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»

ОТЧЁТ

По лабораторной работе №8

По курсу: «Операционные системы»

Тема: «Создание виртуальной файловой системы»

Студент: Керимов А. Ш.

Группа: ИУ7-64Б

Преподаватель: Рязанова Н. Ю.

Москва

2020

Виртуальная файловая система

Листинг 1: `myfs.c`

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

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Kerimov A. IU7-64b");
MODULE_DESCRIPTION("my filesystem");

#define MYFS_MAGIC_NUMBER 0xDEADBEAF
#define SLABNAME "myfs_inode_cache"

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

static struct kmem_cache *myfs_inode_cachep = NULL;
static struct myfs_inode **myfs_inodes = NULL;
static int cache_inode_count = 128;
module_param(cache_inode_count, int, 0);
static int busy = 0;

static struct myfs_inode *cache_get_inode(void)
{
        if (busy == cache_inode_count) {
                return NULL;
        }

        return myfs_inodes[busy++] = kmem_cache_alloc(myfs_inode_cachep, GFP_KERNEL);
}

static struct inode *myfs_make_inode(struct super_block *sb, int mode)
{
        struct myfs_inode *myfs_inodep = NULL;
        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);
                if ((myfs_inodep = cache_get_inode())) {
                        myfs_inodep->i_mode = ret->i_mode;
                        myfs_inodep->i_ino = ret->i_ino;
                }
                ret->i_private = myfs_inodep;
        }

        return ret;
}

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

static const struct super_operations myfs_super_ops = {
        .put_super  = myfs_put_super,
        .statfs     = simple_statfs,
        .drop_inode = generic_delete_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_NUMBER;
        sb->s_op = &myfs_super_ops;

        if (!(root = myfs_make_inode(sb, S_IFDIR | 0755))) {
                printk(KERN_ERR "myfs: inode allocation failed\n");
                return -ENOMEM;
```

```
74          }
75          root->i_op = &simple_dir_inode_operations;
76          root->i_fop = &simple_dir_operations;
77
78          if (!(sb->s_root = d_make_root(root))) {
79                  iput(root);
80                  printk(KERN_ERR "myfs: root creation failed\n");
81                  return -ENOMEM;
82          }
83
84          return 0;
85  }
86
87  static struct dentry *myfs_mount(struct file_system_type *type, int flags, __attribute__((unused))
        const char *dev, void *data)
88  {
89          struct dentry *entry = mount_nodev(type, flags, data, myfs_fill_sb);
90          if (IS_ERR(entry)) {
91                  printk(KERN_ERR "myfs: mounting failed\n");
92          } else {
93                  printk(KERN_DEBUG "myfs: mounted\n");
94          }
95          return entry;
96  }
97
98  static struct file_system_type myfs_type = {
99          .owner   = THIS_MODULE,
100         .name    = "myfs",
101         .mount   = myfs_mount,
102         .kill_sb = kill_anon_super,
103 };
104
105 static int __init myfs_init(void)
106 {
107         int ret = register_filesystem(&myfs_type);
108         if (ret) {
109                 printk(KERN_ERR "myfs: cannot register filesystem\n");
110                 return ret;
111         }
112
113         if (!(myfs_inodes = kmalloc(sizeof(struct myfs_inode *) * cache_inode_count, GFP_KERNEL))) {
114                 printk(KERN_ERR "myfs: kmalloc error\n");
115                 return -ENOMEM;
116         }
117
118         if (!(myfs_inode_cachep = kmem_cache_create(SLABNAME, sizeof(struct myfs_inode), 0,
              SLAB_POISON, NULL))) {
119                 kfree(myfs_inodes);
120                 printk(KERN_ERR "myfs: kmem_cache_create error\n");
121                 return -ENOMEM;
122         }
123
124         printk(KERN_DEBUG "myfs: module loaded\n");
125         return 0;
126 }
127
128 static void __exit myfs_exit(void)
129 {
130         int i;
131         for (i = 0; i < busy; ++i) {
132                 kmem_cache_free(myfs_inode_cachep, myfs_inodes[i]);
133         }
134         kmem_cache_destroy(myfs_inode_cachep);
135         kfree(myfs_inodes);
136
137         if (unregister_filesystem(&myfs_type)) {
138                 printk(KERN_ERR "myfs: cannot unregister filesystem\n");
139         }
140         printk(KERN_DEBUG "myfs: module unloaded\n");
141 }
142
143 module_init(myfs_init)
144 module_exit(myfs_exit)
```

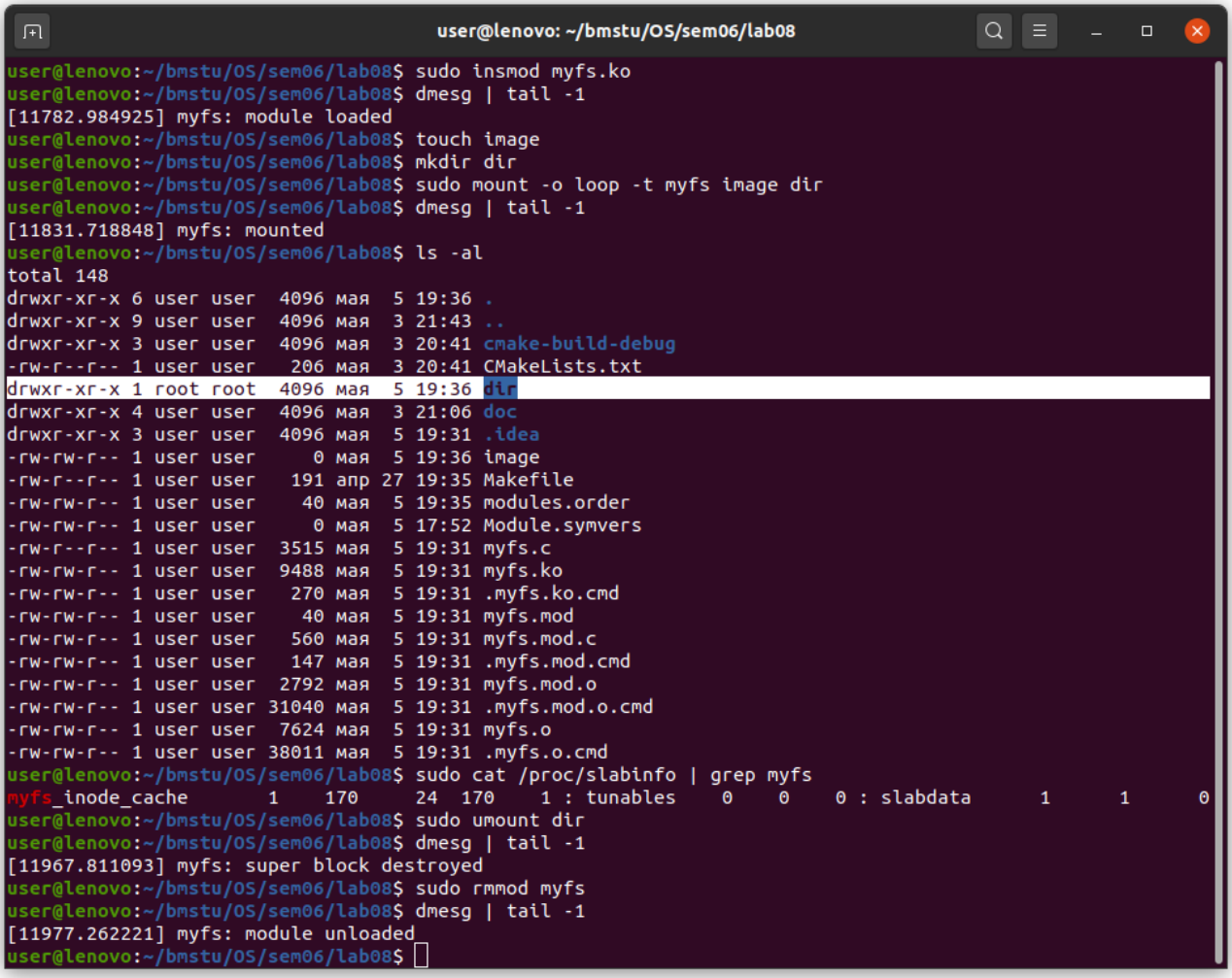

Рис. 1: Демонстрация работы программы