|  |  |
| --- | --- |
|  | **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»

**ОТЧЕТ**

*к лабораторной работе №8*

*По курсу: «Операционные системы»*

*Тема: «Создание виртуальной файловой системы»*

Студент ИУ7-66Б

Дедич Б.Б

Преподаватель

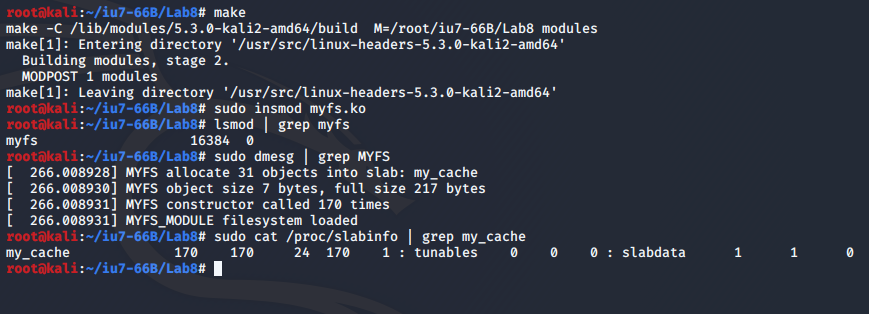
Рязанова Н.Ю.

*Москва, 2021 г.*

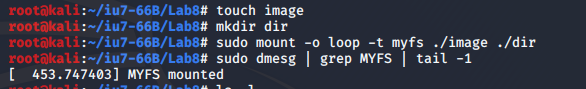
**Текст программы сервера:**

|  |
| --- |
| **#include <linux/module.h>**  #include <linux/kernel.h>  #include <linux/init.h>  #include <linux/fs.h>  #include <linux/time.h>  #include <linux/slab.h>  MODULE\_LICENSE("GPL");  MODULE\_AUTHOR("Dedich Borislav Borislavovich");  #define MYFS\_MAGIC\_NUMBER 0x13131313;  #define SLABNAME "my\_cache"  static int sco = 0;  static struct kmem\_cache \*cache = NULL;  static void\* \*line = NULL;  static int size = 7;  module\_param(size, int, 0);  static int number = 31;  module\_param(number, int, 0);  int free\_allocated\_inodes(struct inode \*inode)  {  kmem\_cache\_free(cache, inode->i\_private);  return 1;  }  // Деструктор суперблока; будет вызван внутри kill\_block\_super перед уничтожением структуры super\_block (при размонтировании ФС)  static void myfs\_put\_super(struct super\_block \*sb)  {  printk(KERN\_DEBUG "MYFS super block destroyed\n");  }  static struct super\_operations const myfs\_super\_ops = {  .put\_super = myfs\_put\_super,  .statfs = simple\_statfs, // заглушка из libfs  .drop\_inode = free\_allocated\_inodes,  };  struct myfs\_inode  {  int i\_mode;  unsigned long i\_ino;  } myfs\_inode;  // Размещает новую структуру inode и заполняет ее значениями  static struct inode \*myfs\_make\_inode(struct super\_block \*sb, int mode)  {  struct inode \*ret = new\_inode(sb);    if (ret)  {  inode\_init\_owner(ret, NULL, mode);  ret->i\_size = PAGE\_SIZE;  ret->i\_atime = ret->i\_mtime = ret->i\_ctime = current\_time(ret);  ret->i\_private = &myfs\_inode;  }  return ret;  }  // Выполняет построение корневого каталога ФС  static int myfs\_fill\_sb(struct super\_block \*sb, void \*data, int silent)  {  struct inode \*root = NULL;  // Заполняется структура super\_block  sb->s\_blocksize = PAGE\_SIZE;  sb->s\_blocksize\_bits = PAGE\_SHIFT;  sb->s\_magic = MYFS\_MAGIC\_NUMBER; // магическое число, по которому драйвер ФС может проверить, что на диске хранится именно та самая ФС, а не прочие данные  sb->s\_op = &myfs\_super\_ops; // операции для суперблока  // Построение корневого каталога ФС  root = myfs\_make\_inode(sb, S\_IFDIR|0755); // создается inode корневого каталога; sb - указатель на суперблок; S\_IFDIR значит, что создаем каталог  if (!root)  {  printk(KERN\_ERR "MYFS inode allocation failed\n");  return -ENOMEM;  }    // файловые и inode операции взяты из libfs (предоставляются ядром)  root->i\_op = &simple\_dir\_inode\_operations;  root->i\_fop = &simple\_dir\_operations;  // Для корневого каталога создается структура dentry, через которую он помещается в directory-кэш  sb->s\_root = d\_make\_root(root); // устанавливаем указатель на dentry корневого каталога  if (!sb->s\_root)  {  printk(KERN\_ERR "MYFS root creation failed\n");  iput(root);  return -ENOMEM;  }  return 0;  }  // Примонтирует устройство и возвращает структуру, описывающую корневой каталог ФС  static struct dentry\* myfs\_mount(struct file\_system\_type \* type, int flags, char const \*dev, void \*data)  {  struct dentry \*const entry = mount\_bdev(type, flags, dev, data, myfs\_fill\_sb); // myfs\_fill\_sb - ук. на ф-ю, которая будет вызвана из mount\_bdev, чтобы проинициализировать суперблок  // dentry представляет корневой каталог ФС  if (IS\_ERR(entry))  printk(KERN\_ERR "MYFS mounting failed!\n");  else  printk(KERN\_DEBUG "MYFS mounted\n");  return entry;  }  // Описывает создаваемую ФС  static struct file\_system\_type myfs\_type = {  .owner = THIS\_MODULE, // отвечает за счетчик ссылок на модуль, чтобы его нельзя было случайно выгрузить  .name = "myfs", // имя ФС; используется при монтировании  .mount = myfs\_mount, // указатель на ф-ю; исп. при монтировании  .kill\_sb = kill\_block\_super, // указатель на ф-ю; исп. при размонтировании; kill\_block\_super предоставляет ядро  };  void co (void \*p)  {  \*(int \*)p = (int)p;  sco++;  }  // Инициализация модуля  static int \_\_init myfs\_init(void)  {  int i, ret;    if(size < 0)  {  printk(KERN\_ERR "MYFS invalid argument %d\n", size);  return -EINVAL;  }    line = kmalloc(sizeof(void\*) \* number, GFP\_KERNEL);  if(!line)  {  printk(KERN\_ERR "MYFS kmalloc error\n");  kfree(line);  return -ENOMEM;  }  for(i = 0; i < number; i++)  line[i] = NULL;    cache = kmem\_cache\_create(SLABNAME, sizeof(struct myfs\_inode), 0, 0, co); // создание кэша slab  if (!cache)  {  printk(KERN\_ERR "MYFS\_MODULE cannot allocate cache\n");  kmem\_cache\_destroy(cache);  return -ENOMEM;  }  for(i = 0; i < number; i++)  {  if(NULL == (line[i] = kmem\_cache\_alloc(cache, GFP\_KERNEL)))  {  printk(KERN\_ERR "MYFS kmem\_cache\_alloc error\n");  for(i = 0; i < number; i++)  kmem\_cache\_free(cache, line[i]);  }  }    ret = register\_filesystem(&myfs\_type);  if (ret != 0)  {  printk(KERN\_ERR "MYFS\_MODULE cannot register filesystem\n");  return ret;  }    printk(KERN\_INFO "MYFS allocate %d objects into slab: %s\n", number, SLABNAME);  printk(KERN\_INFO "MYFS object size %d bytes, full size %ld bytes\n", size, (long)size \* number);  printk(KERN\_INFO "MYFS constructor called %d times\n", sco);  printk(KERN\_INFO "MYFS\_MODULE filesystem loaded\n");  return 0;  }  // Выгрузка модуля  static void \_\_exit myfs\_exit(void)  {  int i, ret;  for(i = 0; i < number; i++)  kmem\_cache\_free(cache, line[i]);  kmem\_cache\_destroy(cache);  kfree(line);    ret = unregister\_filesystem(&myfs\_type);  if (ret != 0)  printk(KERN\_ERR "MYFS\_MODULE cannot unregister filesystem!\n");  printk(KERN\_INFO "MYFS\_MODULE unloaded %d\n", sco);  }  module\_init(myfs\_init);  module\_exit(myfs\_exit); |

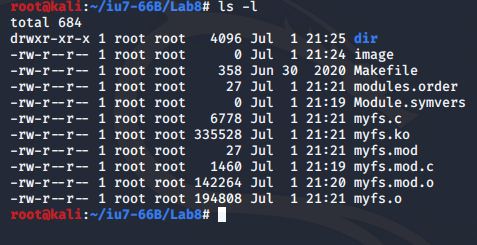
**Рис 1: Загрузка модуля**

****

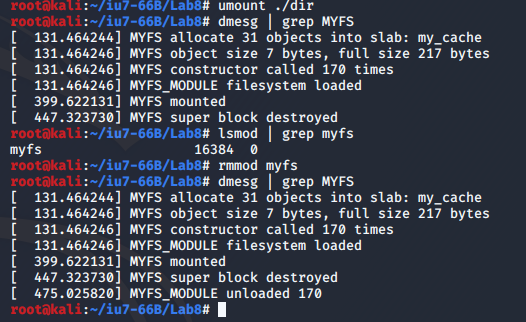
**Рис 2: Монтирование файловой системы**

****

**Рис 3: Файловая система в дереве каталогов**

****

**Рис 4: Выгрузка модуля**



**Рис 5: Загрузка модуля с заданием параметров**

