|  |  |
| --- | --- |
| 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 **ПРОГРАМНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.03 ПРОГРАММНАЯ ИНЖЕНЕРИЯ**

**Отчет**

|  |  |
| --- | --- |
| **По лабораторной работе №** | 8 |

**Дисциплина:** Операционные системы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-62Б |  |  | Н.А. Гарасев |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Н.Ю. Рязанова |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

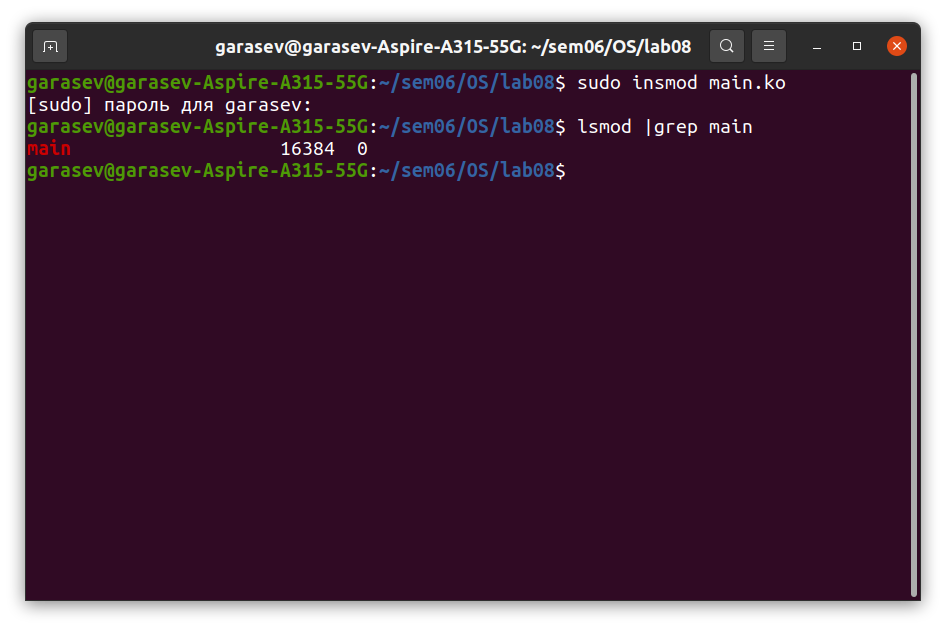
«Создание виртуальной файловой системы»

Код программы.

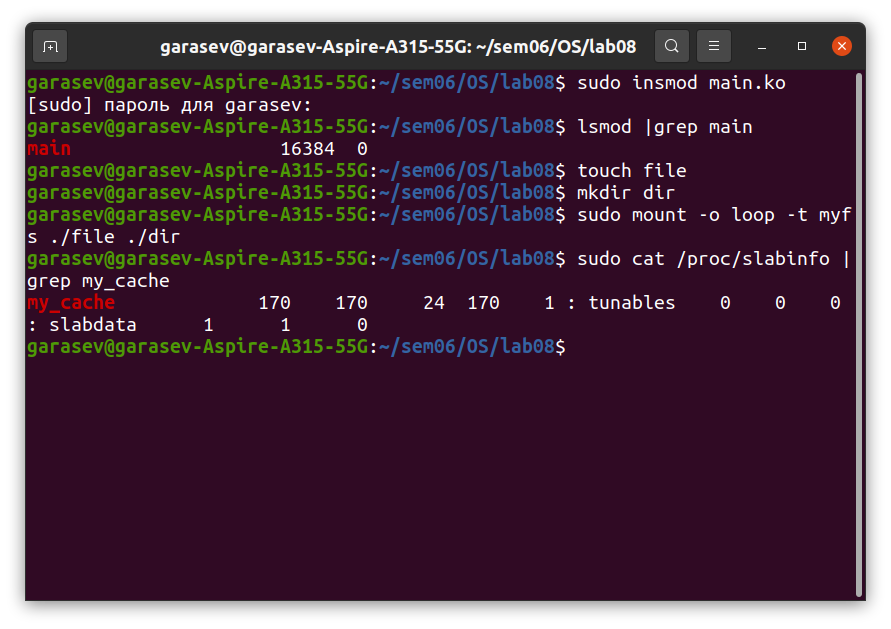
|  |
| --- |
| #include <linux/module.h>  #include <linux/kernel.h>  #include <linux/init.h>  #include <linux/fs.h>  #include <linux/time.h>  #include <linux/slab.h>  #define MYFS\_MAGIC\_NUMBER 0x1313131313;  #define SLABNAME "my\_cache"  struct myfs\_inode  {  int i\_mode;  unsigned long i\_ino;  };  int inode\_number = 0;  static struct kmem\_cache \*cache;  static void myfs\_put\_super(struct super\_block \*sb)  {  printk(KERN\_DEBUG "myfs super block destroyed\n");  }  int free\_alloc\_inodes(struct inode \*inode)  {  kmem\_cache\_free(cache, inode->i\_private);  return 1;  }  static struct super\_operations const myfs\_super\_ops = {  .put\_super = myfs\_put\_super,  .statfs = simple\_statfs,  .drop\_inode = free\_alloc\_inodes,  };  static struct inode \*myfs\_make\_inode(struct super\_block \*sb, int mode)  {  struct inode \*ret = new\_inode(sb);  if (ret)  {  struct myfs\_inode \*my\_inode = kmem\_cache\_alloc(cache, GFP\_KERNEL);  inode\_init\_owner(ret, NULL, mode);  \*my\_inode = (struct myfs\_inode){  .i\_mode = ret->i\_mode,  .i\_ino = ret->i\_ino  };  ret->i\_size = PAGE\_SIZE;  ret->i\_atime = ret->i\_mtime = ret->i\_ctime = current\_time(ret);  ret->i\_private = my\_inode;  }  return ret;  }  static int myfs\_fill\_sb(struct super\_block \*sb, void \*data, int silent)  {  struct inode \*root = NULL;  sb->s\_blocksize = PAGE\_SIZE;  sb->s\_blocksize\_bits = PAGE\_SHIFT;  sb->s\_magic = MYFS\_MAGIC\_NUMBER;  sb->s\_op = &myfs\_super\_ops;  root = myfs\_make\_inode(sb, S\_IFDIR|0755);  if (!root)  {  printk(KERN\_ERR "myfs inode allocation failed\n");  return -ENOMEM;  }  root->i\_op = &simple\_dir\_inode\_operations;  root->i\_fop = &simple\_dir\_operations;  sb->s\_root = d\_make\_root(root);  if (!sb->s\_root)  {  printk(KERN\_ERR "myfs root creation failed\n");  iput(root);  return -ENOMEM;  }  return 0;  }  static struct dentry\* myfs\_mount(struct file\_system\_type \* type, int flags, char const \*dev, void \*data)  {  struct dentry \*const entry = mount\_nodev(type, flags, data, myfs\_fill\_sb);  if (IS\_ERR(entry))  printk(KERN\_ERR "myfs mounting failed!\n");  else  printk(KERN\_DEBUG "myfs mounted");  return entry;    }  static struct file\_system\_type myfs\_type = {  .owner = THIS\_MODULE,  .name = "myfs",  .mount = myfs\_mount,  .kill\_sb = kill\_litter\_super,  };  void co (void \*p)  {  \*(int \*)p = (int)p;  inode\_number++;  }  static int \_\_init myfs\_init(void)  {  int ret = register\_filesystem(&myfs\_type);  cache = kmem\_cache\_create(SLABNAME, sizeof(struct myfs\_inode), 0, 0, co);  if (ret != 0)  {  printk(KERN\_ERR "myfs can't register filesystem\n");  return ret;  }  printk(KERN\_INFO "myfs filesystem registered");  return 0;  }  static void \_\_exit myfs\_exit(void)  {  int ret = unregister\_filesystem(&myfs\_type);  if (ret != 0)  printk(KERN\_ERR "myfs can't unregister filesystem!\n");  kmem\_cache\_destroy(cache);  printk(KERN\_INFO "myfs unregistered %d", inode\_number);  }  MODULE\_LICENSE("GPL");  MODULE\_AUTHOR("Garasev");  module\_init(myfs\_init);  module\_exit(myfs\_exit); |

Демонстрация работы программы.

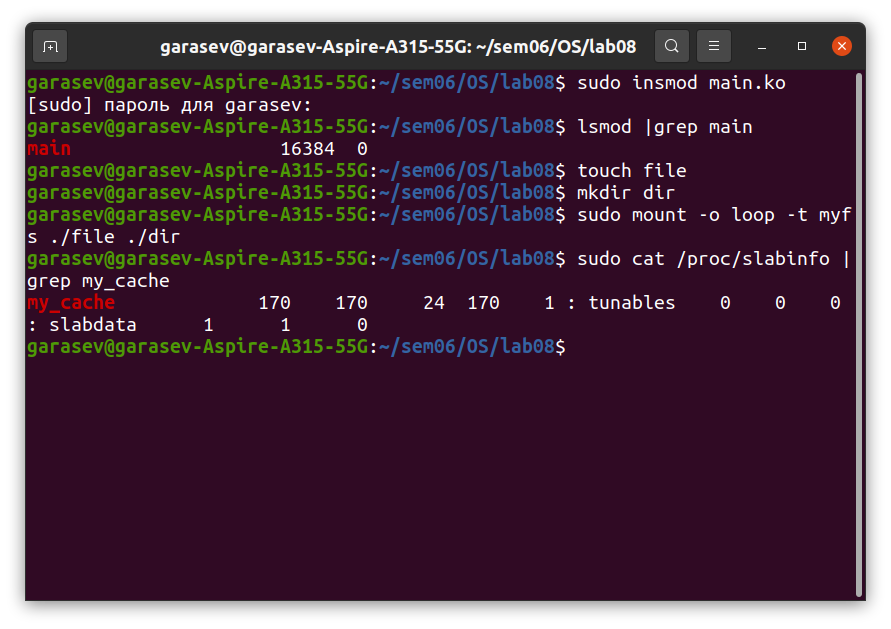
Загрузим модуль ядра командой insmod:



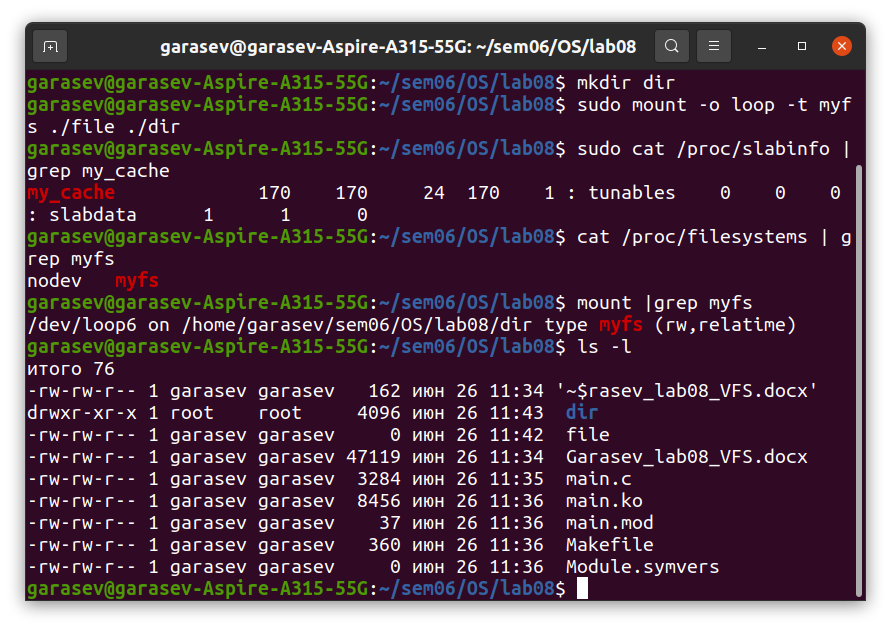
Создадим образ диска и каталог, который будет точкой монтирования фс, а затем примонтируем файловую систему.

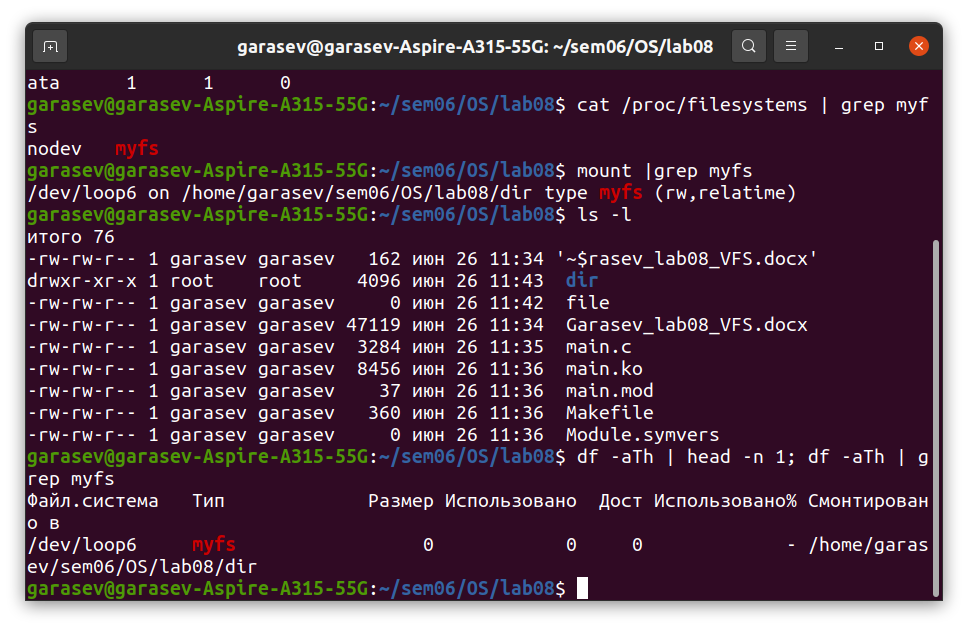


Состояние после монтирования:

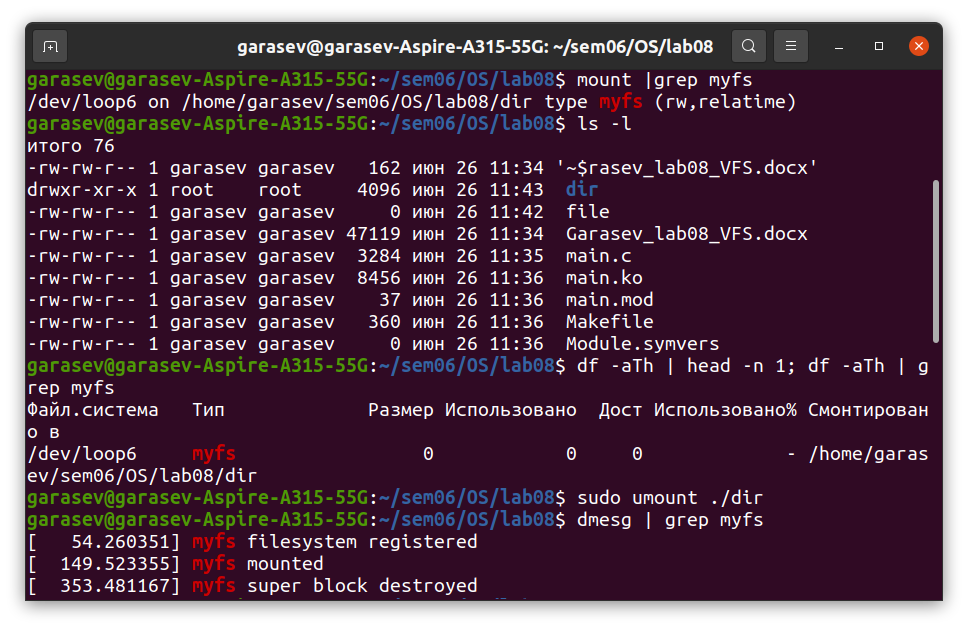


Информация о смонтированной файловой системе:





Файловая система в дереве каталогов:



Размонтирование файловой системы и проверка логов:

