|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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: Создание виртуальной файловой системы.**

Преподаватель: Рязанова Н. Ю.

Студент: Мирзоян С.А.,

Группа: ИУ7-65Б

Москва, 2020 г.

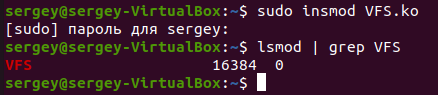
**Листинг**

1. #include <linux/init.h>
2. #include <linux/module.h>
3. #include <linux/kernel.h>
4. #include <linux/pagemap.h>
5. #include <linux/fs.h>
6. #include <asm/atomic.h>
7. #include <asm/uaccess.h>
8. #include <linux/slab.h>
10. #define VFS\_MAGIC\_NUMBER 0x13131313
11. #define SLABNAME "vfs\_cache"
13. MODULE\_LICENSE("GPL");
14. MODULE\_AUTHOR("Sergey Mirzoyan");
15. MODULE\_DESCRIPTION("VFS");

18. **static** **int** size = 7;
19. module\_param(size, **int**, 0);
20. **static** **int** number = 31;
21. module\_param(number, **int**, 0);
23. **static** **void**\* \*line = NULL;
25. **static** **void** co(**void**\* p)
26. {
27. \*(**int**\*)p = (**int**)p;
28. }
29. **struct** kmem\_cache \*cache = NULL;
31. **static** **struct** vfs\_inode
32. {
33. **int** i\_mode;
34. unsigned **long** i\_ino;
35. } vfs\_inode;
37. // Создание inode
38. **static** **struct** inode \* vfs\_make\_inode(**struct** super\_block \*sb, **int** mode)
39. {
40. **struct** inode \*ret = new\_inode(sb);
41. **if** (ret)
42. {
43. inode\_init\_owner(ret, NULL, mode);
44. ret->i\_size = PAGE\_SIZE;
45. ret->i\_atime = ret->i\_mtime = ret->i\_ctime = current\_time(ret);
46. ret->i\_private = &vfs\_inode;
47. }
48. **return** ret;
49. }
51. **static** **void** vfs\_put\_super(**struct** super\_block \*sb)
52. {
53. printk(KERN\_DEBUG "VFS super block destroyed\n");
54. }
56. // Операции структуры суперблок
57. **static** **struct** super\_operations **const** vfs\_super\_ops = {
58. .put\_super = vfs\_put\_super,
59. .statfs = simple\_statfs,
60. .drop\_inode = generic\_delete\_inode,
61. };
63. // Функция инициализации суперблока
64. // Создание корневого каталога ФС
65. **static** **int** vfs\_fill\_sb(**struct** super\_block \*sb, **void** \*data, **int** silent)
66. {
67. **struct** inode\* root = NULL;
69. sb->s\_blocksize = PAGE\_SIZE;
70. sb->s\_blocksize\_bits = PAGE\_SHIFT;
71. sb->s\_magic = VFS\_MAGIC\_NUMBER;
72. sb->s\_op = &vfs\_super\_ops;
74. root = vfs\_make\_inode(sb, S\_IFDIR | 0755);
75. **if** (!root)
76. {
77. printk (KERN\_ERR "VFS inode allocation failed\n");
78. **return** -ENOMEM;
79. }
81. root->i\_op = &simple\_dir\_inode\_operations;
82. root->i\_fop = &simple\_dir\_operations;
84. sb->s\_root = d\_make\_root(root);
85. **if** (!sb->s\_root)
86. {
87. printk(KERN\_ERR "VFS root creation failed\n");
88. iput(root);
89. **return** -ENOMEM;
90. }
91. **return** 0;
92. }
94. // Монтирование ФС
95. **static** **struct** dentry\* vfs\_mount(**struct** file\_system\_type \*type, **int** flags, **const** **char** \*dev, **void** \*data)
96. {
97. **struct** dentry\* **const** entry = mount\_nodev(type, flags, data, vfs\_fill\_sb);
99. **if** (IS\_ERR(entry))
100. printk(KERN\_ERR "VFS mounting failed\n");
101. **else**
102. printk(KERN\_DEBUG "VFS mounted\n");
104. **return** entry;
105. }
107. **static** **struct** file\_system\_type vfs\_type = {
108. .owner = THIS\_MODULE, // Счетчик ссылок на модуль
109. .name = "Sergey\_vfs", // Название ФС
110. .mount = vfs\_mount, // Функция, вызываемая при монтировании ФС
111. .kill\_sb = kill\_litter\_super, // Функция, вызываемая при размонтировании ФС
112. };
114. // Инициализация модуля
115. **static** **int** \_\_init vfs\_module\_init(**void**)
116. {
117. **int** i;
118. **if**(size < 0)
119. {
120. printk(KERN\_ERR "VFS invalid sizeof objects\n");
121. **return** -EINVAL;
122. }
124. line = kmalloc(**sizeof**(**void**\*) \*number, GFP\_KERNEL);
125. **if**(!line)
126. {
127. printk(KERN\_ERR "VFS kmalloc error\n");
128. kfree(line);
129. **return** -ENOMEM;
130. }
132. **for**(i = 0; i < number; i++)
133. line[i] = NULL;
135. // Создание кэша slab
136. cache = kmem\_cache\_create(SLABNAME, size, 0, SLAB\_HWCACHE\_ALIGN, co);
138. **if**(!cache)
139. {
140. printk(KERN\_ERR "VFS cannot create cache\n");
141. // Уничтожение slab
142. kmem\_cache\_destroy(cache);
143. **return** -ENOMEM;
144. }
146. **for**(i = 0; i < number; i++)
147. {
148. **if**(NULL == (line[i] = kmem\_cache\_alloc(cache, GFP\_KERNEL)))
149. {
150. printk(KERN\_ERR "VFS cannot alloc cache\n");
151. **for**(i = 0; i < number; i++ )
152. kmem\_cache\_free(cache, line[i]);
153. **return** -ENOMEM;
154. }
155. }
157. printk(KERN\_INFO "VFS allocate %d objects into slab: %s\n", number, SLABNAME);
158. printk(KERN\_INFO "VFS object size %d bytes, full size %ld bytes\n", size, (**long**)size \*number);
160. // Регистрация файловой системы
161. **int** ret = register\_filesystem(&vfs\_type);
162. **if** (ret != 0)
163. {
164. printk(KERN\_ERR "VFS cannot register filesystem\n");
165. **return** ret;
166. }
168. printk(KERN\_DEBUG "VFS loaded\n");
169. **return** 0;
170. }
172. **static** **void** \_\_exit vfs\_module\_exit(**void**)
173. {
174. **int** i;
175. **for**(i = 0; i < number; i++)
176. {
177. kmem\_cache\_free(cache, line[i]);
178. }
180. kmem\_cache\_destroy(cache);
181. kfree(line);
183. **if** (unregister\_filesystem(&vfs\_type) != 0)
184. {
185. printk(KERN\_ERR "VFS cannot unregister filesystem!\n");
186. }
188. printk(KERN\_DEBUG "VFS unloaded!\n");
189. }
191. module\_init(vfs\_module\_init);
192. module\_exit(vfs\_module\_exit);

**Результат работы программы**

Загрузка модуля.

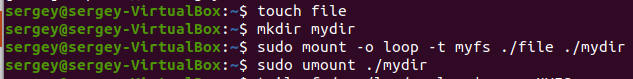


touch file – Создается образ диска

mkdir mydir – Создается корень файловой системы

sudo mount –o loop –t myfs ./file ./mydir – монтирование файловой системы

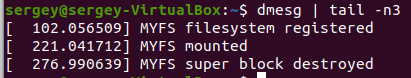
sudo umount ./mydir – размонтирование файловой системы



ls –l



Сообщения о регистрации, монтировании и уничтожении ФС.



Состояние Slab (после монтирования)



Созданная файловая система : df - отчёт об использовании дискового пространства, -a - Включает в список файловых систем те, которые имеют размер в 0 блоков, -T – тип