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

Создание виртуальной файловой системы.

Дисциплина «Операционные системы»

Студент: Зейналов З. Г.

Группа: ИУ7-61Б

Преподаватель: Рязанова Н. Ю.

Москва.
2020 г.

Листинг программы

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

#define MYFS_MAGIC_NUMBER 0x13131313;
#define SLABNAME "my_cache"
struct myfs_inode
{
    int i_mode;
    unsigned long i_ino;
};

int inode_number = 0;
static struct kmem_cache *cache;



static void myfs_put_super(struct super_block *sb)
{
    printk(KERN_DEBUG "MYFS super block destroyed\n");
}

int free_allocated_inodes(struct inode *inode)
{
    kmem_cache_free(cache, inode->i_private);
    return 1;
}

static struct super_operations const myfs_super_ops = {
    .put_super = myfs_put_super,
    .statfs = simple_statfs,
    .drop_inode = free_allocated_inodes,
};

static struct inode *myfs_make_inode(struct super_block *sb, int mode)
{
    struct inode *ret = new_inode(sb);
    if (ret)
    {
        struct myfs_inode *my_inode = kmem_cache_alloc(cache, GFP_KERNEL);
        inode_init_owner(ret, NULL, mode);
        *my_inode = (struct myfs_inode){
            .i_mode = ret->i_mode,
            .i_ino = ret->i_ino
        };
        ret->i_size = PAGE_SIZE;
```

```
        ret->i_atime = ret->i_mtime = ret->i_ctime = current_time(ret);
        ret->i_private = my_inode;

    }
    return ret;
}


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_NUMBER;
    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_bdev(type, flags, dev, data, myfs_fill_sb)
;
    if (IS_ERR(entry))
        printk(KERN_ERR "MYFS mounting failed!\n");
    else
        printk(KERN_DEBUG "MYFS mounted");
    return entry;

}

static struct file_system_type myfs_type = {
```

```
    .owner = THIS_MODULE,
    .name = "myfs",
    .mount = myfs_mount,
    .kill_sb = kill_block_super,
};

void co (void *p)
{
    *(int *)p = (int)p;
    inode_number++;
}

static int __init myfs_init(void)
{
    int ret = register_filesystem(&myfs_type);
    cache = kmem_cache_create(SLABNAME, sizeof(struct myfs_inode), 0, 0, co);
    if (ret != 0)
    {
        printk(KERN_ERR "MYFS can't register filesystem\n");
        return ret;
    }
    printk(KERN_INFO "MYFS filesystem registered");
    return 0;
}

static void __exit myfs_exit(void)
{
    int ret = unregister_filesystem(&myfs_type);
    if (ret != 0)
        printk(KERN_ERR "MYFS can't unregister filesystem!\n");
    kmem_cache_destroy(cache);
    printk(KERN_INFO "MYFS unregistered %d", inode_number);
}

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Zeynalov Zeynal");

module_init(myfs_init);
module_exit(myfs_exit);
```

Листинг Makefile’a

```
ifneq ($(KERNELRELEASE),)
    obj-m   := test.o
else
    CURRENT = $(shell uname -r)
    KDIR = /lib/modules/$(CURRENT)/build
    PWD = $(shell pwd)
default:
    $(MAKE) -C $(KDIR) M=$(PWD) modules
```

```
clean:
    @rm -f *.o .*.cmd .*.flags *.mod.c *.order
    @rm -f .*.*.cmd *~ *.*~ TODO.*
    @rm -fR .tmp*
    @rm -rf .tmp_versions
disclean: clean
    @rm *.ko *.symvers

endif
```

Результаты работы программы

Для создания виртуальной файловой системы необходимо скомпилировать загружаемый модуль ядра и загрузить его с помощью команды insmod.

```
zeynal@zeynal-VirtualBox:~/Zeyanl/sem2/lab8_my_vfs$ make
make -C /lib/modules/5.0.0-23-generic/build  M=/home/zeynal/Zeyanl/sem2/lab8_my_vfs modules
make[1]: вход в каталог «/usr/src/linux-headers-5.0.0-23-generic»
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: выход из каталога «/usr/src/linux-headers-5.0.0-23-generic»
```

Рисунок 1 - сборка модуля ядра

```
zeynal@zeynal-VirtualBox:~/Zeyanl/sem2/lab8_my_vfs$ sudo insmod test.ko
zeynal@zeynal-VirtualBox:~/Zeyanl/sem2/lab8_my_vfs$ lsmod | grep test
test                   16384  0
```

Рисунок 2 - загрузка модуля ядра.

Создадим образ диска (touch image)и корень файловой системы (mkdir dir).

Смонтируем (sudo mount -o loop -t myfs ./image ./dir) файловую систему, размонтируем (sudo umount ./dir) ее и посмотрим в syslog.

```
zeynal@zeynal-VirtualBox:~/Zeyanl/sem2/lab8_my_vfs$ tail -f /var/log/syslog | grep MYFS
Apr 23 01:34:48 zeynal-VirtualBox kernel: [10994.832959] MYFS unregistered 0
Apr 23 01:48:13 zeynal-VirtualBox kernel: [11333.889776] MYFS filesystem registered
Apr 23 01:48:29 zeynal-VirtualBox kernel: [12138.814443] MYFS mounted
Apr 23 01:48:29 zeynal-VirtualBox kernel: [12154.524902] MYFS super block destroyed
Apr 23 01:51:07 zeynal-VirtualBox kernel: [12173.544725] MYFS unregistered 170
```

Рисунок 3 - Вывод отладочной информации в syslog

На рисунке 4 продемонстрировано состояние Slab кэша до монтирования системы.

```
zeynal@zeynal-VirtualBox:~/Zeyanl/sem2/lab8_my_vfs$ sudo cat /proc/slabinfo | grep my_cache
my_cache               0      0     24  170    1 : tunables    0    0    0 : slabdata      0      0
      0
```

Рисунок 4 - Состояние Slab кэша до монтирования системы.

На рисунке 5 представлено состояние Slab кэша после монтирования системы.

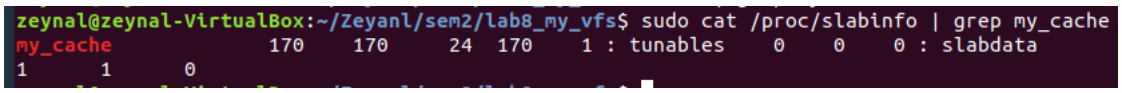

Рисунок 5 - Состояние Slab кэша после монтирования системы.

Информация о смонтированной файловой системе представлена на рисунке 6

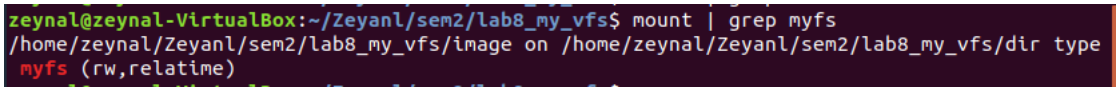

Рисунок 6 - Вывод информации о монтированной файловой системе

Вывод

В результате проделанной работы были получены навыки создания и монтирования виртуальной файловой системы. Также была изучена возможность создания slab кэша для inode.