|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_\_\_\_\_«Информатика и системы управления»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_ «Программное обеспечение ЭВМ и информационные технологии»\_\_\_\_\_\_\_\_\_

**Лабораторная работа № 8**

|  |  |
| --- | --- |
| **Дисциплина** Операционные системы  **Тема** \_Виртуальная файловая система myfs\_  **Студент** \_Ильясов И. М.\_  **Группа** \_ИУ7-63Б\_  **Оценка (баллы)** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Преподаватель** \_Рязанова Н. Ю.\_ |  |

Москва, 2020 г.

**Задание на лабораторную работу**

Создать виртуальную файловую систему myfs, используя наработки лабораторной работы по загружаемым модулям ядра (ЛР3).

**Пример работы программы**

На приведенных ниже рисунках продемонстрирована работа программы из лабораторной работы.

На рисунке 1 приведен результат сборки загружаемого модуля ядра при помощи утилиты make.

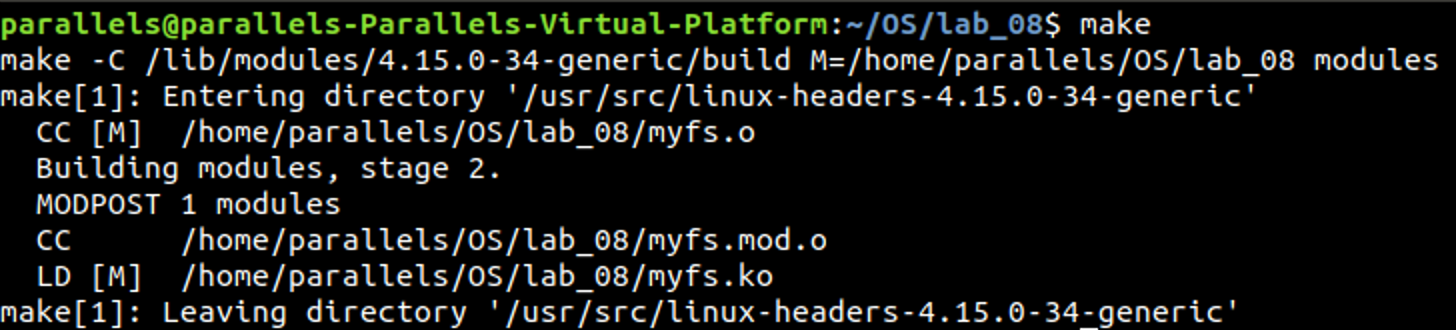


Рисунок 1 – Сборка загружаемого модуля ядра при помощи make

На следующем рисунке продемонстрировано создание образа диска image и каталога dir, который является точкой монтирования.

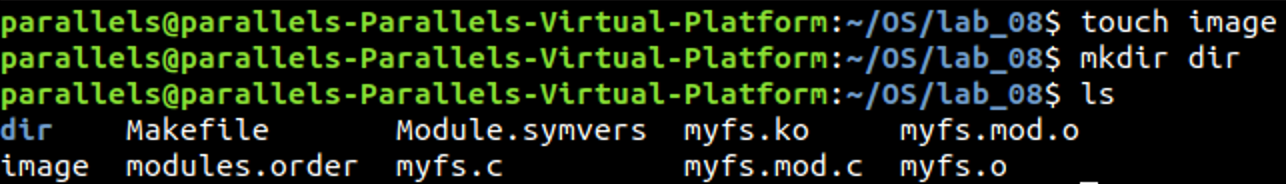


Рисунок 2 – Создание образа диска image и каталога dir

На рисунке 3 представлена команда загрузки модуля и демонстрация успешности этой загрузки.

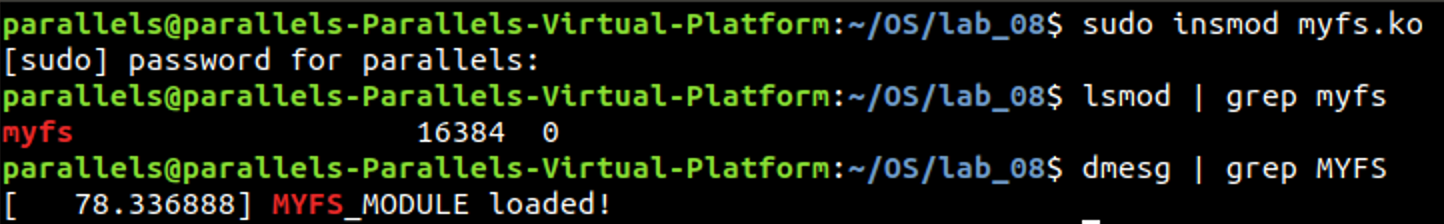


Рисунок 3 – Загрузка модуля myfs

Далее на рисунке 4 показан процесс монтирования виртуальной файловой системы, выведено сообщение об успешности монтирования виртуальной файловой системы и приведена информация о ней.

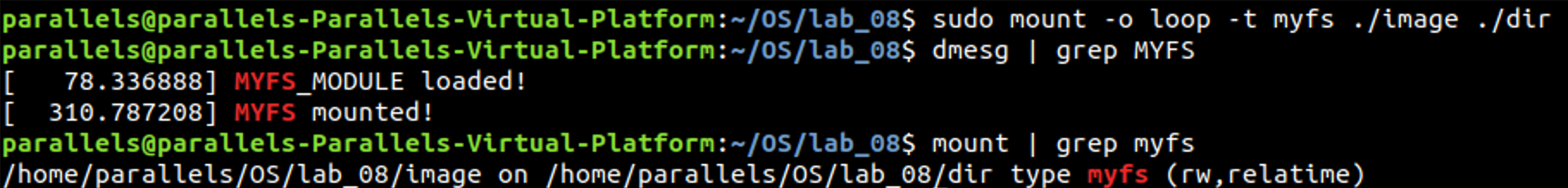


Рисунок 4 – Монтирование виртуальной файловой системы myfs

При этом виртуальная файловая система отобразилась также в проводнике (что показано на рисунке 5).

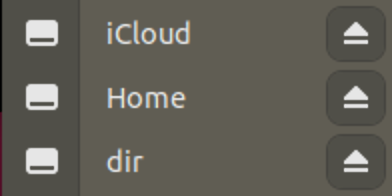


Рисунок 5 – myfs в проводнике

Рисунок 6 – размонтируем виртуальную файловую системы myfs, выгрузим модуль и просмотрим сообщения от модуля.

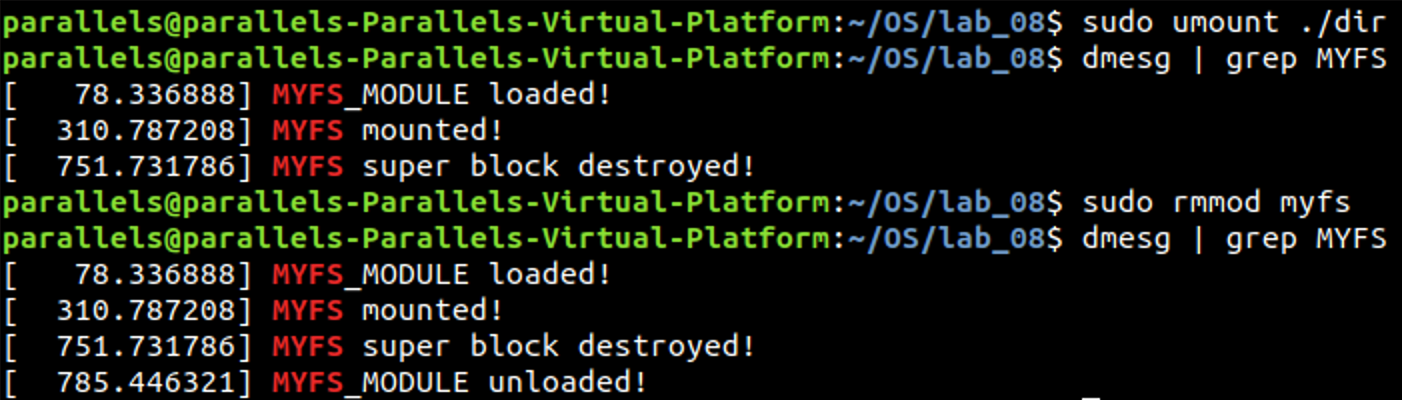


Рисунок 6 – Размонтирование виртуальной файловой системы myfs

При этом стоит отметить, что виртуальная файловая система также исчезла из проводника.

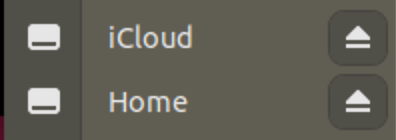


Рисунок 7 – myfs больше не отображается в проводнике

**Листинг программы**

Ниже в листингах приведен код программы и содержимое Makefile. Так, в листинге 1 показано содержимое файла Makefile.

**Листинг 1 – содержимое Makefile**

ifneq ($(KERNELRELEASE),)

obj-m := myfs.o

else

CURRENT = $(shell uname -r)

KDIR = /lib/modules/$(CURRENT)/build

PWD = $(shell pwd)

default:

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

clean:

rm -rf .tmp\_versions

rm \*.ko

rm \*.o

rm \*.mod.c

rm \*.symvers

rm \*.order

endif

В листинге 2 приведено содержимое файла myfs.c.

**Листинг 2 – содержимое myfs.c**

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/fs.h>

#include <linux/time.h>

#include <linux/slab.h>

MODULE\_LICENSE("GPL");

MODULE\_DESCRIPTION("MYFS\_MODULE");

MODULE\_AUTHOR("Ilyasov Idris. ICS7-63B");

#define SLABNAME "myfs\_inode\_cache"

static struct inode \*\*myfs\_inodes;

static const unsigned long MYFS\_MAGIC\_inode\_cnt = 0x13131313;

static int inode\_cnt = 0;

struct kmem\_cache \*cache;

module\_param(inode\_cnt, int, 0);

struct myfs\_inode

{

int i\_mode;

unsigned long i\_ino;

};

static int size = sizeof(struct myfs\_inode);

static struct inode\* myfs\_make\_inode(struct super\_block \*sb, int mode)

{

struct inode\* ret = new\_inode(sb);

struct myfs\_inode \*myfs\_inode;

if (ret)

{

inode\_init\_owner(ret, NULL, mode);

ret->i\_size = PAGE\_SIZE;

ret->i\_atime = ret->i\_mtime = ret->i\_ctime = current\_time(ret);

myfs\_inode = kmem\_cache\_alloc(cache, GFP\_KERNEL);

\*myfs\_inode = (struct myfs\_inode)

{

.i\_mode = ret->i\_mode,

.i\_ino = ret->i\_ino,

};

ret->i\_private = myfs\_inode;

}

return ret;

}

static int myfs\_drop\_inode(struct inode \*inode)

{

kmem\_cache\_free(cache, inode->i\_private);

return generic\_drop\_inode(inode);

}

static void myfs\_put\_super(struct super\_block \*sb)

{

printk(KERN\_DEBUG "MYFS super block destroyed!\n");

}

static struct super\_operations const myfs\_super\_ops = {

.put\_super = myfs\_put\_super,

.statfs = simple\_statfs,

.drop\_inode = myfs\_drop\_inode,

};

static int myfs\_fill\_sb(struct super\_block \*sb, void \*data, int silent)

{

struct inode \*root = NULL;

sb->s\_blocksize = PAGE\_SIZE;

sb->s\_blocksize\_bits = PAGE\_SHIFT;

sb->s\_magic = MYFS\_MAGIC\_inode\_cnt;

sb->s\_op = &myfs\_super\_ops;

root = myfs\_make\_inode(sb, S\_IFDIR | 0755);

if (!root)

{

printk(KERN\_ERR "MYFS inode allocation failed!\n");

return -ENOMEM;

}

root->i\_op = &simple\_dir\_inode\_operations;

root->i\_fop = &simple\_dir\_operations;

sb->s\_root = d\_make\_root(root);

if (!sb->s\_root)

{

printk(KERN\_ERR "MYFS root creation failed!\n");

iput(root);

return -ENOMEM;

}

return 0;

}

static struct dentry\* myfs\_mount(struct file\_system\_type \*type, int flags, char const \*dev, void \*data)

{

struct dentry\* const entry = mount\_nodev(type, flags, data, myfs\_fill\_sb);

if (IS\_ERR(entry))

{

printk(KERN\_ERR "MYFS mounting failed!\n");

}

else

{

printk(KERN\_DEBUG "MYFS mounted!\n");

}

return entry;

}

static struct file\_system\_type myfs\_type = {

.owner = THIS\_MODULE,

.name = "myfs",

.mount = myfs\_mount,

.kill\_sb = kill\_abon\_super,

};

static int \_\_init myfs\_init(void)

{

int ret = register\_filesystem(&myfs\_type);

if (ret != 0)

{

printk(KERN\_ERR "MYFS\_MODULE cannot register filesystem!\n");

return ret;

}

myfs\_inodes = kmalloc(sizeof(struct inode \*) \* inode\_cnt, GFP\_KERNEL);

if (myfs\_inodes == NULL)

{

printk(KERN\_ERR "MYFS kmalloc error!\n");

kfree(myfs\_inodes);

return -ENOMEM;

}

cache = kmem\_cache\_create(SLABNAME, size, 0, SLAB\_POISON, NULL);

if (cache == NULL)

{

printk(KERN\_ERR "MYFS kmem\_cache\_create error!\n");

kmem\_cache\_destroy(cache);

kfree(myfs\_inodes);

return -ENOMEM;

}

printk(KERN\_DEBUG "MYFS\_MODULE loaded!\n");

return 0;

}

static void \_\_exit myfs\_exit(void)

{

int i = 0;

int ret;

for (i = 0; i < inode\_cnt; i++)

{

myfs\_drop\_inode(myfs\_inodes[i]);

}

ret = unregister\_filesystem(&myfs\_type);

if (ret != 0)

{

printk(KERN\_ERR "MYFS\_MODULE cannot unregister filesystem!\n");

}

kmem\_cache\_destroy(cache);

kfree(myfs\_inodes);

printk(KERN\_DEBUG "MYFS\_MODULE unloaded!\n");

}

module\_init(myfs\_init);

module\_exit(myfs\_exit);