9660
[metap@localhost ~]$
[metap@localhost ~]$
```

2、正确编写满足功能的源文件，包括.c 源文件和 Makefile 文件。



C 语言文件

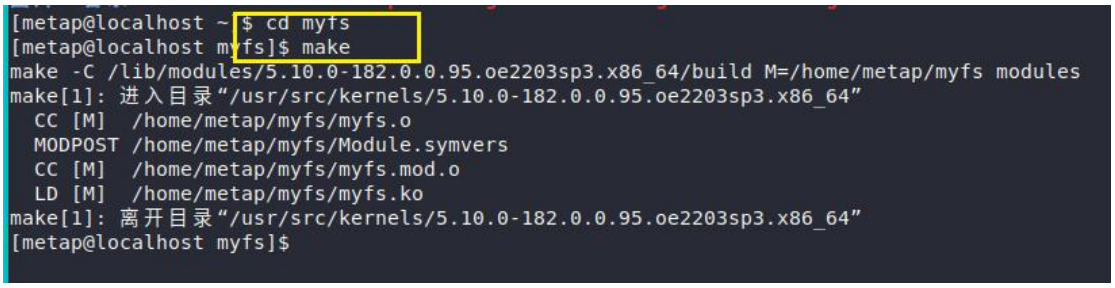
```
#include <linux/module.h>
#include <linux/fs.h>

static struct dentry *myfs_mount(struct file_system_type *fs_type,
                                 int flags, const char *dev_name, void
*data)
{
    /* Your mount logic goes here */
    return mount_bdev(fs_type, flags, dev_name, data, NULL);
}

static struct file_system_type my_fs_type = {
    .owner = THIS_MODULE,
    .name = "myfs",
    .mount = myfs_mount,
    .kill_sb = kill_litter_super,
};

static int __init myfs_init(void)
{
    return register_filesystem(&my_fs_type);
}

static void __exit myfs_exit(void)
{
    unregister_filesystem(&my_fs_type);
}

module_init(myfs_init);
```

```
module_exit(myfs_exit);

MODULE_LICENSE("GPL");
```

Makefile 文件

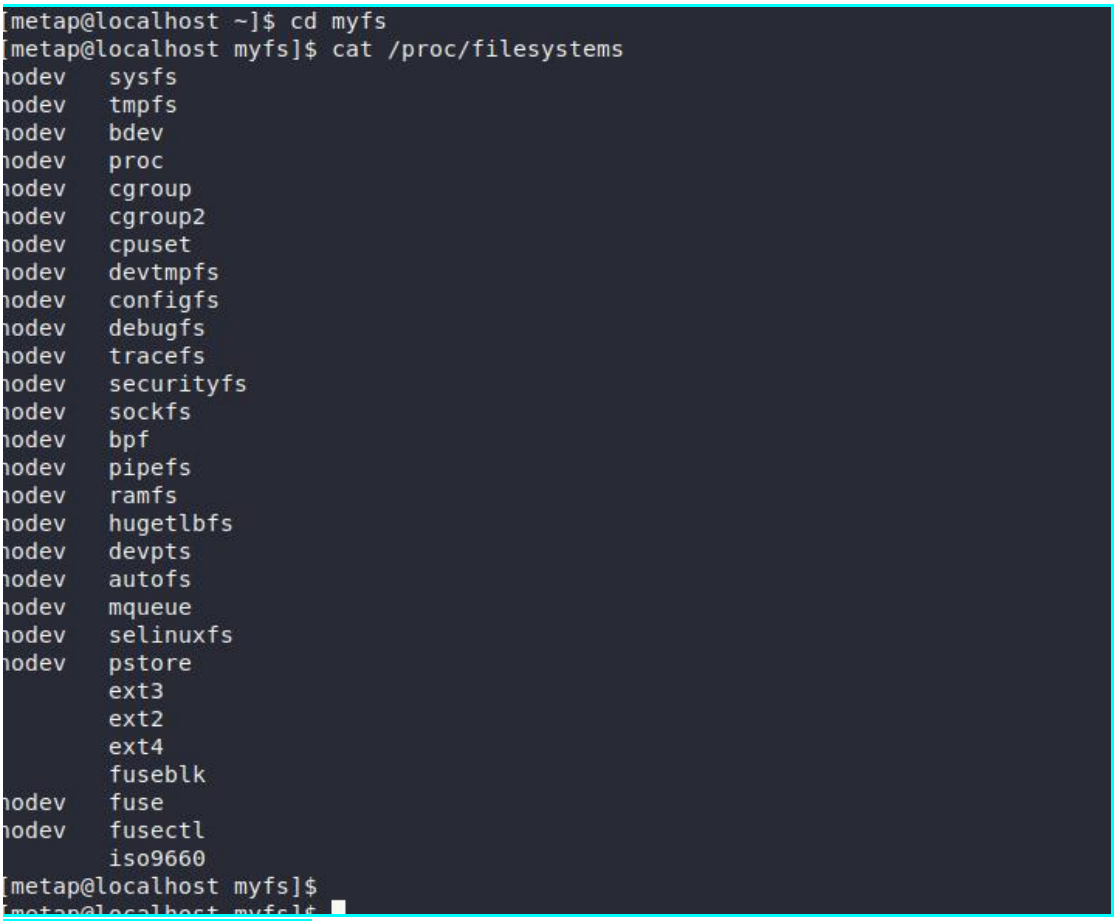
```
obj-m += myfs.o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

3、编译源文件。



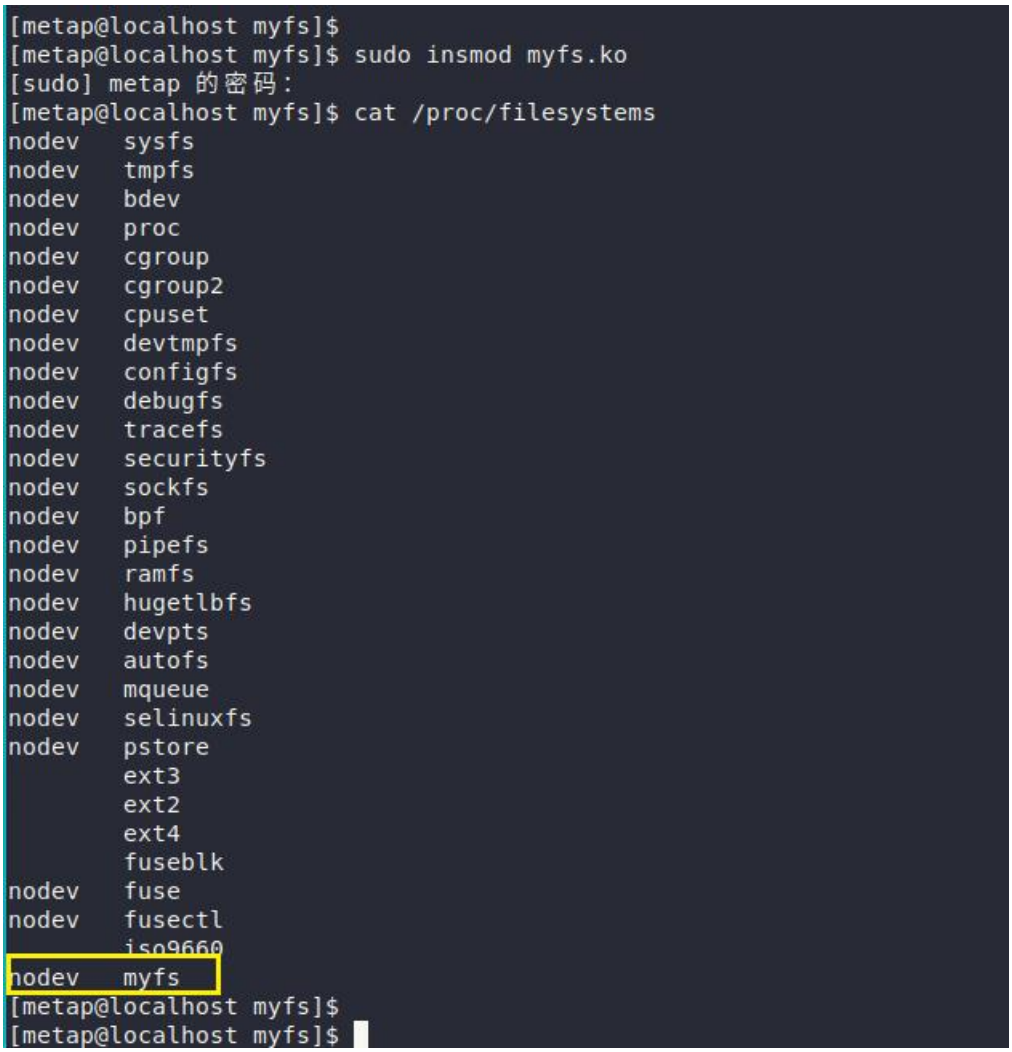
3、对比加载内核模块前后的文件系统结果。

```
cat /proc/filesystems
```



sudo insmod myproc.ko

cat /proc/filesystems



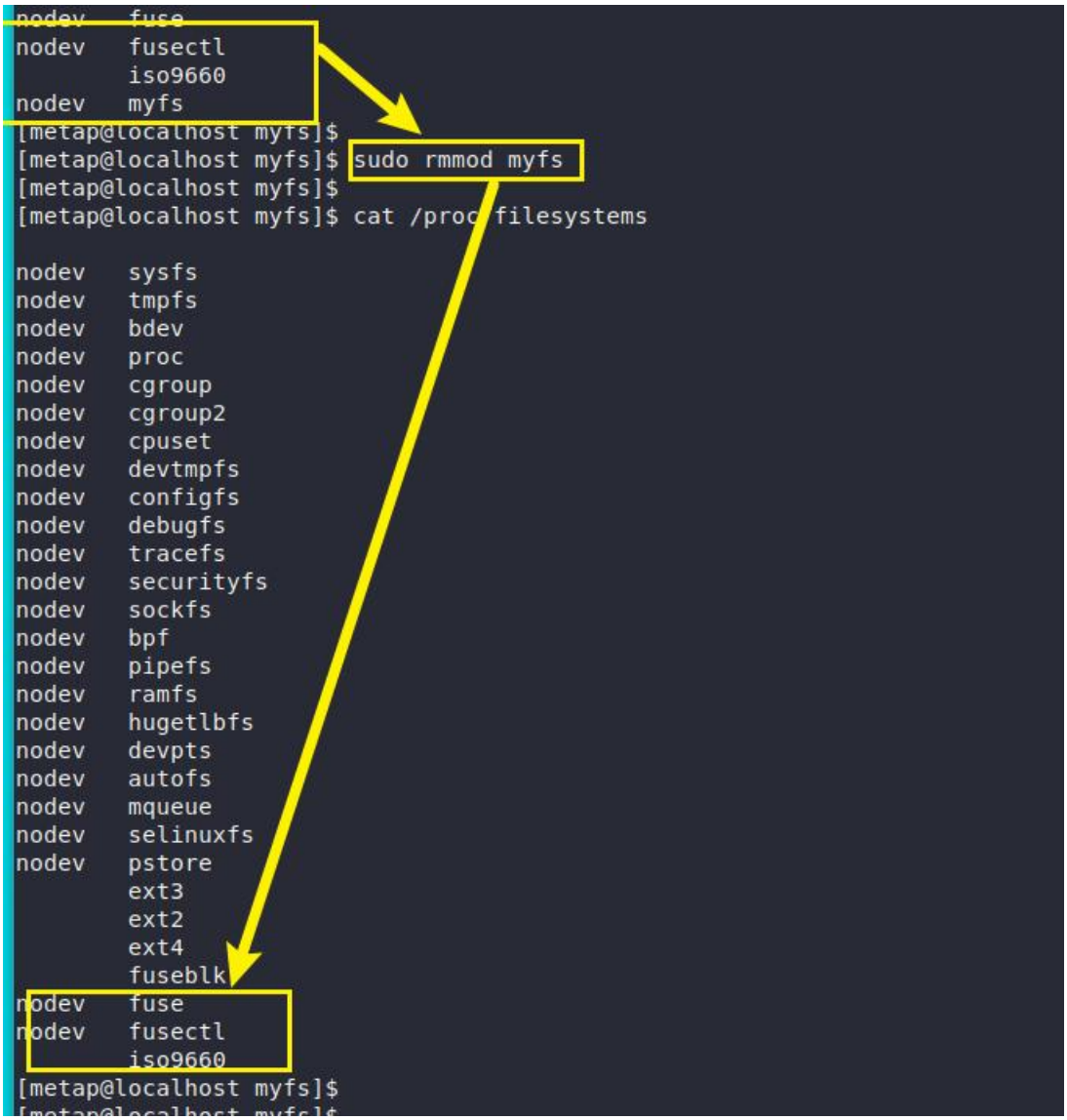
在加载 myfs 模块之前和之后，/proc/filesystems 中**新增了 myfs 文件系统类型**。这表明成功加载了名为 myfs 的自定义文件系统模块

4、卸载内核模块，并查看结果。

sudo rmmod myfs

cat /proc/filesystems

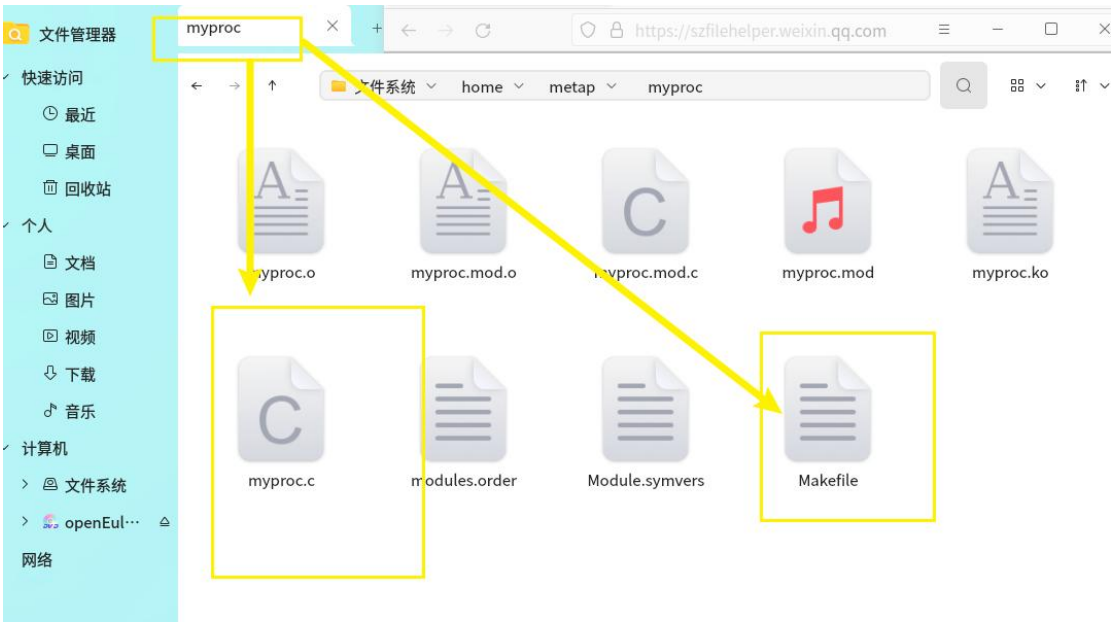
名为 myfs 的自定义文件系统模块在卸载后消失



3 任务 3：在/proc 下创建目录

实验步骤

1、正确编写满足功能的源文件，包括.c 源文件和 Makefile 文件。



C 语言文件

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

static struct proc_dir_entry *my_proc;

static int __init myproc_init(void) {
    my_proc = proc_mkdir("myproc", NULL);
    if (!my_proc) {
        printk(KERN_ERR "Failed to create /proc/myproc directory\n");
        return -ENOMEM;
    }
    printk(KERN_INFO "Created /proc/myproc directory\n");
    return 0;
}

static void __exit myproc_exit(void) {
    proc_remove(my_proc);
    printk(KERN_INFO "Removed /proc/myproc directory\n");
```

}

module_init(myproc_init);

module_exit(myproc_exit);

MODULE_LICENSE("GPL");

MODULE_DESCRIPTION("A simple module to create a directory in /proc");

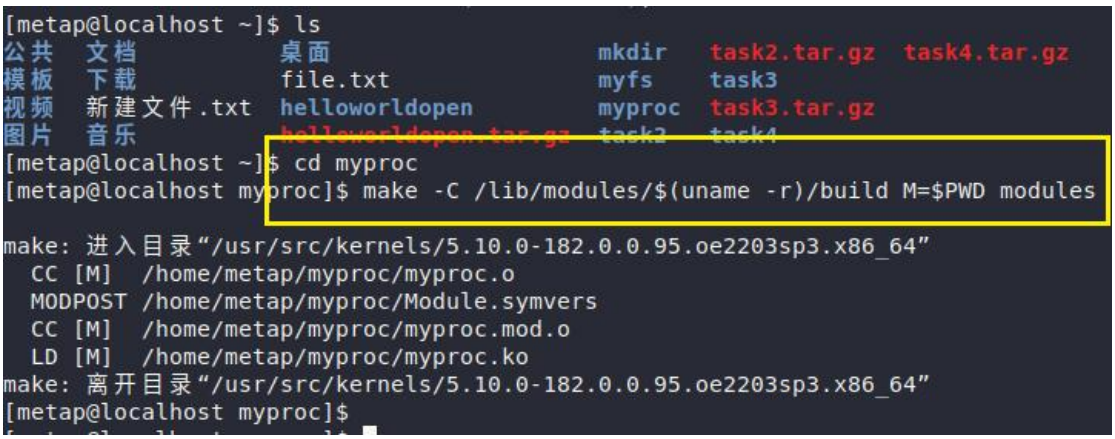
MODULE_AUTHOR("Your Name");

Makefile 文件

obj-m += myproc.o

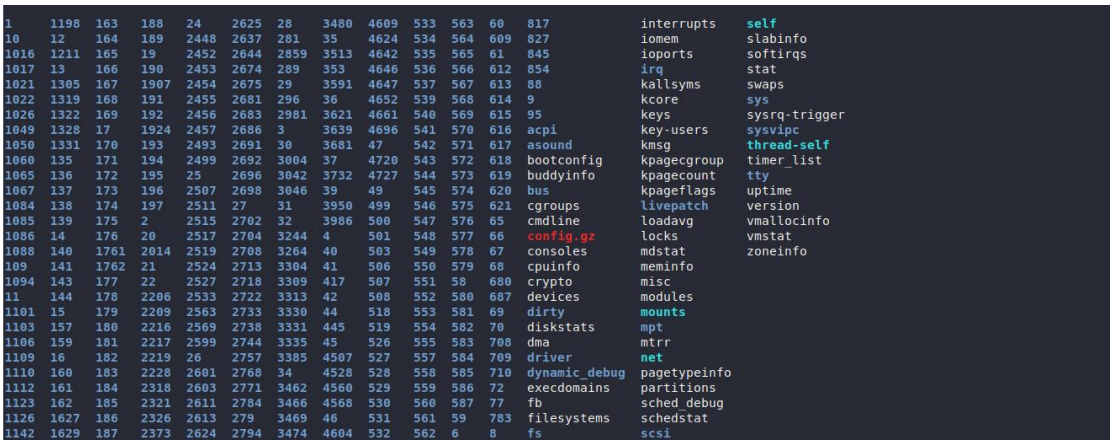
2、编译源文件。

make -C /lib/modules/$(uname -r)/build M=$PWD modules



3、对比加载内核模块前后的文件系统结果。

在加载模块之前



加载模块

sudo insmod myproc.ko

```
[metap@localhost myproc]$ sudo insmod myproc.ko
[sudo] metap 的密码：
[metap@localhost myproc]$
```

在加载模块之后

```
[metap@localhost myproc]$ ls /proc

1     139   186   2507  2771  37    528  571  708           keys
10    14    187   2511  2784  3732  529  572  709           key-users
1016  140   188   2515  279   39    530  573  710           kmsg
1017  141   189   2517  2794  3950  531  574  72            kpagecgroup
1021  143   19    2519  28    3986  532  575  77            kpagecount
1022  144   190   2524  281   4     533  576  783           kpageflags
1026  15    1907  2527  2859  40    534  577  8             livepatch
1049  157   191   2533  289   41    535  578  817           loadavg
1050  159   192   2563  29    417   536  579  827           locks
1060  16    1924  2569  296   42    537  58   845           mdstat
1065  160   193   2599  2981  44    539  580  854           meminfo
1067  161   194   26    3     445   540  581  88            misc
1084  162   195   2601  30    45    541  582  9             modules
1085  1627  196   2603  3004  4507  542  583  95            mounts
1086  1629  197   2611  3042  4528  543  584  acpi          mpt
1088  163   2     2613  3046  4560  544  585  asound        mtrr
109   164   20    2624  31    4568  545  586  bootconfig    myproc
1094  165   2014  2625  32    46    546  587  buddyinfo     net
11    166   21    2637  3244  4604  547  59   bus           pagetypeinfo
1101  167   22    2644  3264  4609  548  6    cgroups       partitions
1103  168   2206  2674  3304  4624  549  60   cmdline       sched_debug
1106  169   2209  2675  3309  4642  550  609  config.gz     schedstat
1109  17    2216  2681  3313  4646  551  61   consoles      scsi
1110  170   2217  2683  3330  4647  552  612  cpuinfo       self
1112  171   2219  2686  3331  4652  553  613  crypto        slabinfo
1123  172   2228  2691  3335  4661  554  614  devices       softirqs
1126  173   2318  2692  3385  4696  555  615  dirty         stat
1142  174   2321  2696  34    4697  557  616  diskstats     swaps
1198  175   2326  2698  3462  47    558  617  dma           sys
12    176   2373  27    3466  49    559  618  driver        sysrq-trigger
1211  1761  24    2702  3469  499   560  619  dynamic_debug sysvipc
13    1762  2448  2704  3474  500   561  620  execdomains   thread-self
1305  177   2452  2708  3480  501   562  621  fb            timer_list
1319  178   2453  2713  35    503   563  65   filesystems   tty
1322  179   2454  2718  3513  506   564  66   fs            uptime
1328  180   2455  2722  353   507   565  67   interrupts    version
1331  181   2456  2733  3591  508   566  68   iomem         vmallocinfo
135   182   2457  2738  36    518   567  680  ioports       vmstat
136   183   2493  2744  3621  519   568  687  irq           zoneinfo
137   184   2499  2757  3639  526   569  69   kallsyms
138   185   25    2768  3681  527   570  70   kcore
```

4、卸载内核模块，并查看结果。

sudo rmmod myproc

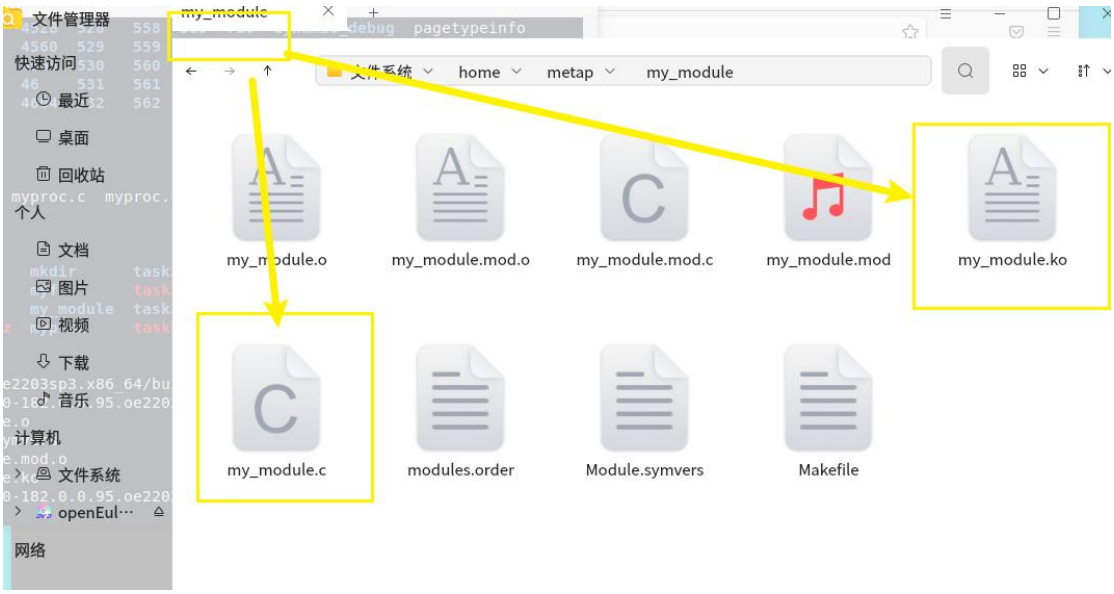
ls /proc

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

[sudo] metap 的密码：
对不起，请重试。
[sudo] metap 的密码：
[metap@localhost myproc]$ ls /proc

1     1198  163   188   24    2625  28    3480  4609  533  563  60   817          interrupts    self
10    12    164   189   2448  2637  281   35    4624  534  564  609  827          iomem         slabinfo
1016  1211  165   19    2452  2644  2859  3513  4642  535  565  61   845          ioports       softirqs
1017  13    166   190   2453  2674  289   353   4646  536  566  612  854          irq           stat
1021  1305  167   1907  2454  2675  29    3591  4647  537  567  613  88           kallsyms      swaps
1022  1319  168   191   2455  2681  296   36    4652  539  568  614  9            kcore         sys
1026  1322  169   192   2456  2683  2981  3621  4661  540  569  615  95           keys          sysrq-trigger
1049  1328  17    1924  2457  2686  3     3639  4696  541  570  616  acpi         key-users     sysvipc
1050  1331  170   193   2493  2691  30    3681  47    542  571  617  asound       kmsg          thread-self
1060  135   171   194   2499  2692  3004  37    4720  543  572  618  bootconfig   kpagecgroup   timer_list
1065  136   172   195   25    2696  3042  3732  4727  544  573  619  buddyinfo    kpagecount    tty
1067  137   173   196   2507  2698  3046  39    49    545  574  620  bus          kpageflags    uptime
1084  138   174   197   2511  27    31    3950  499   546  575  621  cgroups      livepatch     version
1085  139   175   2     2515  2702  32    3986  500   547  576  65   cmdline      loadavg       vmallocinfo
1086  14    176   20    2517  2704  3244  4     501   548  577  66   config.gz    locks         vmstat
1088  140   1761  2014  2519  2708  3264  40    503   549  578  67   consoles     mdstat        zoneinfo
109   141   1762  21    2524  2713  3304  41    506   550  579  68   cpuinfo      meminfo
1094  143   177   22    2527  2718  3309  417   507   551  58   680  crypto       misc
11    144   178   2206  2533  2722  3313  42    508   552  580  687  devices      modules
1101  15    179   2209  2563  2733  3330  44    518   553  581  69   dirty        mounts
1103  157   180   2216  2569  2738  3331  445   519   554  582  70   diskstats    mpt
1106  159   181   2217  2599  2744  3335  45    526   555  583  708  dma          mtrr
1109  16    182   2219  26    2757  3385  4507  527   557  584  709  driver       net
1110  160   183   2228  2601  2768  34    4528  528   558  585  710  dynamic_debug pagetypeinfo
1112  161   184   2318  2603  2771  3462  4560  529   559  586  72   execdomains  partitions
1123  162   185   2321  2611  2784  3466  4568  530   560  587  77   fb           sched_debug
1126  1627  186   2326  2613  279   3469  46    531   561  59   783  filesystems  schedstat
1142  1629  187   2373  2624  2794  3474  4604  532   562  6    8    fs           scsi
[metap@localhost myproc]$
[metap@localhost myproc]$ PP
```

4 任务 4：使用 sysfs 文件系统传递内核模块参数

实验步骤

1、正确编写满足功能的源文件，包括.c 源文件和 Makefile 文件。



C 语言文件

#include <linux/module.h>

```
#include &lt;linux/kernel.h&gt;
#include &lt;linux/init.h&gt;
#include &lt;linux/moduleparam.h&gt;

static char *my_string = "default_value";
module_param(my_string, charp, S_IRUGO);
MODULE_PARM_DESC(my_string, "A string parameter");

static int __init my_module_init(void)
{
printk(KERN_INFO "My module loaded with string parameter: %s\n", my_string);
return 0;
}

static void __exit my_module_exit(void)
{
printk(KERN_INFO "My module unloaded\n");
}

module_init(my_module_init);
module_exit(my_module_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Your Name");
```

Makefile 文件

```
obj-m += my_module.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
```

2、编译源文件。

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

3、加载编译完成的内核模块，并查看加载结果。

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

```
[metap@localhost my_module]$ sudo dmesg | tail
[   37.558383] Bluetooth: RFCOMM TTY layer initialized
[   37.558390] Bluetooth: RFCOMM socket layer initialized
[   37.558438] Bluetooth: RFCOMM ver 1.11
[  490.700432] myproc: loading out-of-tree module taints kernel.
[  490.700491] myproc: module verification failed: signature and/or required key missing - tainting ke
rnel
[  490.729413] Created /proc/myproc directory
[ 1091.647179] e1000: ens33 NIC Link is Down
[ 1097.803023] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: None
[ 1271.407026] Removed /proc/myproc directory
[ 2070.263063] My module loaded with string parameter: default_value
[metap@localhost my_module]$
```

4、卸载内核模块，并查看结果。

```
sudo rmmod my_module
sudo insmod my_module.ko
```

```
[metap@localhost my_module]$
[metap@localhost my_module]$ sudo dmesg | tail
[   37.558390] Bluetooth: RFCOMM socket layer initialized
[   37.558438] Bluetooth: RFCOMM ver 1.11
[  490.700432] myproc: loading out-of-tree module taints kernel.
[  490.700491] myproc: module verification failed: signature and/or required key missing - tainting kernel
[  490.729413] Created /proc/myproc directory
[ 1091.647179] e1000: ens33 NIC Link is Down
[ 1097.803023] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: None
[ 1271.407026] Removed /proc/myproc directory
[ 2070.263063] My module loaded with string parameter: default_value
[ 2400.096753] My module unloaded
[metap@localhost my_module]$
[metap@localhost my_module]$
```

在卸载内核模块之前，最后一行日志是模块加载时打印的消息，指示模块已经加载，并显示了传递的字符串参数的默认值。

在卸载内核模块之后，最后一行日志是模块卸载时打印的消息，指示模块已经成功卸载。这证实了模块的生命周期，从加载到卸载的过程。

给出的代码中，使用了 module_param 宏来声明模块参数，并指定了访问权限为

S_IRUGO，这样就使得这个参数能够通过 sysfs 文件系统传递给内核模块。具体来说，以下这行代码是声明了一个名为 my_string 的模块参数，并指定了访问权限为读取（S_IRUGO）

三、实验分析

本次计算机操作系统实验涉及了文件系统的多个方面，包括扩展属性、自定义文件系统类型的注册、在/proc 目录下创建目录以及使用 sysfs 文件系统传递内核模块参数。以下是对每个任务的实验分析与总结：

任务一：为 Ext4 文件系统添加扩展属性

在这个任务中，我们学习了文件系统扩展属性的概念以及如何为 Ext4 文件系统添加扩展属性。通过使用命令行工具 setfattr，我们可以为文件添加额外的键值对信息，并使用 getfattr 命令来检索这些信息。需要注意的是，要确保文件系统挂载时启用了 user_xattr 选项，并且要注意属性的编码格式。

任务二：注册一个自定义的文件系统类型

这个任务涉及了注册一个自定义的文件系统类型。我们学习了如何使用 register_filesystem 和 unregister_filesystem 函数来注册和注销文件系统类型。在编写文件系统类型时，我们需要填充一个 file_system_type 结构体，并在模块加载时调用 register_filesystem 函数进行注册。

任务三：在/proc 下创建目录

在这个任务中，我们学习了如何在/proc 目录下创建目录。通过使用 proc_mkdir 函数，我们可以在/proc 目录中添加一个新的子目录。这个功能可以用来向用户空间提供内核模块的信息或者控制接口。

任务四：使用 sysfs 文件系统传递内核模块参数

最后一个任务涉及了使用 sysfs 文件系统传递内核模块参数。我们学习了如何使用 module_param 宏来声明模块参数，并在加载模块后通过/sys 目录与模块进行交互。使用 echo 命令可以向内核传递参数，实现对模块参数的设置。

实验总结

本次实验通过涉及文件系统的多个方面，加深了对操作系统中文件系统管理的理解。从添加文件系统扩展属性到注册自定义文件系统类型，再到在/proc 目录下创建目录和使用 sysfs 传递内核模块参数，每个任务都展示了文件系统在操

作系统中的重要作用。通过实际操作，我们掌握了如何与文件系统进行交互，实现对文件和内核模块的管理和控制。这些知识和技能对于深入理解操作系统的内核机制以及开发系统级应用程序都具有重要意义。

五、所遇问题及解决方法

问题

[metap@localhost my_module]$ dmesg | tail dmesg: 读取内核缓冲区失败: Operation not permitted [metap@localhost my_module]$

解决方法

sudo dmesg | tail

问题

[metap@localhost myfs]$ ls Makefile myfs.c [metap@localhost myfs]$ make make -C /lib/modules/5.10.0-182.0.0.95.oe2203sp3.x86_64/build M=/home/metap/myfs modules make[1]: 进入目录“/usr/src/kernels/5.10.0-182.0.0.95.oe2203sp3.x86_64” CC [M] /home/metap/myfs/myfs.o /home/metap/myfs/myfs.c:16:10: 错误：initialization of ‘struct dentry * (*)(struct file_system_type *, int, const char *, void *)’ from incompatible pointer type ‘int (*)(struct file_system_type *, int, const char *, void *)’ [-Werror=incompatible-pointer-types] 16 | .mount = myfs_mount, | ^~~~~~~~~~ /home/metap/myfs/myfs.c:16:10: 附注：(在‘my_fs_type.mount’的初始化附近) cc1：有些警告被当作是错误 make[2]: *** [scripts/Makefile.build:286：/home/metap/myfs/myfs.o] 错误 1 make[1]: *** [Makefile:1850：/home/metap/myfs] 错误 2 make[1]: 离开目录“/usr/src/kernels/5.10.0-182.0.0.95.oe2203sp3.x86_64” make: *** [Makefile:4：all] 错误 2

解决方法

make 前面加上制表符

问题

myfs.c 文件中的 myfs_mount 函数的签名与 struct file_system_type 结构体中的 mount 函数的预期签名不匹配

解决方法

检查 myfs_mount 函数的定义与 struct file_system_type 中 mount 函数的预期签名是否一致。确保它们具有相同的参数和返回类型