Categories

Cybersecurity

Virtualization

Blockchain

C++ tips

Driver development

System development for Windows

Development for Linux

Development for Mac

Mobile development

Web Development

Reverse engineering

QA and testing

White Papers

Top content

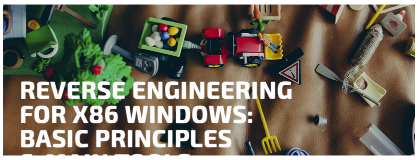
Linux Driver Tutorial: I Simple Linux Device D

This Linux device driver tutorial will provide information about how to write a device driv article includes a practical Linux driver devel follow. We'll discuss the following:

- Kernel logging system
- How to work with character devices
- How to work with user-level memory

We'll [illegible] Linux kernel version [illegible] cou

This article includes
description of simple



How to Reverse Engineer Software (Windows) in a Right Way

This article considers
common reverse engineering

Windows Process Monitoring and Management Tips

The following article will help
you to understand principles

Specifying a name of the device

The file_operations structure

The printk fucntion

Using memory allocated in user mode

Build system of a kernel module

Loading and using a module

References

1. Overview

Linux has a monolithic kernel. For this reasor requires performing a combined compilation is to implement your driver as a kernel modu recompile the kernel to add another driver. W option nel modules.

We run the module code in the kernel contex
very attentive, as it entails extra responsibili
when implementing a user-level application,
the user application in most cases; but if a d
implementing a kernel module, the conseque
level. Luckily for us, the Linux kernel has a n
errors in module code. When the kernel enco
example, null pointer dereferencing), you'll se
malfunctions during Linux operation are calle
malfunctioning module will be unloaded, allo
to work as usual. In addition, you'll be able t
precisely describes this error. But be aware t
message is not recommended, as doing so m
panic.

The kernel and its modules essentially repres
keep in mind that a single program module u
order to minimize it, you must watch what is
exported global characters must be named u
workaround is to simply use the name of the
characters as a prefix) and must be cut to th

```
static void my_exit(void)
{
                        return;
}

module_init(my_init);
module_exit(my_exit);
```

The only two things this module does is load driver, we call the *my_init* function, and to function. The *module_init* and *module_ex* driver loading and unloading. The *my_init* a identical signatures, which must be exactly a

```
int init(void);
void exit(void);
```

If the module requires a certain kernel versio the version, we need to link the **linux/modul** module built for another kernel version will l prohibiting its loading. There's a reason for s API are released quite often, and when you c signature has been changed, you cause dama *module_init* and *module_exit* macros are development process richer.

For a start, here's some useful information a
find device files in the /dev folder. They facil
and the kernel code. If the kernel must recei
device file to pass it to the module serving t
device file originates from the module servin
into two groups: character files and block fil
whereas block files are buffered. As their na
to read and write data character by characte
write only whole blocks of data. We'll leave
the scope of this article, and will get straigh

Linux systems have a way of identifying devi
which identify modules serving device files o
device numbers, which identify a specific de
major device number specifies. In the driver
as constants or they can be allocated dynam
constant has already been used, the system
is allocated dynamically, the function reserve
being used by anything else.

which the device and the *file_operations*
zero to the major parameter, the function wi
the value it returns) on its own. If the value
success, while a negative number signifies ar
specified in the 0–255 range.

We pass the device name as a string value o
can also pass the name of a module if it reg
this string to identify a device in the /sys/de
as read, write, and save are processed by the
file_operations structure. These function
and the pointers to the *module* structure ide
within the *file_operations* structure. Here
version structure:

```
struct file_operations {
       struct module *owner;
       loff_t (*llseek) (struct file *, loff_t
       ssize_t (*read) (struct file *, char *,
       ssize_t (*write) (struct file *, const
       int (*readdir) (struct file *, void *,
       unsigned int (*poll) (struct file *, st
       int (*ioctl) (struct inode *, struct fi
       int (*mmap) (struct file *, struct vm_a
       int (*open) (struct inode *, struct fil
       int (*flush) (struct file *);
       int (*release) (struct inode *, struct
       int (*fsync) (struct file *, struct den
       int (*fasync) (int, struct file *, int)
```

unimplemented function can simply be set tc take care of the implementation of the functi our case, we'll just implement the *read* func

As we're going to ensure the operation of on Linux driver, our *file_operations* structur Correspondingly, after it's created, we'll need how this is done:

```
static struct file_operations simple_driver_fc
{
    .owner   = THIS_MODULE,
    .read    = device_file_read,
};
```

The declaration of the *THIS_MODULE* macro i header file. We transform the macro into the the required module. A bit later, we'll get to a prototype, but right now we have only the *device_file_read*.

```
ssize_t device_file_read (struct file *,
```

The *f*... *operations* structure allo... us t

apriorit R&D Services ▾ Specialties ▾ Competences ▾ About Us ▾

Blog Let's talk

```
            , device_file_major_number );
        return 0;
}
```

The *device_file_major_*number is a globa device number. When the lifetime of the driv revoke the registration of the device file.

6. The printk Fucntion

We've already listed and mentioned almost a *printk* function. The declaration of this func linux/kernel.h file, and its task is simple: to l paid attention to the *KERN_NOTICE* and *KER* present in all listed format strings of printk. and *WARNING* signify the priority level of a m insignificant *KERN_DEBUG* to the critical *KER* instability. This is the only difference betwee *printf* library function.

The *printk* function forms a string, which w the *k* daemon reads it and th

register_chrdev and the *unregister_ch*
ways.

To register a device, we use the following co

```
void unregister_device(void)
{
    printk( KERN_NOTICE "Simple-driver: unregi
    if(device_file_major_number != 0)
    {
        unregister_chrdev(device_file_major_nu
    }
}
```

7. Using Memory Allocated

The function we're going to write will read cl signature of this function must be appropriat *file_operations* structure:

```
ssize_t (*read) (struct file *, char *, size_t
```

Let's have a look at the first parameter, the
file cture allows us to ge y in

address in the kernel address space may hav
cannot simply dereference the pointer. When
have a set of specific macros and functions t
file. The most suitable function in our case is
for itself: it simply copies specific data from
allocated in the user space. In addition, it als
the buffer size is large enough. Thus, errors
relatively easily. Here's the code for the *copy*

```
long copy_to_user( void __user *to, const void
```

First of all, this function must receive three p
the buffer, a pointer to the data source, and
copying. As we've mentioned, an error return
case of successful execution, the value will b
_user macro, whose task is to perform docu
useful application that allows us to analyze i
address space correctly; this is done using th
analysis of static code. Make sure to always
_user.

```
{
    printk( KERN_NOTICE "Simple-driver: Device
bytes count = %u"
                    , (int)*position
                    , (unsigned int)count );
    /* If position is behind the end of a file
    if( *position >= g_s_Hello_World_size )
        return 0;
    /* If a user tries to read more than we ha
ave */
    if( *position + count > g_s_Hello_World_si
        count = g_s_Hello_World_size - *positi
    if( copy_to_user(user_buffer, g_s_Hello_Wo
)
        return -EFAULT;
    /* Move reading position */
    *position += count;
    return count;
}
```

8. Build System of a Kernel

After we've written the code for the driver, it
as we expect. In the earlier kernel versions (
required many more movements from a deve
compilation needed to be prepared individual
the GCC compiler. Only after that would a de
that could be loaded to the kernel. Fortunate
the process is much simpler now. Today, muc
makef t starts the kernel bu , s n an

```
obj-m := module_name.o
module_name-objs := source_1.o source_2.o … sc
```

The *make* command initializes the kernel buil

To build the module:

```
make –C KERNEL_MODULE_BUILD_SYSTEM_FOLDER M=`p
```

To clean up the build folder:

```
make –C KERNEL_MODULES_BUILD_SYSTEM_FOLDER M=`
```

The module build system is commonly locate
Now it's time to prepare the module build sy
execute the following command from the fol
located:

```
#> make modules_prepare
```

Finally combine everything ned

```
endif
```

The *load* target loads the build module and the kernel.

In our tutorial, we've used code from main.c driver. The resulting driver is named simple-r

9. Loading and Using Modul

The following command executed from the s the built module:

```
#> make load
```

After executing this command, the name of t /proc/modules file, while the device that the /proc/devices file. The added records look lik

```
Character devices: 1 mem 4 tty 4 ttyS … 250 Si
```

The fi hree records contain of t

```
Hello world from kernel mode!
```

10. References

- 1. Linux Device Drivers, 3rd Edition by and Greg Kroah-Hartman: http://lwn.net
- 2. The Linux Kernel Module Programn Ori Pomeranz: http://tldp.org/LDP/lkmpg
- Linux Cross Reference http://lxr.free-

Download source code of Simple Linux Driv

We hope this tutorial comes in handy. You ca development.

You may also be ir

R&D Services ▾ Specialties ▾ Competences ▾ About Us ▾

Blog

Let's talk

Get more posts l

Let's talk

Send us a message

4000 chars left

Attach a file Browse

I'm not a robot
reCAPTCHA
Privacy - Terms

By clicking Send you give consent to processing your data

Send