

# Outlines

- Faire des modules
- Interaction via /sys
- Memory alloc
- Interaction via /dev
- Threading
- Hardware?
- Suicide: bottom/top kernel, irq, workqueues etc

Repo du WS: <https://github.com/UrLab/kernelworkshop>

Pad pour échanger du code: <https://pad.lqdn.fr/p/kernelurlab>

# WARNING



# Level<sup>1</sup>: Hello world

- Deux fonctions needed

- `int init_module(void) // return 0 if ok`
- `void cleanup_module(void)`

- Deux headers needed

- `#include <linux/module.h> // requis pour de la magie`
- `#include <linux/kernel.h> // KERN_INFO`

- `printk`: comme `printf`, mais pour le kernel

- `KERN_WARNING`, macro string pour gérer la gravité de la situation  
`printk(KERN_INFO "..", args)`

- `panic()` ;yolooooo

- Compilation: Makefile

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

```
all:
```

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

- `insmod licorn.ko`

- `dmesg ;swag.`

# Level<sup>1</sup>++

- Use `module_init`, `module_exit` macro (exemple: `module_init(start_licorn_factory)`);
- Use macro:
  - `MODULE_AUTHOR("URLAB LICORN CORP");`
  - `MODULE_DESCRIPTION("The end of your life as you know it");`
  - `MODULE_LICENSE("GPL") // in rms we trust.`
- Use `static __init`, `static __exit` before your init code and exit code
  - **`static int __init`** `start_licorn_factory(void)`

# Userspace interaction

- Kernel space vs userspace
- Moyen de transmission:
  - syscall
  - basé sur syscall: mécanisme de “fichier” (open/read/write/ioctl/close).  
Simplification, voir socket etc
- Explorez /sys (le kernel emploie plein d'objets, hiérarchie etc)
- <http://lxr.free-electrons.com/> !!!! votre nouveau meilleur pote

## Level<sup>2</sup>: Hello licorn %name

- L'api principale c'est via des choses qui s'appellent des kobjects (on va y revenir après)
- C'est une hiérarchie. Elle est exportée dans /sysfs. Votre module peut mettre des dossiers (des kobjects) et des fichiers dedans
- Le module doit enregistrer un nouveau kobject, qui va posséder des fichiers (dans le init du module)

- `struct kobject custom_dir;`
- `kobject_init(&custom_dir, &module_type);`
- `kobject_add(&custom_dir, &THIS_MODULE->mkobj.kobj, "custom");`

- `module_type`? Préciser la manière de read/write les fichiers qui sont dedans

```
struct sysfs_ops module_file_ops = {
    .show = licorn_magic_show,
    .store = licorn_store
};
struct kobj_type module_type = {
    .sysfs_ops = &module_file_ops
};
```

- Fonction de read/write:

```
static ssize_t licorn_magic_show(struct kobject *obj, struct
attribute *attr, char *buf) {}
static ssize_t licorn_store(struct kobject *obj, struct attribute
*attr, const char *buf, size_t count) {}
```

## Level<sup>2</sup>: Hello licorn %name

- Résoudre sur base du attr->name, retourner quelque chose en fonction de ça (rien, un string,...)
- pour déclarer un fichier:
- `sysfs_create_file(&custom_dir, &hello_file_attr);`
- ```
struct attribute name_file_attr = {  
    .name = "name",  
    .mode = 0666  
};
```

→ algo:

```
module_init() {  
    kobject_init(...);  
    kobject_add(...);  
    sysfs_create_file(&custom_dir, &name_file_attr);  
}
```

```
module_cleanup() {  
    sysfs_remove_file(&custom_dir, &name_file_attr);  
    kobject_put(&custom_dir);  
}
```

Idea: par exemple, déclarer deux fichiers, un contenant le nom (qui serait copier dans un string dans le module), un uniquement writable, qui quand il est appelé fait un `printk("Hello licorn %s", name)`.

# Memory management

- `#include <slab.h>`
- Kmalloc (kzalloc), kfree (easy)
  - `void * kmalloc(size_t size, int flags); //flags = GFP_KERNEL`
  - `void kfree(void * ptr); // NULL ok.`
- More efficient way, but lot more complex.
- Ldd3 is now your best friend



# Linked liste

- `#include <linux/list.h>`
- Surprise! C'est un truc à mettre à l'intérieur d'une struct, pas un truc qui englobe une struct. Par exemple:
  - ```
struct love {  
    ...  
    struct list_head more_love;  
};
```
- `struct list_head rainbows; init: INIT_LIST_HEAD(&rainbows);`
- `list_add(love_instance->more_love, &rainbows)`
- `struct love * ptr; list_for_each_entry(ptr, &rainbows, more_love)`
- `list_del(&love_instance->more_love)`
- gaffe au segfault, voir `list_empty`.
- LDD3 at page 313

## Level<sup>3</sup>: A very fat licorn

Algo idea:

- dans init: faire un dossier custom, puis initialiser une liste global (une list\_head)
- faire une fonction add\_love(..) qui kcalloc une nouvelle struct love, copie de la data d'userspace dedans, puis fait un list\_add (nouvelle instance de love, liste globale)
- faire une fonction list\_all\_love qui emploie l'itérateur vu plus haut pour en afficher le contenu
- faire un get\_love qui récupère le premier elem de la liste, met son contenu dans buf puis le list\_del && kfree

Warning:

- faire un list\_empty dans le get\_love sinon segfault
- pas multiprocess safe, voir avec des locks (pas sexy)
- pas oublié de kfree une non empty list dans le clean\_up