

Kernel Hacking

Dominik Bacher

UnFUG.org
Unix Friends and User Group

6. Mai 2010

Module

Module

Module laden

Hello World

Makefile

Metadata

load/unload Module

Coding style

memory

timer

ModuleParameter

DefFs

ProcFs

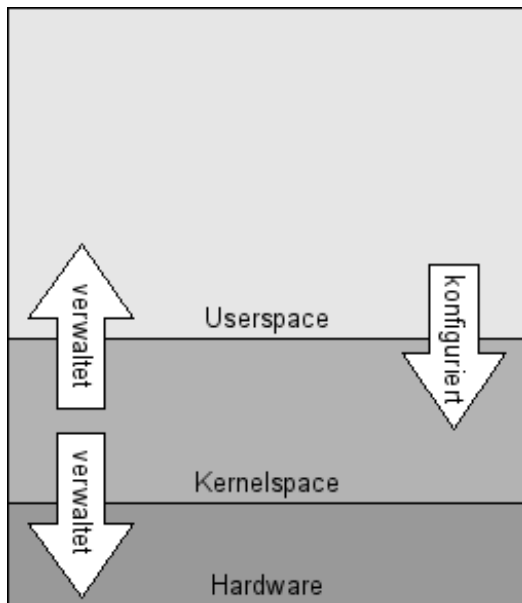
SysFs

Interrupts

Hardware

Hotplug/USB

Kernelspace



Module

Module

Module laden

Hello World

Makefile

Metadata

load/unload Module

Coding style

memory

timer

ModuleParameter

DefFs

ProcFs

SysFs

Interrupts

Hardware

Hotplug/USB

Was ist ein Kernel Module

- Ein teil Code der in den laufenden Kernel geladen werden kann.
- CONFIG_MODULES=y
- installiert in /lib/modules/
- lsmod
- /proc/modules
- /sys/module/

Module

Module

Module laden

Hello World

Makefile

Metadata

load/unload Module

Coding style

memory

timer

ModuleParameter

DefFs

ProcFs

SysFs

Interrupts

Hardware

Hotplug/USB

Module

Module

Module laden

Hello World

Makefile

Metadata

load/unload Module

Coding style

memory

timer

ModuleParameter

DefFs

ProcFs

SysFs

Interrupts

Hardware

Hotplug/USB

Module laden

`modprobe module_name` - vorhandenes modul laden

`depmode` - dependencies auflösen

`insmod` - module zum kernel hinzufügen

`rmmod` - module entfernen

Hello World

```
1 #include <linux/init.h>
2 #include <linux/module.h>
3
4 static int hello_init(void)
5 {
6     printk(KERN_INFO "Hello world!\n");
7     return 0;
8 }
9
10 static void hello_exit(void)
11 {
12     printk(KERN_INFO "Goodby world.\n");
13 }
14
15 module_init(hello_init);
16 module_exit(hello_exit);
```

Module

Module

Module laden

Hello World

Makefile

Metadata

load/unload Module

Coding style

memory

timer

ModuleParameter

DefFs

ProcFs

SysFs

Interrupts

Hardware

Hotplug/USB

Hello World

```
1 printk(KERN_INFO "info text\n");
```

- KERN_NOTICE
- KERN_WARNING
- KERN_ERR
- KERN_CRIT
- KERN_ALERT

Makefile

```
1
2 ifneq ($(KERNELRELEASE),)
3 obj-m := helloworld.o
4
5 else
6 KDIR   := /lib/modules/$(shell uname -r)/build)
7 PWD    := $(shell pwd)
8
9 default:
10        $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
11 endif
```

Module

Module

Module laden

Hello World

Makefile

Metadata

load/unload Module

Coding style

memory

timer

ModuleParameter

DefFs

ProcFs

SysFs

Interrupts

Hardware

Hotplug/USB

```
1 MODULE_AUTHOR("Dominik Bacher");  
2 MODULE_LICENSE("GPL");  
3 MODULE_DESCRIPTION("hello world example");  
4 ...
```

Module

Module

Module laden

Hello World

Makefile

Metadata

load/unload Module

Coding style

memory

timer

ModuleParameter

DefFs

ProcFs

SysFs

Interrupts

Hardware

Hotplug/USB

Metadata

- insmod helloworld.ko
- tail /var/log/messages
- rmmod helloworld
- modinfo helloworld.ko

```
1 filename:      helloworld.ko
2 description:  hello world example
3 license:     GPL
4 author:      Dominik Bacher
5 srcversion:  E0457D07AA88D3950EFB499
6 depends:
7 vermagic:    2.6.31-20-generic SMP
8              mod_unload modversions
```

Module

Module

Module laden

Hello World

Makefile

Metadata

load/unload Module

Coding style

memory

timer

ModuleParameter

DefFs

ProcFs

SysFs

Interrupts

Hardware

Hotplug/USB

Coding style

`/usr/src/linux/Documentation/CodingStyle`

First off, I'd suggest printing out a copy of the GNU coding standards, and NOT read it. Burn them, it's a great symbolic gesture.

`checkpatch.pl`

```
/usr/src/linux-source-2.6.31/scripts/checkpatch.pl -file  
helloworld.c
```

`indent`

```
indent -kr -i8 helloworld.c
```

Module

Module

Module laden

Hello World

Makefile

Metadata

load/unload Module

Coding style

memory

timer

ModuleParameter

DefFs

ProcFs

SysFs

Interrupts

Hardware

Hotplug/USB

Coding style

</usr/src/linux/Documentation/CodingStyle>

First off, I'd suggest printing out a copy of the GNU coding standards, and NOT read it. Burn them, it's a great symbolic gesture.

[checkpatch.pl](#)

```
/usr/src/linux-source-2.6.31/scripts/checkpatch.pl -file  
helloworld.c
```

[indent](#)

```
indent -kr -i8 helloworld.c
```

[Module](#)[Module](#)[Module laden](#)[Hello World](#)[Makefile](#)[Metadata](#)[load/unload Module](#)[Coding style](#)[memory](#)[timer](#)[ModuleParameter](#)[DefFs](#)[ProcFs](#)[SysFs](#)[Interrupts](#)[Hardware](#)[Hotplug/USB](#)

Coding style

[/usr/src/linux/Documentation/CodingStyle](#)

First off, I'd suggest printing out a copy of the GNU coding standards, and NOT read it. Burn them, it's a great symbolic gesture.

[checkpatch.pl](#)

```
/usr/src/linux-source-2.6.31/scripts/checkpatch.pl -file  
helloworld.c
```

[indent](#)

```
indent -kr -i8 helloworld.c
```

[Module](#)[Module](#)[Module laden](#)[Hello World](#)[Makefile](#)[Metadata](#)[load/unload Module](#)[Coding style](#)[memory](#)[timer](#)[ModuleParameter](#)[DefFs](#)[ProcFs](#)[SysFs](#)[Interrupts](#)[Hardware](#)[Hotplug/USB](#)

memory

`kamalloc/kfree` reserviert bis zu 128k physikalischer Speicher.

`kcalloc` gleich wie `kmalloc` nur wird der Speicher auf geleert.

`vmalloc/vfree` reserviert virtuellen Speicher. (mehr als 128k aber kein DMA zugriff)

```
void *kmalloc (size_t size, int flags);
```

- GFP_USER
- GFP_KERNEL
- GFP_ATOMIC

schlafen legen von funktionen während memory reserviert wird.

[Module](#)[Module](#)[Module laden](#)[Hello World](#)[Makefile](#)[Metadata](#)[load/unload Module](#)[Coding style](#)[memory](#)[timer](#)[ModuleParameter](#)[DefFs](#)[ProcFs](#)[SysFs](#)[Interrupts](#)[Hardware](#)[Hotplug/USB](#)

memory

`kamalloc/kfree` reserviert bis zu 128k physikalischer Speicher.

`kcalloc` gleich wie `kmalloc` nur wird der Speicher auf geleert.

`vmalloc/vfree` reserviert virtuellen Speicher. (mehr als 128k aber kein DMA zugriff)

```
void *kmalloc (size_t size, int flags);
```

- GFP_USER
- GFP_KERNEL
- GFP_ATOMIC

schlafen legen von funktionen während memory reserviert wird.

[Module](#)[Module](#)[Module laden](#)[Hello World](#)[Makefile](#)[Metadata](#)[load/unload Module](#)[Coding style](#)[memory](#)[timer](#)[ModuleParameter](#)[DefFs](#)[ProcFs](#)[SysFs](#)[Interrupts](#)[Hardware](#)[Hotplug/USB](#)

`jiffies` Timerinterrupts seit Systemstart

`do_gettimeofday` sekunden und mikrosekunden seit 1.1.1970

`current_kernel_time` sekunden und mikrosekunden seit
1.1.1970

`ndelay` verzögerung in Nanosekunden

`udelay` ... Mikrosekunden

`mdelay` ... Millisekunden

Module

Module

Module laden

Hello World

Makefile

Metadata

load/unload Module

Coding style

memory

timer

ModuleParameter

DefFs

ProcFs

SysFs

Interrupts

Hardware

Hotplug/USB

[Module](#)[Module](#)[Module laden](#)[Hello World](#)[Makefile](#)[Metadata](#)[load/unload Module](#)[Coding style](#)[memory](#)[timer](#)[ModuleParameter](#)[DefFs](#)[ProcFs](#)[SysFs](#)[Interrupts](#)[Hardware](#)[Hotplug/USB](#)

ModuleParameter

- `static int parm1 = 0;`
- `module_param(parm1, int, 0644);`
- `insmod test.ko param1=23`
- `cat /sys/module/test/parameters/parm1`

DefFs

```
1 #include <linux/fs.h>
```

```
1 struct file_operations fops = {  
2     .read      = device_read ,  
3     .write     = device_write ,  
4     .open      = device_open ,  
5     .release   = device_release };
```

- copy_to_user
- copy_from_user
- put_user
- get_user

[Module](#)[Module](#)[Module laden](#)[Hello World](#)[Makefile](#)[Metadata](#)[load/unload Module](#)[Coding style](#)[memory](#)[timer](#)[ModuleParameter](#)[DefFs](#)[ProcFs](#)[SysFs](#)[Interrupts](#)[Hardware](#)[Hotplug/USB](#)

```

1 #include <linux/devfs_fs_kernel.h>
2 ...
3 static int __init treiber_init(void)
4 {
5     ...
6     if(register_chrdev(240, "Treibername", &fops)
7         == 0) {
8         if( devfs_mk_cdev( MKDEV(240,0), S_IFCHR |
9             S_IRUGO | S_IWUGO, "unfug%d", 0 )) )
10            {
11                printk( KERN_ERR "Integration ins
12                    Device-Filesystem " "
13                    fehlgeschlagen.\n");
14            }
15        ...
16    }

```

```

1 devfs_remove( "unfug%d",0 );
2 unregister_chrdev(240,"Treibername");

```

Module

Module

Module laden

Hello World

Makefile

Metadata

load/unload Module

Coding style

memory

timer

ModuleParameter

DefFs

ProcFs

SysFs

Interrupts

Hardware

Hotplug/USB

ProcFs

- Vorgesehen um process information an userspace zu übertragen

```
1 int procfile_read(char *buf, char **loc, off_t off
2     ,
3     int buf_len, int *eof, void *data)
4 {
5     return scnprintf(buf, buf_len - 1, "Unfug\n"
6     );
7 }
8
9 int module_init(void)
10 {
11     struct proc_dir_entry *bi_proc =
12         create_poc_entry("unfug", 0444,
13         NULL);
14     bi_proc->read_proc = procfile_read;
15     bi_proc->write_proc = NULL
16     bi_proc->owner = THIS_MODULE;
17     bi_proc->size = 40;
18     return 0;
19 }
```

Module

Module

Module laden

Hello World

Makefile

Metadata

load/unload Module

Coding style

memory

timer

ModuleParameter

DefFs

ProcFs

SysFs

Interrupts

Hardware

Hotplug/USB

- soll devfs procfs irgendwann mal ablösen.
- ordnet sich automatisch in die passende kategorie ein.
- pro parameter eine Datei die gelesen und geschrieben werden kann.

Module

Module

Module laden

Hello World

Makefile

Metadata

load/unload Module

Coding style

memory

timer

ModuleParameter

DefFs

ProcFs

SysFs

Interrupts

Hardware

Hotplug/USB

SysFs anlegen

```
1 static struct device_driver unfug_driver = {  
2     .name = "unfug_driver",  
3     .bus = &platform_bus_type ,  
4 };
```

```
1 struct platform_device unfug_device = {  
2     .name = "unfug",  
3     .id = 0,  
4     .dev = {  
5         .release = unfug_release ,  
6     }  
7 };
```

```
1 static void unfug_release(struct device *dev)  
2 {  
3     complete(&dev_obj_is_free);  
4 }
```

Module

Module

Module laden

Hello World

Makefile

Metadata

load/unload Module

Coding style

memory

timer

ModuleParameter

DefFs

ProcFs

SysFs

Interrupts

Hardware

Hotplug/USB

Parameter registrieren

```
1 static DEVICE_ATTR(touchtime, S_IRUGO|S_IWUGO,  
    show_touchtime, set_touchtime);
```

```
1 static ssize_t write_sys(struct device *dev,  
    struct device_attribute *attr,  
    const char *buf, size_t count)  
2 {  
3     printk("%s\n", buf);  
4     return count;  
5 }  
6
```

```
1 static ssize_t read_sys(struct device *dev,  
    struct device_attribute *attr, char *buf)  
2 {  
3     return sprintf(buf, "hallo welt\n");  
4 }  
5
```

Module

Module

Module laden

Hello World

Makefile

Metadata

load/unload Module

Coding style

memory

timer

ModuleParameter

DefFs

ProcFs

SysFs

Interrupts

Hardware

Hotplug/USB

Interrupts

Module

Module

Module laden

Hello World

Makefile

Metadata

load/unload Module

Coding style

memory

timer

ModuleParameter

DefFs

ProcFs

SysFs

Interrupts

Hardware

Hotplug/USB

```
1 int isr(int irq, void *dev_id, struct pt_regs *  
    regs)  
2 {  
3     printk("interrupt on irq %d\n", irq);  
4     return 0;  
5 }
```

```
1 request_irq(7, isr, SA_INTERRUPT, "unfug", id);
```

```
1 /proc/interrupts
```

```
1 free_irq(id);
```

- `#include <linux/ioport.h>`
- `request_region(0x378, 3, "unfug");`
- `cat /proc/ioports`
- `release_region(0x3878, 3);`

- `inb(), outb(), outw(), outl() ...`
- `outb(0x378, 42);`
- `char i = inb(0x378);`

- `#include <linux/ioport.h>`
- `request_region(0x378, 3, "unfug");`
- `cat /proc/ioports`
- `release_region(0x3878, 3);`

- `inb(), outb(), outw(), outl() ...`
- `outb(0x378, 42);`
- `char i = inb(0x378);`

Hotplug/USB

```
1 usb_register(&usb_driver);
```

```
1 static struct usb_driver unfug_driver = {  
2     .name = "unfug",  
3     .probe = unfug_probe  
4     .disconnect = unfug_disconnect,  
5     .id_table = unfug_devices,  
6 };  
7 \end{lstlisting}
```

```
8 \begin{lstlisting}  
9 MODULE_DEVICE_TABLE(usb, unfug_devices);  
0
```

```
1 static const struct usb_device_id unfug_devices [] =  
2     {  
3     { USB_DEVICE(VENDOR.ID, DEVICE.ID)},  
4     {}  
5 };
```

[Module](#)[Module](#)[Module laden](#)[Hello World](#)[Makefile](#)[Metadata](#)[load/unload Module](#)[Coding style](#)[memory](#)[timer](#)[ModuleParameter](#)[DefFs](#)[ProcFs](#)[SysFs](#)[Interrupts](#)[Hardware](#)[Hotplug/USB](#)

Quellen I

-  Eva-Katharina Kunst.
Linux-Treiber entwickeln.
dpunkt.verlag, 2006.
-  Robert Love.
Linux Kernel Handbuch.
Addison Wesley Verlag, 2005.
-  Linux cross reference.
<http://lxr.linux.no>.
-  Jürgen Quade und Eva-Katharina Kunst.
Gerätetreiber für kernel 2.6 systematisch eingeführt.
<https://ezs.kr.hsnr.de//TreiberBuch/html//index.html>.
-  Benedikt Waldvogel.
Linux kernel-space programming.
www.unfug.org.

Module

Module

Module laden

Hello World

Makefile

Metadata

load/unload Module

Coding style

memory

timer

ModuleParameter

DefFs

ProcFs

SysFs

Interrupts

Hardware

Hotplug/USB