Kernel Object

It's like an object file, but for kernel.

Jules Aubert

Get ready

```
apt install linux-headers-$(uname -r)
```

Might be something else on other distros.

hello.c

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/printk.h>

static __init int hello_init(void)
{
    pr_info("hello: Hello World!\n");
    return 0;
}

static __exit void hello_exit(void)
{
    pr_info("hello: Goodbye.\n");
}

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

Module information

```
MODULE_LICENSE("Beerware"); // Mandatory
MODULE_DESCRIPTION("Hello module"); // Not mandatory
MODULE_AUTHOR("Jules Aubert"); // Not mandatory
```

About the pr_info function

printk

Makefile

```makefile
obj-m := hello.o
KERNELDIR ?= /lib/modules/$(shell uname -r)/build

PWD := $(shell pwd)

modules:
	$(MAKE) -C $(KERNELDIR) M=$(PWD) $@

clean:
	$(RM) *.o *~ core .depend .*.cmd *.ko *.mod.c \
	            *.tmp_versions *.mod *.order *.symvers

.PHONY: modules clean
```

modinfo

```
modinfo ./hello.ko
```

Load the module into the kernel

```
insmod ./hello.ko
```

Unload the module from the kernel

```
rmmod hello.ko
```

Kernel ring buffer

You can use dmesg(1) to get the logs.

```
dmesg -L -T -W
```

The logs appear at runtime, when using the printk function.

Multiple files

How to compile several source files into one kernel object?

The files

kernel_test.c second.c second.h

The Makefile for several files

```
KERNELDIR = /lib/modules/$(shell uname -r)/build
PWD = $(shell pwd)
obj-m := test.o # IMPORTANT: It has to be named after the following variable!
test-objs := kernel_test.o second.o

modules: $(OBJS)
    $(MAKE) -C $(KERNELDIR) M=$(PWD) $@ $^
```

Arguments

How to put arguments when loading your object?

The code 1/2

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/printk.h>
#include <linux/moduleparam.h>

static int age = 42;
static char *login = "login_x";
static int grades[3] = {0};
static int grd_item = 0; // number of item in grades

module_param(age, int, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP);
MODULE_PARM_DESC(age, "Your age");

module_param(login, charp, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP);
MODULE_PARM_DESC(login, "Your login");

module_param_array(grades, int, &grd_item, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP)
MODULE_PARM_DESC(grades, "Your grades");
```

Module param types

https://github.com/torvalds/linux/blob/master/include/linux/moduleparam.h#L120

```
find / -type f -name 'moduleparam.h' 2>/dev/null
```

The code 2/2

```c
static __init int hello_init(void)
{
    int i = 0;

    pr_info("args: Hello World!\n");
    pr_info("args: My login is %s and I am %d years old!\n", login, age);

    pr_info("args: args equals %d\n", grd_item);
    for (; i < grd_item; ++i)
        pr_info("args:\tgrades[%d] equals %d\n", i, grades[i]);

    return 0;
}

static __exit void hello_exit(void)
{
    pr_info("args: Goodbye %s.\n", login);
}
```

The code 3/2

Never forget

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

Loading without the args

If you don't put the args, or don't put at least one, it will take the default value in the code.

Beware with the array, it can't have more item than declared in the code. 3 here.

Loading with args

Use all the args:

```
insmod ./args.ko login="aubert_o" age=29 grades=7,11,19
```

Use only some of them:

```
insmod ./args.ko grades=11,9 age=29
```

The order is not important.

Drivers

A driver is a code inside the kernel registered with a device.

We developped modules, but not exacty drivers, because we were just writing inside the kernel ring buffer.

You can check the file /proc/devices

Node file

Nodes are all type of file (regulars, directories, specials, sockets, pipes. . .) on the filesystem.

You can check them with the first character when long listing them.

```
ls -lh /dev/
ls -lh /dev/input/
```

mknod(1)

You can create a node with mknod. It needs a **major** and a **minor**.

The major and minor are integers handled by the kernel. The major is associated with a driver, and minors are separated files associated with a major.

Major will be given by your driver, the minor creation is up to you when creating the node.

You can delete a special file with rm(1) (or unlink(1)).

```
mknod /dev/epidriver c MAJOR MINOR
```

How to get the major

There are different ways. The easiest is to call register_chrdev(9). It returns the major chosen by the kernel. Then get it with /proc/devices or print it in the kernel logs.

EpiDriver in action

```
sudo insmod ./epidriver.ko
# check the kernel logs or cat /proc/devices to get the major
sudo mknod /dev/epidriver c $MAJOR 0
cat /dev/epidriver
...
^C
dd if=/dev/epidriver of=test.out bs=512 count=1 status=progress
sudo vim /dev/epidriver
    write something
    :w!
# check the kernel logs :)
sudo rmmod epidriver
```

?

Questions ?