Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Лабораторная работа

Дисциплина: Проектирование ОС и компонентов

Тема: Создание драйвера символьного устройства

Выполнил студент гр. 13541/4 \_\_\_\_\_\_\_\_\_\_ Зорин А.Г.

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_ Душутина Е.В.

(подпись)

«\_\_»\_\_\_\_\_\_\_\_ 2017 г.

Санкт –Петербург

2017

**Цель работы:**

В данной работе необходимо изучить способы создания драйвера символьного устройства и создать собственный драйвер в ОС Linux.

**Выполнение работы:**

В данной работе, для написания драйвера символьного устройства была использована ОС Ubuntu, установленная на виртуальную машину. Сделано это было во избежании нанесения непоправимого ущерба рабочей системе. Информация об использованной системе приведена ниже:

|  |
| --- |
| arseny@ubuntuVM:~$ uname -a  Linux ubuntuVM 4.8.0-51-generic #54~16.04.1-Ubuntu SMP Wed Apr 26 16:00:28 UTC 2017 x86\_64 x86\_64 x86\_64 GNU/Linux |

Драйвер символьного устройства – драйвер для байт-ориентированных операций. Для любого приложения, предназначенного для работы с байт-ориентированным устройством, следует использовать соответствующий драйвер символьного устройства. Использование символьных драйверов осуществляется через соответствующие файлы символьных устройств.

Файлы устройств подключаются к драйверу устройства с помощью специального механизма регистрации, что осуществляется драйвером. Драйвер связывается с устройством с помощью специальных низкоуровневых операций, характерных для конкретного устройства. Таким образом, формируется полное соединение. При этом, файл символьного устройства не является реальным устройством, это просто специальная методика (place-holder) подключения реального устройства.

При подключении приложения к файлу устройства используется имя файла устройства. Но при подключении файла устройства к драйверу устройства используется номер файла устройства, а не имя файла. В результате, приложение пользовательского пространства может использовать для файла устройства любое имя, а в пространстве ядра для связи между файлом устройства и драйвером устройства можно использовать тривиальный механизм индексации. Таким номером файла обычно является пара <major, minor>, т. е. или старший и младший номера файла устройства.

Ранее (вплоть до ядра 2.4) каждый старший номер использовался в качестве указания на отдельный драйвер, а младший номер использовался для указания на конкретное подмножество функциональных возможностей драйвера. В ядре 2.6 такое использование номеров не является обязательным; с одним и тем же старшим номером может быть несколько драйверов, но, очевидно, с различными диапазонами младших номеров.

Однако, такое использование больше характерно для незарезервированных старших номеров, а стандартные старшие номера обычно резервируются для вполне определенных конкретных драйверов. Например, 4 — для последовательных интерфейсов, 13 - для мышей, 14 — для аудио-устройств и так далее.

Для выполнения данной работы был написан простой модуль ядра, работающий с файлом устройства /dev/test

|  |
| --- |
| #include <linux/kernel.h>  #include <linux/module.h>  #include <linux/init.h>  #include <linux/fs.h>  #include <asm/uaccess.h>  // Информация о модуле, которую можно будет увидеть с помощью Modinfo  MODULE\_LICENSE( "GPL" );  MODULE\_AUTHOR( "Zorin Arseny <zorinarseny@yandex.ru>" );  MODULE\_DESCRIPTION( "My test module" );  MODULE\_SUPPORTED\_DEVICE( "test" );  #define SUCCESS 0  #define DEVICE\_NAME "test" /\* Имя устройства \*/  // Поддерживаемые устройством операции  static int device\_open( struct inode \*, struct file \* );  static int device\_release( struct inode \*, struct file \* );  static ssize\_t device\_read( struct file \*, char \*, size\_t, loff\_t \* );  static ssize\_t device\_write( struct file \*, const char \*, size\_t, loff\_t \* );  static int major\_number; /\* Старший номер устройства нашего драйвера \*/  static int is\_device\_open = 0; /\* Используется ли устройство \*/  static char text[ 20 ] = "This is test device\n"; /\* Текст, который будет выводиться при обращении к устройству \*/  static char\* text\_ptr = text; /\* Указатель на текущую позицию в тексте \*/  // Обработчики операций над устройством  static struct file\_operations fops =  {  .read = device\_read,  .write = device\_write,  .open = device\_open,  .release = device\_release  };  // Функция загрузки модуля. Входная точка.  static int \_\_init test\_init( void )  {  printk( KERN\_INFO "TEST driver loaded!\n" );  // Регистрирайия устройсва и получение старшего номера устройства  major\_number = register\_chrdev( 0, DEVICE\_NAME, &fops );  if ( major\_number < 0 )  {  printk( "Character device registration was failed with %d\n", major\_number );  return major\_number;  }  // Сообщаем присвоенный старший номер устройства  printk( KERN\_INFO "Test module was loaded!\n" );  printk( KERN\_WARNING "Please, create a dev file with 'mknod /dev/test c %d 0'.\n", major\_number );  return SUCCESS;  }  // Функция выгрузки модуля  static void \_\_exit test\_exit( void )  {  // Освобождение устройства  unregister\_chrdev( major\_number, DEVICE\_NAME );  printk( KERN\_INFO "Test module was unloaded!\n" );  }  // Указание функций загрузки и выгрузки  module\_init( test\_init );  module\_exit( test\_exit );  static int device\_open( struct inode \*inode, struct file \*file )  {  text\_ptr = text;  if ( is\_device\_open )  return -EBUSY;  is\_device\_open++;  return SUCCESS;  }  static int device\_release( struct inode \*inode, struct file \*file )  {  is\_device\_open--;  return SUCCESS;  }  static ssize\_t  device\_write( struct file \*filp, const char \*buff, size\_t len, loff\_t \* off )  {  printk( KERN\_ERR "Sorry, this operation isn't supported.\n" );  return -EINVAL;  }  static ssize\_t device\_read( struct file \*filp, /\* include/linux/fs.h \*/  char \*buffer, /\* buffer \*/  size\_t length, /\* buffer length \*/  loff\_t \* offset )  {  int byte\_read = 0;  if ( \*text\_ptr == 0 )  return 0;  while ( length && \*text\_ptr )  {  put\_user( \*( text\_ptr++ ), buffer++ );  length--;  byte\_read++;  }  return byte\_read;  } |

Рассмотрим код подробнее.

Первое, что необходимо сделать ­– указать необходимые заголовочные файлы:

|  |
| --- |
| #include <linux/kernel.h>  #include <linux/module.h>  #include <linux/init.h>  #include <linux/fs.h>  #include <asm/uaccess.h> |

Первая строчка подключает linux/kernel.h. Этот заголовочный файл позволяет использовать функцию printk(). Данная функция ядра работает примерно так же, как и функция printf(), только имеет форматирование специально для ядра. Основное назначение этой функции — реализация механизма регистрации событий и предупреждений. Иными словами, эта функция для записи в лог ядра некой информации.

Заголовочные файлы linux/module.h и linux/init.h требуются для всех модулей. Они включают в себя различные определения макросов, таких как module\_init(), module\_exit() и MODULE\_\*().

Заголовочный файл linux/fs.h содержит в себе указатели на функции. Каждое поле структуры соответствует адресу некоторой функции, определенной драйвером для обработки запрошенной операции.

Файл asm/uaccess.h обеспечивает доступ к функции put\_user(). Данная функция необходима, т.к. драйвер работает в пространстве ядра, то он ограничен от адресного пространства пользователя. И для того, чтобы вернуть какой-нибудь результат используется функция put\_user().

Следующее, что хотелось бы рассмотреть – информацию о модуле:

|  |
| --- |
| MODULE\_LICENSE( "GPL" );  MODULE\_AUTHOR( "Zorin Arseny <zorinarseny@yandex.ru>" );  MODULE\_DESCRIPTION( "My test module" );  MODULE\_SUPPORTED\_DEVICE( "test" ); |

Здесь можно указать лицензию, под которой будет распространяться модуль ядра, имя автора, описание модуля и поддерживаемое устройство.

Далее указываем операции, поддерживаемые устройством и их обработчики:

|  |
| --- |
| static int device\_open( struct inode \*, struct file \* );  static int device\_release( struct inode \*, struct file \* );  static ssize\_t device\_read( struct file \*, char \*, size\_t, loff\_t \* );  static ssize\_t device\_write( struct file \*, const char \*, size\_t, loff\_t \* );  static struct file\_operations fops =  {  .read = device\_read,  .write = device\_write,  .open = device\_open,  .release = device\_release  }; |

Здесь указываются возможные операции чтения, записи, открытия и закрытия.

Далее создаются точки входа и выхода из модуля. Определяются макросы \_\_init, \_\_exit.

|  |
| --- |
| // Функция загрузки модуля. Входная точка.  static int \_\_init test\_init( void )  {  printk( KERN\_INFO "TEST driver loaded!\n" );  // Регистрирайия устройсва и получение старшего номера устройства  major\_number = register\_chrdev( 0, DEVICE\_NAME, &fops );  if ( major\_number < 0 )  {  printk( "Character device registration was failed with %d\n", major\_number );  return major\_number;  }  // Сообщаем присвоенный старший номер устройства  printk( KERN\_INFO "Test module was loaded!\n" );  printk( KERN\_WARNING "Please, create a dev file with 'mknod /dev/test c %d 0'.\n", major\_number );  return SUCCESS;  }  // Функция выгрузки модуля  static void \_\_exit test\_exit( void )  {  // Освобождение устройства  unregister\_chrdev( major\_number, DEVICE\_NAME );  printk( KERN\_INFO "Test module was unloaded!\n" );  }  // Указание функций загрузки и выгрузки  module\_init( test\_init );  module\_exit( test\_exit ); |

В функции загрузки производится регистрация устройства и присвоение ему пары номеров (старшего и младшего). Сообщение пользователю присвоенных старшего и младшего номеров.

В функции выгрузки происходит только освобождение устройства.

Далее производим реализацию методов, открытия, закрытия, чтения и записи в файл.

|  |
| --- |
| static int device\_open( struct inode \*inode, struct file \*file )  {  text\_ptr = text;  if ( is\_device\_open )  return -EBUSY;  is\_device\_open++;  return SUCCESS;  }  static int device\_release( struct inode \*inode, struct file \*file )  {  is\_device\_open--;  return SUCCESS;  } |

В методе открытия устройства задаем отображаемый текст, и меняем флаг открытого устройства. В методе закрытия – устанавливается флаг открытого устройства в 0.

Далее – реализация методов чтения и записи:

|  |
| --- |
| device\_write( struct file \*filp, const char \*buff, size\_t len, loff\_t \* off )  {  printk( KERN\_ERR "Sorry, this operation isn't supported.\n" );  return -EINVAL;  }  static ssize\_t device\_read( struct file \*filp, char \*buffer, size\_t length, loff\_t \* offset )  {  int byte\_read = 0;  if ( \*text\_ptr == 0 )  return 0;  while ( length && \*text\_ptr )  {  put\_user( \*( text\_ptr++ ), buffer++ );  length--;  byte\_read++;  }  printk( KERN\_INFO "Read from device\n" );  return byte\_read;  } |

В методе записи в файл будет только уведомление о неподдерживаемой операции. А в методе чтения – считывание всей информации и запись в dmesg сообщения о том, что данный метод был вызван.

Теперь, после написания кода модуля ядра, его необходимо собрать. Для этого используется Makefile следующего содержания:

|  |
| --- |
| obj-m += test.o  all:  make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules  clean:  make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean |

Теперь запускаем его командой make:

|  |
| --- |
| arseny@ubuntuVM:~/Downloads/testModule$ make  make -C /lib/modules/4.8.0-51-generic/build M=/home/arseny/Downloads/testModule modules  make[1]: Entering directory '/usr/src/linux-headers-4.8.0-51-generic'  CC [M] /home/arseny/Downloads/testModule/test.o  Building modules, stage 2.  MODPOST 1 modules  CC /home/arseny/Downloads/testModule/test.mod.o  LD [M] /home/arseny/Downloads/testModule/test.ko  make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-51-generic' |

Для работы с модулем понадобятся следующие команды:

* insmod – добавление модуля в ядро
* rmmod – удаление модуля из ядра
* lsmod – вывод списка текущих модулей
* modinfo – вывод информации о модуле

Для начала, посмотрим информацию о полученном модуле:

|  |
| --- |
| arseny@ubuntuVM:~/Downloads/testModule$ modinfo test.ko  filename: /home/arseny/Downloads/testModule/test.ko  description: My test module  author: Zorin Arseny <zorinarseny@yandex.ru>  license: GPL  srcversion: D2503754193EC6FCAFF4DF0  depends:  vermagic: 4.8.0-51-generic SMP mod\_unload modversions |

Из полученной информации можно увидеть, что указана именно та информация, которая была указана в коде. Далее произведем вставку модуля в ядро и проверим его работу:

|  |
| --- |
| arseny@ubuntuVM:~/Downloads/testModule$ sudo insmod test.ko  arseny@ubuntuVM:~/Downloads/testModule$ dmesg | tail  [ 30.920236] IPv6: ADDRCONF(NETDEV\_UP): ens33: link is not ready  [ 30.922023] pcnet32 0000:02:01.0 ens33: link up  [ 35.516034] NET: Registered protocol family 40  [ 488.290192] test: loading out-of-tree module taints kernel.  [ 488.290569] test: module verification failed: signature and/or required key missing - tainting kernel  [ 488.294948] TEST driver loaded!  [ 488.294957] Test module was loaded!  [ 488.294960] Please, create a dev file with 'mknod /dev/test c 245 0'. |

Так как в системе отсутствует файл /dev/test, создадим его с помощью команды mknod и протестируем методы чтения и записи:

|  |
| --- |
| arseny@ubuntuVM:~/Downloads/testModule$ sudo mknod /dev/test c 245 0  arseny@ubuntuVM:~/Downloads/testModule$ cat /dev/test  This is test device  arseny@ubuntuVM:~/Downloads/testModule$ sudo su  root@ubuntuVM:/home/arseny/Downloads/testModule# echo 1 > /dev/test  bash: echo: write error: Invalid argument  root@ubuntuVM:/home/arseny/Downloads/testModule# exit  arseny@ubuntuVM:~/Downloads/testModule$ dmesg | tail  [ 30.920236] IPv6: ADDRCONF(NETDEV\_UP): ens33: link is not ready  [ 30.922023] pcnet32 0000:02:01.0 ens33: link up  [ 35.516034] NET: Registered protocol family 40  [ 488.290192] test: loading out-of-tree module taints kernel.  [ 488.290569] test: module verification failed: signature and/or required key missing - tainting kernel  [ 488.294948] TEST driver loaded!  [ 488.294957] Test module was loaded!  [ 488.294960] Please, create a dev file with 'mknod /dev/test c 245 0'.  [ 725.128774] Read from device  [ 737.951468] Sorry, this operation isn't supported. |

После того, как была произведена проверка работоспособности написанного модуля, его можно удалить из ядра:

|  |
| --- |
| arseny@ubuntuVM:~/Downloads/testModule$ lsmod | grep test  test 16384 0  arseny@ubuntuVM:~/Downloads/testModule$ sudo rmmod test  arseny@ubuntuVM:~/Downloads/testModule$ lsmod | grep test  arseny@ubuntuVM:~/Downloads/testModule$ dmesg | tail  [ 30.922023] pcnet32 0000:02:01.0 ens33: link up  [ 35.516034] NET: Registered protocol family 40  [ 488.290192] test: loading out-of-tree module taints kernel.  [ 488.290569] test: module verification failed: signature and/or required key missing - tainting kernel  [ 488.294948] TEST driver loaded!  [ 488.294957] Test module was loaded!  [ 488.294960] Please, create a dev file with 'mknod /dev/test c 245 0'.  [ 725.128774] Read from device  [ 737.951468] Sorry, this operation isn't supported.  [ 1178.005155] Test module was unloaded! |

**Вывод:**

В ходе данной лабораторной работы были изучены методы создания и запуска модулей ядра (драйверов символьного устройства).

Помимо изучения теоретической части, была произведена практическая реализация. То есть, был создан простой модуль ядра, взаимодействующий с файлом устройства /dev/test.

**Список литературы:**

1. Обфускация и защита программных продуктов http://citforum.ru/security/articles/obfus/
2. AppFuscator http://appfuscator.com
3. CXX-OBFUS http://stunnix.com/prod/cxxo/