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
По курсу «Операционные системы»
Тема: Виртуальная файловая система

Студент: Кондрашова О.П.
Группа: ИУ7-65Б
Преподаватель: Рязанова Н.Ю.

Москва, 2020г.

Листинг 1. Код программы

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/pagemap.h>
#include <linux/fs.h>
#include <asm/atomic.h>
#include <asm/uaccess.h>
#include <linux/slab.h>

#define VFS_MAGIC_NUMBER 0x13131313
#define SLABNAME "vfs_cache"

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Kondrashova");
MODULE_DESCRIPTION("lab8");


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

static void* *line = NULL;

static void co(void* p)
{
  *(int*)p = (int)p;
}
struct kmem_cache *cache = NULL;

static struct vfs_inode
{
  int i_mode;
  unsigned long i_ino;
} vfs_inode;

// Создание inode
static struct inode * vfs_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 = &vfs_inode;
  }
  return ret;
}

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

// Операции структуры суперблок
static struct super_operations const vfs_super_ops = {
    .put_super = vfs_put_super,
    .statfs = simple_statfs,
    .drop_inode = generic_delete_inode,
};

// Функция инициализации суперблока
```

```
64.  // Создание корневого каталога ФС
65.  static int vfs_fill_sb(struct super_block *sb, void *data, int silent)
66.  {
67.    struct inode* root = NULL;
68.
69.    sb->s_blocksize = PAGE_SIZE;
70.    sb->s_blocksize_bits = PAGE_SHIFT;
71.    sb->s_magic = VFS_MAGIC_NUMBER;
72.    sb->s_op = &vfs_super_ops;
73.
74.    root = vfs_make_inode(sb, S_IFDIR | 0755);
75.    if (!root)
76.    {
77.      printk (KERN_ERR "VFS inode allocation failed\n");
78.      return -ENOMEM;
79.    }
80.
81.    root->i_op = &simple_dir_inode_operations;
82.    root->i_fop = &simple_dir_operations;
83.
84.    sb->s_root = d_make_root(root);
85.    if (!sb->s_root)
86.    {
87.      printk(KERN_ERR "VFS root creation failed\n");
88.      iput(root);
89.      return -ENOMEM;
90.    }
91.    return 0;
92.  }
93.
94.  // Монтирование ФС
95.  static struct dentry* vfs_mount(struct file_system_type *type, int flags, const char *d
ev, void *data)
96.  {
97.    struct dentry* const entry = mount_nodev(type, flags, data, vfs_fill_sb);
98.
99.    if (IS_ERR(entry))
100.     printk(KERN_ERR  "VFS mounting failed\n");
101.   else
102.     printk(KERN_DEBUG "VFS mounted\n");
103.
104.   return entry;
105. }
106.
107. static struct file_system_type vfs_type  =  {
108.   .owner  =  THIS_MODULE,       // Счетчик ссылок на модуль
109.   .name  =  "vfs",          // Название ФС
110.   .mount  =  vfs_mount,         // Функция, вызываемая при монтировании ФС
111.   .kill_sb  =  kill_litter_super,   // Функция, вызываемая при размонтировании ФС
112. };
113.
114. // Инициализация модуля
115. static int __init vfs_module_init(void)
116. {
117.   int i;
118.
119.   if(size < 0)
120.   {
121.     printk(KERN_ERR "VFS invalid sizeof objects\n");
122.     return -EINVAL;
123.   }
124.
125.   line = kmalloc(sizeof(void*) *number, GFP_KERNEL);
126.   if(!line)
127.   {
128.     printk(KERN_ERR "VFS kmalloc error\n");
129.     kfree(line);
130.     return -ENOMEM;
131.   }
```

```
132.
133.    for(i = 0; i < number; i++)
134.      line[i] = NULL;
135.
136.    // Создание кэша slab
137.    cache = kmem_cache_create(SLABNAME, size, 0, SLAB_HWCACHE_ALIGN, co);
138.
139.    if(!cache)
140.    {
141.      printk(KERN_ERR "VFS cannot create cache\n");
142.      // Уничтожение slab
143.      kmem_cache_destroy(cache);
144.      return -ENOMEM;
145.    }
146.
147.    for(i = 0; i < number; i++)
148.    {
149.      if(NULL == (line[i] = kmem_cache_alloc(cache, GFP_KERNEL)))
150.      {
151.        printk(KERN_ERR "VFS cannot alloc cache\n");
152.        for(i = 0; i < number; i++ )
153.          kmem_cache_free(cache, line[i]);
154.        return -ENOMEM;
155.      }
156.    }
157.
158.    printk(KERN_INFO "VFS allocate %d objects into slab: %s\n", number, SLABNAME);
159.    printk(KERN_INFO "VFS object size %d bytes, full size %ld bytes\n", size, (long)size
*number);
160.
161.    // Регистрация файловой системы
162.    int ret = register_filesystem(&vfs_type);
163.    if (ret != 0)
164.    {
165.      printk(KERN_ERR "VFS cannot register filesystem\n");
166.      return ret;
167.    }
168.
169.    printk(KERN_DEBUG "VFS loaded\n");
170.    return 0;
171.  }
172.
173.  static void __exit vfs_module_exit(void)
174.  {
175.    int i;
176.    for(i = 0; i < number; i++)
177.    {
178.      kmem_cache_free(cache, line[i]);
179.    }
180.
181.    kmem_cache_destroy(cache);
182.    kfree(line);
183.
184.    if (unregister_filesystem(&vfs_type) != 0)
185.    {
186.      printk(KERN_ERR "VFS cannot unregister filesystem!\n");
187.    }
188.
189.    printk(KERN_DEBUG "VFS unloaded!\n");
190.  }
191.
192.  module_init(vfs_module_init);
193.  module_exit(vfs_module_exit);
```

Загрузка модуля ядра:

```
olga@olga-VirtualBox:~/Documents/lab8$ sudo insmod vfs.ko
[sudo] password for olga:
olga@olga-VirtualBox:~/Documents/lab8$ lsmod | grep vfs
vfs                    16384  0
olga@olga-VirtualBox:~/Documents/lab8$ sudo dmesg | tail -3
[sudo] password for olga:
[ 4444.505790] VFS allocate 31 objects into slab: vfs_cache
[ 4444.505791] VFS object size 7 bytes, full size 217 bytes
[ 4444.505795] VFS loaded
```

Содержимое файла /proc/slabinfo с информацией о кэше:

```
olga@olga-VirtualBox:~/Documents/lab8$ sudo cat /proc/slabinfo | grep vfs
vfs_cache            256    256     16  256    1 : tunables    0    0    0 : slabdata      1      1      0
```

Создание образа диска и создание каталога, который будет точкой монтирования (корнем) файловой системы. Используя созданный образ, примонтируем файловую систему:

```
olga@olga-VirtualBox:~/Documents/lab8$ touch image
olga@olga-VirtualBox:~/Documents/lab8$ mkdir dir
olga@olga-VirtualBox:~/Documents/lab8$ sudo mount -o loop -t vfs ./image ./dir
olga@olga-VirtualBox:~/Documents/lab8$ sudo dmesg | tail -4
[ 4444.505790] VFS allocate 31 objects into slab: vfs_cache
[ 4444.505791] VFS object size 7 bytes, full size 217 bytes
[ 4444.505795] VFS loaded
[ 6295.063442] VFS mounted
```

В дереве каталогов:

```
olga@olga-VirtualBox:~/Documents/lab8$ ll
total 148
drwxrwx---  3 olga olga  4096 мая 12 12:57 ./
drwxr-xr-x 15 olga olga  4096 мая 10 01:33 ../
drwxr-xr-x  1 root root  4096 мая 12 12:53 dir/
-rwxrwx---  1 olga olga  6148 мая 10 01:28 .DS_Store*
-rw-r--r--  1 olga olga     0 мая 12 12:50 image
```

Размонтирование файловой системы и выгрузка модуля:

```
olga@olga-VirtualBox:~/Documents/lab8$ sudo umount ./dir
olga@olga-VirtualBox:~/Documents/lab8$ sudo rmmod vfs
olga@olga-VirtualBox:~/Documents/lab8$ sudo dmesg | tail -6
[ 4444.505790] VFS allocate 31 objects into slab: vfs_cache
[ 4444.505791] VFS object size 7 bytes, full size 217 bytes
[ 4444.505795] VFS loaded
[ 6295.063442] VFS mounted
[ 6806.352811] VFS super block destroyed
[ 6818.308567] VFS unloaded!
```

Загрузка модуля с заданными параметрами размера и количества элементов кэша:

```
olga@olga-VirtualBox:~/Documents/lab8$ sudo insmod vfs.ko size=16 number=64
olga@olga-VirtualBox:~/Documents/lab8$ sudo dmesg | tail -3
[ 6963.185087] VFS allocate 64 objects into slab: vfs_cache
[ 6963.185088] VFS object size 16 bytes, full size 1024 bytes
[ 6963.185092] VFS loaded
olga@olga-VirtualBox:~/Documents/lab8$ sudo cat /proc/slabinfo | grep vfs
vfs_cache            128    128     32  128    1 : tunables    0    0    0 : slabdata      1      1      0
```