«Разработка драйвера символьного устройства под ОС Linux»

**Цель работы:** Разработать драйвер для символьного устройства в ОС Linux, продемонстрировать его работоспособность. В качестве символьного устройства создадим свое устройство. Задача драйвера будет считывать записанные в устройство символы и записывать их в файл лога.

* Используемые системы

Linux

|  |
| --- |
| System**: Linux Mint 18** Sarah x86\_64  Kernel: 4.4.0-34-generic DE: X-Cinnamon Session: cinnamon  ----------------------------------  Processor: Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz  Memory (Gb): 3.7  Video: 00:02.0 VGA compatible controller: Intel Corporation 3rd Gen Core processor Graphics Controller (rev 09) Subsystem: Acer Incorporated [ALI] 3rd Gen Core processor Graphics Controller Kernel driver in use: i915 -- 01:00.0 VGA compatible controller: NVIDIA Corporation GF108M [GeForce GT 630M] (rev a1) Subsystem: Acer Incorporated [ALI] GF108M [GeForce GT 630M] Kernel driver in use: nvidia  ---------------------------------- |

Windows

|  |
| --- |
| C:\Users\Saboteur>systeminfo  Имя узла: SABOTEUR-PC  Название ОС: **Microsoft Windows 7** Максимальная  Версия ОС: 6.1.7601 Service Pack 1 сборка 7601  Изготовитель ОС: Microsoft Corporation  Параметры ОС: Изолированная рабочая станция  Сборка ОС: Multiprocessor Free  Зарегистрированный владелец: Saboteur  Зарегистрированная организация:  Код продукта: 00426-OEM-8992662-00006  Дата установки: 20.09.2015, 2:18:48  Время загрузки системы: 19.12.2015, 11:38:47  Изготовитель системы: Acer  Модель системы: Aspire V3-571G  Тип системы: x64-based PC  Процессор(ы): Число процессоров - 1.  [01]: Intel64 Family 6 Model 58 Stepping 9 Gen  uineIntel ~2501 МГц  Версия BIOS: Acer V1.13, 09.10.2012  Папка Windows: C:\Windows  Системная папка: C:\Windows\system32  Устройство загрузки: \Device\HarddiskVolume1  Язык системы: ru;Русский  Язык ввода: ru;Русский  Часовой пояс: (UTC+04:00) Волгоград, Москва, Санкт-Петербург  Полный объем физической памяти: 3 934 МБ  Доступная физическая память: 1 027 МБ  Виртуальная память: Макс. размер: 7 867 МБ  Виртуальная память: Доступна: 4 672 МБ  Виртуальная память: Используется: 3 195 МБ  Расположение файла подкачки: C:\pagefile.sys |

Разработка драйвера символьного устройства

Существует два пути для общения модуля с процессами. Первый идет через файлы устройства (подобно файлам в каталоге /dev), другой должен использовать файловую систему proc. Первоначальная цель файлов устройства состоит в том, чтобы позволить процессам связываться с драйверами устройства в ядре, и через них с физическими устройствами (модемы, терминалы, и т.д.).

Каждый драйвер устройства, который является ответственным за некоторый тип аппаратных средств, имеет собственный главный номер. Список драйверов и их главных номеров доступен в /proc/devices. Каждое физическое устройство, управляемое драйвером устройства имеет собственный номер. Каталог /dev включает специальный файл, названный файлом устройства, для каждого из тех устройств, которые реально установлены в системе.

Устройства разделены на два типа: символьные и блочные. Различие в том, что блочные имеют буфер для запросов, так что они могут выбирать в каком порядке отвечать. Это важно в случае устройств памяти, где скорее понадобится читать или писать сектора, которые ближе друг к другу, чем те, которые находятся далеко. Другое различие: блочные устройства могут принимать ввод и возвращать вывод только в блоках (чей размер может измениться согласно устройству), в то время как символьные устройства могут использовать столько байтов, сколько нужно. Большинство устройств в мире символьные, потому что они не нуждаются в этом типе буферизации и не работают с фиксированным размером блока. Для того чтобы узнать, является ли устройство блочным или символьным, можно обратить внимание на первый символ в выводе ls -l. Если это "b", значит устройство блочное, а если "c", то символьное.

Драйвер символьного устройства является модулем ядра, следовательно, в нем обязательно должны быть определены функции загрузки модуля в ядро (module\_init()) и выгрузки модуля из ядра (module\_exit()). Для того, что бы ядро знала операции, которые драйвер будет выполнять, они прописываются в структуре file\_operations, определенной в заголовочном файле linux/fs.h:

|  |
| --- |
| struct file\_operations {  struct module \*owner;  loff\_t (\*llseek) (struct file \*, loff\_t, int);  ssize\_t (\*read) (struct file \*, char \_\_user \*, size\_t, loff\_t \*);  ssize\_t (\*write) (struct file \*, const char \_\_user \*, size\_t, loff\_t \*);  ssize\_t (\*aio\_read) (struct kiocb \*, const struct iovec \*, unsigned long, loff\_t);  ssize\_t (\*aio\_write) (struct kiocb \*, const struct iovec \*, unsigned long, loff\_t);  ssize\_t (\*read\_iter) (struct kiocb \*, struct iov\_iter \*);  ssize\_t (\*write\_iter) (struct kiocb \*, struct iov\_iter \*);  int (\*iterate) (struct file \*, struct dir\_context \*);  unsigned int (\*poll) (struct file \*, struct poll\_table\_struct \*);  long (\*unlocked\_ioctl) (struct file \*, unsigned int, unsigned long);  long (\*compat\_ioctl) (struct file \*, unsigned int, unsigned long);  int (\*mmap) (struct file \*, struct vm\_area\_struct \*);  int (\*open) (struct inode \*, struct file \*);  int (\*flush) (struct file \*, fl\_owner\_t id);  int (\*release) (struct inode \*, struct file \*);  int (\*fsync) (struct file \*, loff\_t, loff\_t, int datasync);  int (\*aio\_fsync) (struct kiocb \*, int datasync);  int (\*fasync) (int, struct file \*, int);  int (\*lock) (struct file \*, int, struct file\_lock \*);  ssize\_t (\*sendpage) (struct file \*, struct page \*, int, size\_t, loff\_t \*, int);  unsigned long (\*get\_unmapped\_area)(struct file \*, unsigned long, unsigned long, unsigned long, unsigned long);  int (\*check\_flags)(int);  int (\*flock) (struct file \*, int, struct file\_lock \*);  ssize\_t (\*splice\_write)(struct pipe\_inode\_info \*, struct file \*, loff\_t \*, size\_t, unsigned int);  ssize\_t (\*splice\_read)(struct file \*, loff\_t \*, struct pipe\_inode\_info \*, size\_t, unsigned int);  int (\*setlease)(struct file \*, long, struct file\_lock \*\*);  long (\*fallocate)(struct file \*file, int mode, loff\_t offset,  loff\_t len);  int (\*show\_fdinfo)(struct seq\_file \*m, struct file \*f);  }; |

Данная структура содержит указатели на функции драйвера, которые отвечают за выполнение различных операций с устройством. Драйвер зачастую реализует далеко не все функции, предусмотренные данной структурой. Поля структуры, соответствующие нереализованным функциям, заполняются ”пустыми” указателями - NULL. Для простейшего драйвера реализуем 2 функции:

• open - открытие устройства;

• release - закрытие устройства.

Далее представлен код драйвера.

|  |
| --- |
| #include <linux/kernel.h>  #include <linux/module.h>  #include <linux/fs.h>  #include <asm/uaccess.h>  #include <linux/init.h>  #include <linux/slab.h>  #include <linux/errno.h>  #include <linux/types.h>  #define SUCCESS 0  #define DEVICE\_NAME "chardev" /\* Имя устройства, будет отображаться в /proc/devices \*/  #define BUF\_LEN 180 /\* Максимальная длина сообщения \*/  int device\_init(void);  void device\_exit(void);  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 \*);  module\_init(device\_init);  module\_exit(device\_exit);  static int Major; /\* Старший номер устройства - драйвера \*/  static int Device\_Open = 0; /\* Счетчик открытия устройства  \* Используется для предотвращения обращения  \* из нескольких процессов \*/  static char Message[BUF\_LEN]; /\* Текст сообщения \*/  static char \*Message\_Ptr;  static struct file\_operations fops = {  .read = device\_read,  .write = device\_write,  .open = device\_open,  .release = device\_release  };  // Ниже мы задаём информацию о модуле, которую можно будет увидеть с помощью Modinfo  MODULE\_LICENSE("GPL");  MODULE\_AUTHOR("lab3 Vasilev");  /\* Загрузка модуля в ядро \*/  int device\_init(void)  {  /\* Регистрация устройства \*/  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);  printk(KERN\_INFO "Try various minor numbers. Try to cat and echo to\n");  printk(KERN\_INFO "the device file.\n");  printk(KERN\_INFO "Remove the device file and module when done.\n");  return SUCCESS;  }  /\* Выгрузка модуля из ядра \*/  void device\_exit(void)  {  /\* Освобождение старшего номера устройства \*/  unregister\_chrdev(Major, DEVICE\_NAME);  printk(KERN\_ALERT "Removing character device /dev/%s c %d 0'.\n", DEVICE\_NAME, Major);  return;  }  /\* Открытие файла устройства процессом "cat /dev/chardev" \*/  static int device\_open(struct inode \*inode, struct file \*file)  {  printk(KERN\_INFO "Try to open character device /dev/%s c %d 0'.\n", DEVICE\_NAME, Major);  if (Device\_Open)  return -EBUSY;  Device\_Open++; // счетчик открытия устройства  try\_module\_get(THIS\_MODULE);  return SUCCESS;  }  /\* Закрытие файла устройства процессом \*/  static int device\_release(struct inode \*inode, struct file \*file)  {  printk(KERN\_ALERT "Try to close character device /dev/%s c %d 0'.\n", DEVICE\_NAME, Major);  Device\_Open--; /\* Возможно обслуживание другого процесса \*/  /\*  \* Уменьшить счетчик обращений, иначе после успешного  \* открытия не сможем больше выгрузить модуль.  \*/  module\_put(THIS\_MODULE);  return SUCCESS;  }  /\* Открытие файла устройства процессом для чтения \*/  static ssize\_t device\_read(struct file \*filp, /\*указатель на структуру file\*/  char \*buffer, /\* буфер, куда надо положить данные \*/  size\_t length, /\* длина буфера \*/  loff\_t \* offset)  {  int bytes\_read = 0; /\* Количество байт, записанных в буфер \*/  printk("Try to read character device /dev/%s c %d 0'.\n", DEVICE\_NAME, Major);  if (\*Message\_Ptr == 0) /\* Если достигли конца сообщения, \*/  return 0; /\*вернуть ноль, как признак конца файла\*/  /\* Помещение данных в буфер\*/  while (length && \*Message\_Ptr) {  put\_user(\*(Message\_Ptr++), buffer++);  length--;  bytes\_read++;  }  printk("Read %d bytes, %d left.\n", bytes\_read, length);  /\* Возвращаем число байт, записанных в буфер \*/  return bytes\_read;  }  /\* Открытие файла устройства процессом для записи \*/  static ssize\_t device\_write(struct file \*filp, const char \*buffer, size\_t length, loff\_t \* offset)  {  static int counter = 0;  int i;  printk("Try to write character device /dev/%s (%d).\n", DEVICE\_NAME, length);  memset(Message, 0, BUF\_LEN);  for (i = 0; i < length && i < (BUF\_LEN - 40); i++)  get\_user(Message[i], buffer + i);  counter++;  sprintf(Message + strlen(Message), "\nI already told you %d times HELLO!\n", counter);  Message\_Ptr = Message;  return i;  } |

Для компиляции драйвера используется следующий Makefile:

|  |
| --- |
| obj-m += chardev.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 |

Тестирование драйвера символьного устройства

|  |
| --- |
| **Компиляция драйера**  **user@ubuntu:~/Documents/driver$ make**  make -C /lib/modules/4.8.0-36-generic/build M=/home/user/Documents/driver modules  make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'  CC [M] /home/user/Documents/driver/chardev.o  /home/user/Documents/driver/chardev.c: In function ‘device\_read’:  /home/user/Documents/driver/chardev.c:124:9: warning: format ‘%d’ expects argument of type ‘int’, but argument 3 has type ‘size\_t {aka long unsigned int}’ [-Wformat=]  printk("Read %d bytes, %d left.\n", bytes\_read, length);  ^  /home/user/Documents/driver/chardev.c: In function ‘device\_write’:  /home/user/Documents/driver/chardev.c:134:9: warning: format ‘%d’ expects argument of type ‘int’, but argument 3 has type ‘size\_t {aka long unsigned int}’ [-Wformat=]  printk("Try to write character device /dev/%s (%d).\n", DEVICE\_NAME, length);  ^  Building modules, stage 2.  MODPOST 1 modules  CC /home/user/Documents/driver/chardev.mod.o  LD [M] /home/user/Documents/driver/chardev.ko  make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'  **Загрузка драйвера в ядро**  **user@ubuntu:~/Documents/driver$ sudo insmod chardev.ko**  [sudo] password for user:  **user@ubuntu:~/Documents/driver$ tail /var/log/syslog**  May 13 12:50:52 ubuntu systemd-tmpfiles[3278]: [/usr/lib/tmpfiles.d/var.conf:14] Duplicate line for path "/var/log", ignoring.  May 13 12:50:52 ubuntu systemd[1]: Started Cleanup of Temporary Directories.  May 13 12:51:59 ubuntu kernel: [ 977.201764] chardev: loading out-of-tree module taints kernel.  May 13 12:51:59 ubuntu kernel: [ 977.201938] chardev: module verification failed: signature and/or required key missing - tainting kernel  May 13 12:51:59 ubuntu kernel: [ 977.203269] I was assigned major number 245. To talk to  May 13 12:51:59 ubuntu kernel: [ 977.203270] the driver, create a dev file with  May 13 12:51:59 ubuntu kernel: [ 977.203270] 'mknod /dev/chardev c 245 0'.  May 13 12:51:59 ubuntu kernel: [ 977.203270] Try various minor numbers. Try to cat and echo to  May 13 12:51:59 ubuntu kernel: [ 977.203271] the device file.  May 13 12:51:59 ubuntu kernel: [ 977.203271] Remove the device file and module when done.  **Создание файла символьного устройства**  **user@ubuntu:~/Documents/driver$ sudo mknod /dev/chardev c 245 0**  **user@ubuntu:~/Documents/driver$ lsmod | grep chardev**  chardev 16384 0  **user@ubuntu:~/Documents/driver$ ls -l /dev/chardev**  crw-r--r-- 1 root root 245, 0 May 13 12:54 /dev/chardev  **Изменение прав доступа к файлу для поддержки возможности записи в файл**  **user@ubuntu:~/Documents/driver$ sudo chmod 666 /dev/chardev**  **user@ubuntu:~/Documents/driver$ ls -l /dev/chardev**  crw-rw-rw- 1 root root 245, 0 May 13 12:54 /dev/chardev  **user@ubuntu:~/Documents/driver$ cat /proc/devices | grep chardev**  245 chardev  **Запись в файл устройства**  **user@ubuntu:~/Documents/driver$ echo "device Hello world" > /dev/chardev**  **user@ubuntu:~/Documents/driver$ cat /var/log/syslog**  **…**  May 13 12:49:47 ubuntu gnome-session[2025]: \*\* (soffice:2835): WARNING \*\*: Unknown event notification 36  May 13 12:50:45 ubuntu gnome-session[2025]: message repeated 32 times: [ \*\* (soffice:2835): WARNING \*\*: Unknown event notification 36]  May 13 12:50:52 ubuntu systemd[1]: Starting Cleanup of Temporary Directories...  May 13 12:50:52 ubuntu systemd-tmpfiles[3278]: [/usr/lib/tmpfiles.d/var.conf:14] Duplicate line for path "/var/log", ignoring.  May 13 12:50:52 ubuntu systemd[1]: Started Cleanup of Temporary Directories.  May 13 12:51:59 ubuntu kernel: [ 977.201764] chardev: loading out-of-tree module taints kernel.  May 13 12:51:59 ubuntu kernel: [ 977.201938] chardev: module verification failed: signature and/or required key missing - tainting kernel  May 13 12:51:59 ubuntu kernel: [ 977.203269] I was assigned major number 245. To talk to  May 13 12:51:59 ubuntu kernel: [ 977.203270] the driver, create a dev file with  May 13 12:51:59 ubuntu kernel: [ 977.203270] 'mknod /dev/chardev c 245 0'.  May 13 12:51:59 ubuntu kernel: [ 977.203270] Try various minor numbers. Try to cat and echo to  May 13 12:51:59 ubuntu kernel: [ 977.203271] the device file.  May 13 12:51:59 ubuntu kernel: [ 977.203271] Remove the device file and module when done.  May 13 12:56:34 ubuntu kernel: [ 1252.091928] Try to open character device /dev/chardev c 245 0'.  May 13 12:56:34 ubuntu kernel: [ 1252.091939] Try to write character device /dev/chardev (19).  May 13 12:56:34 ubuntu kernel: [ 1252.091943] Try to close character device /dev/chardev c 245 0'.  **Чтение файла устройства**  **user@ubuntu:~/Documents/driver$ cat /dev/chardev**  device Hello world  I already told you 1 times HELLO!  **user@ubuntu:~/Documents/driver$ cat /var/log/syslog**  …  May 13 12:57:16 ubuntu kernel: [ 1293.665511] Try to read character device /dev/chardev c 245 0'.  May 13 12:57:16 ubuntu kernel: [ 1293.665513] Read 54 bytes, 131018 left.  May 13 12:57:16 ubuntu kernel: [ 1293.665720] Try to read character device /dev/chardev c 245 0'.  May 13 12:57:16 ubuntu kernel: [ 1293.665732] Try to close character device /dev/chardev c 245 0'.  May 13 12:57:50 ubuntu kernel: [ 1327.716227] Try to open character device /dev/chardev c 245 0'.  May 13 12:57:50 ubuntu kernel: [ 1327.716237] Try to read character device /dev/chardev c 245 0'.  May 13 12:57:50 ubuntu kernel: [ 1327.716245] Try to close character device /dev/chardev c 245 0'.  **Выгрузка модуля из ядра**  **user@ubuntu:~/Documents/driver$ sudo rmmod chardev.ko**  **Удаление файла устройства**  **user@ubuntu:~/Documents/driver$ sudo rm -i /dev/chardev**  rm: remove character special file '/dev/chardev'? y  **user@ubuntu:~/Documents/driver$ lsmod | grep chardev**  **Очистка каталога от исполняемых файлов**  **user@ubuntu:~/Documents/driver$ make clean**  make -C /lib/modules/4.8.0-36-generic/build M=/home/user/Documents/driver clean  make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'  CLEAN /home/user/Documents/driver/.tmp\_versions  CLEAN /home/user/Documents/driver/Module.symvers  make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'  **user@ubuntu:~/Documents/driver$ ls -l**  total 12  -rw-rw-r-- 1 user user 5164 May 13 12:49 chardev.c  -rw-rw-r-- 1 user user 157 May 13 12:49 Makefile |

В результате проведенной работы драйвер символьного устройства был успешно скомпилирован и загружен в ядро. После этого была проверена его работоспособность. Далее драйвер был успешно выгружен.

Разработка драйвера клавиатуры

Создадим драйвер, который будет перехватывать прерывание нажатия клавиши и подсчитывать количество нажатий.

Код программы

|  |
| --- |
| #include <linux/module.h>  #include <linux/init.h>  #include <linux/interrupt.h>  #include <linux/stat.h>  #define SHARED\_IRQ 17  static int irq = SHARED\_IRQ, my\_dev\_id, irq\_counter = 0;  module\_param( irq, int, S\_IRUGO );  //Действие при прерывании  static irqreturn\_t my\_interrupt( int irq, void \*dev\_id ) {  irq\_counter++;  printk( KERN\_INFO "In the ISR: counter = %d\n", irq\_counter );  return IRQ\_NONE;  }  //Функция инициализации модуля  static int \_\_init my\_init( void ) {  if ( request\_irq( irq, my\_interrupt, IRQF\_SHARED, "my\_interrupt", &my\_dev\_id ) )  return -1;  printk( KERN\_INFO "Successfully loading ISR handler on IRQ %d\n", irq );  return 0;  }  //Функция удаления модуля  static void \_\_exit my\_exit( void ) {  synchronize\_irq( irq );  free\_irq( irq, &my\_dev\_id );  printk( KERN\_INFO "Successfully unloading, irq\_counter = %d\n", irq\_counter );  }  module\_init( my\_init );  module\_exit( my\_exit ); |

Инициализация модуля и вывод буфера сообщений ядра в стандартный поток вывода

|  |
| --- |
| root@debian:~/Documents/SPO# sudo /sbin/insmod irq.ko irq=1  root@debian:~/Documents/SPO# dmesg | tail -n50  [ 31.470919] cfg80211: (5170000 KHz - 5250000 KHz @ 80000 KHz, 160000 KHz AUTO), (N/A, 2000 mBm), (N/A)  [ 31.470920] cfg80211: (5250000 KHz - 5330000 KHz @ 80000 KHz, 160000 KHz  ...  [ 31.771290] IPv6: ADDRCONF(NETDEV\_UP): eth0: link is not ready  [ 31.771299] IPv6: ADDRCONF(NETDEV\_CHANGE): eth0: link becomes ready  **[ 2693.921167] Successfully loading ISR handler on IRQ 1**  [ 2694.034352] In the ISR: counter = 1  [ 2707.823574] In the ISR: counter = 2  [ 2707.823626] In the ISR: counter = 3  ...  [ 2714.815690] In the ISR: counter = 41 |

Проверяем наличие обработчика прерывания

|  |
| --- |
| root@debian:~/Documents/SPO# cat /proc/interrupts  CPU0  0: 31 XT-PIC-XT-PIC timer  1: 159 XT-PIC-XT-PIC i8042, **my\_interrupt**  2: 0 XT-PIC-XT-PIC cascade  ...  HYP: 0 Hypervisor callback interrupts  ERR: 0  MIS: 0 |

Удаляем модуль и проверяем буфер ядра.

|  |
| --- |
| root@debian:~/Documents/SPO# sudo /sbin/rmmod irq  root@debian:~/Documents/SPO# dmesg | tail -n5  [ 2765.900879] In the ISR: counter = 66  [ 2765.919637] In the ISR: counter = 67  [ 2765.933030] In the ISR: counter = 68  [ 2766.437647] In the ISR: counter = 69  [ **2766.443093] Successfully unloading, irq\_counter = 69** |

Вывод

В данной работе изучены основные принципы разработки драйвера ОС Linux. Создан простой драйвер для символьного устройства. Драйвер в ОС Linux является модулем ядра, при его создании необходимо указать, какие функции он будет выполнять, а также, задать ему идентификатор. Минимальное требование к драйверу – описать функции загрузки и выгрузки. Созданный в лабораторной драйвер также осуществлял операции чтения и записи.