```
/**********
* molab.h
 * sampl
 * *******/

#define DEVNAM "olab" // имя устройства
#define LEN_MSG 255    // длины буферов устройства
#define LOG(...) if( debug !=0 ) printk( KERN_INFO "! "__VA_ARGS__ )
#define MIN( A, B ) (( A<B ) ? A : B )

MODULE_LICENSE( "GPL" );
MODULE_AUTHOR( "author" );
MODULE_VERSION( "1.2" );
/**********
* molab.c
* sampl
 * ********/

#include <linux/module.h>
#include <linux/fs.h>
#include <asm/uaccess.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include "./molab.h"


static int mode = 0; // открытие: 0 - без контроля, 1 - единичное, 2 - множественное
module_param( mode, int, S_IRUGO );
static int debug = 0;
module_param( debug, int, S_IRUGO );

static int dev_open = 0;
struct molab_data {         // область данных драйвера:
    char buf[ LEN_MSG + 1 ]; // буфер данных
    int odd;                 // признак начала чтения
};

static int molab_open( struct inode *n, struct file *f ) {
    LOG( "open - node: %p, file: %p, refcount: %d", n, f,
module_refcount( THIS_MODULE ) );
    if( dev_open ) {
        LOG( "device /dev/%s is busy", DEVNAM );
        return -EBUSY;
    }
    if( 1 == mode ) dev_open++;
    if( 2 == mode ) {
          struct molab_data *data;
          f->private_data = kmalloc( sizeof( struct molab_data ), GFP_KERNEL );
          if( NULL == f->private_data ) {
              LOG( "memory allocation error" );
              return -ENOMEM;
          }
          data = (struct molab_data*)f->private_data;
          strcpy( data->buf, "dynamic: not initialized!" ); // динамический буфер
          data->odd = 0;
   }
    return 0;
}

static int molab_release( struct inode *n, struct file *f ) {
    LOG( "close - node: %p, file: %p, refcount: %d", n, f,
module_refcount( THIS_MODULE ) );
    if( 1 == mode ) dev_open--;
    if( 2 == mode ) kfree( f->private_data );
    return 0;
}
static struct molab_data* get_buffer( struct file *f ) {
    static struct molab_data static_buf = { "static: not initialized!", 0 }; //
статический буфер
```

```
67      return 2 == mode ? (struct molab_data*)f->private_data : &static_buf;
68  }
69
70  // чтение из /dev/mopen :
71  static ssize_t molab_read( struct file *f, char *buf, size_t count, loff_t *pos ) {
72      struct molab_data* data = get_buffer( f );
73      LOG( "read - file: %p, read from %p bytes %d; refcount: %d",
74            f, data, count, module_refcount( THIS_MODULE ) );
75      if( 0 == data->odd ) {
76          int res = copy_to_user( (void*)buf, data->buf, strlen( data->buf ) );
77          data->odd = 1;
78          put_user( '\n', buf + strlen( data->buf ) );
79          res = strlen( data->buf ) + 1;
80          LOG( "return bytes : %d", res );
81          return res;
82      }
83      data->odd = 0;
84      LOG( "return : EOF" );
85      return 0;
86  }
87
88  // запись в /dev/mopen :
89  static ssize_t molab_write( struct file *f, const char *buf, size_t count, loff_t *pos )
    {
90      int res, len = MIN( count, LEN_MSG );
91      struct molab_data* data = get_buffer( f );
92      LOG( "write - file: %p, write to %p bytes %d; refcount: %d",
93            f, data, count, module_refcount( THIS_MODULE ) );
94      res = copy_from_user( data->buf, (void*)buf, len );
95      if( '\n' == data->buf[ len -1 ] ) data->buf[ len -1 ] = '\0';
96      else data->buf[ len ] = '\0';
97      LOG( "put bytes : %d", len );
98      return len;
99  }
100
101 static const struct file_operations molab_fops = {
102     .owner = THIS_MODULE,
103     .open =    molab_open,
104     .release = molab_release,
105     .read   = molab_read,
106     .write = molab_write,
107 };
108
109 static struct miscdevice molab_dev = {
110     MISC_DYNAMIC_MINOR, DEVNAM, &molab_fops
111 };
112
113 static int __init molab_init( void ) {
114     int ret = misc_register( &molab_dev );
115     if( ret ) { LOG( "unable to register %s misc device", DEVNAM );
116     }else{ LOG( "installed device /dev/%s in mode %d", DEVNAM, mode ); }
117     return ret;
118 }
119
120 static void __exit molab_exit( void ) {
121      LOG( "released device /dev/%s", DEVNAM );
122      misc_deregister( &molab_dev );
123 }
124 module_init( molab_init );
125 module_exit( molab_exit );
126
127 /* Makefile */
128 CURRENT = $(shell uname -r)
129 KDIR = /lib/modules/$(CURRENT)/build
130 PWD = $(shell pwd)
131 DEST = /lib/modules/$(CURRENT)/misc
132
133 TARGET = molab
134 obj-m   := $(TARGET).o
```

```
135
136  default:
137          $(MAKE) -C $(KDIR) M=$(PWD) modules
138
139  clean:
140          @rm -f *.o .*.cmd .*.flags *.mod.c *.order
141          @rm -f .*.*.cmd *.symvers *~ *.*~ TODO.*
142          @rm -fR .tmp*
143          @rm -rf .tmp_versions
144
145
```