INTERNALS SESSIONS 03:

Linux Kernel Modules 101

Abolfazl Kazemi

mahsan.co
info@mahsan.co

Agenda

1. Introducing Loadable Kernel Modules
2. Exporting Functions in LKMs
3. What is Procfs?
4. What are character devices?
5. How to write a simple kernel rootkit?

Kernel Modules Intro

- Pieces of code, that can be loaded and unloaded from kernel on demand.
- Offers an easy way to extend the functionality of the base kernel without having to rebuild or recompile the kernel again
- Most of the drivers are implemented as a Linux kernel modules. When those drivers are not needed, we can unload only that specific driver.
- The kernel modules will have a .ko extension.
- On a normal linux system, the kernel modules will reside inside /lib/modules/<kernel_version>/kernel/ directory

Module Structure

```c
#include <linux/module.h>    // included for all kernel modules
#include <linux/kernel.h>    // included for KERN_INFO
#include <linux/init.h>      // included for __init and __exit macros

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Abolfazl Kazemi");
MODULE_DESCRIPTION("Hello World Module!");

static int __init hello_init(void) {
    printk(KERN_INFO "Hello World!\n");
    return 0;            //Nonzero return means that the module couldn't be loaded.
}

static void __exit hello_cleanup(void) {
    printk(KERN_INFO "Good Bye!\n");
}

module_init(hello_init);
module_exit(hello_cleanup);
```

Module Makefile

```
obj-m += hello.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
```

-C $KDIR

The directory where the kernel source is located. “make” will actually change to the specified directory when executing and will change back when finished.

M=$PWD

Informs kbuild that an external module is being built. The value given to “M” is the absolute path of the directory where the external module (kbuild file) is located.

Module Parameters

module_param(): This macro takes 3 arguments: the *name* of the variable, its *type* and *permissions* for the corresponding file in sysfs

module_param_array(): The parameters are like the previous macro but for getting *the number of parameters* you need to pass a pointer to a count variable as third parameter

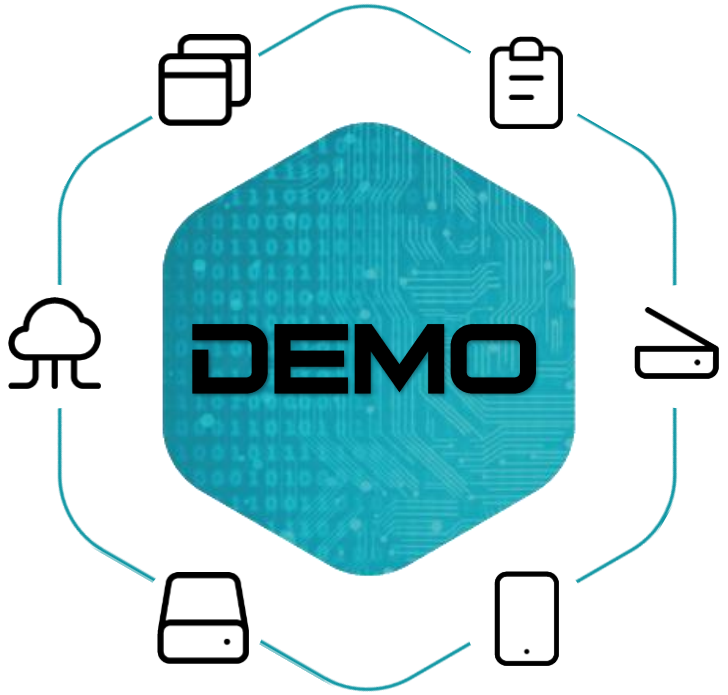
```
static char *hellstr = "CharParam";
static int hellarray[5] = {1,2,3,4,5};
static int arr_argc = 0;

module_param(hellstr, charp, 0);
MODULE_PARM_DESC(hellstr, "A character string");

module_param_array(hellarray, int, &arr_argc, 0);
MODULE_PARM_DESC(hellarray, "An array of integers");
```

Manipulating Kernel Modules

- lsmod
  cat /proc/modules

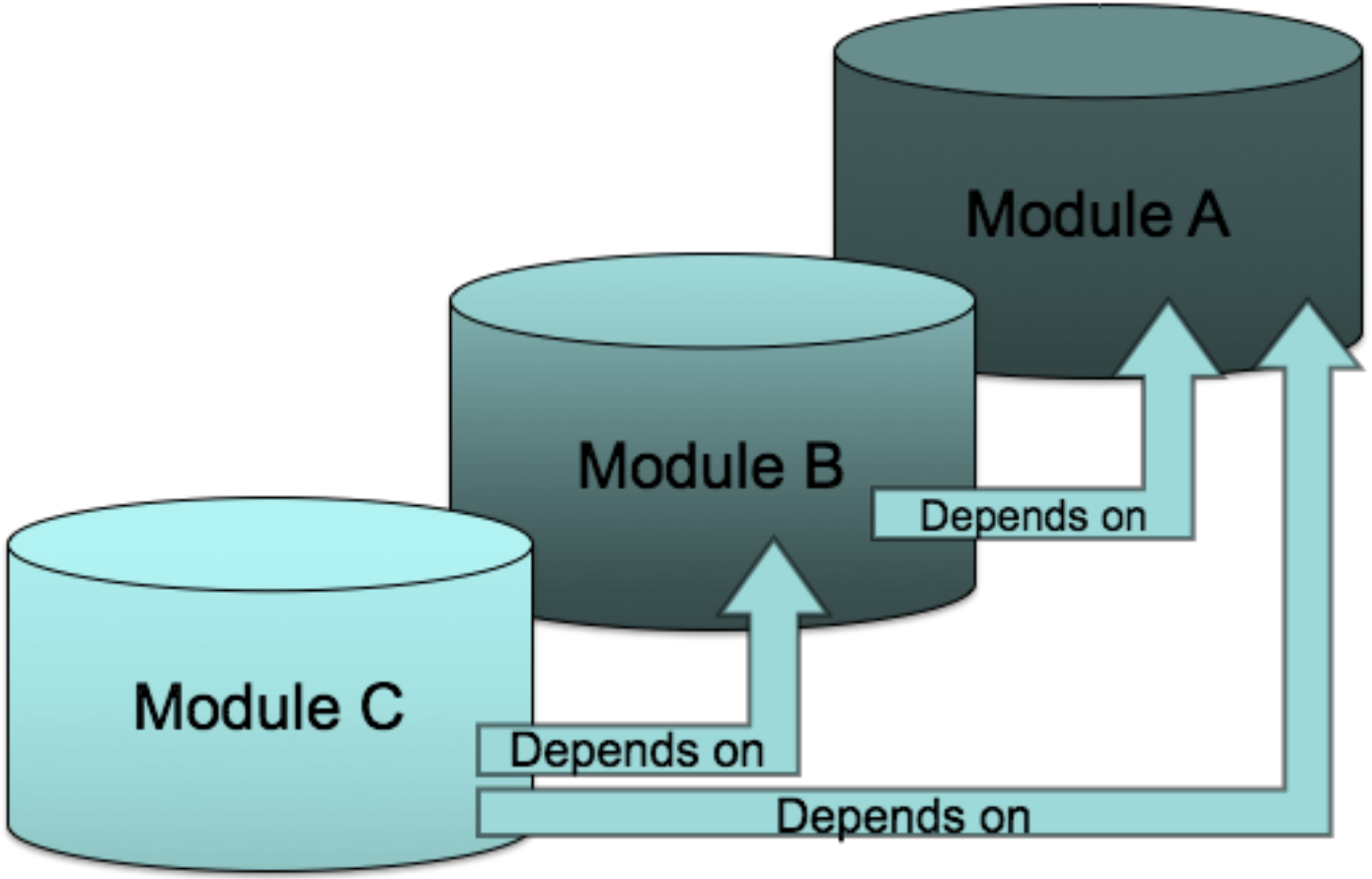
- modinfo
- insmod
- rmmod

- modprobe



Compiling and Running a Simple Module

Module Dependency



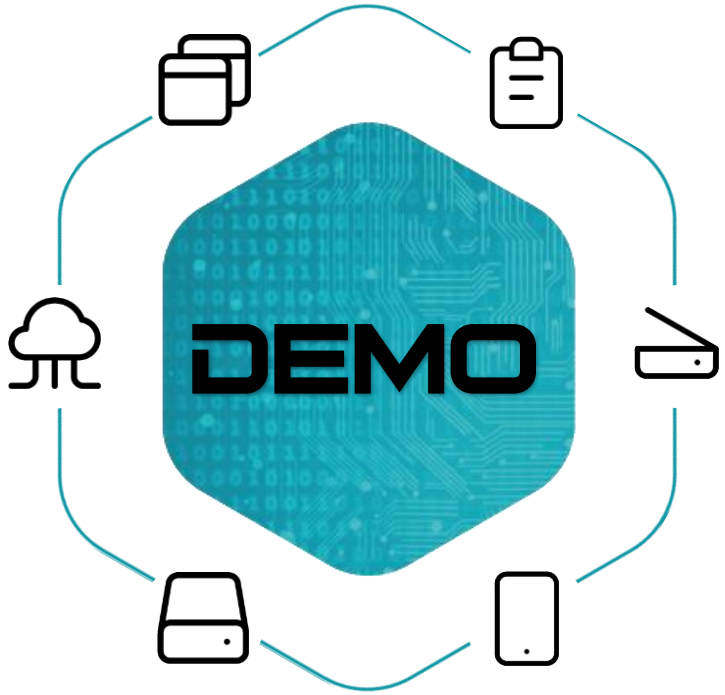
Module Dependency
(export part)

```
int MAH_GLOBAL_VAR = 1337;
EXPORT_SYMBOL(MAH_GLOBAL_VAR);

void print_hello(int num) {
    while (num--) {
        printk(KERN_INFO "Just Saying Hello!!!\n");
    }
}
EXPORT_SYMBOL(print_hello);
```

Module Dependency
(import part)

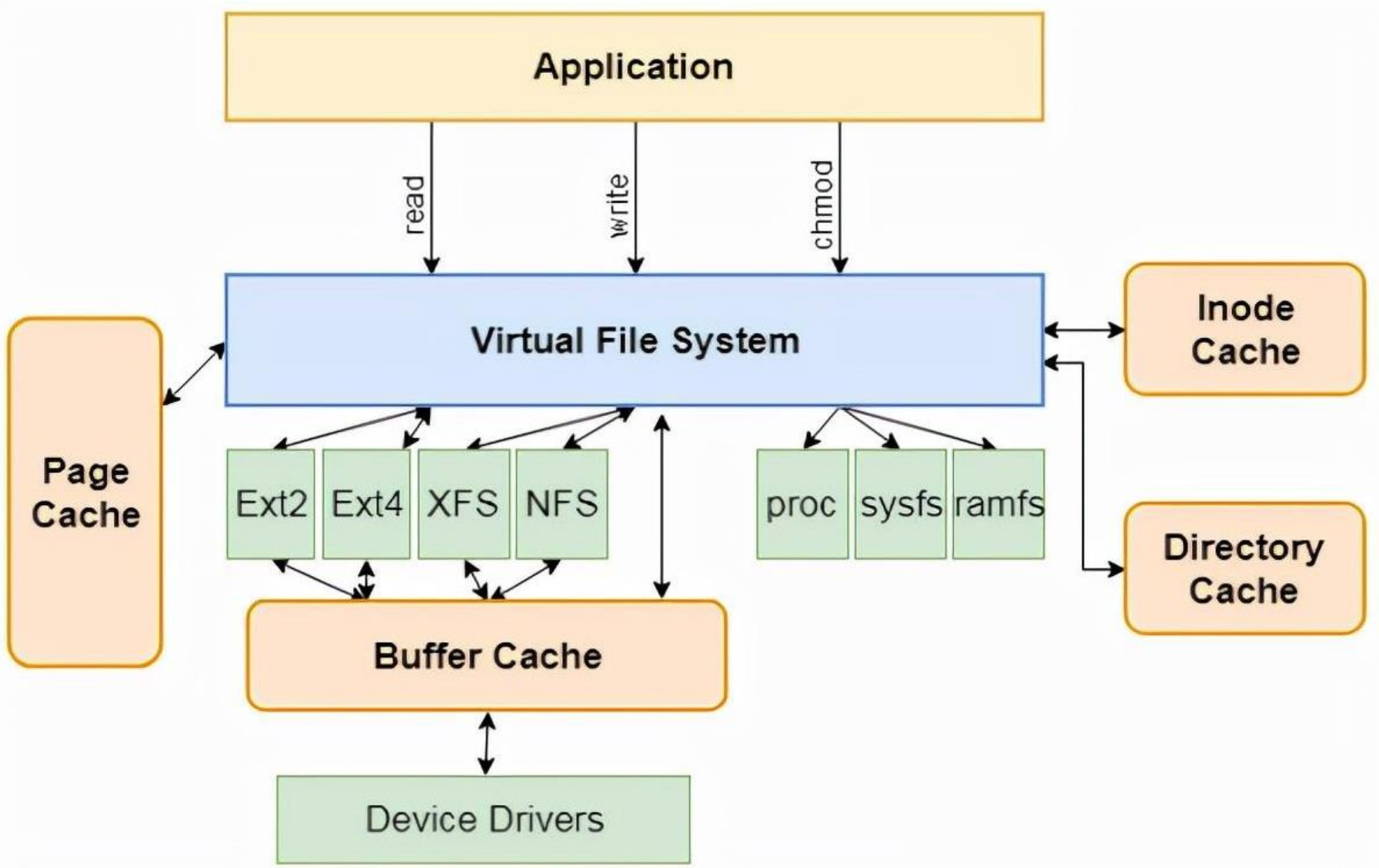
```c
extern void print_hello(int);
extern void add_two_numbers(int, int);
extern int MAH_GLOBAL_VAR;

static int __init im_init(void) {
    printk(KERN_INFO "Hello from Importer Module.\n");
    print_hello(5);
    add_two_numbers(11, 28);
    printk(KERN_INFO "Value of MAH_GLOBAL_VAR %d\n", MAH_GLOBAL_VAR);
    return 0;
}
```

```
user@pc1:~$ lsmod | grep -E "imp|exp|Mod"
Module                  Size  Used by
mod_importer           16384  0
mod_exporter           16384  1 mod_importer
```



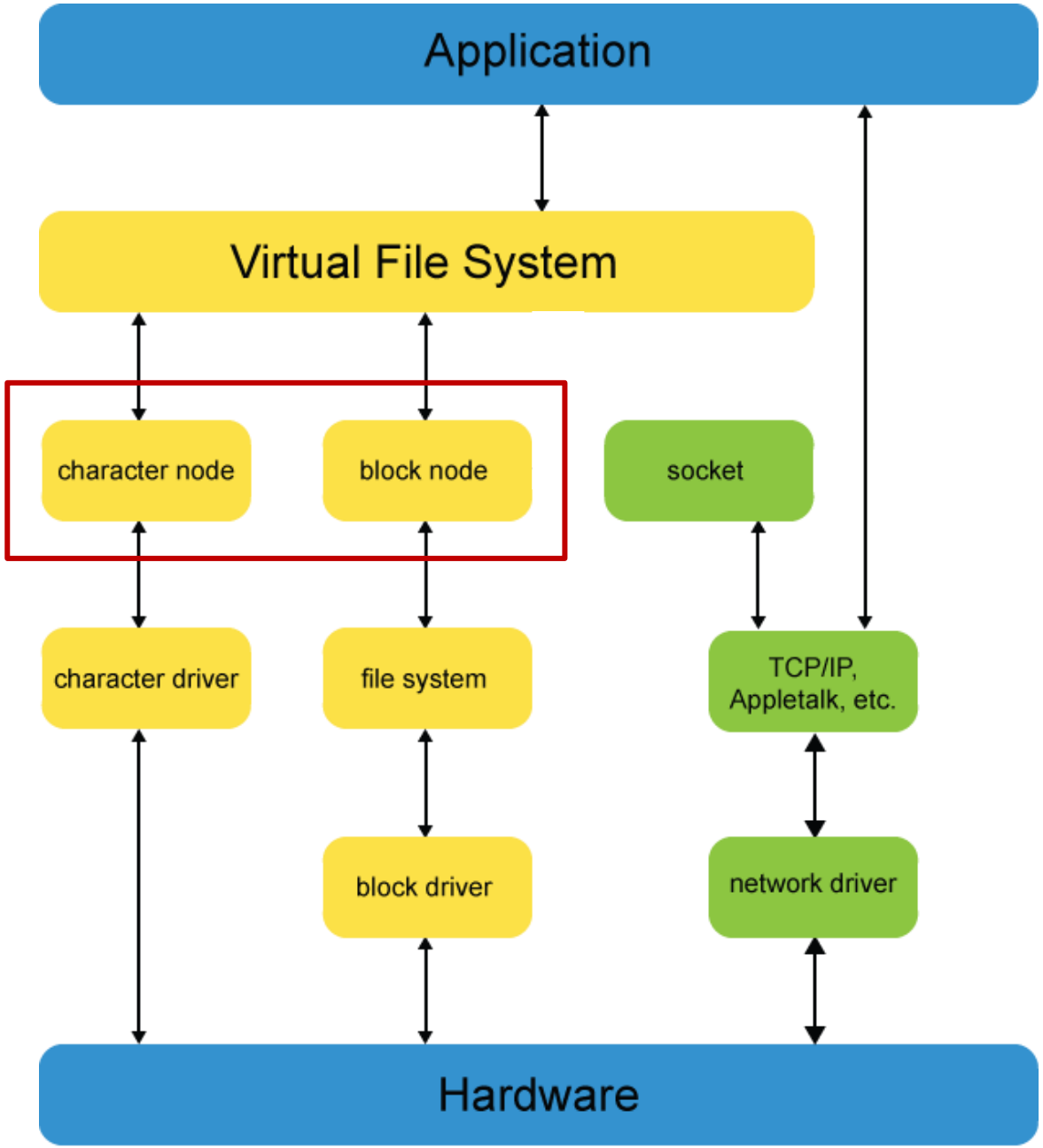
Dependent Modules

Virtual File System



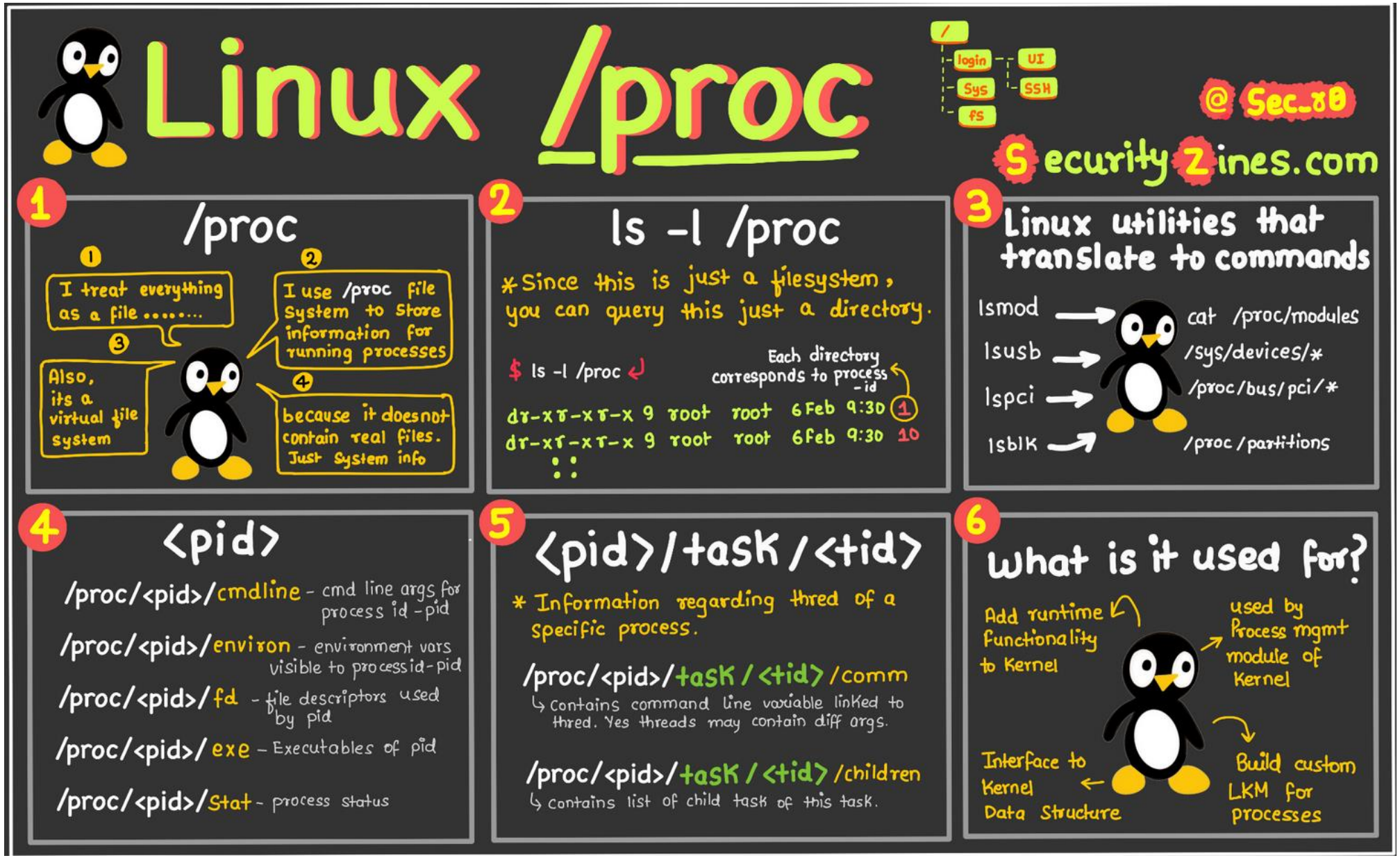
source

Virtual File System



source

procfs



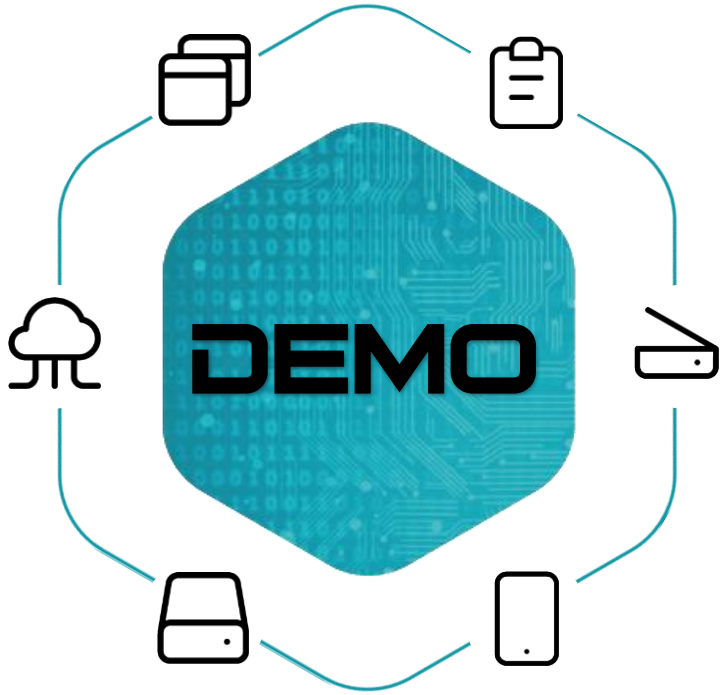
source

Adding a new /proc entry

```
#ifdef HAVE_PROC_OPS /* kernel>=5.6 */
static const struct proc_ops proc_file_fops = {
    .proc_read = procfile_read,
    .proc_write = procfile_write,
};
#else
static const struct file_operations proc_file_fops = {
    .read = procfile_read,
    .write = procfile_write,
};
#endif
```

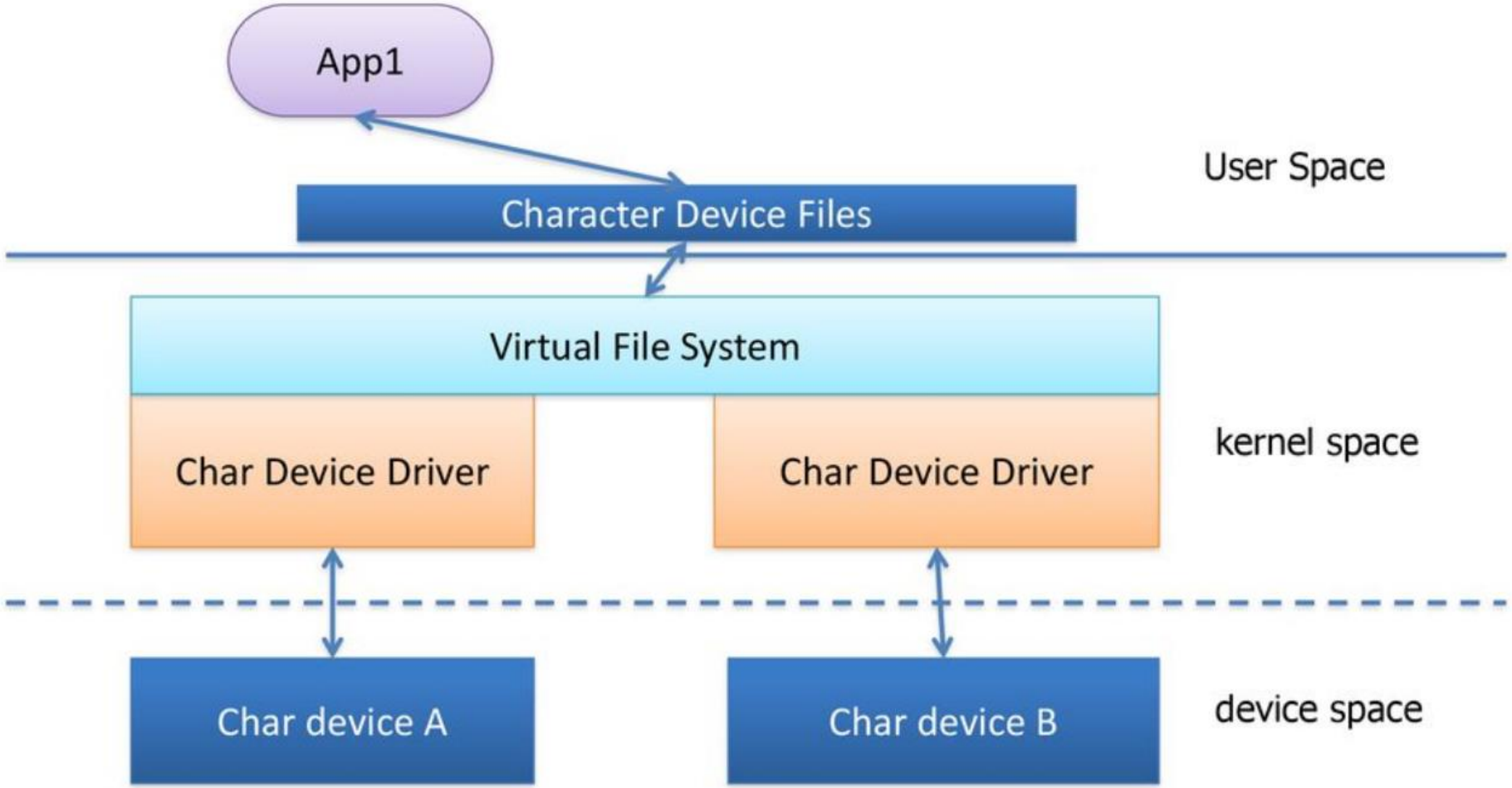
Adding a new /proc entry

```c
static int __init procfs2_init(void) {
    our_proc_file = proc_create(PROCFS_NAME, 0644, NULL, &proc_file_fops);
    if (!our_proc_file) {
        pr_alert("Error:Could not initialize /proc/%s\n", PROCFS_NAME);
        return -ENOMEM;
    }

    pr_info("/proc/%s created\n", PROCFS_NAME);
    return 0;
}

static void __exit procfs2_exit(void)  {
    proc_remove(our_proc_file);
    pr_info("/proc/%s removed\n", PROCFS_NAME);
}

module_init(procfs2_init);
module_exit(procfs2_exit);
```



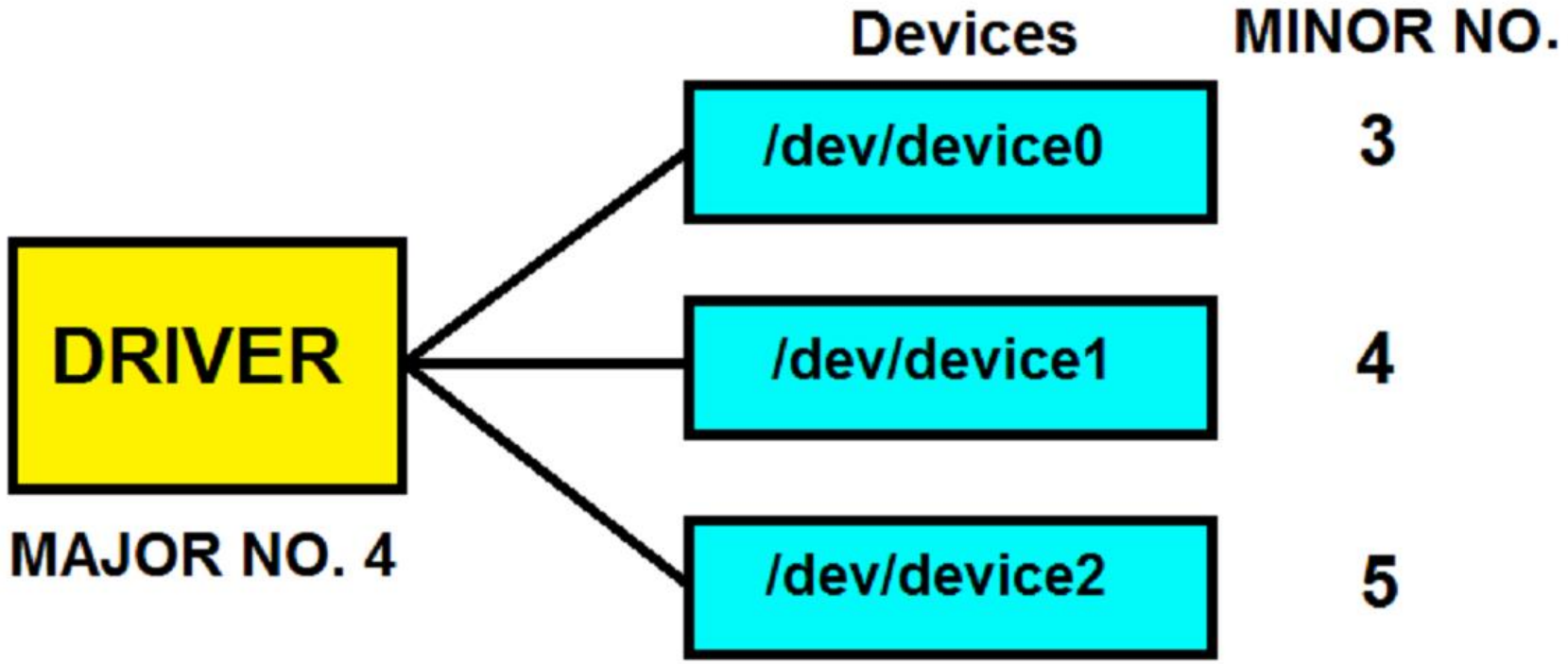
Adding a New Proc Entry

Character Devices



source

Major/Minor Numbers



source

Character Devices Creation

```c
static int mah_cdev_init(void) {
    int err;
    int i;

    err = register_chrdev_region(MKDEV(MY_MAJOR, 0), NUM_MINORS, MODULE_NAME);
    if(err){
        pr_err("MAHSAN: Error creating char device. code: %d\n", err);
        return err;
    }

    for (i = 0; i < NUM_MINORS; i++) {
        cdev_init(&devs[i].cdev, &mah_fops);
        cdev_add(&devs[i].cdev, MKDEV(MY_MAJOR, i), 1);
    }
    pr_info("MAHSAN: Successfully initialized char device.\n");
    return 0;
}
```

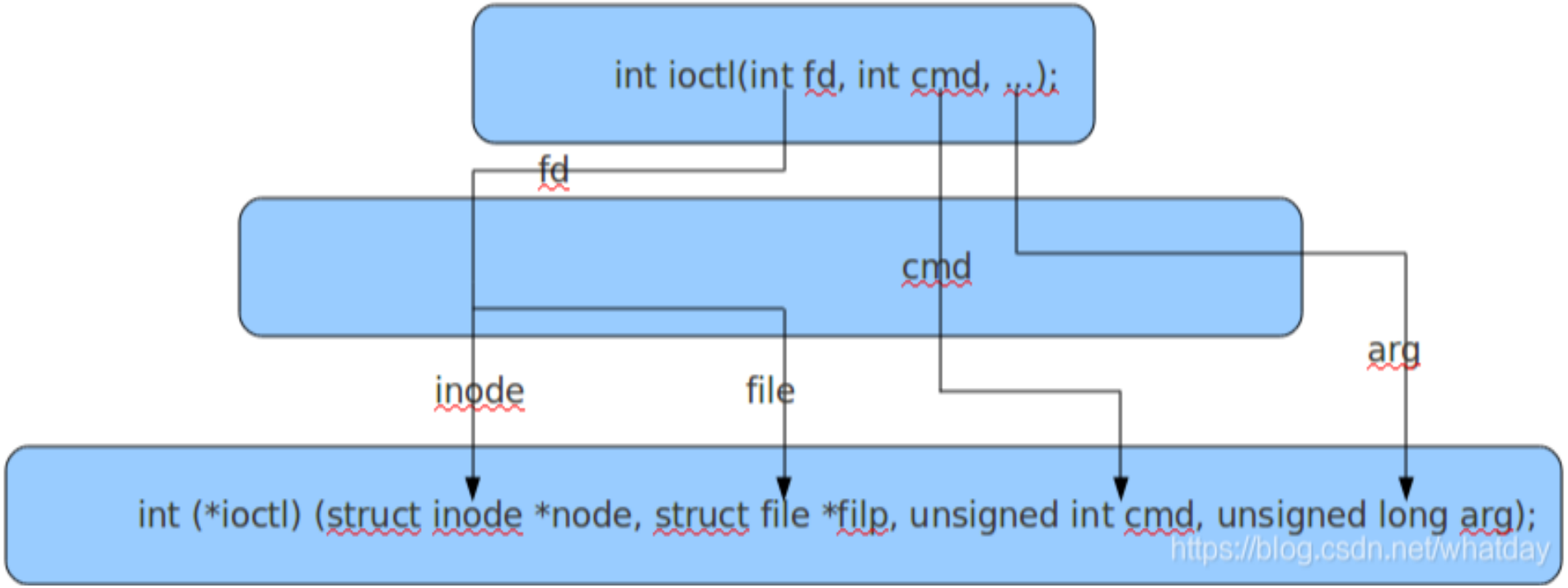
Callback Functions

Driver's Internal Data

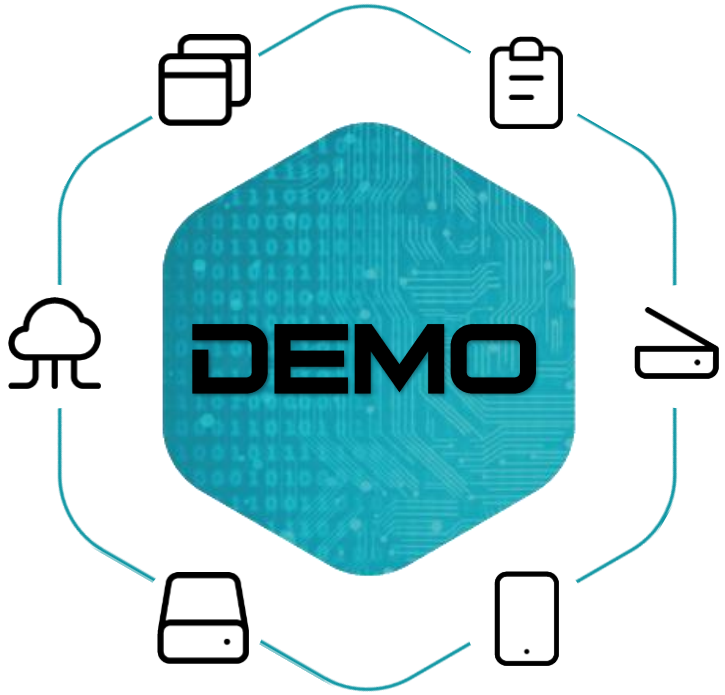
Character Devices Structures

```
struct mah_device_data {
    struct cdev cdev;
    char buffer[BUFSIZ];
    ssize_t size;
    atomic_t is_locked;
    atomic_t is_up;
};

static const struct file_operations mah_fops = {
    .owner = THIS_MODULE,
    .open = mah_cdev_open,
    .read = mah_cdev_read,
    .write = mah_cdev_write,
    .release = mah_cdev_release,
    .unlocked_ioctl = mah_cdev_ioctl
};
```

Character Devices
IOCTL



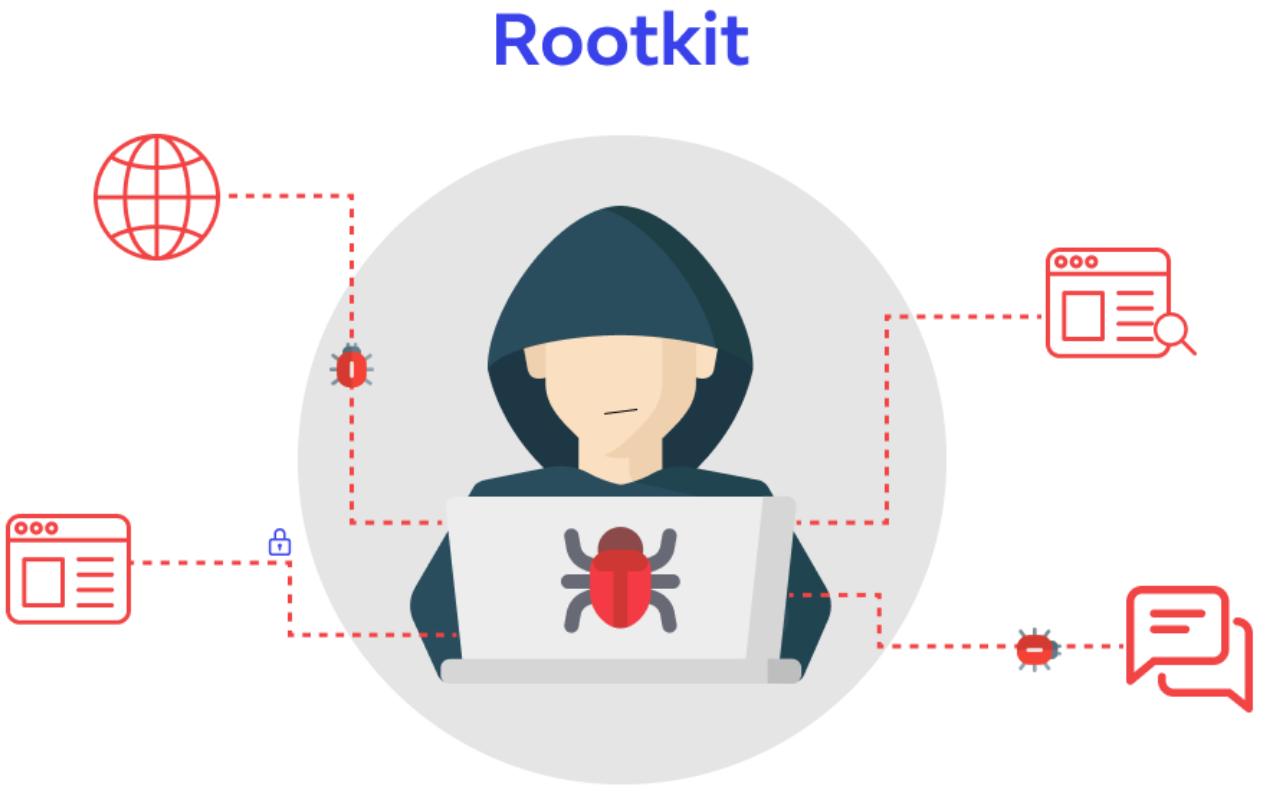

```
#define _IOC(dir, type, nr, size)
```



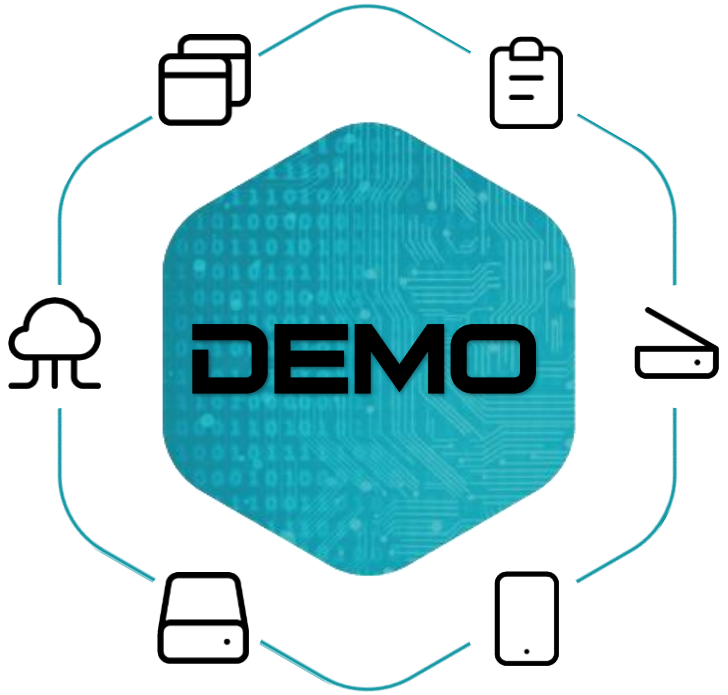
Character Devices

Kernel Rootkits

A rootkit is a clandestine computer program designed to provide continued privileged access to a computer while actively hiding its presence



source



Rooting a Process!!!

Further Readings

- Linux Kernel Teaching
- Linux Kernel Module Programming
- Linux Device Drivers Tutorials


Thanks
Mahsan
Your Support In Digital World