Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

Факультет «Информатика и системы управления»
Кафедра «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №8 по курсу:

«Операционные системы»

«Создание виртуальной файловой системы»»

Студент группы ИУ7-63Б: Фурдик Н. О.
(Фамилия И.О.)
Преподаватель: Рязанова Н.Ю.
(Фамилия И.О.)

Москва, 2020

Оглавление

Листинг кода

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/pagemap.h>  /* PAGE_CACHE_SIZE */
#include <linux/fs.h>       /* This is where libfs stuff is declared */ #include <asm/
    atomic.h>
#include <asm/uaccess.h>  /* copy_to_user */
#include <linux/slab.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Furdik");
MODULE_DESCRIPTION("Lab8");

#define VFS_MAGIC_NUMBER 0x13131313
#define SLABNAME "vfs_cache"

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

static void* obj = 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;

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)
```

```
51 {
52   printk(KERN_DEBUG "VFS super block destroyed!\n");
53 }
54
55 static struct super_operations const vfs_super_ops = {
56   .put_super = vfs_put_super,
57   .statfs = simple_statfs,
58   .drop_inode = generic_delete_inode,
59 };
60
61 static int vfs_fill_sb(struct super_block *sb, void *data, int silent)
62 {
63   struct inode* root = NULL;
64
65   sb->s_blocksize = PAGE_SIZE;
66   sb->s_blocksize_bits = PAGE_SHIFT;
67   sb->s_magic = VFS_MAGIC_NUMBER;
68   sb->s_op = &vfs_super_ops;
69
70
71   root = vfs_make_inode(sb, S_IFDIR | 0755);
72   if (!root)
73   {
74     printk (KERN_ERR "VFS inode allocation failed !\n");
75     return -ENOMEM;
76   }
77
78   root->i_op = &simple_dir_inode_operations;
79   root->i_fop = &simple_dir_operations;
80
81   sb->s_root = d_make_root(root);
82   if (!sb->s_root)
83   {
84       printk(KERN_ERR "VFS root creation failed!\n");
85       iput(root);
86       return -ENOMEM;
87   }
88
89   return 0;
90 }
91
92 static struct dentry* vfs_mount(struct file_system_type *type, int flags, const char *dev
      , void *data)
93 {
94   struct dentry* const entry = mount_nodev(type, flags, data, vfs_fill_sb);
95
96   if (IS_ERR(entry))
97     printk(KERN_ERR  "VFS mounting failed!\n");
98   else
99     printk(KERN_DEBUG "VFS mounted!\n");
100
101
102   return entry;
```

```
103 }
104
105 static struct file_system_type vfs_type = {
106   .owner = THIS_MODULE,
107   .name = "vfs",
108   .mount = vfs_mount,
109   .kill_sb = kill_litter_super,
110   };
111
112 static int __init vfs_module_init(void)
113   {
114
115   if (size < 0)
116   {
117     printk(KERN_ERR "VFS_MODULE invalid sizeof objects\n");
118     return -EINVAL;
119   }
120
121   obj = kmalloc(sizeof(void*), GFP_KERNEL);
122   if (!obj)
123   {
124     printk(KERN_ERR "VFS_MODULE kmalloc error\n");
125     kfree(obj);
126     return -ENOMEM;
127   }
128
129   cache = kmem_cache_create(SLABNAME, size, 0, SLAB_HWCACHE_ALIGN, co);
130
131   if (!cache)
132   {
133     printk(KERN_ERR "VFS_MODULE cannot create cache\n");
134     kmem_cache_destroy(cache);
135     return -ENOMEM;
136   }
137
138   if (NULL == (obj = kmem_cache_alloc(cache, GFP_KERNEL)))
139   {
140     printk(KERN_ERR "VFS_MODULE cannot alloc object\n");
141     kmem_cache_destroy(cache);
142     return -ENOMEM;
143   }
144
145   printk(KERN_INFO "VFS_MODULE allocate %d objects into slab: %s\n", number, SLABNAME);
146   printk(KERN_INFO "VFS_MODULE object size %d bytes, full size %ld bytes\n", size, (long)
        size *number);
147
148   int ret = register_filesystem(&vfs_type);
149   if (ret != 0)
150   {
151     printk(KERN_ERR "VFS_MODULE cannot register filesystem!\n");
152     return ret;
153   }
154
```

```
155   printk(KERN_DEBUG "VFS_MODULE loaded!\n");
156   return 0;
157 }
158
159 static void __exit vfs_module_exit(void)
160 {
161   kmem_cache_free(cache, obj);
162
163   kmem_cache_destroy(cache);
164   kfree(obj);
165
166   if (unregister_filesystem(&vfs_type) != 0)
167   {
168     printk(KERN_ERR "VFS_MODULE cannot unregister filesystem!\n");
169   }
170
171   printk(KERN_DEBUG "VFS_MODULE unloaded!\n");
172 }
173
174 module_init(vfs_module_init);
175 module_exit(vfs_module_exit);
```

Результат работы

1. Скомпилируем модуль ядра и загрузим его, проверив его в списке загруженных модулей ядра.

```
schoolboychik@schoolboychik-VirtualBox:~/lab8$ make
make -C /lib/modules/5.3.0-51-generic/build M=/home/schoolboychik/lab8 modules
make[1]: вход в каталог «/usr/src/linux-headers-5.3.0-51-generic»
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: выход из каталога «/usr/src/linux-headers-5.3.0-51-generic»
```

```
schoolboychik@schoolboychik-VirtualBox:~/lab8$ sudo insmod mod.ko
schoolboychik@schoolboychik-VirtualBox:~/lab8$ lsmod | grep mod
mod                    16384  0
```

2. Выведем содержание буфера сообщений ядра, а также проверим содержимое файла /proc/slabinfo, в котором хранится информация о кэшах.

```
schoolboychik@schoolboychik-VirtualBox:~/lab8$ sudo dmesg | tail -3
[  736.242694] VFS_MODULE allocate 31 objects into slab: vfs_cache
[  736.242695] VFS_MODULE object size 7 bytes, full size 217 bytes
[  736.242697] VFS_MODULE loaded!
```

```
schoolboychik@schoolboychik-VirtualBox:~/lab8$ sudo cat /proc/slabinfo | grep vf
s
vfs_cache            256    256     16  256    1 : tunables    0    0    0 : sla
bdata      1      1      0
```

3. Создадим образ диска. Кроме того, нужно создать каталог, который будет точкой монтирования (корнем) файловой системы. Далее, используя этот образ, примонтируем файловую систему. Проверим успешное монтирование, для чего выведем содержания буфера сообщений ядра.

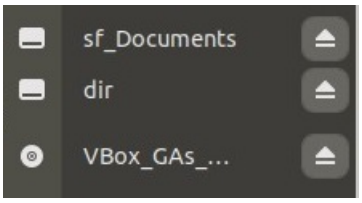
```
schoolboychik@schoolboychik-VirtualBox:~/lab8$ touch file
schoolboychik@schoolboychik-VirtualBox:~/lab8$ mkdir test_dir
```

```
schoolboychik@schoolboychik-VirtualBox:~/lab8$ sudo mount -o loop -t vfs ./file
./test_dir
schoolboychik@schoolboychik-VirtualBox:~/lab8$ sudo dmesg | tail -1
[ 1387.869147] VFS mounted!
```

4. Покажем созданную ФС в дереве каталогов

```
schoolboychik@schoolboychik-VirtualBox:~/lab8$ ls -al
итого 384
drwxrwx---  5 schoolboychik schoolboychik  4096 мая 15 16:11 .
drwxr-xr-x 18 schoolboychik schoolboychik  4096 мая 15 16:08 ..
-rwxrwx---  1 schoolboychik schoolboychik 76958 мая 11 14:21 bmstu.jpg
drwxr-xr-x  1 root          root           4096 мая 15 16:13 dir
-rw-r--r--  1 schoolboychik schoolboychik     0 мая 15 16:11 image
```

При этом она также отобразилась в проводнике:



5. Размонтируем ФС и выгрузим модуль ядра, также проверив, что выгрузка прошла успешно (для чего выведем содержание буфера сообщений ядра).

```
schoolboychik@schoolboychik-VirtualBox:~/lab8$ sudo umount ./test_dir
schoolboychik@schoolboychik-VirtualBox:~/lab8$ sudo rmmod mod
```

```
[ 1387.869147] VFS mounted!
[ 1454.008676] VFS super block destroyed!
[ 1462.103475] VFS unmounted!
```

6. Также продемонстрируем загрузку модуля ядра с заданными размером и количеством элементов кэша.

```
schoolboychik@schoolboychik-VirtualBox:~/lab8$ sudo insmod md.ko size=100 number=200
[sudo] пароль для schoolboychik:
schoolboychik@schoolboychik-VirtualBox:~/lab8$ sudo dmesg | tail -3
[  979.719336] VFS_MODULE allocate 200 objects into slab: wfs_cache
[  979.719337] VFS_MODULE object size 100 bytes, full size 20000 bytes
[  979.719339] VFS_MODULE loaded!
```

```
schoolboychik@schoolboychik-VirtualBox:~/lab8$ sudo cat /proc/slabinfo | grep wfs
wfs_cache            224    224    128   32    1 : tunables    0    0    0 : slabdata      7      7      0
schoolboychik@schoolboychik-VirtualBox:~/lab8$
```

Литература

1. Рязанова Н.Ю. - Курс лекций по "Операционным системам"[Текст], Москва 2020 год.

2. У. Ричард Стивенс, Стивен А. Раго - UNIX. Профессиональное программирование. 3-е изд. - Москва: Питер, 2018. - 944 с.