Gabriele **Proietti Mattia**

Advanced Operating Systems and Virtualization

[Lab 04] Kernel Modules

DIAG
Department of Computer, Control and Management Engineering "A. Ruberti", Sapienza University of Rome

gpm.name · proiettimattia@diag.uniroma1.it

A.Y. 2020/2021 · v1

LKMs

Loadable Kernel Modules

A **Loadable Kernel Module** (LKM) is a software component which can be added to the memory image of the Kernel while it is already running. For adding a kernel module the kernel does not need to be recompiled.

LKMs are also used to develop new parts of the Kernel that can be then integrated in the final image once stable. They are also used to tailor the start up of a kernel configuration, depending on specific needs

Code Examples are in this public repository

https://github.com/gabrielepmattia/aosv-code-examples

Hello World

Your first kernel module

The boilerplate for starting a new kernel modules requires only a few lines.

module.c

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Author <me@example.com>");
MODULE_DESCRIPTION("My Fancy Module");
module_init(my_module_init);
module_exit(my_module_cleanup);
```

Reference Counters

The Kernel keeps a reference counter for each loaded LKM. If the reference counter is greater than zero, then the module is locked. This means that there are other services in the system which rely on facilities exposed by the module. If not forced, unloading of the module fails. The `lsmod` utility gives also information on who is using the module.

APIs

- `try_module_get()`: try to increment the reference counter
- `module_put()`: decrement the reference counter

The CONFIG_MODULE_UNLOAD symbol is a global macro which allows the kernel to unload modules it can be used to check unloadability.

Module Parameters

Parameters can be passed to modules but they are not like function parameters, they are initialized as global variables declared in source code of the module. For declaring a parameter you can use the following macros defined in `include/linux/module.h`:

- `module_param(variable, type, perm)`
- `module_param_array(name, type, num, perm)`

These macros specify the name of the global variable which "receives" the input, its type, and its permission (when mapped to a pseudofile). Pseudo-files for parameters are located in `/sys`. Initialization is done upon module load.

Module Parameters

```
#define S_IRWXUGO     (S_IRWXU|S_IRWXG|S_IRWXO)
#define S_IALLUGO     (S_ISUID|S_ISGID|S_ISVTX|S_IRWXUGO)
#define S_IRUGO       (S_IRUSR|S_IRGRP|S_IROTH)
#define S_IWUGO       (S_IWUSR|S_IWGRP|S_IWOTH)
#define S_IXUGO       (S_IXUSR|S_IXGRP|S_IXOTH)
```

Permissions

Permission refers to the permission bits of the GNU C library.

- S_IRUSR, S_IREAD Read permission bit for the owner of the file. On many systems this bit is 0400. S_IREAD is an obsolete synonym provided for BSD compatibility.
- S_IWUSR, S_IWRITE Write permission bit for the owner of the file. Usually 0200. S_IWRITE is an obsolete synonym provided for BSD compatibility.
- S_IXUSR, S_IEXEC Execute (for ordinary files) or search (for directories) permission bit for the owner of the file. Usually 0100. S_IEXEC is an obsolete synonym provided for BSD compatibility.
- S_IRWXU This is equivalent to '(S_IRUSR | S_IWUSR | S_IXUSR)'.
- S_IRGRP Read permission bit for the group owner of the file. Usually 040.
- S_IWGRP Write permission bit for the group owner of the file. Usually 020.
- S_IXGRP Execute or search permission bit for the group owner of the file. Usually 010.
- S_IRWXG This is equivalent to '(S_IRGRP | S_IWGRP | S_IXGRP)'.
- S_IROTH Read permission bit for other users. Usually 04.
- S_IWOTH Write permission bit for other users. Usually 02.
- S_IXOTH Execute or search permission bit for other users. Usually 01.
- S_IRWXO This is equivalent to '(S_IROTH | S_IWOTH | S_IXOTH)'.
- S_ISUID This is the set-user-ID on execute bit, usually 04000. See How Change Persona.
- S_ISGID This is the set-group-ID on execute bit, usually 02000. See How Change Persona.

Building

For building a module you need the source of the kernel to be present in the folder pointed by

`/lib/modules/$(uname -r)/build`

That is a symbolic link and points to different locations according to your Linux distribution, in general to the source of the linux kernel.

A typical makefile is the following

```
KDIR = /lib/modules/$(shell uname -r)/build
obj-m += hello.o

all:
	make -C $(KDIR) M=$(PWD) modules
clean:
	make -C $(KDIR) M=$(PWD) clean
```

Loading and Unloading

A module is loaded by the administrator via the shell command **`insmod`**. It can also be used to pass parameters (variable=value).

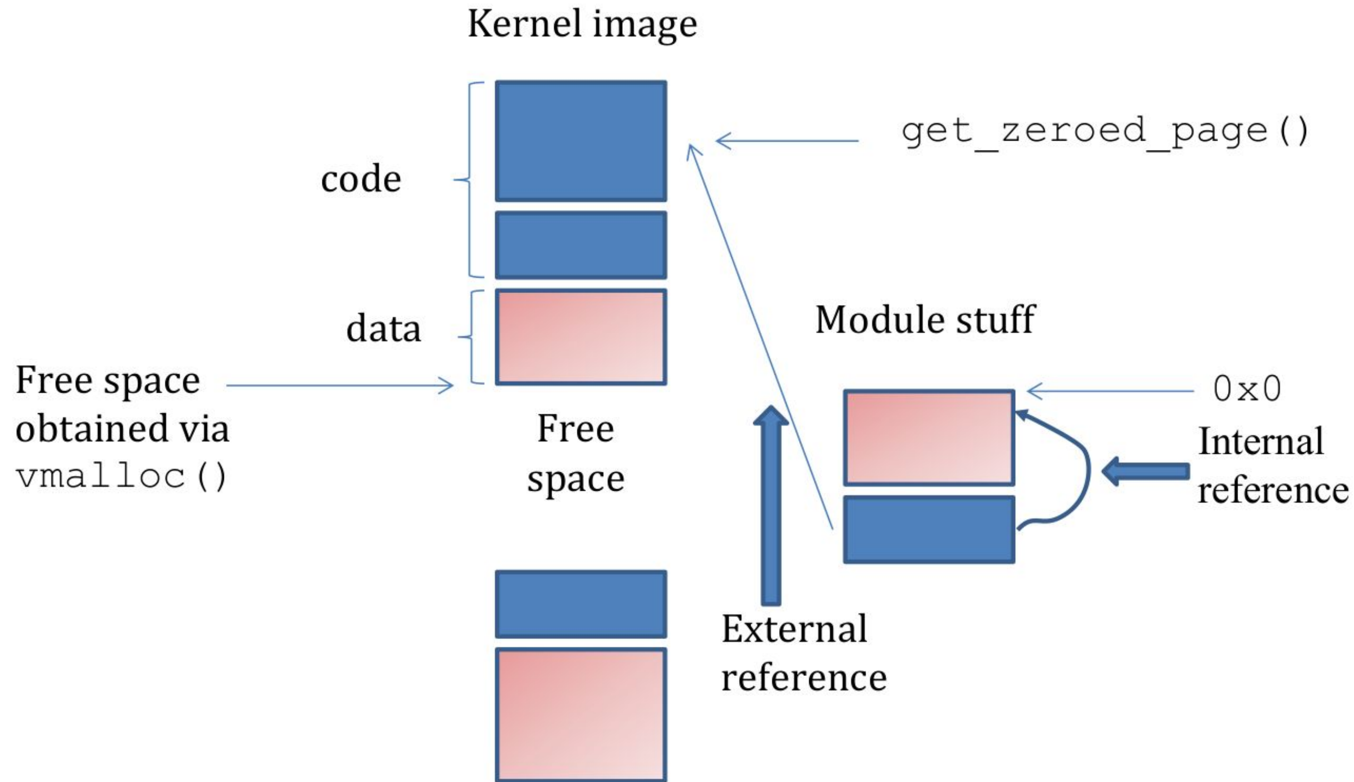
It takes as a parameter the path to the object file generated when compiling the module. A module is unloaded via the shell command **`rmmod`**. We can also use modprobe, which by default looks for the actual module in the directory /lib/modules/$(uname –r).

Steps for loading a module

When loading a module we need memory to load in RAM both code and data structures. We need to know several memory locations to perform a dynamic resolution:

- base address of the module, for internal references
- locations in memory of static Kernel facilities (functions and data)

Loading a Module



Who does the job?

Up to kernel 2.4 most of the job is done is userspace

- a module is a **.o ELF**
- applications reserve memory, resolve the symbols' addresses and load the module in RAM

From kernel 2.6 most of the job is kernel-internal

- a module is a **.ko** ELF
- shell commands are used to trigger the kernel actions for memory allocation, module loading, and address resolution

Module Loading in 2.4

System calls

System calls used for loading a module in kernel 2.4 were

```
#include <linux/module.h>
caddr_t create_module(const char *name, size_t size);
int init_module(const char *name, struct module *image);
int delete_module(const char *name);
```

- `create_module` attempts to create a loadable module entry and reserve the kernel memory that will be needed to hold the module. This system call is only open to the superuser.
- `init_module` loads the **relocated module image** into kernel space and runs the module's init function. The module image begins with a module structure and is followed by code and data as appropriate.
- `delete_module` attempts to remove an unused loadable module entry. If name is NULL, all unused modules marked autoclean will be removed. This system call is only open to the superuser.

Module loading in 2.6

System calls

SYNOPSIS

```c
int init_module(void *module_image, unsigned long len, const char *param_values);

int finit_module(int fd, const char *param_values, int flags);

int delete_module(const char *name, int flags);
```

DESCRIPTION

`init_module()` loads an ELF image into kernel space, performs any necessary symbol relocations, initializes module parameters to values provided by the caller, and then runs the module's init function. This system call requires privilege.

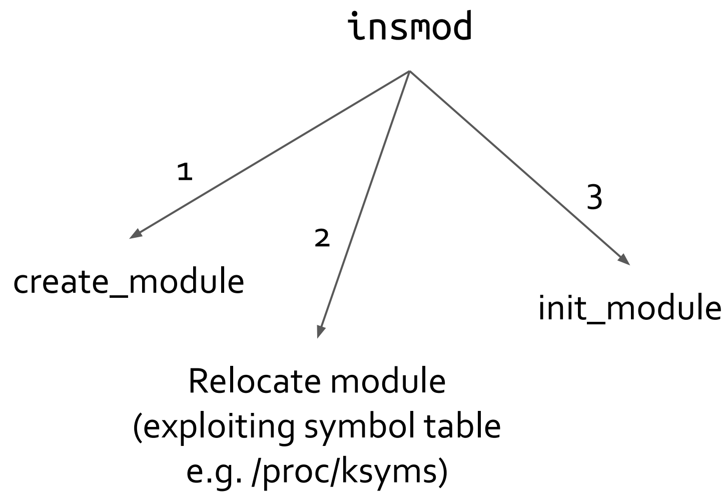
The `module_image` argument points to a buffer containing the binary image to be loaded; len specifies the size of that buffer. The module image should be a valid ELF image, built for the running kernel.

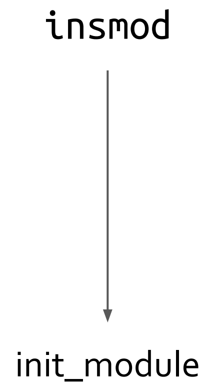
Module loading in 2.6

To make a .ko file, we start with a regular .o file. The **modpost** program creates (from the .o file) a C source file that describes the additional sections that are required for the .ko file

The C file is called .mod file and this .mod file is compiled and linked with the original .o file to make a .ko file

insmod comparison





/sys/module

The `/sys` fileystem is a RAM filesystem which essential contains configurable parameters of the kernel in form of files, for a generic module we have:

- `/sys/module/MODULENAME`: The name of the module that is in the kernel. This module name will always show up if the module is loaded as a dynamic module.

- `/sys/module/MODULENAME/parameters`: This directory contains individual files that are each individual parameters of the module that are able to be changed at runtime.

- `/sys/module/MODULENAME/refcnt`: If the module is able to be unloaded from the kernel, this file will contain the current reference count of the module. Note: If CONFIG_MODULE_UNLOAD kernel configuration value is not enabled, this file will not be present

printk

The printk function can be used as the standard printf, but you can specify the log level.

Table 18.1 **Available Loglevels**

Loglevel	Description
KERN_EMERG	An emergency condition; the system is probably dead.
KERN_ALERT	A problem that requires immediate attention.
KERN_CRIT	A critical condition.
KERN_ERR	An error.
KERN_WARNING	A warning.
KERN_NOTICE	A normal, but perhaps noteworthy, condition.
KERN_INFO	An informational message.
KERN_DEBUG	A debug message—typically superfluous.

Love, Robert. *Linux kernel development*. Pearson Education, 2010.

The output can be seen with the command `dmesg`.

Advanced Operating Systems and Virtualization

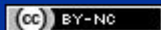
[Lab 04] Kernel Modules

LECTURER

Gabriele **Proietti Mattia**

BASED ON WORK BY

http://www.ce.uniroma2.it/~pellegrini/



gpm.name · proiettimattia@diag.uniroma1.it

DIAG