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.

```
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_08$ make
make -C /lib/modules/4.15.0-34-generic/build M=/home/parallels/OS/lab_08 modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-34-generic'
  CC [M]  /home/parallels/OS/lab_08/myfs.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/parallels/OS/lab_08/myfs.mod.o
  LD [M]  /home/parallels/OS/lab_08/myfs.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-34-generic'
```

Рисунок 1 – Сборка загружаемого модуля ядра при помощи make

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

```
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_08$ touch image
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_08$ mkdir dir
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_08$ ls
dir    Makefile       Module.symvers  myfs.ko     myfs.mod.o
image  modules.order  myfs.c          myfs.mod.c  myfs.o
```

Рисунок 2 – Создание образа диска image и каталога dir

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

```
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_08$ sudo insmod myfs.ko
[sudo] password for parallels:
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_08$ lsmod | grep myfs
myfs                   16384  0
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_08$ dmesg | grep MYFS
[   78.336888] MYFS_MODULE loaded!
```

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.

```
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_08$ sudo mount -o loop -t myfs ./image ./dir
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_08$ dmesg | grep MYFS
[   78.336888] MYFS_MODULE loaded!
[  310.787208] MYFS mounted!
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_08$ mount | grep myfs
/home/parallels/OS/lab_08/image on /home/parallels/OS/lab_08/dir type myfs (rw,relatime)
```

Рисунок 4 – Монтирование виртуальной файловой системы myfs

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

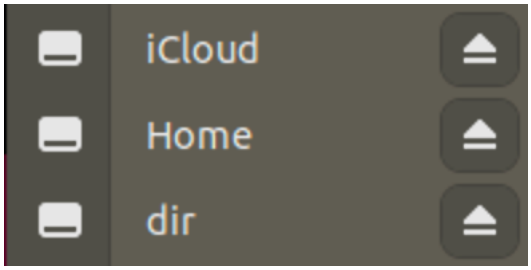

Рисунок 5 – myfs в проводнике

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

```
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_08$ sudo umount ./dir
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_08$ dmesg | grep MYFS
[   78.336888] MYFS_MODULE loaded!
[  310.787208] MYFS mounted!
[  751.731786] MYFS super block destroyed!
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_08$ sudo rmmod myfs
parallels@parallels-Parallels-Virtual-Platform:~/OS/lab_08$ dmesg | grep MYFS
[   78.336888] MYFS_MODULE loaded!
[  310.787208] MYFS mounted!
[  751.731786] MYFS super block destroyed!
[  785.446321] MYFS_MODULE unloaded!
```

Рисунок 6 – Размонтирование виртуальной файловой системы myfs

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

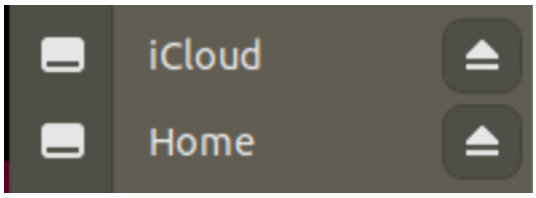

Рисунок 7 – myfs больше не отображается в проводнике

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);
```