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 операционным системам

Студент Колганов О.С.

Группа ИУ7 — 62Б

Преподаватель Рязанова Н.Ю.

Москва.
2020 г.

Задание:
Используя наработки из лабораторной работы по загружаемым модулям ядра, создать виртуальную файловую систему и slab-кэш для inode.

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.
8. MODULE_LICENSE("GPL");
9. MODULE_DESCRIPTION("MYFS_MODULE");
10. MODULE_AUTHOR("Moxxx1e");
11.
12. static int number = 0;
13. module_param(number, int, 0);
14. static struct inode **myfs_inodes = NULL;
15.
16. struct myfs_inode {
17.     int i_mode;
18.     unsigned long i_ino;
19. };
20. struct kmem_cache *cache = NULL;
21.
22. static struct inode *myfs_make_inode(struct super_block *sb,
int mode) {
23.     struct inode *ret = new_inode(sb);
24.     struct myfs_inode *myfs_inode = NULL;
25.     if (ret) {
26.         inode_init_owner(ret, NULL, mode);
27.         ret->i_size = PAGE_SIZE;
28.         ret->i_atime = ret->i_mtime = ret->i_ctime =
```

```
current_time(ret);
    29.         myfs_inode = kmem_cache_alloc(cache, GFP_KERNEL);
    30.         *myfs_inode = (struct myfs_inode){
    31.             .i_mode = ret->i_mode,
    32.             .i_ino = ret->i_ino,
    33.         };
    34.         ret->i_private = myfs_inode;
    35.     }
    36.
    37.     return ret;
    38. }
    39.
    40. static void myfs_put_super(struct super_block *sb) {
    41.     printk(KERN_DEBUG "MYFS super block destroyed!\n");
    42. }
    43.
    44. static int myfs_drop_inode(struct inode *inode) {
    45.     kmem_cache_free(cache, inode->i_private);
    46.     return generic_drop_inode(inode);
    47. }
    48.
    49. static struct super_operations const myfs_super_ops = {
    50.     .put_super = myfs_put_super,
    51.     .statfs = simple_statfs,
    52.     .drop_inode = myfs_drop_inode,
    53. };
    54.
    55. #define MYFS_MAGIC_NUMBER 0x13131313
    56. static int myfs_fill_sb(struct super_block *sb, void *data,
int silent) {
    57.     struct inode *root = NULL;
    58.    int i = 0;
```

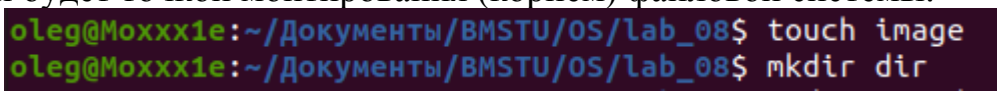
```
59.
60.     sb->s_blocksize = PAGE_SIZE;
61.     sb->s_blocksize_bits = PAGE_SHIFT;
62.     sb->s_magic = MYFS_MAGIC_NUMBER;
63.     sb->s_op = &myfs_super_ops;
64.
65.     root = myfs_make_inode(sb, S_IFDIR | 0755);
66.     if (!root) {
67.         printk(KERN_ERR "MYFS inode allocation failed!\n");
68.         return -ENOMEM;
69.     }
70.
71.     root->i_op = &simple_dir_inode_operations;
72.     root->i_fop = &simple_dir_operations;
73.
74.     sb->s_root = d_make_root(root);
75.     if (!sb->s_root) {
76.         printk(KERN_ERR "MYFS root creation failed!\n");
77.         iput(root);
78.         return -ENOMEM;
79.     }
80.
81.     for (i = 0; i < number; i++) {
82.         if (!(myfs_inodes[i] = myfs_make_inode(sb, S_IFDIR |
0755))) {
83.             printk(KERN_ERR "MYFS kmem_cache_alloc error\n");
84.             for (i = 0; i < number; i++) {
85.                 myfs_drop_inode(myfs_inodes[i]);
86.             }
87.
88.             kmem_cache_destroy(cache);
89.             kfree(myfs_inodes);
```

```
    90.             return -ENOMEM;
    91.         }
    92.     }
    93.
    94.     return 0;
    95. }
    96.
    97. static struct dentry *myfs_mount(struct file_system_type
*type, int flags, char const *dev, void *data) {
    98.     struct dentry *const entry = mount_nodev(type, flags,
data, myfs_fill_sb);
    99.
    100. if (IS_ERR(entry)) {
    101.         printk(KERN_ERR "MYFS mounting failed!\n");
    102.     } else {
    103.         printk(KERN_DEBUG "MYFS mounted!\n");
    104.     }
    105.
    106.     return entry;
    107. }
    108.
    109. static struct file_system_type myfs_type = {
    110.     .owner = THIS_MODULE,
    111.     .name = "myfs",
    112.     .mount = myfs_mount,
    113.     .kill_sb = kill_litter_super,
    114. };
    115.
    116.
    117. #define SLABNAME "myfs_inode_cache"
    118. static int size = sizeof(struct myfs_inode);
    119. static int __init myfs_init(void) {
    120.     int ret = register_filesystem(&myfs_type);
```

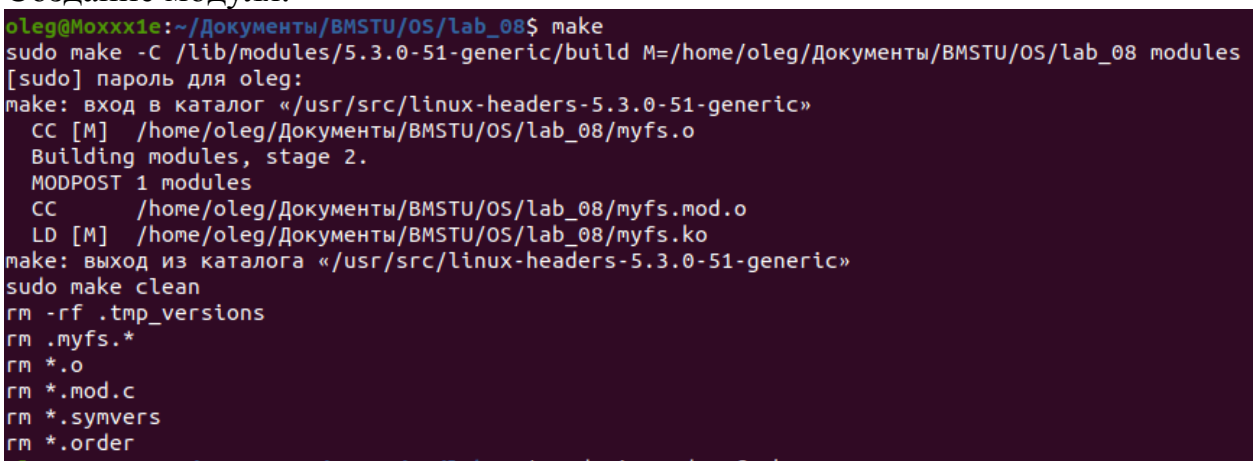
```
    121.     if (ret != 0) {
    122.         printk(KERN_ERR "MYFS_MODULE cannot register
filesystem!\n");
    123.         return ret;
    124.     }
    125.
    126.     myfs_inodes = kmalloc(sizeof(struct inode *) * number,
GFP_KERNEL);
    127.     if (!myfs_inodes) {
    128.         printk(KERN_ERR "MYFS kmalloc error\n");
    129.         kfree(myfs_inodes);
    130.         return -ENOMEM;
    131.     }
    132.
    133.     cache = kmem_cache_create(SLABNAME, size, 0, SLAB_POISON,
NULL);
    134.     if (!cache) {
    135.         printk(KERN_ERR "MYFS kmem_cache_create error\n");
    136.         kfree(myfs_inodes);
    137.         kmem_cache_destroy(cache);
    138.         return -ENOMEM;
    139.     }
    140.
    141.     printk(KERN_DEBUG "MYFS_MODULE loaded\n");
    142.
    143.     return 0;
    144. }
    145.
    146. static void __exit myfs_exit(void) {
    147.     int i = 0;
    148.     int ret = unregister_filesystem(&myfs_type);
    149.     if (ret != 0) {
    150.         printk(KERN_ERR "MYFS_MODULE cannot unregister
filesystem!\n");
```

```
151.     }
152.
153.     for (i = 0; i < number; i++) {
154.         myfs_drop_inode(myfs_inodes[i]);
155.     }
156.
157.     kmem_cache_destroy(cache);
158.     kfree(myfs_inodes);
159.     printk(KERN_DEBUG "MYFS_MODULE unloaded\n");
160. }
161.
162. module_init(myfs_init);
163. module_exit(myfs_exit);
```

На скриншоте показано создание образа диска, а также создание каталога dir, который будет точкой монтирования (корнем) файловой системы.



Создание модуля:



Загрузка модуля:

```
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo insmod myfs.ko
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ ls -al
итого 752
drwxr-xr-x 3 oleg oleg   4096 мая 21 14:26 .
drwxr-xr-x 7 oleg oleg   4096 мая 21 14:24 ..
drwxr-xr-x 2 oleg oleg   4096 мая 21 14:26 dir
-rw-r--r-- 1 oleg oleg      0 мая 21 14:26 image
-rw-r--r-- 1 oleg oleg     71 мая 21 14:25 .~lock.report_8.odt#
-rw-r--r-- 1 oleg oleg    321 мая 19 23:06 Makefile
-rw-r--r-- 1 oleg oleg   4497 мая 21 14:21 myfs.c
-rw-r--r-- 1 root root  10312 мая 21 14:26 myfs.ko
-rw-r--r-- 1 root root     54 мая 21 14:26 myfs.mod
-rw-r--r-- 1 oleg oleg 721885 мая 21 14:25 report_8.odt
```

Состояние кэша до монтирования:

```
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo cat /proc/slabinfo
| grep myfs
myfs_inode_cache       0      0     24  170    1 : tunables    0
  0    0 : slabdata      0      0      0
```

Монтирование файловой системы.

```
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo mount -o loop -t myfs ./image ./dir
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ ls -al
итого 748
drwxr-xr-x 3 oleg oleg   4096 мая 21 14:26 .
drwxr-xr-x 7 oleg oleg   4096 мая 21 14:24 ..
drwxr-xr-x 1 root root   4096 мая 21 14:26 dir
-rw-r--r-- 1 oleg oleg      0 мая 21 14:26 image
-rw-r--r-- 1 oleg oleg     71 мая 21 14:25 .~lock.report_8.odt#
-rw-r--r-- 1 oleg oleg    321 мая 19 23:06 Makefile
-rw-r--r-- 1 oleg oleg   4497 мая 21 14:21 myfs.c
-rw-r--r-- 1 root root  10312 мая 21 14:26 myfs.ko
-rw-r--r-- 1 root root     54 мая 21 14:26 myfs.mod
-rw-r--r-- 1 oleg oleg 721885 мая 21 14:25 report_8.odt
```

Создаётся inode для корневого каталога, поэтому в кэше значится 1 элемент.

```
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ dmesg | grep MYFS
[  266.249534] MYFS_MODULE loaded
[  713.828982] MYFS mounted!
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo cat /proc/slabinfo | grep myfs
myfs_inode_cache       1    170     24  170    1 : tunables    0    0    0 : slabdata      1      1      0
```

Информация о примонтированной файловой системе:

```
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ mount | grep myfs
/dev/loop28 on /home/oleg/Документы/BMSTU/OS/lab_08/dir type myfs (rw,relatime)
```

Дополнительная информация о примонтированной файловой системе: имя, тип, размер, использованная память, доступная память, процент использования, корневой каталог.

```
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ df -aTh | grep myfs
/dev/loop28    myfs               0          0     0            - /home/oleg/Документы/BMSTU/OS/lab_0
8/dir
```

После размонтирования ФС кэш возвращается в прежнее состояние (0 элементов):

```
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo umount ./dir
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo cat /proc/slabinfo | grep myfs
myfs_inode_cache       0    170     24  170    1 : tunables    0    0    0 : slabdata      1      1      0
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ dmesg | grep MYFS
[  266.249534] MYFS_MODULE loaded
[  713.828982] MYFS mounted!
[ 1522.612199] MYFS super block destroyed!
```

Выгрузка модуля:

```
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo rmmod myfs.ko
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ dmesg | grep MYFS
[  266.249534] MYFS_MODULE loaded
[  713.828982] MYFS mounted!
[ 1522.612199] MYFS super block destroyed!
[ 1628.312690] MYFS_MODULE unloaded
```

В программе реализовано заполнение кэша элементами inode.
Slab - это динамическая структура, «добирающая» столько страниц памяти, сколько нужно для поддержания размещения требуемого числа элементов данных (с учётом их размера). Это продемонстрировано на следующем скриншоте: количество выделенных объектов (число, следующее за количеством активных элементов) увеличилось в два раза при увеличении количества активных элеметов на один.

```
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo insmod myfs.ko number=169
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo mount -o loop -t myfs ./image ./dir
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo cat /proc/slabinfo | grep myfs
myfs_inode_cache     170    170     24  170    1 : tunables    0    0    0 : slabdata      1      1      0
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo umount ./dir
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo rmmod myfs.ko
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo insmod myfs.ko number=170
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo mount -o loop -t myfs ./image ./dir
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo cat /proc/slabinfo | grep myfs
myfs_inode_cache     171    340     24  170    1 : tunables    0    0    0 : slabdata      2      2      0
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo umount ./dir
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo rmmod myfs.ko
```

Тестирование программы при разном количестве элементов: заполнение кэша 1000 элементами.

```
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo insmod myfs.ko number=1000
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo mount -o loop -t myfs ./image ./dir
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo cat /proc/slabinfo | grep myfs
myfs_inode_cache    1001   1020     24  170    1 : tunables    0    0    0 : slabdata      6      6      0
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo umount ./dir
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo rmmod myfs.ko
```

Заполнение кэша 10000 элементами.

```
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo insmod myfs.ko number=10000
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo mount -o loop -t myfs ./image ./dir
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo cat /proc/slabinfo | grep myfs
myfs_inode_cache   10001  10030     24  170    1 : tunables    0    0    0 : slabdata     59     59      0
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo umount ./dir
oleg@Moxxx1e:~/Документы/BMSTU/OS/lab_08$ sudo rmmod myfs.ko
```