



Weeze

Viele Möglichkeiten! ...

114 m²
Wohnfläche

85.000,00 €
Kaufpreis



Wiehl

Fachwerkhaus mit Garten und
Scheune !!! ...

90 m²
Wohnfläche

48.000,00 €
Kaufpreis



immonet.de
Wir sind Immobilien

Home » Articles » Xavier Calbet's articles

Writing device drivers in Linux: A brief tutorial

Short URL: <http://fsmsh.com/1238>

Like Share 497

+364 Auf Google empfehlen

Wed, 2006-04-26 11:03 -- Xavier Calbet

“Do you pine for the nice days of Minix-1.1, when men were men and wrote their own device drivers?” *Linus Torvalds*

Pre-requisites

In order to develop Linux device drivers, it is necessary to have an understanding of the following:

- **C programming.** Some in-depth knowledge of C programming is needed, like pointer usage, bit manipulating functions, etc.
- **Microprocessor programming.** It is necessary to know how microcomputers work internally: memory addressing, interrupts, etc. All of these concepts should be familiar to an assembler programmer.

There are several different devices in Linux. For simplicity, this brief tutorial will only cover type `char` devices loaded as modules. Kernel 2.6.x will be used (in particular, kernel 2.6.8 under Debian Sarge, which is now Debian Stable).

User space and kernel space

When you write device drivers, it's important to make the distinction between “user space” and “kernel space”.

- **Kernel space.** Linux (which is a kernel) manages the machine's hardware in a simple and efficient manner, offering the user a simple and uniform programming interface. In the same way, the kernel, and in particular its device drivers, form a bridge or interface between the end-user/programmer and the hardware. Any subroutines or functions forming part of the kernel (modules and device drivers, for example) are considered to be part of kernel space.
- **User space.** End-user programs, like the UNIX `shell` or other GUI based applications (`kpresenter` for example), are part of the user space. Obviously, these applications need to interact with the system's hardware . However, they don't do so directly, but through the kernel supported functions.

All of this is shown in figure 1.

Interfacing functions between user space and kernel space

The kernel offers several subroutines or functions in user space, which allow the end-user application programmer to interact with the hardware. Usually, in UNIX or Linux systems, this dialogue is performed through functions or subroutines in order to

read and write files. The reason for this is that in Unix devices are seen, from the point of view of the user, as files.

On the other hand, in kernel space Linux also offers several functions or subroutines to perform the low level interactions directly with the hardware, and allow the transfer of information from kernel to user space.

Usually, for each function in user space (allowing the use of devices or files), there exists an equivalent in kernel space (allowing the transfer of information from the kernel to the user and vice-versa). This is shown in Table 1, which is, at this point, empty. It will be filled when the different device drivers concepts are introduced.

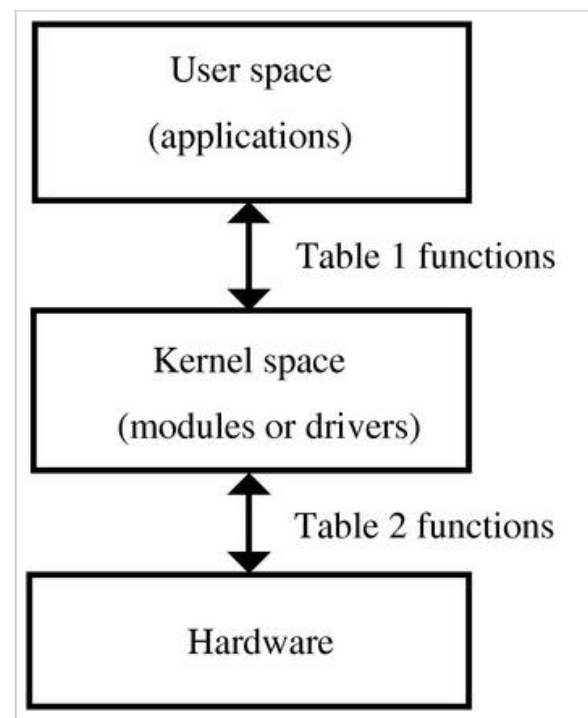


Figure 1: User space where applications reside, and kernel space where modules or device drivers reside

Events	User functions	Kernel functions
Load module		
Open device		
Read device		
Write device		
Close device		
Remove module		

Table 1. Device driver events and their associated interfacing functions in kernel space and user space.

Interfacing functions between kernel space and the hardware device

There are also functions in kernel space which control the device or exchange information between the kernel and the hardware. Table 2 illustrates these concepts. This table will also be filled as the concepts are introduced.

Events	Kernel functions
Read data	
Write data	

Table 2. Device driver events and their associated functions between kernel space and the hardware device.

The first driver: loading and removing the driver in user space

I'll now show you how to develop your first Linux device driver, which will be introduced in the kernel as a module.

For this purpose I'll write the following program in a file named `nothing.c`

<nothing.c> =

```
#include <linux/module.h>

MODULE_LICENSE("Dual BSD/GPL");
```

Since the release of kernel version 2.6.x, compiling modules has become slightly more complicated. First, you need to have a complete, compiled kernel source-code-tree. If you have a Debian Sarge system, you can follow the steps in Appendix B (towards the end of this article). In the following, I'll assume that a kernel version 2.6.8 is being used.

Next, you need to generate a makefile. The makefile for this example, which should be named `Makefile`, will be:

```
<Makefile> =

obj-m := nothing.o
```

Unlike with previous versions of the kernel, it's now also necessary to compile the module using the same kernel that you're going to load and use the module with. To compile it, you can type:

```
$ make -C /usr/src/kernel-source-2.6.8 M=pwd modules
```

This extremely simple module belongs to kernel space and will form part of it once it's loaded.

In user space, you can load the module as root by typing the following into the command line:

```
# insmod nothing.ko
```

The `insmod` command allows the installation of the module in the kernel. However, this particular module isn't of much use.

It is possible to check that the module has been installed correctly by looking at all installed modules:

```
# lsmod
```

Finally, the module can be removed from the kernel using the command:

```
# rmmod nothing
```

By issuing the `lsmod` command again, you can verify that the module is no longer in the kernel.

The summary of all this is shown in Table 3.

Events	User functions	Kernel functions
Load module	insmod	
Open device		
Read device		
Write device		
Close device		
Remove module	rmmod	

Table 3. Device driver events and their associated interfacing functions between kernel space and user space.

The “Hello world” driver: loading and removing the driver in kernel space

When a module device driver is loaded into the kernel, some preliminary tasks are usually performed like resetting the device, reserving RAM, reserving interrupts, and reserving input/output ports, etc.

These tasks are performed, in kernel space, by two functions which need to be present (and explicitly declared): `module_init` and `module_exit`; they correspond to the user space commands `insmod` and `rmmod`, which are used when installing or removing a module. To sum up, the user commands `insmod` and `rmmod` use the kernel space functions `module_init` and `module_exit`.

Let's see a practical example with the classic program `Hello world`:

<hello.c> =

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("Dual BSD/GPL");

static int hello_init(void) {
    printk("<1> Hello world!\n");
    return 0;
}

static void hello_exit(void) {
    printk("<1> Bye, cruel world\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

The actual functions `hello_init` and `hello_exit` can be given any name desired. However, in order for them to be identified as the corresponding loading and removing functions, they have to be passed as parameters to the functions `module_init` and `module_exit`.

The `printk` function has also been introduced. It is very similar to the well known `printf` apart from the fact that it only works inside the kernel. The `<1>` symbol shows the high priority of the message (low number). In this way, besides getting the message in the kernel system log files, you should also receive this message in the system console.

This module can be compiled using the same command as before, after adding its name into the Makefile.

<Makefile2> =

```
obj-m := nothing.o hello.o
```

In the rest of the article, I have left the Makefiles as an exercise for the reader. A complete Makefile that will compile all of the modules of this tutorial is shown in Appendix A.

When the module is loaded or removed, the messages that were written in the `printk` statement will be displayed in the system console. If these messages do not appear in the console, you can view them by issuing the `dmesg` command or by looking at the system log file with `cat /var/log/syslog`.

Table 4 shows these two new functions.

Events	User functions	Kernel functions
Load module	insmod	module_init()

Open device		
Read device		
Write device		
Close device		
Remove module	rmmod	module_exit()

Table 4. Device driver events and their associated interfacing functions between kernel space and user space.

The complete driver “memory”: initial part of the driver

I'll now show how to build a complete device driver: `memory.c`. This device will allow a character to be read from or written into it. This device, while normally not very useful, provides a very illustrative example since it is a complete driver; it's also easy to implement, since it doesn't interface to a real hardware device (besides the computer itself).

To develop this driver, several new `#include` statements which appear frequently in device drivers need to be added:

<memory initial> =

```
/* Necessary includes for device drivers */
#include <linux/init.h>
#include <linux/config.h>
#include <linux/module.h>
#include <linux/kernel.h> /* printk() */
#include <linux/slab.h> /* kmalloc() */
#include <linux/fs.h> /* everything... */
#include <linux/errno.h> /* error codes */
#include <linux/types.h> /* size_t */
#include <linux/proc_fs.h>
#include <linux/fcntl.h> /* O_ACCMODE */
#include <asm/system.h> /* cli(), *_flags */
#include <asm/uaccess.h> /* copy_from/to_user */

MODULE_LICENSE("Dual BSD/GPL");

/* Declaration of memory.c functions */
int memory_open(struct inode *inode, struct file *filp);
int memory_release(struct inode *inode, struct file *filp);
ssize_t memory_read(struct file *filp, char *buf, size_t count, loff_t *f_pos);
ssize_t memory_write(struct file *filp, char *buf, size_t count, loff_t *f_pos);
void memory_exit(void);
int memory_init(void);

/* Structure that declares the usual file */
/* access functions */
struct file_operations memory_fops = {
    read: memory_read,
    write: memory_write,
    open: memory_open,
    release: memory_release
};

/* Declaration of the init and exit functions */
module_init(memory_init);
module_exit(memory_exit);

/* Global variables of the driver */
```

```

/* Major number */
int memory_major = 60;
/* Buffer to store data */
char *memory_buffer;

```

After the `#include` files, the functions that will be defined later are declared. The common functions which are typically used to manipulate files are declared in the definition of the `file_operations` structure. These will also be explained in detail later. Next, the initialization and exit functions—used when loading and removing the module—are declared to the kernel. Finally, the global variables of the driver are declared: one of them is the `major number` of the driver, the other is a pointer to a region in memory, `memory_buffer`, which will be used as storage for the driver data.

The “memory” driver: connection of the device with its files

In UNIX and Linux, devices are accessed from user space in exactly the same way as files are accessed. These device files are normally subdirectories of the `/dev` directory.

To link normal files with a kernel module two numbers are used: `major number` and `minor number`. The `major number` is the one the kernel uses to link a file with its driver. The `minor number` is for internal use of the device and for simplicity it won't be covered in this article.

To achieve this, a file (which will be used to access the device driver) must be created, by typing the following command as root:

```
# mknod /dev/memory c 60 0
```

In the above, `c` means that a `char` device is to be created, `60` is the `major number` and `0` is the `minor number`.

Within the driver, in order to link it with its corresponding `/dev` file in kernel space, the `register_chrdev` function is used. It is called with three arguments: `major number`, a string of characters showing the module name, and a `file_operations` structure which links the call with the file functions it defines. It is invoked, when installing the module, in this way:

<memory init module> =

```

int memory_init(void) {
    int result;

    /* Registering device */
    result = register_chrdev(memory_major, "memory", &memory_fops);
    if (result < 0) {
        printk(
            "<1>memory: cannot obtain major number %d\n", memory_major);
        return result;
    }

    /* Allocating memory for the buffer */
    memory_buffer = kmalloc(1, GFP_KERNEL);
    if (!memory_buffer) {
        result = -ENOMEM;
        goto fail;
    }
    memset(memory_buffer, 0, 1);

    printk("<1>Inserting memory module\n");
    return 0;

fail:

```

```
memory_exit();
return result;
}
```

Also, note the use of the `kmalloc` function. This function is used for memory allocation of the buffer in the device driver which resides in kernel space. Its use is very similar to the well known `malloc` function. Finally, if registering the `major number` or allocating the memory fails, the module acts accordingly.

The “memory” driver: removing the driver

In order to remove the module inside the `memory_exit` function, the function `unregister_chrdev` needs to be present. This will free the `major number` for the kernel.

<memory exit module> =

```
void memory_exit(void) {
    /* Freeing the major number */
    unregister_chrdev(memory_major, "memory");

    /* Freeing buffer memory */
    if (memory_buffer) {
        kfree(memory_buffer);
    }

    printk("<1>Removing memory module\n");
}
```

The buffer memory is also freed in this function, in order to leave a clean kernel when removing the device driver.

The “memory” driver: opening the device as a file

The kernel space function, which corresponds to opening a file in user space (`fopen`), is the member `open:` of the `file_operations` structure in the call to `register_chrdev`. In this case, it is the `memory_open` function. It takes as arguments: an `inode` structure, which sends information to the kernel regarding the `major number` and `minor number`; and a `file` structure with information relative to the different operations that can be performed on a file. Neither of these functions will be covered in depth within this article.

When a file is opened, it’s normally necessary to initialize driver variables or reset the device. In this simple example, though, these operations are not performed.

The `memory_open` function can be seen below:

<memory open> =

```
int memory_open(struct inode *inode, struct file *filp) {

    /* Success */
    return 0;
}
```

This new function is now shown in Table 5.

Events	User functions	Kernel functions
Load module	<code>insmod</code>	<code>module_init()</code>

Open device	fopen	file_operations: open
Read device		
Write device		
Close device		
Remove module	rmmod	module_exit()

Table 5. Device driver events and their associated interfacing functions between kernel space and user space.

The “memory” driver: closing the device as a file

The corresponding function for closing a file in user space (`fclose`) is the `release:` member of the `file_operations` structure in the call to `register_chrdev`. In this particular case, it is the function `memory_release`, which has as arguments an `inode` structure and a `file` structure, just like before.

When a file is closed, it’s usually necessary to free the used memory and any variables related to the opening of the device. But, once again, due to the simplicity of this example, none of these operations are performed.

The `memory_release` function is shown below:

<memory_release> =

```
int memory_release(struct inode *inode, struct file *filp) {

    /* Success */
    return 0;

}
```

This new function is shown in Table 6.

Events	User functions	Kernel functions
Load module	insmod	module_init()
Open device	fopen	file_operations: open
Read device		
Write device		
Close device	fclose	file_operations: release
Remove module	rmmod	module_exit()

Table 6. Device driver events and their associated interfacing functions between kernel space and user space.

The “memory” driver: reading the device

To read a device with the user function `fread` or similar, the member `read:` of the `file_operations` structure is used in the call to `register_chrdev`. This time, it is the function `memory_read`. Its arguments are: a type file structure; a buffer (`buf`), from which the user space function (`fread`) will read; a counter with the number of bytes to transfer (`count`), which has the same value as the usual counter in the user space function (`fread`); and finally, the position of where to start reading the file (`f_pos`).

In this simple case, the `memory_read` function transfers a single byte from the driver buffer (`memory_buffer`) to user space with the function `copy_to_user:`

<memory read> =

```
ssize_t memory_read(struct file *filp, char *buf,
                    size_t count, loff_t *f_pos) {

    /* Transferring data to user space */
    copy_to_user(buf,memory_buffer,1);

    /* Changing reading position as best suits */
    if (*f_pos == 0) {
        *f_pos+=1;
        return 1;
    } else {
        return 0;
    }
}
```

The reading position in the file (`f_pos`) is also changed. If the position is at the beginning of the file, it is increased by one and the number of bytes that have been properly read is given as a return value, `1` . If not at the beginning of the file, an end of file (`0`) is returned since the file only stores one byte.

In Table 7 this new function has been added.

Events	User functions	Kernel functions
Load module	insmod	module_init()
Open device	fopen	file_operations: open
Read device	fread	file_operations: read
Write device		
Close device	fclose	file_operations: release
Remove modules	rmmmod	module_exit()

Table 7. Device driver events and their associated interfacing functions between kernel space and user space.

The “memory” driver: writing to a device

To write to a device with the user function `fwrite` or similar, the member `write:` of the `file_operations` structure is used in the call to `register_chrdev`. It is the function `memory_write`, in this particular example, which has the following as arguments: a type file structure; `buf`, a buffer in which the user space function (`fwrite`) will write; `count`, a counter with the number of bytes to transfer, which has the same values as the usual counter in the user space function (`fwrite`); and finally, `f_pos`, the position of where to start writing in the file.

<memory write> =

```
ssize_t memory_write( struct file *filp, char *buf,
                     size_t count, loff_t *f_pos) {

    char *tmp;

    tmp=buf+count-1;
    copy_from_user(memory_buffer,tmp,1);
    return 1;
}
```

In this case, the function `copy_from_user` transfers the data from user space to kernel space.

In Table 8 this new function is shown.

Events	User functions	Kernel functions
Load module	<code>insmod</code>	<code>module_init()</code>
Open device	<code>fopen</code>	<code>file_operations: open</code>
Close device	<code>fread</code>	<code>file_operations: read</code>
Write device	<code>fwrite</code>	<code>file_operations: write</code>
Close device	<code>fclose</code>	<code>file_operations: release</code>
Remove module	<code>rmmod</code>	<code>module_exit()</code>

Device driver events and their associated interfacing functions between kernel space and user space.

The complete “memory” driver

By joining all of the previously shown code, the complete driver is achieved:

<memory.c> =

```
<memory initial>
<memory init module>
<memory exit module>
<memory open>
<memory release>
<memory read>
<memory write>
```

Before this module can be used, you will need to compile it in the same way as with previous modules. The module can then be loaded with:

```
# insmod memory.ko
```

It's also convenient to unprotected the device:

```
# chmod 666 /dev/memory
```

If everything went well, you will have a device `/dev/memory` to which you can write a string of characters and it will store the last one of them. You can perform the operation like this:

```
$ echo -n abcdef >/dev/memory
```

To check the content of the device you can use a simple `cat`:

```
$ cat /dev/memory
```

The stored character will not change until it is overwritten or the module is removed.

The real “parlelport” driver: description of the parallel port

I'll now proceed by modifying the driver that I just created to develop one that does a real task on a real device. I'll use the simple and ubiquitous computer parallel port and the driver will be called `parlelport`.

The parallel port is effectively a device that allows the input and output of digital information. More specifically it has a female D-25 connector with twenty-five pins. Internally, from the point of view of the CPU, it uses three bytes of memory. In a PC, the base address (the one from the first byte of the device) is usually `0x378`. In this basic example, I'll use just the first byte, which consists entirely of digital outputs.

The connection of the above-mentioned byte with the external connector pins is shown in figure 2.

The “parlelport” driver: initializing the module

The previous `memory_init` function needs modification—changing the RAM memory allocation for the reservation of the memory address of the parallel port (`0x378`). To achieve this, use the function for checking the availability of a memory region (`check_region`), and the function to reserve the memory region for this device (`request_region`). Both have as arguments the base address of the memory region and its length. The `request_region` function also accepts a string which defines the module.

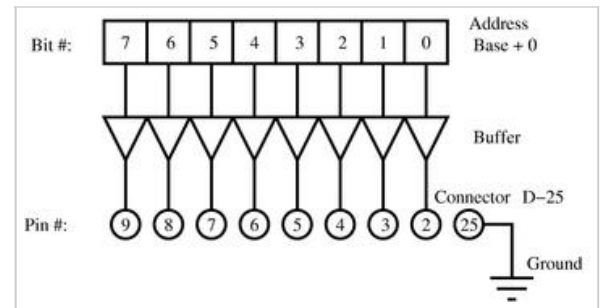


Figure 2: The first byte of the parallel port and its pin connections with the external female D-25 connector

<parlelport modified init module> =

```
/* Registering port */
port = check_region(0x378, 1);
if (port) {
    printk("<1>parlelport: cannot reserve 0x378\n");
    result = port;
    goto fail;
}
request_region(0x378, 1, "parlelport");
```

The “parlelport” driver: removing the module

It will be very similar to the `memory` module but substituting the freeing of memory with the removal of the reserved memory of the parallel port. This is done by the `release_region` function, which has the same arguments as `check_region`.

<parlelport modified exit module> =

```
/* Make port free! */
if (!port) {
    release_region(0x378, 1);
}
```

The “parlelport” driver: reading the device

In this case, a real device reading action needs to be added to allow the transfer of this information to user space. The `inb` function achieves this; its arguments are the address of the parallel port and it returns the content of the port.

<parlelport inport> =

```
/* Reading port */
parlelport_buffer = inb(0x378);
```

Table 9 (the equivalent of Table 2) shows this new function.

Events	Kernel functions
Read data	inb
Write data	

Device driver events and their associated functions between kernel space and the hardware device.

The “parlelport” driver: writing to the device

Again, you have to add the “writing to the device” function to be able to transfer later this data to user space. The function `outb` accomplishes this; it takes as arguments the content to write in the port and its address.

<parlelport *outport*> =

```
/* Writing to the port */
outb(parlelport_buffer,0x378);
```

Table 10 summarizes this new function.

Events	Kernel functions
Read data	inb
Write data	outb

Device driver events and their associated functions between kernel space and the hardware device.

The complete “parlelport” driver

I’ll proceed by looking at the whole code of the `parlelport` module. You have to replace the word `memory` for the word `parlelport` throughout the code for the `memory` module. The final result is shown below:

<parlelport.c> =

```
<parlelport initial>
<parlelport init module>
<parlelport exit module>
<parlelport open>
<parlelport release>
<parlelport read>
<parlelport write>
```

Initial section

In the initial section of the driver a different `major` number is used (`61`). Also, the global variable `memory_buffer` is changed to `port` and two more `#include` lines are added: `ioport.h` and `io.h`.

<parlelport *initial*> =

```
/* Necessary includes for drivers */
#include <linux/init.h>
#include <linux/config.h>
#include <linux/module.h>
#include <linux/kernel.h> /* printk() */
#include <linux/slab.h> /* kmalloc() */
```

```

#include <linux/fs.h> /* everything... */
#include <linux/errno.h> /* error codes */
#include <linux/types.h> /* size_t */
#include <linux/proc_fs.h>
#include <linux/fcntl.h> /* O_ACCMODE */
#include <linux/ioport.h>
#include <asm/system.h> /* cli(), *_flags */
#include <asm/uaccess.h> /* copy_from/to_user */
#include <asm/io.h> /* inb, outb */

MODULE_LICENSE("Dual BSD/GPL");

/* Function declaration of parlelport.c */
int parlelport_open(struct inode *inode, struct file *filp);
int parlelport_release(struct inode *inode, struct file *filp);
ssize_t parlelport_read(struct file *filp, char *buf,
                        size_t count, loff_t *f_pos);
ssize_t parlelport_write(struct file *filp, char *buf,
                        size_t count, loff_t *f_pos);
void parlelport_exit(void);
int parlelport_init(void);

/* Structure that declares the common */
/* file access fcuntions */
struct file_operations parlelport_fops = {
    read: parlelport_read,
    write: parlelport_write,
    open: parlelport_open,
    release: parlelport_release
};

/* Driver global variables */
/* Major number */
int parlelport_major = 61;

/* Control variable for memory */
/* reservation of the parallel port*/
int port;

module_init(parlelport_init);
module_exit(parlelport_exit);

```

Module init

In this module-initializing-routine I'll introduce the memory reserve of the parallel port as was described before.

<parlelport init module> =

```

int parlelport_init(void) {
    int result;

    /* Registering device */
    result = register_chrdev(parlelport_major, "parlelport",
                            &parlelport_fops);
    if (result < 0) {
        printk(
            "<1>parlelport: cannot obtain major number %d\n",
            parlelport_major);
        return result;
    }
}

```

```

    }

    <parlelport modified init module>

    printk("<1>Inserting parlelport module\n");
    return 0;

    fail:
        parlelport_exit();
        return result;
}

```

Removing the module

This routine will include the modifications previously mentioned.

<parlelport exit module> =

```

void parlelport_exit(void) {

    /* Make major number free! */
    unregister_chrdev(parlelport_major, "parlelport");

    <parlelport modified exit module>

    printk("<1>Removing parlelport module\n");
}

```

Opening the device as a file

This routine is identical to the `memory` driver.

<parlelport open> =

```

int parlelport_open(struct inode *inode, struct file *filp) {

    /* Success */
    return 0;

}

```

Closing the device as a file

Again, the match is perfect.

<parlelport release> =

```

int parlelport_release(struct inode *inode, struct file *filp) {

    /* Success */
    return 0;

}

```

Reading the device

The reading function is similar to the `memory` one with the corresponding modifications to read from the port of a device.

<parlelport read> =

```
ssize_t parlelport_read(struct file *filp, char *buf,
    size_t count, loff_t *f_pos) {

    /* Buffer to read the device */
    char parlelport_buffer;

    <parlelport inport>

    /* We transfer data to user space */
    copy_to_user(buf,&parlelport_buffer,1);

    /* We change the reading position as best suits */
    if (*f_pos == 0) {
        *f_pos+=1;
        return 1;
    } else {
        return 0;
    }
}
```

Writing to the device

It is analogous to the `memory` one except for writing to a device.

<parlelport write> =

```
ssize_t parlelport_write( struct file *filp, char *buf,
    size_t count, loff_t *f_pos) {

    char *tmp;

    /* Buffer writing to the device */
    char parlelport_buffer;

    tmp=buf+count-1;
    copy_from_user(&parlelport_buffer,tmp,1);

    <parlelport outport>

    return 1;
}
```

LEDs to test the use of the parallel port

In this section I'll detail the construction of a piece of hardware that can be used to visualize the state of the parallel port with some simple LEDs.

WARNING: Connecting devices to the parallel port can harm your computer. Make sure that you are properly earthed and your computer is turned off when connecting the device. Any problems that arise due to undertaking these experiments is your sole responsibility.

The circuit to build is shown in figure 3 You can also read "PC & Electronics: Connecting Your PC to the Outside World" by Zoller as reference.

In order to use it, you must first ensure that all hardware is correctly connected. Next, switch off the PC and connect the

device to the parallel port. The PC can then be turned on and all device drivers related to the parallel port should be removed (for example, `lp`, `parport`, `parport_pc`, etc.). The `hotplug` module of the Debian Sarge distribution is particularly annoying and should be removed. If the file `/dev/parlelport` does not exist, it must be created as root with the command:

```
# mknod /dev/parlelport c 61 0
```

Then it needs to be made readable and writable by anybody with:

```
# chmod 666 /dev/parlelport
```

The module can now be installed, `parlelport`. You can check that it is effectively reserving the input/output port addresses `0x378` with the command:

```
$ cat /proc/ioports
```

To turn on the LEDs and check that the system is working, execute the command:

```
$ echo -n A >/dev/parlelport
```

This should turn on LED zero and six, leaving all of the others off.

You can check the state of the parallel port issuing the command:

```
$ cat /dev/parlelport
```

Final application: flashing lights

Finally, I'll develop a pretty application which will make the LEDs flash in succession. To achieve this, a program in user space needs to be written with which only one bit at a time will be written to the `/dev/parlelport` device.

<lights.c> =

```
#include <stdio.h>
#include <unistd.h>

int main() {
    unsigned char byte,dummy;
    FILE * PARLELPORT;

    /* Opening the device parlelport */
    PARLELPORT=fopen("/dev/parlelport","w");
    /* We remove the buffer from the file i/o */
    setvbuf(PARLELPORT,&dummy,_IONBF,1);

    /* Initializing the variable to one */
    byte=1;

    /* We make an infinite loop */
    while (1) {
        /* Writing to the parallel port */
        /* to turn on a LED */
        printf("Byte value is %d\n",byte);
        fwrite(&byte,1,1,PARLELPORT);
        sleep(1);
    }
}
```

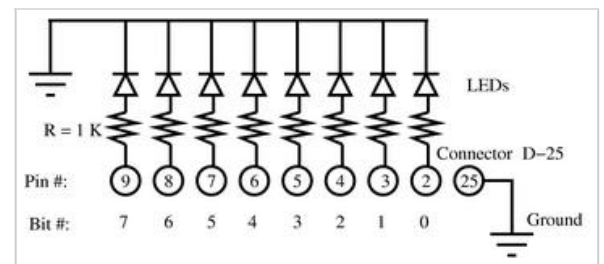


Figure 3: Electronic diagram of the LED matrix to monitor the parallel port


```

    /* Updating the byte value */
    byte<<=1;
    if (byte == 0) byte = 1;
}

fclose(PARLELPORT);
}

```

It can be compiled in the usual way:

```
$ gcc -o lights lights.c
```

and can be executed with the command:

```
$ lights
```

The lights will flash successively one after the other! The flashing LEDs and the Linux computer running this program are shown in figure 4.

Conclusion

Having followed this brief tutorial you should now be capable of writing your own complete device driver for simple hardware like a relay board (see Appendix C), or a minimal device driver for complex hardware. Learning to understand some of these simple concepts behind the Linux kernel allows you, in a quick and easy way, to get up to speed with respect to writing device drivers. And, this will bring you another step closer to becoming a true Linux kernel developer.

Bibliography

A. Rubini, J. Corbet. 2001. [Linux device drivers \(second edition\)](#). Ed. O'Reilly. This book is available for free on the internet.

Jonathan Corbet. 2003/2004. [Porting device drivers to the 2.6 kernel](#). This is a very valuable resource for porting drivers to the new 2.6 Linux kernel and also for learning about Linux device drivers.

B. Zoller. 1998. PC & Electronics: Connecting Your PC to the Outside World (Productivity Series). Nowadays it is probably easier to surf the web for hardware projects like this one.

M. Waite, S. Prata. 1990. C Programming. Any other good book on C programming would suffice.

Appendix A. Complete Makefile

<Makefile> =

```
obj-m := nothing.o hello.o memory.o parlelport.o
```

Appendix B. Compiling the kernel on a Debian Sarge system

To compile a 2.6.x kernel on a Debian Sarge system you need to perform the following steps, which should be run as root:

- Install the "kernel-image-2.6.x" package.

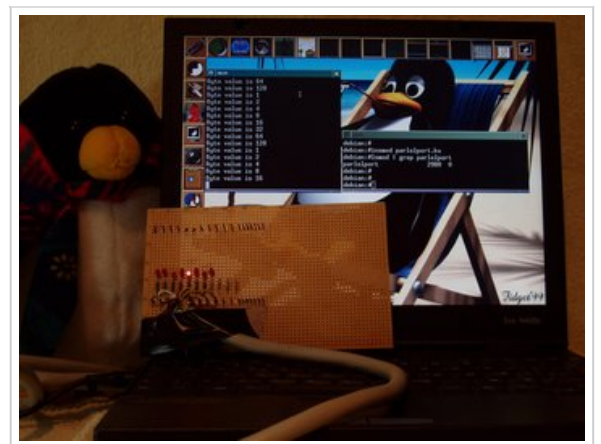


Figure 4: Flashing LEDs mounted on the circuit board and the computer running Linux. Two terminals are shown: one where the "parlelport" module is loaded and another one where the "lights" program is run. Tux is closely following what is going on

- Reboot the machine to make this the running kernel image. This is done semi-automatically by Debian. You may need to tweak the lilo configuration file `/etc/lilo.conf` and then run `lilo` to achieve this.
- Install the “kernel-source-2.6.x” package.
- Change to the source code directory, `cd /usr/src` and unzip and untar the source code with `bunzip2 kernel-source-2.6.x.tar.bz2` and `tar xvf kernel-source-2.6.x.tar`. Change to the kernel source directory with `cd /usr/src/kernel-source-2.6.x`
- Copy the default Debian kernel configuration file to your local kernel source directory `cp /boot/config-2.6.x.config`.
- Make the kernel and the modules with `make` and then `make modules`.

Appendix C. Exercises

If you would like to take on some bigger challenges, here are a couple of exercises you can do:

- I once wrote two device drivers for two ISA [Meilhaus](#) boards, an analog to digital converter (ME26) and a relay control board (ME53). The software is available from the [ADQ](#) project. Get the newer PCI versions of these Meilhaus boards and update the software.
- Take any device that doesn’t work on Linux, but has a very similar chipset to another device which does have a proven device driver for Linux. Try to modify the working device driver to make it work for the new device. If you achieve this, submit your code to the kernel and become a kernel developer yourself!

Comments and acknowledgements

Three years have elapsed since the [first version](#) of this document was written. It was originally written in Spanish and intended for version 2.2 of the kernel, but kernel 2.4 was already making its first steps at that time. The reason for this choice is that good documentation for writing device drivers, the **Linux device drivers** book (see bibliography), lagged the release of the kernel in some months. This new version is also coming out soon after the release of the new 2.6 kernel, but up to date documentation is now readily available in [Linux Weekly News](#) making it possible to have this document synchronized with the newest kernel.

Fortunately enough, PCs still come with a built-in parallel port, despite the actual trend of changing everything inside a PC to render it obsolete in no time. Let us hope that PCs still continue to have built-in parallel ports for some time in the future, or that at least, parallel port PCI cards are still being sold.

This tutorial has been originally typed using a text editor (i.e. `emacs`) in `noweb` format. This text is then processed with the `noweb` tool to create a LaTeX file (`.tex`) and the source code files (`.c`). All this can be done using the supplied `makefile.document` with the command `make -f makefile.document`.

I would like to thank the “Instituto Politécnico de Bragança”, the “Núcleo Estudantil de Linux del Instituto Politécnico de Bragança (NUX)”, the “Asociación de Software Libre de León (SLeón)” and the “Núcleo de Estudantes de Engenharia Informática da Universidade de Évora” for making this update possible.

Category: [Hacking](#)

License:

[GFDL](#)

[Xavier Calbet’s articles](#) [Log in](#) or [register](#) to post comments

Comments

Great,

Submitted by [circulus](#) on Thu, 2006-04-27 11:03



Very nice well written, this nice step by step tutorial is exactly for me.

"... Nothing Runs Like A Python ..."

[Log in](#) or [register](#) to post comments

Excellent job

Submitted by Anonymous visitor (not verified) on Wed, 2006-10-04 11:31



Thanks for the Excellent job of making LINUX drivers easy to understand in a very very short time.

[Log in](#) or [register](#) to post comments

Great.....

Submitted by deepak george (not verified) on Thu, 2007-09-13 09:50

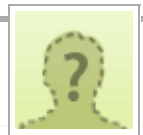


Great.....

[Log in](#) or [register](#) to post comments

Thank you

Submitted by Anonymous visitor (not verified) on Mon, 2006-08-14 20:13

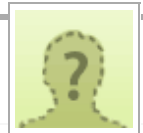


Great tutorial!

[Log in](#) or [register](#) to post comments

Amazing

Submitted by Anonymous visitor (not verified) on Wed, 2006-08-16 13:10



I am amazed!! How have you managed to get this thing so simple? A real good job.

Sundip Sharma

[Log in](#) or [register](#) to post comments

WOW!!!!!! simply fantastic work

Submitted by Anonymous visitor (not verified) on Wed, 2006-08-30 08:12



Thanks

a great article with ample scope of learning the skeleton of linux device drivers and the commonly used functions in them.....so simple and presented in quite an interesting way, so never lets u feel bored.
great work

[Log in](#) or [register](#) to post comments

Great simple tutorial

Submitted by Anonymous visitor (not verified) on Thu, 2006-08-31 00:29

After kernel 2.4 the kernel compilation and adding up of modules has changed drastically. This tutorial helped me compile and insert my module. Thanks. Effort whole heartedly appreciated.



Kabeer Ahmed.

[Log in](#) or [register](#) to post comments

WOW!!!!!!!!!!!!1 Great....

Submitted by Anonymous visitor (not verified) on Thu, 2006-08-31 17:30



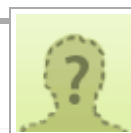
A great article with ample scope of learning with nice step by step procedure..

Thank's
Prakash.

[Log in](#) or [register](#) to post comments

Its a good

Submitted by Anonymous visitor (not verified) on Fri, 2006-09-08 12:40



Its a good information.

Regards, Rajesh

[Log in](#) or [register](#) to post comments

No word's to say

Submitted by Anonymous visitor (not verified) on Sun, 2006-09-17 17:37

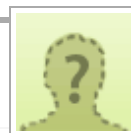


Great Work !!!

[Log in](#) or [register](#) to post comments

Linux is not an OS

Submitted by Anonymous visitor (not verified) on Fri, 2006-09-22 11:41

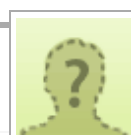


"Linux's kernel" has no sense, because Linux is a kernel

[Log in](#) or [register](#) to post comments

Query on reading from the device

Submitted by Vishw anatha K on Fri, 2006-09-29 10:16



Hi All,

I actually tried memory.ko as below:

```
$echo -n abcdef > /dev/memory
```

```
$cat /dev/memory
```

```
f
```

But I've got only the last written character i.e., 'f'.

I changed the code so that the buffer can hold upto 10 bytes (by giving 10 as a parameter to kmalloc and read,write calls) but still I'm getting the last input character. Can any one post the answer here...

--

Regards,
Vishwa

[Log in](#) or [register](#) to post comments

Character driver...

Submitted by bzamora on Tue, 2008-01-08 23:32



Hi Vishwa,

On page 4 of the tutorial, where the author shows the mknod command, he states that the c identifies this driver as a character driver. My guess is that is the reason that you only see the last letter. I believe character drivers will only allow you to receive the most recent character that was written... ie the 'f'.

[Log in](#) or [register](#) to post comments

m getting the full string

Submitted by chandan_001 on Sat, 2013-03-16 10:23



Hi all,
I've got the full string and don't change the code.
using

```
$echo "chandan kumar bera" > /dev/memory  
$cat /dev/memory  
try it.
```

[Log in](#) or [register](#) to post comments

A really Helpful link

Submitted by Anonymous visitor (not verified) on Tue, 2006-10-03 10:08



A really good and Helpful link for newbies!!!!

[Log in](#) or [register](#) to post comments

To start my career as programmer in device drivers

Submitted by jadhav_raghav on Thu, 2006-10-12 14:48



hi,

This one is good article.

I want to learn more on this, and can u help me get more info about device drivers.

Thanks,
Raghavendra

[Log in](#) or [register](#) to post comments

WOW A MASTERPIECE

Submitted by Anonymous visitor (not verified) on Fri, 2006-10-20 11:06



Gud work by u. A gud guide for a layman in the fiels of device drivers

Heartly appreciated

[Log in](#) or [register](#) to post comments

problem with driver software

Submitted by Anonymous visitor (not verified) on Tue, 2006-10-31 10:23



hello sir whhen i have compiled my sorce code then i face an error that "fops size is not known " pleee send me some information to remove this error
mu email id is erabhinandan@gmail.com

[Log in](#) or [register](#) to post comments

Linux extra-versioning problem and another useful link

Submitted by Anonymous visitor (not verified) on Tue, 2006-10-31 11:10



Thank you for your excellent tutorial!

That's true that the first step is always the more hard to achieve, and guides like this really simplify the achievement of the basic know-how necessary to get real job done.

When I first tried the tutorial I get this error when trying to insmod my module nothing.ko:
insmod: error inserting nothing.ko: -1 Invalid module format

while dmesg gave me:

```
nothing: version magic '2.6.17.11.061003.eraamil SMP mod_unload PENTIUM4 REGPARM gcc-4.1' should be  
'2.6.17.11.061003.eraamil SMP mod_unload PENTIUM4 REGPARM gcc-4.0'
```

It was that I like to add a customized extra version to my kernel compiling it, while the extra version in the source tree Makefile was still ".11".

To fix this I changed manually the Makefile, updating the extra version line with the current linux extra version, then I started to compile with make (but I control-C-ed quite soon, after the compilation of the version.h file, that you can eventually change by hand).

Then, since I compiled the kernel with gcc-4.0 and I had installed the gcc-4.1 version, I changed the gcc symlink with:
`sudo ln -sf /usr/bin/gcc-4.0 /usr/bin/gcc`

Now the gcc symlink (used to compile the module) points to the same version of gcc used to compile the kernel: trick done!

Another useful link:

<http://www.openaddict.com/documents/lkmpg/x30.html>

HTH

Cheers

[Log in](#) or [register](#) to post comments

Compiling with Kernel Version 2.4.20-8

Submitted by Anonymous visitor (not verified) on Thu, 2006-11-02 04:54



hi,
This is Priya here. I have installed Redhat Linux version 9 on my system. Kernel version 2.4.20-8. Could I Please know how to compile and insert the modules given in this tutorial, with the associated makefiles. Please help me. My mail id : priyasophy_1982@yahoo.com

[Log in](#) or [register](#) to post comments

Reply

Submitted by Anonymous visitor (not verified) on Fri, 2007-01-26 15:37



Its better if you use kernel 2.6.x.x.2. Because the Linux systems now supports 2.6 kernel and the interface for 2.6 is simple and totally different than 2.4. Mention the topic on which you are working.
If I have a knowledge about that i will surely help you. Do post ur queries on prem38_kamble @rediffmail.com

[Log in](#) or [register](#) to post comments

This makefile worked for me.

Submitted by Zafeer (not verified) on Thu, 2007-11-01 05:22



This makefile worked for me. I got it off another site:

```
obj-m += memory.o
```

```
all:
```

```
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
```

```
clean:
```

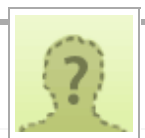
```
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

So the only difference is you only compile memory.c, although you can just add more files after memory.o like hello.o, etc to compile hello.c
Hope this helps.

[Log in](#) or [register](#) to post comments

Works with CentOS 5...

Submitted by bzamora on Tue, 2008-01-08 23:25



Hi...

Thanks for this sample. This worked great with CentOS 5.

[Log in](#) or [register](#) to post comments

very nice work !!!!!!!!

Submitted by Anonymous visitor (not verified) on Mon, 2006-11-06 08:35



I appreciate for the effort took for explaining device driver writing in step by step manner...

really great work...

Thank you,
Murali

[Log in](#) or [register](#) to post comments

thanks alot

Submitted by Anonymous visitor (not verified) on Tue, 2006-11-14 10:50



thanks man nice tutorial

regards
SuLtAn

[Log in](#) or [register](#) to post comments

REALLY AGREAT HELP FOR

Submitted by Anonymous visitor (not verified) on Tue, 2006-11-21 11:43



REALLY A GREAT HELP FOR STARTERS...

ARUN

[Log in](#) or [register](#) to post comments

Cannot unload module

Submitted by Anonymous visitor (not verified) on Sat, 2006-11-25 06:31



Dear sir,

This is a very good tutorial you have here. It is very cool as it explains the fundamental really well.

I tried out the first 2 section of the tutorial but am stuck.

I have created the module nothing.ko and inserted it into the kernel.
I am able to see the module using lsmod

When I tried to rmmod, the system keeps giving me the following error:

ERROR: Removing 'nothing': Device or resource busy

My lsmod | grep nothing gives me
nothing 2688 0 [permanent]

Is there a way to unload this module

Very much appreciate your help

Thanks,
Alex

[Log in](#) or [register](#) to post comments

Not accurate article!

Submitted by Anonymous visitor (not verified) on Sat, 2006-11-25 19:21



My comments:

To be complete tutorial it must be accurate. It is not so.

For example, memory driver part is completely hide extensions of files and creating makefile in case if you build module from multiply files.

I think it should be added directly in article or,as minimum, in comments.

[Log in](#) or [register](#) to post comments

d BEST

Submitted by Anonymous visitor (not verified) on Tue, 2006-11-28 12:22



Thank you very much for posting such an astonish article for all..

God Bless all..

[Log in](#) or [register](#) to post comments

GOOD

Submitted by Anonymous visitor (not verified) on Tue, 2006-11-28 13:10



Thank u for giving an easy to learn tutorial about device drivers..

U can also add some more details about functions which will be more helpful...

[Log in](#) or [register](#) to post comments

VeryGood

Submitted by Anonymous visitor (not verified) on Tue, 2006-12-05 10:01



The Tutorial help me lot

[Log in](#) or [register](#) to post comments

good article

Submitted by Anonymous visitor (not verified) on Wed, 2006-12-20 01:47



As the title says,....
Simple to understand and grasp the important things.
Thanks a lot m8
George

[Log in](#) or [register](#) to post comments

NICE ONE

Submitted by Anonymous visitor (not verified) on Wed, 2006-12-20 13:39



VERY NICE ARTICLE.

THANKS A LOT.
K.RAJASEKAR

[Log in](#) or [register](#) to post comments

good job

Submitted by Anonymous visitor (not verified) on Sun, 2006-12-24 15:25



good job

[Log in](#) or [register](#) to post comments

Its the Best that I have come accross

Submitted by Anonymous visitor (not verified) on Mon, 2006-12-25 06:52



It motivates a non-driver-writer to jump into the arena that is always thought to be only for the real techies.

[Log in](#) or [register](#) to post comments

Verywell done

Submitted by Anonymous visitor (not verified) on Tue, 2006-12-26 00:02



Really good tutorial on drivers. I challenge us to do the exercises, specifically a device that doesn't work yet.

Thanks for this important contribution!

Regards,

[Log in](#) or [register](#) to post comments

Help on Kernel Keyboard Driver

Submitted by Anonymous visitor (not verified) on Tue, 2006-12-26 05:33



Hai everyone,

IN LINUX KERNEL VERSION-2.6.8-24 THE KEYBOARD DRIVER FILE KEYBOARD.C IS USED.I WOULD LIKE TO MODIFY THIS FILE SUCH THAT ON A TOGGLE KEY I WOULD LIKE TO READ DATA FROM A FILE.I USED SYSTEM CALLS LIKE OPEN,CLOSE,READ.BUT AN ERROR MESSAGE SAYING THAT THE HEADER FILE UNISTD.H INCLUDING THIS CALLS IS NOT AVAILABLE IS ENCOUNTERED.CAN ANY BODY HELP ME IN THIS CONTEXT.

[Log in](#) or [register](#) to post comments

After you've done that

Submitted by Anonymous visitor (not verified) on Fri, 2007-04-27 22:07



After you've done that you could write a driver that prevents your caps-lock key from getting stuck.

[Log in](#) or [register](#) to post comments

hi

Submitted by Anonymous visitor (not verified) on Tue, 2007-01-02 17:12



good job done

[Log in](#) or [register](#) to post comments

Only bit-1 lights up after module loads

Submitted by Anonymous visitor (not verified) on Fri, 2007-01-19 01:42



First off, thanks for the great tutorial, Xavier.

I am trying to complete the LED section of the tutorial. I am running Ubuntu Server 6.10, kernel 2.6.17-10, and I removed lp, parport_pc, and parport. I have successfully compiled parlepport.ko and I can verify that the module has loaded with:

```
$ cat /proc/ioprots
```

```
..
```

```
0378-0378 : parlepport
```

The circuit shows many lights blinking at boot time. After boot, bit-2 and bit-3 stay lit. They remain that way until the computer is rebooted – or until the monitor goes to sleep. All lights go black while the monitor sleeps. The only response I can generate is that bit-1 lights after inserting parlepport.ko. It stays lit until the computer is rebooted.

The only thing that looks suspicious is this:

```
$:~/dev$ make -C /usr/src/linux-source-2.6.17 M=`pwd` modules
```

```
make: Entering directory `/usr/src/linux-source-2.6.17'
```

```
CC [M] /home/sam/dev/parlepport.o
```

```
/home/sam/dev/parlepport.c:31: warning: initialization from incompatible pointer type
```

```
..
```

Line 31 corresponds to:

```
write: parlepport_write
```

But my parlepport_write() is the same as the tutorial's.

Any pointers?

Thanks,
Sam

[Log in](#) or [register](#) to post comments

I have more or less the same

Submitted by Anonymous visitor (not verified) on Sun, 2007-02-11 12:00



I have more or less the same issue.

after booting the LED is OFF but when I remove all parport related modules, the LED gets ON and the parlepport driver does not work. If I manage to avoid parport module at boot and insmod parlepport after boot, things seems OK (I can set On or Off the LED).

I guess the parport driver when manually unloaded (rmmod) let the chipset in a state that prevent from outputting data from the port (may in input mode in place of output mode)

I tried to force the output mode (0x37A control reg) with no success

[Log in](#) or [register](#) to post comments

how did you remove lp, parport_pc, and parport

Submitted by leneth (not verified) on Wed, 2007-08-22 10:19



Hi!

How did you remove lp, parport_pc, and parport ? I think i need to do this becuse i will get a device or resource busy error.

Thanks,
Leneth

[Log in](#) or [register](#) to post comments

VeryWell Explained

Submitted by Anonymous visitor (not verified) on Thu, 2007-01-25 23:29



Fantastic Job. You have described the driver structure in simple and best manner.
Thanks.AJ

[Log in](#) or [register](#) to post comments

Damn useful !!! Well done

Submitted by Anonymous visitor (not verified) on Thu, 2007-02-01 19:41



Damn useful !!! Well done buddy :-*

[Log in](#) or [register](#) to post comments

Good one.

Submitted by Anonymous visitor (not verified) on Mon, 2007-02-05 07:18



Hello,

Really nice tutorial...it helped me a lot..Practical Eggs are good..

Keep the good work going..

[Log in](#) or [register](#) to post comments

Excellent, Quick & Handy.,

Submitted by Anonymous visitor (not verified) on Thu, 2007-02-08 07:05



This is really nice & handy tutorial.

Can this guy give similar tutorials for GCC,Shell Scripting..? It would be really good to have such tutorials..

[Log in](#) or [register](#) to post comments

Very Helpful!

Submitted by Anonymous visitor (not verified) on Mon, 2007-02-12 22:14



A really handy overview!

[Log in](#) or [register](#) to post comments

write a kernel for linux

Submitted by Anonymous visitor (not verified) on Sun, 2007-03-04 20:06



how we write a kernel for linux? what is code of this problem?

[Log in](#) or [register](#) to post comments

SIMPLE & COOL

Submitted by Anonymous visitor (not verified) on Mon, 2007-03-05 10:50



The tutorial has been simple, I had no idea where to start with.

This has given me a good introduction to DD programming & kernel modules.

-Arjun

[Log in](#) or [register](#) to post comments

Compilation problem

Submitted by Anonymous visitor (not verified) on Thu, 2007-03-08 21:58



I am using Fedora core 5. I could not compile the very first version of "nothing" with just licence line in it.

Here is what I got.

```
make -C /usr/src/linux/kernel M=`pwd` modules
make: Entering directory `/usr/src/redhat/BUILD/kernel-2.6.15/linux-2.6.15.i686/kernel'
make: *** No rule to make target `modules'. Stop.
make: Leaving directory `/usr/src/redhat/BUILD/kernel-2.6.15/linux-2.6.15.i686/kernel'
```

Is the procedure different for Fedora? Thanks in advance.

[Log in](#) or [register](#) to post comments

Great

Submitted by kirz on Fri, 2007-03-16 12:42

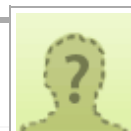


Its really simple and informative.....

[Log in](#) or [register](#) to post comments

Good work

Submitted by Anonymous visitor (not verified) on Fri, 2007-03-16 17:19



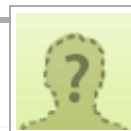
A really interesting tutorial. A little short but the main explanations are present.
Thanks.

KirgliZ

[Log in](#) or [register](#) to post comments

tutorial(device driver)

Submitted by Anonymous visitor (not verified) on Thu, 2007-03-22 05:31

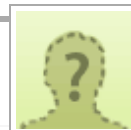


its really excellent....
amazing...
His presentation is too good and simple....

[Log in](#) or [register](#) to post comments

Yeah

Submitted by kirz on Thu, 2007-03-29 11:44

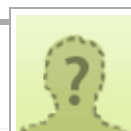


Simply simple :)

[Log in](#) or [register](#) to post comments

Efficient work

Submitted by Anonymous visitor (not verified) on Wed, 2007-04-04 05:55



Really good work.

[Log in](#) or [register](#) to post comments

Thank you so much!

Submitted by Anonymous visitor (not verified) on Sat, 2007-04-07 04:37



You wrote so clearly and easily to understand.

[Log in](#) or [register](#) to post comments

great tutorial for newbies

Submitted by Anonymous visitor (not verified) on Sat, 2007-04-07 07:56



This tutorial gives great confidence to develop device driver for someone who is neo in this field. Great job !!!!!!!!

[Log in](#) or [register](#) to post comments

Great job.

Submitted by solar_sea (not verified) on Tue, 2007-04-17 18:22



I really like it and it's written in a very nice way, thanks! :)

[Log in](#) or [register](#) to post comments

Great Work Sir. Its truly

Submitted by Anonymous visitor (not verified) on Wed, 2007-04-18 13:07



Great Work Sir. Its truly helpful for beginneers like us. We hope to have some more from you.

[Log in](#) or [register](#) to post comments

Pretty Good and Easy to Understand

Submitted by Anonymous visitor (not verified) on Tue, 2007-04-24 14:32



Dear sir...
its really very useful.can u elaborate the drivers on keyboard also.

[Log in](#) or [register](#) to post comments

Very interesting. Have never

Submitted by Anonymous visitor (not verified) on Tue, 2007-05-01 06:10



Very interesting. Have never really read much about driver development. Nice work.

[Log in](#) or [register](#) to post comments

Printable version?

Submitted by gandy909 (not verified) on Tue, 2007-05-01 18:15



Is there a printable version of this article? I don't find a link to one...
Thanks.

[Log in](#) or [register](#) to post comments

There is no such thing as

Submitted by Raquel (not verified) on Tue, 2007-05-01 19:12



There is no such thing as "quick" and "easy" when it comes to writing drivers. Thank you :)

P.S: great tutorial

[Log in](#) or [register](#) to post comments

Debian etch, is now the

Submitted by Anonymous visitor (not verified) on Wed, 2007-05-02 00:49

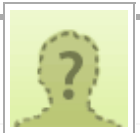


Debian etch, is now the stable release version.

[Log in](#) or [register](#) to post comments

Similar Tutorial for NT

Submitted by Anonymous visitor (not verified) on Thu, 2007-05-10 12:33



I can write a similar tutorial using LEDs if anyone is interested for DOS.

I am interested in Windows NT, please give us similar simple tutorial for Windows NT device drivers.

Great Work.

Ja

[Log in](#) or [register](#) to post comments

Great work.

Submitted by Anonymous visitor (not verified) on Sat, 2007-06-02 14:45



I am really impressed the way author has presented this wonderful article..he has explained in such a simple and nice manner..

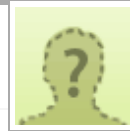
Great going..

so happy hacking..

[Log in](#) or [register](#) to post comments

Submitted by Anonymous

Submitted by Anonymous visitor (not verified) on Tue, 2007-06-12 15:45



Submitted by Anonymous visitor (not verified) on Thu, 2007-05-10 12:33.

I can write a similar tutorial using LEDs if anyone is interested for DOS.

I am interested in Windows NT, please give us similar simple tutorial for Windows NT device drivers.

Great Work.

[Log in](#) or [register](#) to post comments

This is FREE SOFTWARE Magazine

Submitted by Danboy on Tue, 2007-06-12 23:31



I'm afraid that if you are interested in Windows NT you'll have to find another magazine to make your requests to. You won't be seeing any Windows NT tutorials on this site unless they tell you how to wipe it from your computer and replace it with a nice GNU/Linux replacement.

It's amazing that you are smart enough to understand this article but yet you can't work out that no one here would want to write NT tutorials.

[Log in](#) or [register](#) to post comments

Gud tutorial ,keep the good work going!

Submitted by Ulhas Dhuri (not verified) on Fri, 2007-06-29 09:19



Hi,

Its a good tutorial, I must say. You have gracefully handled this linux device driver tutorial. It would be a great benefit to newbies ,if u can come up with futher advance topics of kernel programming in your own style of writing.

[Log in](#) or [register](#) to post comments

French tutorial

Submitted by Metoule (not verified) on Mon, 2007-07-09 20:58



Amazing tutorial ! It made me want to develop my own driver!

For French speaking people, you can find another tutorial here: <http://broux.developpez.com/articles/c/driver-c-linux/>

[Log in](#) or [register](#) to post comments

Excellent article

Submitted by dalex on Wed, 2007-07-11 14:16



I only wish there were more tutorials like that...

[Log in](#) or [register](#) to post comments

Tip-Top

Submitted by skypjack on Sat, 2007-07-14 08:45



I quote above comment, an excellent article!
Congratulations and ... Thanks! :-)

[Log in](#) or [register](#) to post comments

Very nice tutorial that gets you up and running quickly

Submitted by Jakobularius on Tue, 2007-07-17 06:17



The tutorial provides a nice intro to really mastering your Linux machine!

[Log in](#) or [register](#) to post comments

Its

Submitted by Anonymous visitor (not verified) on Tue, 2007-07-24 06:26



Its fantastic...!!!!!!!!!!!!!!

[Log in](#) or [register](#) to post comments

HELP needed about device driver writing

Submitted by prince2007 (not verified) on Mon, 2007-07-30 01:07



hi,
this ia a excellent tutorial, i am using rhel5 2.6.18-8.el5
i like to learn this device driver programming but i am gating
error: linux/module.h: No such file or directory
so how to get this file. if i download this file will it work?
plz inform me princei007@yahoo.com

[Log in](#) or [register](#) to post comments

I'm having the same problem

Submitted by Anonymous visitor (not verified) on Wed, 2007-08-01 08:25



I'm having the same problem with ubuntu

[Log in](#) or [register](#) to post comments

Good tutorial

Submitted by Anonymous visitor (not verified) on Wed, 2007-08-08 18:16



Good job.

I have searched and read many,

Atlast I found this tutorial, its amazingly easy to get the picture of writing a driver.

thanks
by
karthi

[Log in](#) or [register](#) to post comments

outb is not working

Submitted by Mohan (not verified) on Tue, 2007-08-28 14:24

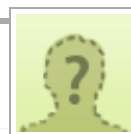


I wrote a small application to test the parallel port and whatever data I am writting, it is getting displayed on LED array. However when I tried to use outb from driver, it is not working. I have removed lp, parport_pc, parport modules. I put some debug messages and it is showing outb is executing properly. Even /proc/ioports is showing ports are allocated for me. I am using Fedora v6. Do you have any clue on this?

[Log in](#) or [register](#) to post comments

help needed

Submitted by Anonymous visitor (not verified) on Thu, 2007-09-06 07:19



Hi
help me for writng USB device driver. from where i start my work.
thanks

[Log in](#) or [register](#) to post comments

Good article!

Submitted by Anonymous visitor (not verified) on Tue, 2007-09-25 06:37



Good starting point for beginners. Thank you!

[Log in](#) or [register](#) to post comments

Is the code available?

Submitted by vonbrand on Wed, 2007-09-26 19:58

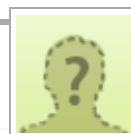


I'm interested both in the source code (C, Makefile) and in the article itself.

[Log in](#) or [register](#) to post comments

i was running awayfrom

Submitted by Anonymous visitor (not verified) on Fri, 2007-09-28 19:17



i was running away from device drivers from last 3 weeks but this tutorial made it so easy. withing 2 hrs i had the idea wat dey r nd hw to make dem work..excellent tutorial. great work.thanx a lot

[Log in](#) or [register](#) to post comments

How I got it working

Submitted by Redmond (not verified) on Sat, 2007-09-29 08:06



I had some trouble getting the parport driver working on my slackware 12.2.6.21. The problem was that every time I boot my machine the pin 4 led would not shut off, and all other leds could not be turned on or off by echo or lights.o. All of this was caused by modules lp, parport, and parport_pc. The solution was to comment out the lines that load the modules in /etc/rc.d/rc.modules-[kernel version here]. There pretty hard to miss. Add the modules to /etc/modprobe.d/blacklist did not help though I did that and it may be part of the solution.

P.S. I love this tutorial, you wrote that it was originally written in Spanish, If English is not your first language I can't tell.

[Log in](#) or [register](#) to post comments

The simplest, compact

Submitted by Anonymous visitor (not verified) on Mon, 2007-10-01 10:54



The simplest, compact tutorial to anyone to jump start on the driver.

[Log in](#) or [register](#) to post comments

Best tutorial

Submitted by Manish A (not verified) on Wed, 2007-10-10 22:37



This is the best tutorial I've read on the web. And I'm not speaking only of linux device driver tutorials.

Thanks!!

[Log in](#) or [register](#) to post comments

device driver

Submitted by shekhar (not verified) on Thu, 2007-10-18 09:12



Thx.
this article is very useful.
it helped me a lot in understanding character driver.

but i have problem with assigning the more space to memory buffer.
even i increased it to 10 from 1 i am able to get only last character,
why this is so?

[Log in](#) or [register](#) to post comments

well written

Submitted by anantbasu (not verified) on Mon, 2007-10-22 07:47



quite a precise and well written startup material for device drivers' developers
-AB

[Log in](#) or [register](#) to post comments

compile process

Submitted by Oliver Thane (not verified) on Fri, 2007-10-26 22:28



Thanks for the excellent tutorial. However I am stuck in the position where by I must use Kernel 2.4 for the module I am working on. The driver it self has been mostly developed and I am just debugging some glitches.

I have the problem that in order to compile the module to work on the target embedded device I must be in Kernel version 2.4.20-20. Despite my efforts my network card only works in 2.4.20-8SMP and when I compile the drive in this mode it will not run on the embedded target. The solution currently is to reboot after every build into the 2.4.20-8SMP, which is not great.

I will look into fixing the network card problem but I am quit interested in learning how to compile drivers for versions of the OS not currently running (and especially in 2.4). I cannot seem to find much on the net so if anyone knows where to look or can help it would be much appreciated.

[Log in](#) or [register](#) to post comments

Could someone give some

Submitted by Anonymous visitor (not verified) on Wed, 2007-10-31 21:18



Could someone give some inside or where to find detailed info on supporting/writing drivers for dual core or multiprocessor HW with under 2.6? TIA

[Log in](#) or [register](#) to post comments

do u have a simple source code of usb or pci modules?

Submitted by Roya_Gh (not verified) on Fri, 2007-11-09 11:02



hello,

i'm a student of computer eng(hardware)and at the moment i'm working on a project which is about USB driver programming in linux.i use kernell 2.6.18(fedora core). i have run a simple module in it(i mean that famous hello module)but after that i tried to run some other modules such as pci modules but i faced with some problems...maybe the program it self has some problems...so...is there any persons here that has a source code of a simple usb or pci drivers?i would be really thankful for your attention.

[Log in](#) or [register](#) to post comments

Great Work!!!

Submitted by Sid B (not verified) on Mon, 2007-11-12 05:39



Its really a wonderfull piece of documentation for beginners trying to develop Linux device drivers.

[Log in](#) or [register](#) to post comments

Verywell written...

Submitted by Brett Zamora (not verified) on Tue, 2007-11-20 22:11



Excellent presentation. I look forward to actually building the examples.

[Log in](#) or [register](#) to post comments

Help

Submitted by Anonymous visitor (not verified) on Fri, 2007-11-23 14:29



I am using knoppix Live CD. And I don't know how to compile my modules in knoppix. If anyone can help I will be thankful.

[Log in](#) or [register](#) to post comments

Awsome!

Submitted by Anonymous visitor (not verified) on Tue, 2007-11-27 18:36



Never seen an article describing device driver writing this simple.

[Log in](#) or [register](#) to post comments

Excellent article!

Submitted by metacrease (not verified) on Mon, 2007-12-03 08:17



Excellent article!

[Log in](#) or [register](#) to post comments

Really nice article

Submitted by sow mya.kethireddy on Tue, 2008-01-29 08:57



hi,
I'm new to linux. I'm searching for easily understandable article. I found this. ThanQ verymuch to the author.

[Log in](#) or [register](#) to post comments

Problems in Ubuntu on memory and parlepport drivers

Submitted by new to linux on Thu, 2008-02-14 16:08



Hello, I am new to linux, so please forgive me.

I have been able to compile and run the first two drivers in the tutorial but once I try the multi file ones i have issues the memory drive and the parlepport driver will not compile.

The lines in brackets like this one: '**<parlepport modified init module>**' ...without the " I just have them so the brackets will show up and not be filtered.

that refrence other files I have created like the tutorial said to do are generating the following error:

In file included from /usr/src/parport/parlepport.c:2:

/usr/src/parport/parlepport init module: In function 'parlepport_init':

/usr/src/parport/parlelport init module:14: error: expected expression before '<' token

I have literally copied and pasted the example code... what am I missing?

I am using UBUNTU

Thanks

[Log in](#) or [register](#) to post comments

Solution

Submitted by admin on Fri, 2008-02-15 14:30



Hi,

I urge you to ask the same question in the Ubuntu Forums (<http://ubuntuforums.org/>). They are fantastic, and will definitely help you!

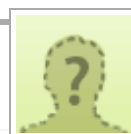
Bye,

Merc.

[Log in](#) or [register](#) to post comments

Compile these examples for ARM architecture

Submitted by thainam on Mon, 2008-08-11 11:31



Hi, thank you for your article, it's great. I compiled them and work ok on Ubuntu:
obj-m += parlelport.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

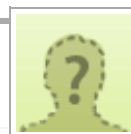
However, I installed them on Iliad tablet, it said different architecture(it bases on ARM architecture). Could you show me how to these drivers for Iliad tablet?

Once again, thank you very much.

[Log in](#) or [register](#) to post comments

driver

Submitted by azad on Thu, 2008-09-11 10:33



Hello to everybody,

I have read the script to Linux driver. I would like to know how one makes the circuit LEDs.

Many greetings

Azad

[Log in](#) or [register](#) to post comments

Nonexistant resource 00000000378

Submitted by BilalMehdi on Fri, 2008-10-03 17:41



Hi...

I tried the parallel port driver .. but ... I am having some problems...

1. I dont see 0378 memory allocated to my module with `cat /proc/ioports` command. (I dont see memory location 0378 in the list)
2. I cant see the state of the port with `cat /dev/parlel_port`
3. When I remove my module .. I get a message in the log "Trying to free nonexistent resource 00000000378-00000000378" ... (No error message is logged during `insmod`)

Although I did remove `parport` and `parport_pc` module... (I was unable to load my module without unloading these modules)

If someone knows how to fix it.. please post a reply here..

Thanx in advance

[Log in](#) or [register](#) to post comments

Procedure to unload the current parallel port driver

Submitted by saurabhg84 on Mon, 2010-02-08 10:28



Firstly, let me say...this is an awesome tutorial but the only thing missing out here is an elaborate procedure to unload the driver which is present by default on all systems having a parallel port. For some reason `rmmod/modprobe` did not work out here and the solution as someone has pointed out in one of the earlier comments is to modify the startup script to NOT load the parallel port driver at startup. In Ubuntu, I used the following steps to unload the driver during startup and now it works brilliantly:

Steps to unload paralel port driver:

1. Make the following changes in the `/etc/rc1.d` :
`K80 cups: modprobe -q lp || true`
`K80 cups: modprobe -q ppdev || true`
Uncomment the above 2 lines of the scripts
2. Make the following changes in the `/etc/rc2.d` :
`S50 cups: modprobe -q ppdev || true`
`S50 cups: modprobe -q lp || true`
Uncomment the above 2 lines of the scripts
3. Add the following lines to the file `/etc/modprobe.d/blacklist.conf`:
`blacklist ppdev`
`blacklist parport_pc`
`blacklist parport`
`blacklist lp`
Restart PC at this point
4. Unload the remaining modules manually using `rmmod` (if any)
5. Follow remaining instructions given in the tutorial
6. Load custom module (`sudo insmod parlepport.ko`)

Hope this helps...

Courtesy: Redmond who has posted one of the earlier comment

[Log in](#) or [register](#) to post comments

in reply to above

Submitted by aditya84 on Mon, 2010-10-11 11:20



Awesome tutorial, thanks Xavier
I think saurabhg84 means "comment" in points 1 & 2.
I think just blacklisting might also work, but in the right order.
Anyway, will try it out and get back.... \m/

[Log in](#) or [register](#) to post comments

Great!

Submitted by nagabhushana on Thu, 2010-10-14 13:40



I am grateful to you for this excellent tutorial.

[Log in](#) or [register](#) to post comments

Mounting the memory device

Submitted by demohis on Thu, 2011-04-21 11:55



I don't understand the step between the command `insmod memory.ko` and the command `chmod 666 /dev/memory`. I don't find the device memory in the folder `/dev/`. What do I wrong?

[Log in](#) or [register](#) to post comments

mknod

Submitted by shr10 on Thu, 2011-08-18 08:35



try to create the device manually:
`mknod --mode=0666 /dev/memory c 60 0`
description:
`mknod` - the command name to create device file
`--mode=0666` - (optional) save the `chmod` command
`/dev/memory` - device file name (to use several devices add minor version to name; i.e: `/dev/memory0`)
`c` - stand for char device
`60` - major version (u actually should check the `register_chrdev` return value for actual major number, if the 60 was in use u will get other major number)
`0` - minor version (if u use several devices , increment this for each)

I don't know why it wasn't mentioned in the article, maybe in old versions of kernel the device file had been created by `register_chrdev`?

[Log in](#) or [register](#) to post comments

Great work

Submitted by sb_78 on Tue, 2011-06-14 10:42



Nicely written.
Great tutorial exactly what i needed.
Thanks a million.

[Log in](#) or [register](#) to post comments

Great job.

Submitted by tkman on Tue, 2011-08-02 21:36



Very good introduction.

Now to create a fantastic device driver everyone wants. ;)

[Log in](#) or [register](#) to post comments

memory256 (more than 1 char) Debian 2.6.26-2 on hppa

Submitted by kevenm on Thu, 2011-11-17 19:13



Im working on an rp2470 A500 hppa box. Loaded Debian 2.6.26-2.
Loved this article and had to get started.
I have and am now reading Linux Device Drivers 2nd ed (Linux 2.4) to
learn other driver concepts, and callable routines.

first make attempt failed.
I had to get kernel sources,
make oldconfig
make prepare
then edit Makefile removing -fno-strict-overflow
(my gcc version is 4.3.2-1.1)
then had to make the kernel: make

THEN the only error I had was something about
.../linux-source-2.6.26/pwd/ no Makefile found

The article on page 2 as I saw listed the make command as
\$ make -C /usr/src/kernel-source-2.6.8 M=pwd modules
I changed to
\$ make -C /usr/src/kernel-source-2.6.8 M=\$(pwd) modules
and nothing.c built, loaded, and worked!

I worked memory.c into memory256.c to handle 256 char storage.
Too large to post here, but you can request it from me,
Ranger1@3kranger.com
Keven

[Log in](#) or [register](#) to post comments

help need

Submitted by kuru90 on Thu, 2012-01-26 05:54



sir, i want to develop a keyboard interface, which should be platform independent...that is, it should take calls from linux/windows and map the corresponding keys and display...please help to do this....

functions are,

```
int Key_Init()
```

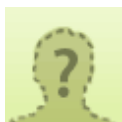
```
int Key_Register(KEYCALLBACK fnCB)
```

```
int Key_Poll(sKey_t *pKey)
```

```
void Key_Capture_Event(DWORD charType, DWORD charValue)
```

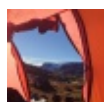
[Log in](#) or [register](#) to post comments

Author information



Biography

Xavier Calbet (xcalbet AT gmail DOT com) is a long time free software user who started using Linux distributions in the ancient times when Slackware had to be installed using tens of floppy disks. He is currently working on his two pet projects: a meteorological field and satellite image display system, SAPO, and the best available free numerical computer language to date, PDL (Perl Data Language).



Add a comment...

☒ Also post on Facebook

Posting as Martin Hoffmann ([Not you?](#))

[Comment](#)



Rangaswamy Gowda · [Follow](#) · Embedded Firmware Developer at Ezetap Mobile solution Pvt Ltd

thanks for make our way easy.

[Reply](#) · [Like](#) · [Follow Post](#) · 13 January 2012 at 09:05



Sumein Momin · SSE at TPVision, Bangalore

Nice Atricle...When we will et the Next version or updated version? Please Include drivers for UART USB PCI SPI etc...

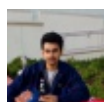
[Reply](#) · [Like](#) · [Follow Post](#) · 13 February 2012 at 10:27



Omveer Chaudhary · Jai Narayan Vyas University, Jodhpur

Really nice and helpful artical.
Thanks

[Reply](#) · [Like](#) · [Follow Post](#) · 25 March 2012 at 05:49



Kaushiik Baskaran · [Follow](#)

Amazingly portrayed! The device driver development procedure can be quickly grasped due to the eloquent explanation of the author. Thank you very much.

[Reply](#) · [Like](#) · [Follow Post](#) · 5 April 2012 at 18:32



Mmk Fx

Thanks a lot for the tutorial..it worked like a gem...

[Reply](#) · [Like](#) · [Follow Post](#) · 19 April 2012 at 11:01



Malin Nisha Shaik Chandini · [Follow](#) · Software Engineer at Sasken Communication Technologies · 108 subscribers

This is really best explanation....

I want to know about Codec driver. Could you please guide me regarding codec driver.

[Reply](#) · [Like](#) · [Follow Post](#) · 28 April 2012 at 05:07



Rupali Garg · Mdu

amazing.....but I want dfd for my project.

[Reply](#) · [Like](#) · [Follow Post](#) · 28 April 2012 at 10:04



Brijendra Sharma · GUEST FACULTY LECTURER at Guest Faculty Lecturer for Post graduate Student

This is really best explanation....

I want to know about mobile driver. Could you please guide me regarding mobile driver.

[Reply](#) · [Like](#) · [Follow Post](#) · 17 May 2012 at 06:02



Nousilal Badavath · [Follow](#) · Engineer, Staff I - Software Systems at Broadcom

nice article.....good work.Thanks.

[Reply](#) · [Like](#) · [Follow Post](#) · 17 May 2012 at 04:47



Chết Mát Ấc Ấc · Làm Cu li at Công Ty CP Viễn Thông Tecapro

thank you for your information.

[Reply](#) · [Like](#) · [Follow Post](#) · 30 May 2012 at 05:36



Basile Mbali · Université de yaoundé 1

It's really best explanation...

Thanks for your article.

[Reply](#) · [Like](#) · [Follow Post](#) · 30 June 2012 at 20:53



Nitin Gharia · Software Engineer at GlobalLogic

Really Good Work. Thanks!

[Reply](#) · [Like](#) · [Follow Post](#) · 3 July 2012 at 13:06



Nerds On Call

Thanks for the excellent tutorial. However I am stuck in the position where by I must use Kernel 2.4 for the module I am working on. The driver it self has been mostly developed and I am just debugging some glitches.

I have the problem that in order to compile the module to work on the target embedded device I must be in Kernel version 2.4.20-20. Despite my efforts my network card only works in 2.4.20-8SMP and when I compile the drive in this mode it will not run on the embedded target. The solution currently is to reboot after every build into the 2.4.20-8SMP, which is not great.

I will look into fixing the network card problem but I am quit interested in learning how to compile drivers for versions of the OS not currently running (and especially in 2.4). I cannot seem to find much on the net so if anyone knows where to look or can help it would be much appreciated.
www.callnerds.com/portland

[Reply](#) · [Like](#) · [Follow Post](#) · 16 August 2012 at 00:12



John Calcote · Senior Software Engineer at Fusion-io, Inc.

Excellent introduction to a topic that's rather hard to get into. Thanks!

[Reply](#) · [Like](#) · [Follow Post](#) · 21 August 2012 at 17:28



Joseph Russell

Nice job! Well written.

[Reply](#) · [Like](#) · [Follow Post](#) · 3 November 2012 at 22:28



Vuppalapati Srinivasa Raju · Works at Anits engg college

Great Tutorial for beginners, Thanks a lot.

[Reply](#) · [Like](#) · [Follow Post](#) · 9 November 2012 at 00:26



Vuppalapati Srinivasa Raju · Works at Anits engg college



Great Tutorial for Beginners. thanks a lot.

[Reply](#) · [Like](#) · [Follow Post](#) · 9 November 2012 at 00:25



Rakesh Venkatesh · RVCE, Bangalore

Awesome tutorial..this helped me a lot in developing char drivers for my lab project..

There is one slight mistake at the Table 8 of memory driver. The first occurrence of "Close device" should be replaced by "Read device".

[Reply](#) · [Like](#) · [Follow Post](#) · 15 November 2012 at 01:35



Varsha Dhage · University of Pune

Great tutorial, every step for DD has been compiled nicely with the code and the explanation... Very helpful to learn as well as revise the steps...

[Reply](#) · [Like](#) · [Follow Post](#) · 3 December 2012 at 14:49

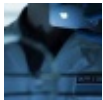


Sunitha Pn · Jeevan Jyothi Convent School

Great and easy to understand!

Thank you very much!

[Reply](#) · [1](#) · [Like](#) · [Follow Post](#) · 9 January 2013 at 11:48



Abby Gautam · St. Joseph School

I am not a programmer but I have this C language subject this session and have to prepare for it. What all topics should be covered in it?

And has anyone studied from this course <http://www.wiziq.com/course/2118-learn-how-to-program-in-c-language> of C tutorial online? or tell me any other guidance... would really appreciate help.

[Reply](#) · [Like](#) · [Follow Post](#) · 18 January 2013 at 15:51

Facebook social plugin

MOST FORWARDED

Anybody up to writing good directory software?

Tue, 2007-02-20 11:17 -- David Jonathan

Since the very beginning, directories (of any kind) have had a very central role in the internet. (I have recently grown fond of Free Web Directory. Even Slashdot can be considered a directory: a collection of great news and invaluable user-generated comments. As far as software is concerned, doing a quick search on Google about software directories will return the free (as in freedom) software directories like [Savannah](#), [SourceForge](#), [Freshmeat](#) and so on, followed by shareware and freeware sites such as [FileBuzz](#), PCWin Download Center and Freeware Downloads (great if you're looking for shareware and freeware, but definitely less comprehensive than their free-as-in-freedom counterparts).

Interview with Dave Mohyla, of DTIDATA

Wed, 2007-12-05 05:17 -- Tony Mobily

Dave Mohyla is the president and founder of [dtidata.com](#), a hard drive recovery facility based in Tampa, Florida.

TM: Where are you based? What does your company do?

DTI Data recovery is based in South Pasadena, Florida which is a suburb of Tampa. We have been here for over 10 years. We operate a bio-metrically secured class 100 clean room where we perform [hard drive recovery](#) on all types of hard disks, from laptop hard drives to multi drive RAID systems.

Interview with Mark Shuttleworth

Mon, 2006-02-20 11:17 -- Tony Mobily

Mark Shuttleworth is the founder of Thawte, the first Certification Authority to sell public SSL certificates. After selling Thawte to Verisign, Mark moved on to training as an astronaut in Russia and visiting space. Once he got back he founded Ubuntu, the leading GNU/Linux distribution. He agreed on releasing a quick interview to Free Software Magazine.

MOST EMAILED

Free Open Document label templates

Tue, 2006-08-22 18:41 -- Solveig Haugland

If you've ever spent hours at work doing mailings, cursed your printer for printing outside the lines on your labels, or moaned "There has got to be a better way to do this," here's the solution you've been looking for. Working smarter, not harder! Worldlabel.com, a manufacture of [labels](#) offers Open Office / Libre Office [labels templates](#) for downloading in ODF format which will save you time, effort, and (if you want) make really cool-looking labels

Creating a user-centric site in Drupal

Fri, 2008-10-24 15:31 -- Tony Mobily

A little while ago, while talking in the #drupal mailing list, I showed my latest creation to one of the core developers there. His reaction was "Wow, I am always surprised what people use Drupal for". His surprise is somehow justified: I did create a site for a bunch of [entertainers in Perth](#), a company set to use Drupal to take over the world with [Entertainers.Biz](#).

Update: since writing this article, I have updated the system so that the whole booking process happens online. I will update the article accordingly!

So, why, why do people and companies develop free software?

Wed, 2007-11-07 14:01 -- Tony Mobily

More and more people are discovering free software. Many people only do so after weeks, or even months, of using it. I wonder, for example, how many Firefox users actually know *how free* Firefox really is—many of them realise that you can get it for free, but find it hard to believe that anybody can modify it and even redistribute it legally.

When the discovery is made, the first instinct is to ask: why do they do it? Programming is hard work. Even though most (if not all) programmers are driven by their higher-than-normal IQs and their amazing passion for solving problems, it's still hard to understand why so many of them would donate so much of their time to creating something that they can't really show off to anybody but their colleagues or geek friends.

Sure, anybody can buy laptops, and *just program*. No need to get a full-on lab or spend thousands of dollars in equipment. But... is that the full story?



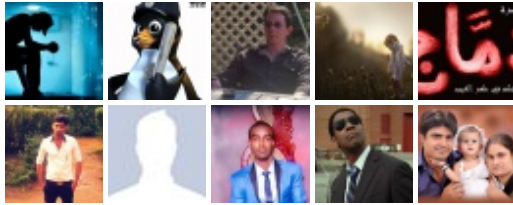
Hosted by [CariNet: Cloud computing](#) is a reality.



Free Software Magazine

Like

1,619 people like Free Softw are Magazine.



Facebook social plugin

Free Software Mag...



Folgen

+1

+ 2.891



<http://www.freesoftwaremagazine.com>

Copyright © 2011 Ryan F. Carberry CC: BY-NC-SA

POPULARCONTENT

Today's:

- [Issues](#)
- [About Free Software Magazine](#)
- [Web site blocking techniques](#)
- [We are looking for tutorial-style articles!](#)
- [The XWindow innovation: welcome to the new Xorg](#)
- [Free Software Magazine: articles on free software](#)
- [Linux performance: is Linux becoming just too slow and bloated?](#)


All time:

- [Writing device drivers in Linux: A brief tutorial](#)
- [Managing your iPod without iTunes](#)
- [Changing the Ubuntu look](#)

- [GRUB tips and tricks](#)
- [Managing users in Ubuntu](#)
- [Printing with Ubuntu](#)



Industriebau
Gewerbebau
Verwaltungsbau



Ihr Partner für
modernen
Hallenbau

althoff ■

INDUSTRIE- UND VERWALTUNGSBAU

*Wir
machen
Projekte*

g+1 x

USER LOGIN

Username *

Password *

- [Create new account](#)
- [Request new password](#)

Log in

141	A First - KDE and the Outreach Program for Women
4	Open Speech Initiative announced
140	NVIDIA to partially open up their GPU specification
28	Has the NSA "poisoned the well" for responsible disclosure?
138	WPA2 is vulnerable (hole196)
142	City of Munich: "Migration to sustainable desktop completed successfully"
5	Using Solr With TYPO3 On Debian Wheezy
143	Skype with care – Microsoft is reading everything you write

Get this widget »