Лекция 1.2

Символьные драйверы

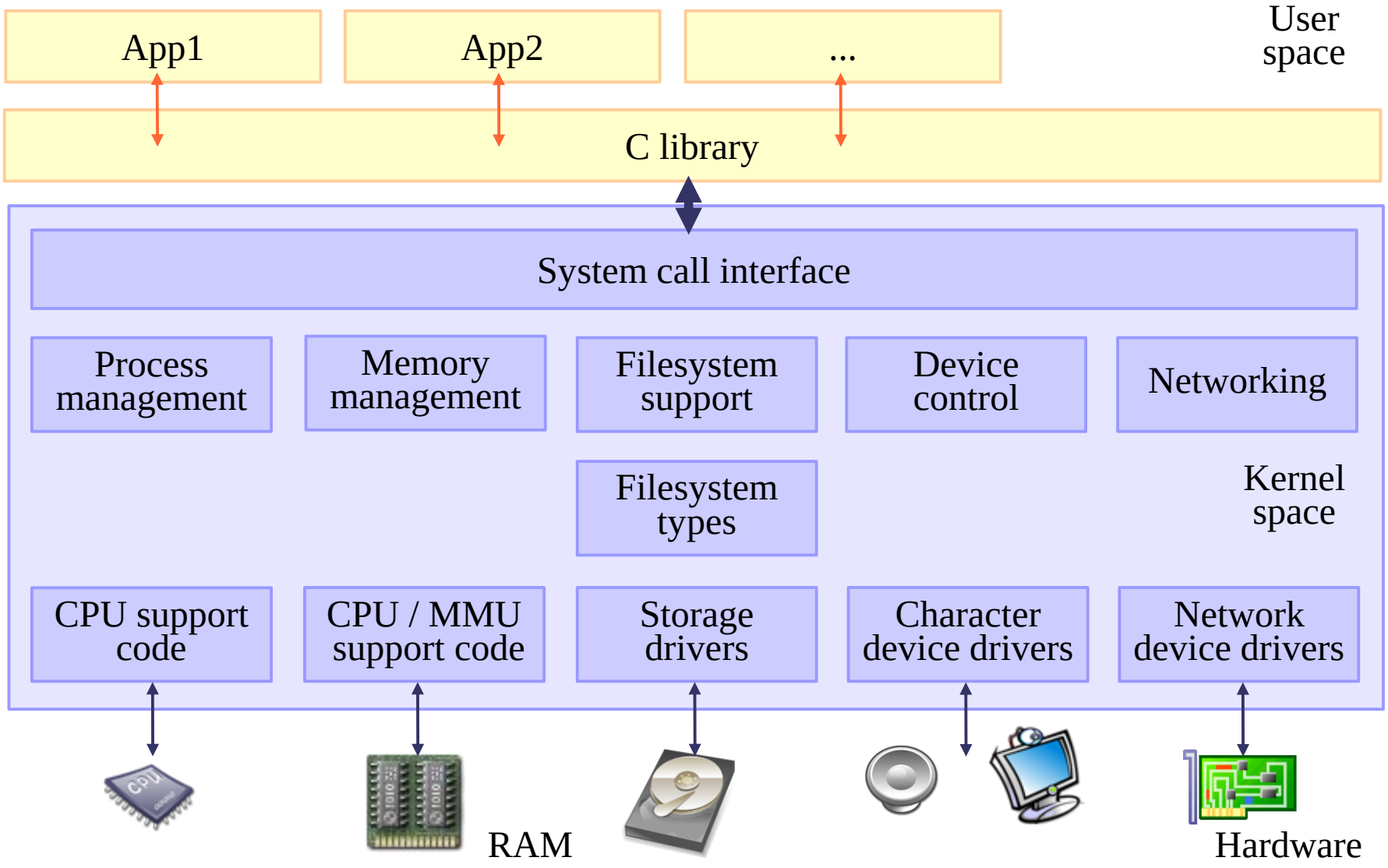
Разработал: Максимов А.Н.

Версия 1.9. 02.2025

Содержание

- Символьный драйвер

Часть 1. Архитектура ядра



Символьный драйвер

Символьный драйвер обрабатывает поток байт. Обычно драйвер отвечает за реализацию следующих системных вызовов open, close, read, and write .

Примеры:

console	/dev/console
serial ports	/dev/ttyS0, /dev/ttyUSB0
IrDA	
Bluetooth	
web камера	/dev/video0
GPIO	/dev/gpiochip0
...	

Доступ к символьным устройствам осуществляется через файловую систему. Они представляются в виде файла.

User space API для работы с символьным драйвером

```
int open(char *path,int oflags,mode_t permission)
int close(int fd)
int read(int fd,char *buff,int count)
int write(int fd,char *buff,int count)
```

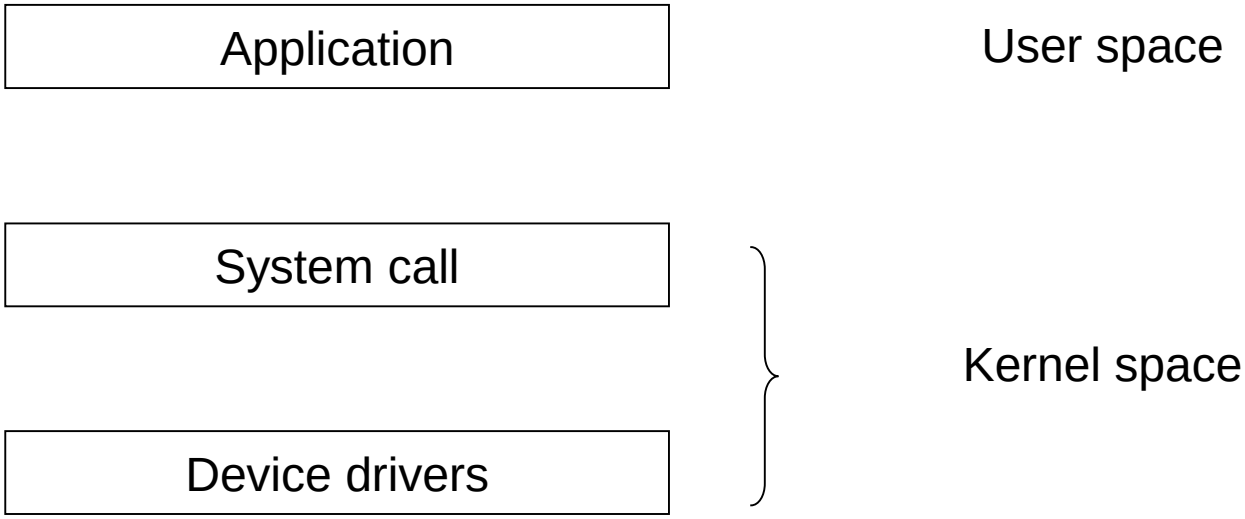
и др.

Приме пользовательского приложения для работы с символьным драйвером

```
#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>

int main() {
    int fd;
    char buf[100];
    fd = open("/dev/com1",O_RDWR);
    read(fd,buf,20);
    buf[20]=0;
    printf("Input: >>> %s <<<\n",buf);
  close(fd);
}
```

Архитектура ввода-вывода



Идентификация символьных драйверов

Для пользовательских процессов символьный драйвер представляется в виде файла устройства в каталоге /dev

Если выполнить команду ls -l мы, то символьные драйверы будут помечены c, блочные b

```
crw------- 1 root tty      5,   1 2008-10-30 13:07 console
crw-rw---- 1 root root    10, 252 2008-10-30 16:06 dac960_gam
crw-rw---- 1 root audio   14,   9 2008-10-30 16:06 dmmidi
brw-rw---- 1 root floppy   2,   4 2008-10-30 16:06 fd0d360
brw-rw---- 1 root floppy   2,   8 2008-10-30 16:06 fd0h1200
brw-rw---- 1 root floppy   2,  40 2008-10-30 16:06 fd0h1440
crw-r----- 1 root kmem     1,   2 2008-10-30 16:06 kmem
crw-rw---- 1 root root     1,  11 2008-10-30 16:06 kmsg
crw-rw-r-- 1 root lp       6,   0 2008-10-30 13:07 lp0
crw-r----- 1 root kmem     1,   1 2008-10-30 16:06 mem
crw-rw---- 1 root audio   14,   2 2008-10-30 16:06 midi
crw-rw---- 1 root uucp   108,   0 2005-11-21 08:29 ppp
crw-rw-rw- 1 root tty      5,   2 2008-10-30 16:06 ptmx
crw-rw-r-- 1 root root     1,   9 2008-10-30 13:07 urandom
crw-rw---- 1 root root   189,   0 2008-10-30 16:06 usbdev1.1
crw-rw---- 1 root root   250,   0 2008-10-30 16:06 usbdev1.1_ep00
crw-rw---- 1 root root   250,   1 2008-10-30 16:06 usbdev1.1_ep81
crw-rw-rw- 1 root root     1,   5 2005-11-21 06:22 zero
```

Драйвер имеет :

major_number -

идентификации ядром ОС

minor_number -

ядро не использует,

использует сам драйвер для того, чтобы различать устройства)

RISoft

Компоненты символьного драйвера драйвера

- Инициализация (init) инициализация устройства, регистрация драйвера в ядре
- Реализация функций для точек входа (open, close, read, write, ioctl), соответствующих системным вызовам
- Обработчики прерываний, обработчики таймеров, bottom halves, kernel threads

Структуры данных драйвера

1. struct file_operations
2. struct file

Интерфейс символьного драйвера **file_operations** (/linux/fs.h)

```
struct file_operations {
   struct module *owner;                        // Pointer to the LKM that owns the structure
   loff_t (*llseek) (struct file *, loff_t, int);    // Change current read/write position in a file
   ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);    // Used to retrieve data from the device
   ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);   // Used to send data to the device
   ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);  // Asynchronous read
   ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t); // Asynchronous write
   ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);          // possibly asynchronous read
   ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);         // possibly asynchronous write
   int (*iterate) (struct file *, struct dir_context *);              // called when VFS needs to read the directory
contents
   unsigned int (*poll) (struct file *, struct poll_table_struct *);    // Does a read or write block?
   long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long); // Called by the ioctl system call
   long (*compat_ioctl) (struct file *, unsigned int, unsigned long);   // Called by the ioctl system call
   int (*mmap) (struct file *, struct vm_area_struct *);              // Called by mmap system call
   int (*mremap)(struct file *, struct vm_area_struct *);             // Called by memory remap system call
   int (*open) (struct inode *, struct file *);           // first operation performed on a device file
   int (*flush) (struct file *, fl_owner_t id);           // called when a process closes its copy of the descriptor
   int (*release) (struct inode *, struct file *);        // called when a file structure is being released
   int (*fsync) (struct file *, loff_t, loff_t, int datasync);  // notify device of change in its FASYNC flag
   int (*aio_fsync) (struct kiocb *, int datasync);       // synchronous notify device of change in its FASYNC flag
   int (*fasync) (int, struct file *, int);               // asynchronous notify device of change in its FASYNC flag
   int (*lock) (struct file *, int, struct file_lock *);    // used to implement file locking
   …
};
```

Реализация функций драйвером

Для реализациии итерфейса драйвер создает структуру file_operations.

Символьный драйвер реализует необходимые ему функции.

Для остальных указатеть в file_operations устанавливается в NULL.

```
struct file_operations fops = {
   .read = device_read,
   .write = device_write,
   .open = device_open,
   .release = device_release
};
```

инициализация структуры с ипользованием синтаксисиса C99.

Номер устройства

- В ядре для представления пары старший и младший номер устройства используется тип dev_t
- Определен в <linux/kdev_t.h> в Linux 2.6: 32 bit size (major: 12 bits, minor: 20 bits)
- Макрос для создания номера устройства MKDEV(int major, int minor);
- Макрос для извлечения номера устройства:

MAJOR(dev_t dev);

MINOR(dev_t dev);

Пример использования

```
int init_module(void)
{
   Major = register_chrdev(0, DEVICE_NAME, &fops);
   if (Major < 0) {
       printk(KERN_ALERT "Registering char device failed with %d\n", Major);
       return Major;
   }
   printk(KERN_INFO "I was assigned major number %d. To talk to\n", Major);
   printk(KERN_INFO "the driver, create a dev file with\n");
   printk(KERN_INFO "'mknod /dev/%s c %d 0'.\n", DEVICE_NAME, Major)

   return 0;
}
```

Деинициализация драйвера

```
void cleanup_module(void) {
  /* Удаление символьного драйвера  */
  unregister_chrdev(Major, DEVICE_NAME);
  printk(KERN_ALERT "Cleanup_module OK \n");
}
```

Детально file_operation

int (*open) (struct inode *, struct file *);

Вызывается, когда открывается файл.

int (*release) (struct inode *, struct file *);

Вызывается, когда файл закрывается.

Пример

```
static int device_open(struct inode *inode, struct file *file){
   static int counter = 0;
   if (Device_Open)
     return -EBUSY;
   Device_Open++;
   sprintf(msg, "I already told you %d times Hello world!\n", counter++);
   msg_Ptr = msg;
   try_module_get(THIS_MODULE);
   return SUCCESS;
}
static int device_release(struct inode *inode, struct file *file){
  Device_Open--; /* We're now ready for our next caller */
    /** Decrement the usage count, or else once you opened the file, you'll
       never get get rid of the module.   */
   module_put(THIS_MODULE);
  return 0;
}
```

Структура file

Структура file, создается когда происходит вызов open

mode_t f_mode;

Режим октрытия файла (FMODE_READ and/or FMODE_WRITE)

loff_t f_pos; текущий offset в файле.

struct file_operations *f_op;

struct dentry *f_dentry

Операции записи и чтения

```
ssize_t (*read) (
   struct file *, /* Open file descriptor */
   __user char *, /* Userspace buffer to fill up */
   size_t,        /* Size of the userspace buffer */
   loff_t *);     /* Offset in the open file */
```

Вызывается при чтении из файла

```
ssize_t (*write) (
   struct file *,       /* Open file descriptor */
   __user const char *, /* Userspace  buffer to write to the device */
   size_t,              /* Size of the userspace  buffer */
   loff_t* );           /* Offset in the open file */
```

Пример реализации

```
static ssize_t device_read(struct file *filp,    char *buffer, /* buffer to fill with data */
                size_t length, /* length of the buffer */     loff_t * offset)
{
  int bytes_read = 0; /* Number of bytes actually written to the buffer*/
     /* If we're at the end of the message, * return 0 signifying end of file */
   if (*msg_Ptr == 0)  return 0;
   /* Actually put the data into the buffer */
   while (length && *msg_Ptr) {
    /** The buffer is in the user data segment, not the kernel
    * segment so "*" assignment won't work. We have to use
    * put_user which copies data from the kernel data segment to
    * the user data segment.   */
     put_user(*(msg_Ptr++), buffer++);
     length--;
     bytes_read++;
  }
  /* Most read functions return the number of bytes put into the buffer */
  return bytes_read;
}
```

Обмен данными с пользовательским процессом

Драйвер не может использовать memcpy для обмена между памятью user space и памяться kernel space

В функция read и write необходимо использовать специальные функции:

include <asm/uaccess.h>

unsigned long **copy_to_user** (void __user *to, const void *from, unsigned long n);

unsigned long **copy_from_user** (void *to, const void __user *from, unsigned long n);

Функции в случае успеха возвращают 0

Установка драйвера в ядро

1. Вставить модуль:

insmod foodrv.ko

2. Посмотреть в /proc/devices, какой у драйвера major_number

(предположим 100)

3. Создать файл устройства:

mknod /dev/foodev c 100 0

(для автоматического создания файла устройства можно использовать udev)

Динамическое выделение памяти

malloc - не существует.

kmalloc/kfree выделяет до 128k физической памяти в последовательном блоке

kcalloc(. . .) аналогично kmalloc, но память обнуляется

void *kmalloc (size t size, int flags);

flags определены в <linux/mm.h>

GFP_USER associated userspace process sleeps until free memory available

GFP_KERNEL associated kernel function sleeps until free memory available

GFP_ATOMIC doesn't sleep (used in ISRs)

Литература

1. Пособие по программированию модулей ядра Linux. Ч.1 https://habr.com/ru/company/ruvds/blog/681880/
2. Пособие по программированию модулей ядра Linux. Ч.2 https://habr.com/ru/company/ruvds/blog/683106/
3. Writing a Linux Kernel Module — Part 2: A Character Device http://derekmolloy.ie/writing-a-linux-kernel-module-part-2-a-character-device/
4. The linux-kernel mailing list FAQ http://www.tux.org/lkml.
5. Rob Day The Kernel Newbie Corner: What's in That Loadable Module, Anyway? http://www.linux.com/learn/linux-training/32867-the-kernel-newbie-corner-whats-in-that-loadable-module-anyway
6. Andrew Murray Init Call Mechanism in the Linux Kernel

http://linuxgazette.net/157/amurray.html

7. Kernel modules https://wiki.archlinux.org/index.php/Kernel_modules (полезные команды и конфигурирование)
8. https://linux-kernel-labs.github.io/refs/heads/master/labs/device_drivers.html#overview
9. Automatic create /dev file https://gist.github.com/strezh/b01fcd50875c214e510a81c6aa6d2a2a
10. https://embetronicx.com/tutorials/linux/device-drivers/device-file-creation-for-character-drivers/#Automatically_Creating_Device_File
11. https://sysprog21.github.io/lkmpg/#avoiding-collisions-and-deadlocks
12. Understanding the Structure of a Linux Kernel Device Driver https://www.youtube.com/watch?v=XoYkHUnmpQo