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»

О Т Ч Е Т

по лабораторной работе № ____

Дисциплина: *Операционные системы*

Студент	***ИУ7И-66Б***		**Нгуен Ф. С.**
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			**Рязанова Н. Ю.**
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2021

код 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"

MODULE_LICENSE("Dual BSD/GPL");
MODULE_AUTHOR("Nguyensanghso@gmail.com");

static int sco = 0;

static struct kmem_cache *cache = NULL;
static void* *line = NULL;

static int size = 7;
module_param(size, int, 0);
static int number = 31;
module_param(number, int, 0);


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


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 = free_allocated_inodes,
};


struct myfs_inode
{
    int i_mode;
    unsigned long i_ino;
} myfs_inode;


static struct inode *myfs_make_inode(struct super_block *sb, int mode)
{
    struct inode *ret = new_inode(sb);

    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);
        ret->i_private = &myfs_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\n");
    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;
    sco++;
}


static int __init myfs_init(void)
{
    int i, ret;
```

```
    if(size < 0)
    {
        printk(KERN_ERR "MYFS invalid argument %d\n", size);
        return -EINVAL;
    }

    line = kmalloc(sizeof(void*) * number, GFP_KERNEL);
    if(!line)
    {
        printk(KERN_ERR "MYFS kmalloc error\n");
        kfree(line);
        return -ENOMEM;
    }
    for(i = 0; i < number; i++)
        line[i] = NULL;

    cache = kmem_cache_create(SLABNAME, sizeof(struct myfs_inode), 0, 0, co);
    if (!cache)
    {
        printk(KERN_ERR "MYFS_MODULE cannot allocate cache\n");
        kmem_cache_destroy(cache);
        return -ENOMEM;
    }
    for(i = 0; i < number; i++)
    {
        if(NULL == (line[i] = kmem_cache_alloc(cache, GFP_KERNEL)))
        {
            printk(KERN_ERR "MYFS kmem_cache_alloc error\n");
            for(i = 0; i < number; i++)
            kmem_cache_free(cache, line[i]);
        }
    }

    ret = register_filesystem(&myfs_type);
    if (ret != 0)
    {
        printk(KERN_ERR "MYFS_MODULE cannot register filesystem\n");
        return ret;
    }

    printk(KERN_INFO "MYFS allocate %d objects into slab: %s\n", number,
SLABNAME);
    printk(KERN_INFO "MYFS object size %d bytes, full size %ld bytes\n", size,
(long)size * number);
    printk(KERN_INFO "MYFS constructor called %d times\n", sco);
    printk(KERN_INFO "MYFS_MODULE filesystem loaded\n");
    return 0;
}


static void __exit myfs_exit(void)
{
    int i, ret;
    for(i = 0; i < number; i++)
        kmem_cache_free(cache, line[i]);
    kmem_cache_destroy(cache);
    kfree(line);

    ret = unregister_filesystem(&myfs_type);
    if (ret != 0)
        printk(KERN_ERR "MYFS_MODULE cannot unregister filesystem!\n");
    printk(KERN_INFO "MYFS_MODULE unloaded %d\n", sco);
}


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

```
nguyensang@K-virtual-machine:~/Desktop/OS2021/lab7/Test$ sudo insmod myfs.ko
[sudo] password for nguyensang:
nguyensang@K-virtual-machine:~/Desktop/OS2021/lab7/Test$ lsmod | head -5
Module                  Size  Used by
myfs                   16384  0
nls_utf8               16384  1
isofs                  49152  1
rfcomm                 81920  4
nguyensang@K-virtual-machine:~/Desktop/OS2021/lab7/Test$ touch image
nguyensang@K-virtual-machine:~/Desktop/OS2021/lab7/Test$ sudo mount -o loop -t myfs ./image ./Rdir
nguyensang@K-virtual-machine:~/Desktop/OS2021/lab7/Test$
```

```
nguyensang@K-virtual-machine:~/Desktop/OS2021/lab7/Test$ dmesg | tail -5
[ 6579.111186] MYFS allocate 31 objects into slab: my_cache
[ 6579.111187] MYFS object size 7 bytes, full size 217 bytes
[ 6579.111187] MYFS constructor called 170 times
[ 6579.111188] MYFS_MODULE filesystem loaded
[ 6698.057273] MYFS mounted
nguyensang@K-virtual-machine:~/Desktop/OS2021/lab7/Test$
```