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

Дисциплина Операционные системы

Тема Создание виртуальной файловой системы

Студент Набиев Ф.М.
Группа ИУ7–63Б
Оценка (баллы)
Преподаватель Рязанова Н.Ю.

Москва, 2020 г.

1 Реализация

```
1  #include <linux/fs.h>
2  #include <linux/init.h>
3  #include <linux/time.h>
4  #include <linux/slab.h>
5  #include <linux/kernel.h>
6  #include <linux/module.h>
7  #include <linux/version.h>
8
9  MODULE_LICENSE("GPL");
10 MODULE_AUTHOR("Faris Nabiev");
11
12 static int __init myfs_module_init(void);
13 static void __exit myfs_module_exit(void);
14
15 module_init(myfs_module_init);
16 module_exit(myfs_module_exit);
17
18 #define MYFS_MAGIC_NUMBER 0x13131313
19 #define SLABNAME "myfs_cache"
20
21 static int sco = 0;
22 // Размер элементов кэша
23 static int size = 7;
24 static int number = 31;
25 // Получение значения параметра командной строки
26 // если он передан
27 module_param(size, int, 0);
28 module_param(number, int, 0);
29
30 static void **line = NULL;
31
32 struct kmem_cache *cache = NULL;
33
34 static struct myfs_inode
35 {
36      int i_mode;
37      unsigned long i_ino;
38 } myfs_inode;
39
40 // Создание inode
41 static struct inode *myfs_make_inode(struct super_block *sb, int mode)
42 {
43     // Размещение новой структуры inode
44     struct inode *ret = new_inode(sb);
45
46     if (ret)
```

```
47      {
48          inode_init_owner(ret, NULL, mode);
49
50          // Заполнение значениями
51          ret->i_size    = PAGE_SIZE;
52          ret->i_atime   = ret->i_mtime = ret->i_ctime = current_time(ret);
53          ret->i_private = &myfs_inode;
54      }
55
56      return ret;
57  }
58
59  // Деструктор суперблока, вызываемый перед уничтожением
60  // структуры super_block (при размонтировании ФС)
61  static void myfs_put_super(struct super_block * sb)
62  {
63      printk(KERN_DEBUG "myfs: super block destroyed\n");
64  }
65
66  // Операции структуры суперблок
67  static struct super_operations const myfs_super_ops = {
68      .put_super  = myfs_put_super,
69      .statfs     = simple_statfs,
70      .drop_inode = generic_delete_inode,
71  };
72
73  // Функция инициализации суперблока
74  // Выполняет построение корневого каталога ФС
75  static int myfs_fill_sb(struct super_block *sb, void *data, int silent)
76  {
77      struct inode* root = NULL;
78
79      // Заполняется структура super_block
80      sb->s_blocksize      = PAGE_SIZE;
81      sb->s_blocksize_bits = PAGE_SHIFT;
82      sb->s_magic          = MYFS_MAGIC_NUMBER;
83      sb->s_op             = &myfs_super_ops;
84
85      // Создание inode каталога ФС (указывает на это S_IFDIR)
86      root = myfs_make_inode(sb, S_IFDIR | 0755);
87      if (!root)
88      {
89          printk (KERN_ERR "myfs: inode allocation failed\n");
90          return -ENOMEM;
91      }
92
93      // Файловые и inode операции из libfs
94      root->i_op  = &simple_dir_inode_operations;
```

```
95      root->i_fop = &simple_dir_operations;
96
97      // Создание структуры dentry для
98      // представления корневого каталога в ядре
99      sb->s_root = d_make_root(root);
100     if (!sb->s_root)
101     {
102         printk(KERN_ERR "myfs: root creation failed\n");
103         iput(root);
104         return -ENOMEM;
105     }
106
107     return 0;
108 }
109
110 // Монтирование ФС
111 // возвращает структуру, описывающую корневой каталог ФС
112 static struct dentry* myfs_mount(struct file_system_type *type,
113                                  int flags, char const *dev, void *data)
114 {
115     // myfs_fill_sb будет вызвана по переданному указателю
116     // из mount_bdev, чтобы проинициализировать суперблок
117     struct dentry* const entry = mount_nodev(type, flags,
118                                              data, myfs_fill_sb);
119
120     if (IS_ERR(entry))
121         printk(KERN_ERR "myfs: mounting failed\n");
122     else
123         printk(KERN_DEBUG "myfs: mounted\n");
124
125     return entry;
126 }
127
128 // Описание создаваемой ФС
129 static struct file_system_type myfs_type = {
130     // счетчик ссылок на модуль
131     .owner   = THIS_MODULE,
132     // название ФС
133     .name    = "myfs",
134     // указатель на функцию, вызываемую при монтировании ФС
135     .mount   = myfs_mount,
136     // указатель на функцию, вызываемую при размонтировании ФС
137     .kill_sb = kill_litter_super,
138 };
139
140 // Конструктор, вызываемый при размещении каждого элемента
141 static void co(void* p)
142 {
```

```
143     *(int*)p = (int)p;
144     sco++;
145 }
146 
147 // Инициализация модуля
148 static int __init myfs_module_init(void)
149 {
150     int i;
151     int ret;
152 
153     if (size < 0)
154     {
155         printk(KERN_ERR "myfs: invalid argument\n");
156         return -EINVAL;
157     }
158 
159     line = kmalloc(sizeof(void*) * number, GFP_KERNEL);
160 
161     if (line == NULL)
162     {
163         printk(KERN_ERR "myfs: kmalloc error\n" );
164         kfree(line);
165         return -ENOMEM;
166     }
167 
168     for (i = 0; i < number; i++)
169     {
170         line[i] = NULL;
171     }
172 
173     cache = kmem_cache_create(SLABNAME, size, 0, SLAB_HWCACHE_ALIGN, co);
174 
175     if (cache == NULL)
176     {
177         printk(KERN_ERR "myfs: kmem_cache_create error\n" );
178         kmem_cache_destroy(cache);
179         kfree(line);
180         return -ENOMEM;
181     }
182 
183     for (i = 0; i < number; i++)
184     {
185         if (NULL == (line[i] = kmem_cache_alloc(cache, GFP_KERNEL))) {
186             printk(KERN_ERR "myfs: kmem_cache_alloc error\n");
187 
188             for (i = 0; i < number; i++)
189             {
190                 kmem_cache_free(cache, line[i]);
```

```
191             }
192
193             kmem_cache_destroy(cache);
194             kfree(line);
195             return -ENOMEM;
196         }
197     }
198
199     printk(KERN_INFO "myfs: allocate %d objects into slab: %s\n",
200            number, SLABNAME);
201     printk(KERN_INFO "myfs: object size %d bytes, full size %ld bytes\n",
202            size, (long)size * number);
203     printk(KERN_INFO "myfs: constructor called %d times\n", sco);
204
205     ret = register_filesystem(&myfs_type);
206
207     if (ret != 0)
208     {
209         printk(KERN_ERR "myfs: module cannot register filesystem\n");
210         return ret;
211     }
212
213     printk(KERN_DEBUG "myfs: module loaded\n");
214     return 0;
215 }
216
217 // Выгрузка модуля
218 static void __exit myfs_module_exit(void)
219 {
220     int i;
221     int ret;
222
223     for (i = 0; i < number; i++)
224         kmem_cache_free(cache, line[i]);
225
226     kmem_cache_destroy(cache);
227     kfree(line);
228
229     ret = unregister_filesystem(&myfs_type);
230
231     if (ret != 0)
232         printk(KERN_ERR "myfs: module cannot unregister filesystem\n");
233
234     printk(KERN_DEBUG "myfs: module unloaded\n");
235 }
```

2 Результаты работы

Рис. 2.1 – Сборка

```
 # make
make -C /lib/modules/5.5.0-0.bpo.2-amd64/build M=/home/faris/Documents/Repositories/b
make[1]: Entering directory '/usr/src/linux-headers-5.5.0-0.bpo.2-amd64'
  CC [M]  /home/faris/Documents/Repositories/bmstu/OperatingSystems/lab_18/myfs.o
/home/faris/Documents/Repositories/bmstu/OperatingSystems/lab_18/myfs.c: In function
/home/faris/Documents/Repositories/bmstu/OperatingSystems/lab_18/myfs.c:141:16: warni
 [-Wpointer-to-int-cast]
     *(int*)p = (int)p;
                ^
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/faris/Documents/Repositories/bmstu/OperatingSystems/lab_18/myfs.mod.o
  LD [M]  /home/faris/Documents/Repositories/bmstu/OperatingSystems/lab_18/myfs.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.5.0-0.bpo.2-amd64'
```

Рис. 2.2 – Загрузка модуля ядра и проверка списка загруженных модулей ядра

```
 ~/Documents/Repositories/bmstu/OperatingSystems/lab_18  master ? ⇡1
 # sudo insmod myfs.ko

 ~/Documents/Repositories/bmstu/OperatingSystems/lab_18  master ? ⇡1
 # lsmod | grep myfs
myfs                   16384  0
```

Рис. 2.3 – Вывод буфера сообщений ядра

```
 # sudo dmesg | tail -4
[50751.108826] myfs: allocate 31 objects into slab: myfs_cache
[50751.108830] myfs: object size 7 bytes, full size 217 bytes
[50751.108832] myfs: constructor called 256 times
[50751.108839] myfs: module loaded
```

Рис. 2.4 – Состояние slab-кэша

```
 # sudo cat /proc/slabinfo | grep myfs
myfs_cache           256    256     16  256    1 : tunables    0    0    0 : slabdata      1      1      0
```

Рис. 2.5 – Создание образа диска, корня файловой системы; монтирование файловой системы

```
 # touch image

  ~/Documents/Repositories/bmstu/OperatingSystems/lab_18
 # mkdir dir

  ~/Documents/Repositories/bmstu/OperatingSystems/lab_18
 # sudo mount -o loop -t myfs ./image ./dir
```

Рис. 2.6 – Вывод буфера сообщений ядра

```
 # sudo dmesg | tail -1
[51202.610017] myfs: mounted
```

Рис. 2.7 – Вывод дерева каталогов

```
 # "ls" -l
total 704
-rw-r--r-- 1 faris faris   6198 Jun 20 15:01 compile_commands.json
drwxr-xr-x 1 root  root    4096 Jun 22 19:17 dir
-rw-r--r-- 1 faris faris      0 Jun 22 19:16 image
-rw-r--r-- 1 faris faris    348 Jun 21 19:40 Makefile
-rw-r--r-- 1 faris faris     73 Jun 22 19:04 modules.order
-rw-r--r-- 1 faris faris      0 Jun 20 15:00 Module.symvers
-rw-r--r-- 1 faris faris   6531 Jun 22 15:58 myfs.c
-rw-r--r-- 1 faris faris 338136 Jun 22 19:04 myfs.ko
```

Рис. 2.8 – Размонтирование ФС и выгрузка модуля

```
# sudo umount ./dir

 ~/Documents/Repositories/bmstu/OperatingSystems/lab_18
# sudo rmmod myfs
```

Рис. 2.9 – Вывод буфера сообщений ядра

```
 # sudo dmesg | tail -2
[51566.104551] myfs: super block destroyed
[51572.598161] myfs: module unloaded
```

Рис. 2.10 – Загрузка модуля с заданным размером и количеством элементов кэша

```
 # sudo insmod myfs.ko size=32 number=256

  ~/Documents/Repositories/bmstu/OperatingSystems/lab_18  master ? ⇡1
 # sudo dmesg | tail -4
[51715.810296] myfs: allocate 256 objects into slab: myfs_cache
[51715.810297] myfs: object size 32 bytes, full size 8192 bytes
[51715.810298] myfs: constructor called 256 times
[51715.810300] myfs: module loaded

  ~/Documents/Repositories/bmstu/OperatingSystems/lab_18  master ? ⇡1
 # sudo cat /proc/slabinfo | grep myfs
myfs_cache           256    256     64   64    1 : tunables    0    0    0 : slabdata      4      4      0
```