|  |  |
| --- | --- |
| **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»(ИУ7)**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 8 |

**Название:** Создание виртуальной файловой системы

**Дисциплина:** Операционные системы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-62б |  |  | Блохин Д.М. |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Рязанова Н.Ю. |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2021

**Код программы**

#include <linux/module.h> #include <linux/fs.h> #include <linux/slab.h>

#define VFS\_MAGIC\_NUMBER 0x13131313; #define SLABNAME "my\_cache"

static int sco = 0;  
static struct kmem\_cache \*cache = NULL; static void\* line = NULL;

static void VFS\_put\_sup(struct super\_block \*sup\_block) {

printk(KERN\_DEBUG "VFS super block destroyed\n"); }

static struct super\_operations const VFS\_super\_ops = { .put\_super = VFS\_put\_sup,  
.statfs = simple\_statfs,  
.drop\_inode = generic\_delete\_inode,

};

struct VFS\_inode {

int i\_mode; unsigned

long i\_ino; } VFS\_inode;

static struct inode \*VFS\_create\_inode(struct super\_block \*sup\_block, int mode) {

struct inode \*root = new\_inode(sup\_block);

if (root) {

inode\_init\_owner(root, NULL, mode); root->i\_size = PAGE\_SIZE; root->i\_atime = current\_time(root); root->i\_mtime = current\_time(root); root->i\_ctime = current\_time(root); root->i\_private = &VFS\_inode;

}

return root;

}

static int VFS\_fill\_sup\_block(struct super\_block \*sup\_block, void \*data, int silent) {

struct inode \*root = NULL;

sup\_block->s\_blocksize = PAGE\_SIZE; sup\_block- >s\_blocksize\_bits = PAGE\_SHIFT; sup\_block-

>s\_magic = VFS\_MAGIC\_NUMBER; sup\_block->s\_op = &VFS\_super\_ops;  
root = VFS\_create\_inode(sup\_block, S\_IFDIR|0755);

if (!root) {

printk(KERN\_ERR "VFS inode allocation failed\n");

return -ENOMEM;

}

root->i\_op = &simple\_dir\_inode\_operations; root->i\_fop = &simple\_dir\_operations; sup\_block->s\_root = d\_make\_root(root);

if (!sup\_block->s\_root) {

printk(KERN\_ERR "VFS root creation failed\n"); iput(root);  
return -ENOMEM;

}

return 0; }

static struct dentry\* VFS\_mount(struct file\_system\_type \* type, int flags, char const \*dev, void  
\*data)  
{

struct dentry \*const entry = mount\_nodev(type, flags, data, VFS\_fill\_sup\_block); if (IS\_ERR(entry))

printk(KERN\_ERR "VFS mounting failed!\n"); else

printk(KERN\_DEBUG "VFS mounted\n"); return entry;

}

static struct file\_system\_type VFS\_type = { .owner = THIS\_MODULE,

.name = "VFS",  
.mount = VFS\_mount, .kill\_sb = kill\_block\_super,

};

void co (void \*p)

{

\*(int \*)p = (int)p; sco++; }

static int init VFS\_init(void) {

int ret;

line = kmem\_cache\_alloc(cache, GFP\_KERNEL); if (!line)

{  
printk(KERN\_ERR "VFS kmem\_cache\_alloc error\n"); kmem\_cache\_free(cache, line);

}

cache = kmem\_cache\_create(SLABNAME, sizeof(struct VFS\_inode), 0, 0, co); if (!cache)  
{

printk(KERN\_ERR "VFS\_MODULE cannot allocate cache\n"); kmem\_cache\_destroy(cache);  
return -ENOMEM;

}

ret = register\_filesystem(&VFS\_type); if (ret != 0)  
{

printk(KERN\_ERR "VFS\_MODULE cannot register filesystem\n");

return ret; }

printk(KERN\_INFO "VFS\_MODULE filesystem loaded\n");

return 0; }

static void exit VFS\_exit(void) {

int ret; kmem\_cache\_free(cache, line); kmem\_cache\_destroy(cache);

ret = unregister\_filesystem(&VFS\_type); if (ret != 0)

printk(KERN\_ERR "VFS\_MODULE cannot unregister filesystem!\n"); printk(KERN\_INFO "VFS\_MODULE unloaded\n");

}

MODULE\_LICENSE("Dual BSD/GPL");   
MODULE\_AUTHOR("Blokhin Dmitry");   
module\_init(VFS\_init);   
module\_exit(VFS\_exit);