

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический  
университет имени Н.Э. Баумана»  
(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	Куприй А. А.
Группа	ИУ7-63Б
Преподаватель	Рязанова Н.Ю.

Москва, 2020 г.

## ВВЕДЕНИЕ

**Задание:** Создать виртуальную файловую систему wfs, используя наработки лабораторной работы по загружаемым модулям ядра.

## 1 Практическая часть

На листинге ниже представлен текст программы.

Листинг 1.1 — Текст программы

```
1 #include <linux/module.h>
2 #include <linux/kernel.h>
3 #include <linux/init.h>
4 #include <linux/fs.h>
5 #include <linux/time.h>
6 #include <linux/slab.h>
7 #include <linux/version.h>
8
9 #define wfs_MAGIC_NUMBER 0x13131313
10 #define SLABNAME "wfs_cache"
11
12 MODULE_LICENSE("GPL");
13 MODULE_AUTHOR("Alexander Kupry");
14 MODULE_DESCRIPTION("lab_8");
15
16 static int size = 7;
17 module_param(size, int, 0);
18 static int number = 31;
19 module_param(number, int, 0);
20 static int sco = 0;
21
22 static void* *line = NULL;
23
24 static void co(void* p)
25 {
26     *(int*)p = (int)p;
27     sco++;
28 }
29
30 struct kmem_cache *cache = NULL;
31
32 static struct wfs_inode
33 {
34     int i_mode;
35     unsigned long i_ino;
36 } wfs_inode;
37
38 static struct inode *wfs_make_inode(struct super_block *sb, int mode)
39 {
40     struct inode *ret = new_inode(sb);
41
```

```

42     if (ret)
43     {
44         inode_init_owner(ret, NULL, mode);
45         ret->i_size = PAGE_SIZE;
46         ret->i_atime = ret->i_mtime = ret->i_ctime = current_time(ret);
47         ret->i_private = &wfs_inode;
48     }
49
50     return ret;
51 }
52
53 static void wfs_put_super(struct super_block * sb)
54 {
55     printk(KERN_DEBUG "[wfs] super block destroyed!\n");
56 }
57
58 static struct super_operations const wfs_super_ops = {
59     .put_super = wfs_put_super,
60     .statfs = simple_statfs,
61     .drop_inode = generic_delete_inode,
62 };
63
64 static int wfs_fill_sb(struct super_block *sb, void *data, int silent)
65 {
66     struct inode* root = NULL;
67
68     sb->s_blocksize = PAGE_SIZE;
69     sb->s_blocksize_bits = PAGE_SHIFT;
70     sb->s_magic = wfs_MAGIC_NUMBER;
71     sb->s_op = &wfs_super_ops;
72
73     root = wfs_make_inode(sb, S_IFDIR | 0755);
74     if (!root)
75     {
76         printk (KERN_ERR "[wfs] inode allocation failed!\n");
77         return -ENOMEM;
78     }
79
80     root->i_op = &simple_dir_inode_operations;
81     root->i_fop = &simple_dir_operations;
82
83     sb->s_root = d_make_root(root);
84     if (!sb->s_root)
85     {
86         printk(KERN_ERR "[wfs] root creation failed!\n");
87         iput(root);

```

```

88         return -ENOMEM;
89     }
90
91     return 0;
92 }
93
94 static struct dentry* wfs_mount(struct file_system_type *type, int flags,
95     char const *dev, void *data)
96 {
97     struct dentry* const entry = mount_nodev(type, flags, data,
98         wfs_fill_sb);
99
100     if (IS_ERR(entry))
101         printk(KERN_ERR "[wfs] mounting failed!\n");
102     else
103         printk(KERN_DEBUG "[wfs] mounted!\n");
104
105     return entry;
106 }
107
108 static struct file_system_type wfs_type = {
109     .owner = THIS_MODULE,
110     .name = "wfs",
111     .mount = wfs_mount,
112     .kill_sb = kill_litter_super,
113 };
114
115 static int __init wfs_init(void)
116 {
117     int i;
118     int ret;
119
120     if (size < 0)
121     {
122         printk(KERN_ERR "[wfs] invalid argument\n");
123         return -EINVAL;
124     }
125
126     line = kmalloc(sizeof(void*) * number, GFP_KERNEL);
127
128     if (line == NULL)
129     {
130         printk(KERN_ERR "[wfs] kmalloc error\n");
131         kfree(line);
132         return -ENOMEM;
133     }

```

```

132
133     for (i = 0; i < number; i++)
134     {
135         line[i] = NULL;
136     }
137
138     cache = kmem_cache_create(SLABNAME, size, 0, SLAB_HWCACHE_ALIGN, co);
139
140     if (cache == NULL)
141     {
142         printk(KERN_ERR "[wfs] kmem_cache_create error\n");
143         kmem_cache_destroy(cache);
144         kfree(line);
145         return -ENOMEM;
146     }
147
148     for (i = 0; i < number; i++)
149     {
150         if (NULL == (line[i] = kmem_cache_alloc(cache, GFP_KERNEL))) {
151             printk(KERN_ERR "[wfs] kmem_cache_alloc error\n");
152
153             for (i = 0; i < number; i++)
154             {
155                 kmem_cache_free(cache, line[i]);
156             }
157
158             kmem_cache_destroy(cache);
159             kfree(line);
160             return -ENOMEM;
161         }
162     }
163
164     printk(KERN_INFO "[wfs] allocate %d objects into slab: %s\n", number,
165           SLABNAME);
166     printk(KERN_INFO "[wfs] object size %d bytes, full size %ld bytes\n",
167           size, (long)size * number);
168     printk(KERN_INFO "[wfs] constructor called %d times\n", sco);
169
170     ret = register_filesystem(&wfs_type);
171
172     if (ret != 0)
173     {
174         printk(KERN_ERR "[wfs] module cannot register filesystem!\n");
175         return ret;
176     }

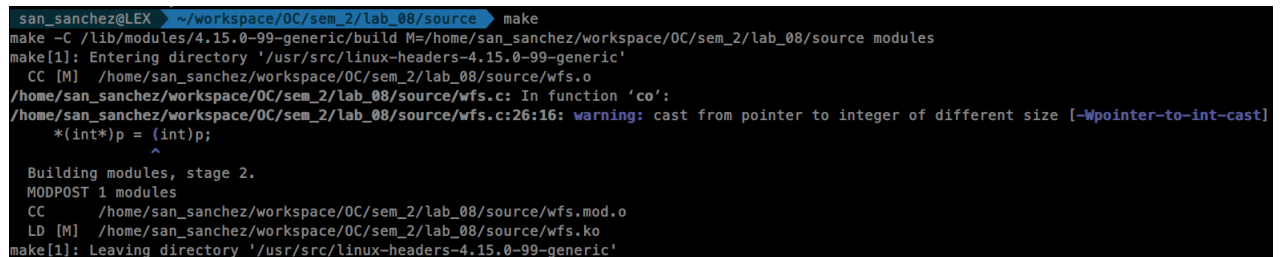
```

```

176     printk(KERN_DEBUG "[wfs] module loaded!\n");
177     return 0;
178 }
179
180 static void __exit wfs_exit(void)
181 {
182     int i;
183     int ret;
184
185     for (i = 0; i < number; i++)
186         kmem_cache_free(cache, line[i]);
187
188     kmem_cache_destroy(cache);
189     kfree(line);
190
191     ret = unregister_filesystem(&wfs_type);
192
193     if (ret != 0)
194         printk(KERN_ERR "[wfs] module cannot unregister filesystem!\n");
195
196     printk(KERN_DEBUG "[wfs] module unloaded!\n");
197 }
198
199 module_init(wfs_init);
200 module_exit(wfs_exit);

```

Компиляция загружаемого модуля ядра при помощи Makefile:



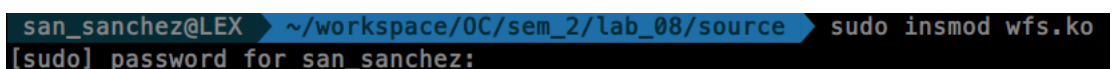
```

san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/source make
make -C /lib/modules/4.15.0-99-generic/build M=/home/san_sanchez/workspace/OC/sem_2/lab_08/source modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-99-generic'
CC [M] /home/san_sanchez/workspace/OC/sem_2/lab_08/source/wfs.o
/home/san_sanchez/workspace/OC/sem_2/lab_08/source/wfs.c: In function 'co':
/home/san_sanchez/workspace/OC/sem_2/lab_08/source/wfs.c:26:16: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
    *(int*)p = (int)p;
    ^
Building modules, stage 2.
MODPOST 1 modules
CC /home/san_sanchez/workspace/OC/sem_2/lab_08/source/wfs.mod.o
LD [M] /home/san_sanchez/workspace/OC/sem_2/lab_08/source/wfs.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-99-generic'

```

Рисунок 1.1 — Скришот результата работы make.

Загрузка модуля ядра с помощью команды insmod:



```

san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/source sudo insmod wfs.ko
[sudo] password for san_sanchez:

```

Рисунок 1.2 — Скришот результата работы команды insmod.

Демонстрация успешной загрузки:

```

san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/source sudo lsmod | grep wfs
[sudo] password for san_sanchez:
wfs 16384 0
san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/source sudo dmesg | grep wfs
[214846.418749] [wfs] allocate 31 objects into slab: wfs_cache
[214846.418751] [wfs] object size 7 bytes, full size 217 bytes
[214846.418752] [wfs] constructor called 256 times
[214846.418756] [wfs] module loaded!

```

Рисунок 1.3 — Скришот результата загрузки ядра.

Состояние slab-кэша:

```

san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/source sudo cat /proc/slabinfo | grep wfs
wfs_cache 256 256 16 256 1 : tunables 0 0 0 : slabdata 1 1 0

```

Рисунок 1.4 — Скришот содержимого /proc/slabinfo.

Создадим образ диска и корень файловой системы:

```

san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/res touch image
san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/res mkdir dir
san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/res ls -la
total 12
drwxrwxr-x 3 san_sanchez san_sanchez 4096 мая 20 18:00 .
drwxrwxr-x 5 san_sanchez san_sanchez 4096 мая 20 18:00 ..
drwxrwxr-x 2 san_sanchez san_sanchez 4096 мая 20 18:00 dir
-rw-rw-r-- 1 san_sanchez san_sanchez 0 мая 20 18:00 image

```

Рисунок 1.5 — Скришот команд touch, mkdir и ls -la.

Используя данный образ монтируем файловую систему:

```

san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/res sudo mount -o loop -t wfs ./image ./dir
[sudo] password for san_sanchez:
san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/res sudo dmesg | grep wfs
[214846.418749] [wfs] allocate 31 objects into slab: wfs_cache
[214846.418751] [wfs] object size 7 bytes, full size 217 bytes
[214846.418752] [wfs] constructor called 256 times
[214846.418756] [wfs] module loaded!
[219067.601312] [wfs] mounted!
san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/res mount | grep wfs
/dev/loop29 on /home/san_sanchez/workspace/OC/sem_2/lab_08/res/dir type wfs (rw,relatime)
san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/res ls -al
total 8
drwxrwxr-x 3 san_sanchez san_sanchez 4096 мая 20 18:00 .
drwxrwxr-x 5 san_sanchez san_sanchez 4096 мая 20 18:00 ..
drwxr-xr-x 1 root root 4096 мая 20 18:02 dir
-rw-rw-r-- 1 san_sanchez san_sanchez 0 мая 20 18:00 image

```

Рисунок 1.6 — Скришот результата работы mount.

Также эта файловая система отобразилась в файловом менеджере:



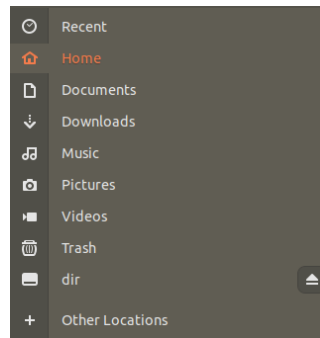


Рисунок 1.7 — Скриншот файлового менеджера.

Размонтирование файловой системы и выгрузка модуля:

```
san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/res sudo umount ./dir
[sudo] password for san_sanchez:
san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/res sudo rmmod wfs
san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/res dmesg | grep wfs
[214846.418749] [wfs] allocate 31 objects into slab: wfs_cache
[214846.418751] [wfs] object size 7 bytes, full size 217 bytes
[214846.418752] [wfs] constructor called 256 times
[214846.418756] [wfs] module loaded!
[219067.601312] [wfs] mounted!
[221322.267547] [wfs] super block destroyed!
[221333.203665] [wfs] module unloaded!
```

Рисунок 1.8 — Скриншот результата работы команд umount и rmmod.

Пример загрузки модуля ядра с параметрами:

```
san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/source sudo insmod wfs.ko size=16 number=32
san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/source sudo dmesg | grep wfs | tail -4
[222236.733329] [wfs] allocate 32 objects into slab: wfs_cache
[222236.733331] [wfs] object size 16 bytes, full size 512 bytes
[222236.733331] [wfs] constructor called 128 times
[222236.733333] [wfs] module loaded!
san_sanchez@LEX ~/workspace/OC/sem_2/lab_08/source sudo cat /proc/slabinfo | grep wfs
wfs_cache 128 128 32 128 1 : tunables 0 0 0 : slabdata 1 1 0
```

Рисунок 1.9 — Скриншот результата работы команды insmod.