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 предмету: «Операционные системы»

Тема: Виртуальная файловая система

Студент: Лаврова А. А.,
Группа: ИУ7-65Б
Преподаватель: Рязанова Н. Ю.

Москва, 2020 г.

Листинг программы:

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

#define VFS_MAGIC_NUMBER 0x13131313
#define SLABNAME "vfs_cache"

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Lavrova");

static void* *line = NULL;
struct kmem_cache *cache = NULL;
static int size = 7;
module_param(size, int, 0);
static int number = 31;
module_param(number, int, 0);


static void co(void* p)
{
  *(int*)p = (int)p;
}

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

// Создание структуры inode и заполнение значениями
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");
}

// Структура суперблок
static struct super_operations const vfs_super_ops = {
        .put_super = vfs_put_super,
        .statfs = simple_statfs,
        .drop_inode = generic_delete_inode,
```

```
};

// Инициализация суперблока и создание корневого каталога
static int vfs_fill_sb(struct super_block *sb, void *data, int silent)
{
  struct inode* root = NULL;

  // Заполнение структуры суперблок
  sb->s_blocksize = PAGE_SIZE;
  sb->s_blocksize_bits = PAGE_SHIFT;
  sb->s_magic = VFS_MAGIC_NUMBER;
  sb->s_op = &vfs_super_ops;

  // Построение корневого каталога ФС
  root = vfs_make_inode(sb, S_IFDIR | 0755);
  if (!root)
  {
    printk (KERN_ERR "VFS inode allocation failed\n");
    return -ENOMEM;
  }

  root->i_op = &simple_dir_inode_operations;
  root->i_fop = &simple_dir_operations;

  // Создание структуры dentry для корневого каталога (устанавливаем указатель)
  sb->s_root = d_make_root(root);
  if (!sb->s_root)
  {
    printk(KERN_ERR "VFS root creation failed\n");
    iput(root);
    return -ENOMEM;
  }
  return 0;
}

// Монтирование ФС (возвращает структуру, описывающую корневой каталог ФС)
static struct dentry* vfs_mount(struct file_system_type *type, int flags, const char *dev, void
*data)
{
  struct dentry* const entry = mount_nodev(type, flags, data, vfs_fill_sb);

  if (IS_ERR(entry))
    printk(KERN_ERR  "VFS mounting failed\n");
  else
    printk(KERN_DEBUG "VFS mounted\n");

  return entry;
}

// Описание ФС
static struct file_system_type vfs_type  =  {
  .owner  =  THIS_MODULE,   // Счетчик ссылок на модуль
  .name  =  "vfs",                          // Имя ФС
  .mount  =  vfs_mount,           // Функция, используемая для монтировании ФС
  .kill_sb  =  kill_litter_super, // Функция, используемая для размонтировании ФС
};

// Инициализация модуля
static int __init vfs_module_init(void)
{
```

```
  int i;

  if(size < 0)
  {
    printk(KERN_ERR "VFS invalid sizeof objects\n");
    return -EINVAL;
  }

  line = kmalloc(sizeof(void*) *number, GFP_KERNEL);
  if(!line)
  {
    printk(KERN_ERR "VFS kmalloc error\n");
    kfree(line);
    return -ENOMEM;
  }

  for(i = 0; i < number; i++)
    line[i] = NULL;

  // Кэш slab
  cache = kmem_cache_create(SLABNAME, size, 0, SLAB_HWCACHE_ALIGN, co);

  if(!cache)
  {
    printk(KERN_ERR "VFS cannot create 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 "VFS cannot alloc cache\n");
      for(i = 0; i < number; i++ )
        kmem_cache_free(cache, line[i]);
      return -ENOMEM;
    }
  }

  // Регистрация файловой системы
  int ret = register_filesystem(&vfs_type);
  if (ret != 0)
  {
    printk(KERN_ERR "VFS cannot register filesystem\n");
    return ret;
  }

  printk(KERN_INFO "VFS allocate %d objects into slab: %s\n", number, SLABNAME);
  printk(KERN_INFO "VFS object size %d bytes, full size %ld bytes\n", size, (long)size *number);

  printk(KERN_DEBUG "VFS loaded\n");
  return 0;
}

// Выгрузка модуля
static void __exit vfs_module_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(&vfs_type);
  if (ret != 0)
  {
    printk(KERN_ERR "VFS cannot unregister filesystem!\n");
  }

  printk(KERN_DEBUG "VFS unloaded!\n");
}

module_init(vfs_module_init);
module_exit(vfs_module_exit);
```

Результаты работы программы:

```
parallels@parallels-Parallels-Virtual-Platform:~/lab_8/new$ sudo insmod vfs.ko
parallels@parallels-Parallels-Virtual-Platform:~/lab_8/new$ lsmod | grep vfs
vfs                    16384  0
parallels@parallels-Parallels-Virtual-Platform:~/lab_8/new$ sudo dmesg | tail -3
[  202.742460] VFS allocate 31 objects into slab: vfs_cache
[  202.742460] VFS object size 7 bytes, full size 217 bytes
[  202.742464] VFS loaded
```

Рис. 1: Загрузка модуля ядра.

```
parallels@parallels-Parallels-Virtual-Platform:~/lab_8/new$ sudo cat /proc/slabin
fo| grep vfs
vfs_cache            256    256     16  256    1 : tunables    0    0    0 : slab
data      1      1      0
parallels@parallels-Parallels-Virtual-Platform:~/lab_8/new$
```

Рис. 2: Демонстрация состояния slab кэша.

```
parallels@parallels-Parallels-Virtual-Platform:~/lab_8/new$ touch image
parallels@parallels-Parallels-Virtual-Platform:~/lab_8/new$ mkdir dir
parallels@parallels-Parallels-Virtual-Platform:~/lab_8/new$ sudo mount -o loop -t
 vfs ./image ./dir
parallels@parallels-Parallels-Virtual-Platform:~/lab_8/new$ sudo dmesg | tail -4
[  202.742460] VFS allocate 31 objects into slab: vfs_cache
[  202.742460] VFS object size 7 bytes, full size 217 bytes
[  202.742464] VFS loaded
[  361.328124] VFS mounted
```

Рис. 3 и 4: Создание образа диска и каталога (будет являться точкой монтирования файловой системы). Монтирование файловой системы с использованием созданного образа.

```
parallels@parallels-Parallels-Virtual-Platform:~/lab_8/new$ ll
total 200
drwxr-xr-x 4 parallels parallels  4096 May 17 17:04 ./
drwxr-xr-x 7 parallels parallels  4096 May 17 17:01 ../
-rw-r--r-- 1 parallels parallels 63758 May 17 17:01 .cache.mk
drwxr-xr-x 1 root      root       4096 May 17 17:04 dir/
-rw-r--r-- 1 parallels parallels     0 May 17 17:03 image
```

Рис. 5: Демонстрация дерева каталогов

```
parallels@parallels-Parallels-Virtual-Platform:~/lab_8/new$ sudo umount ./dir
parallels@parallels-Parallels-Virtual-Platform:~/lab_8/new$ sudo rmmod vfs
parallels@parallels-Parallels-Virtual-Platform:~/lab_8/new$ sudo dmesg | tail -6
[  202.742460] VFS allocate 31 objects into slab: vfs_cache
[  202.742460] VFS object size 7 bytes, full size 217 bytes
[  202.742464] VFS loaded
[  361.328124] VFS mounted
[  445.082960] VFS super block destroyed
[  452.459529] VFS unloaded!
```

Рис. 6: Размонтирование файловой системы и выгрузка модуля

```
parallels@parallels-Parallels-Virtual-Platform:~/lab_8/new$ sudo insmod vfs.ko si
ze=16 number=64
parallels@parallels-Parallels-Virtual-Platform:~/lab_8/new$ sudo dmesg | tail -3
[  559.090173] VFS allocate 64 objects into slab: vfs_cache
[  559.090175] VFS object size 16 bytes, full size 1024 bytes
[  559.090179] VFS loaded
parallels@parallels-Parallels-Virtual-Platform:~/lab_8/new$ sudo cat /proc/slabin
fo| grep vfs
vfs_cache            128    128     32  128    1 : tunables    0    0    0 : slab
data      1      1      0
parallels@parallels-Parallels-Virtual-Platform:~/lab_8/new$
```

Рис. 7: Загрузка модуля с параметрами (параметры – размер и количество элементов кэша)