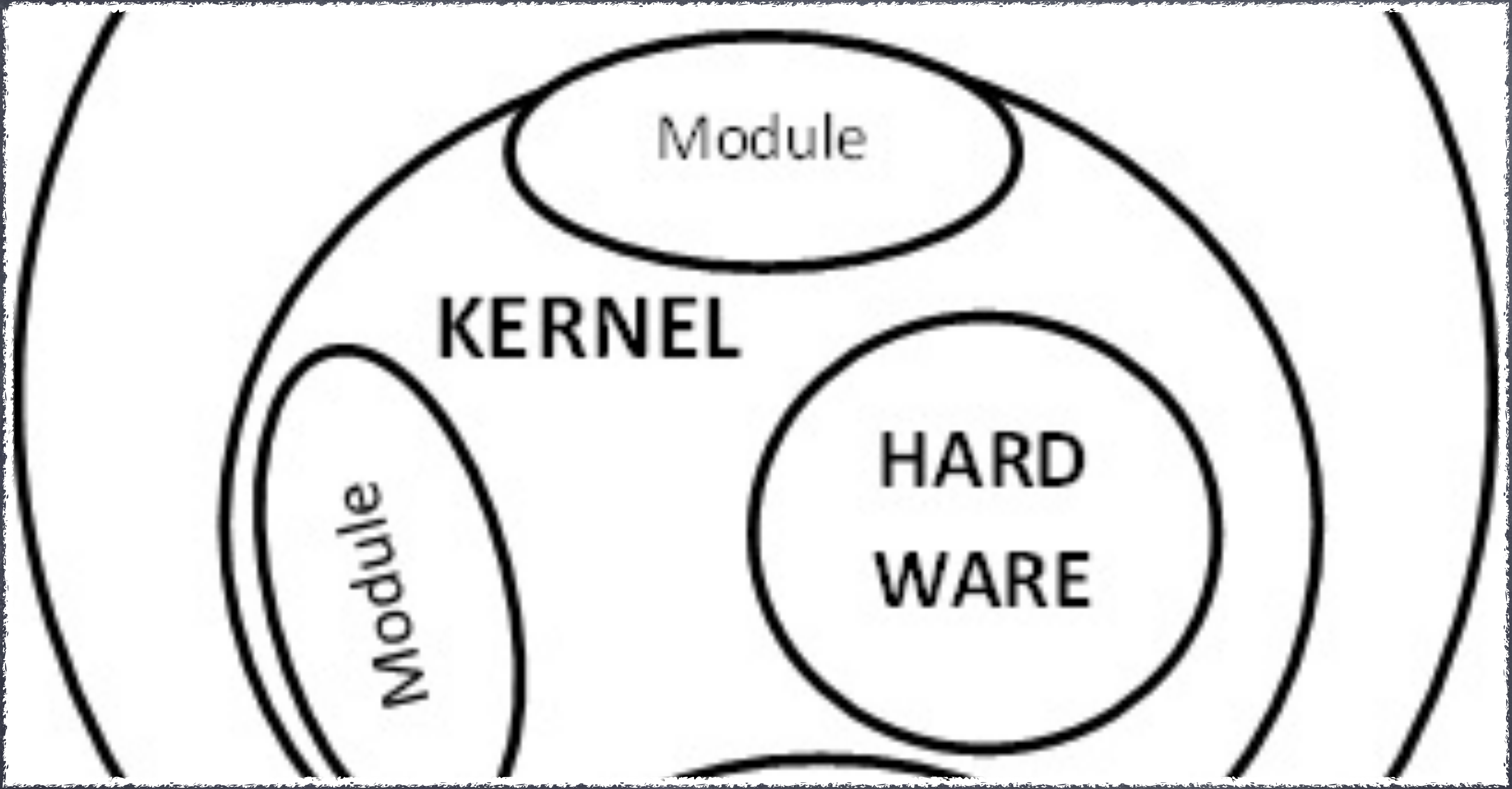


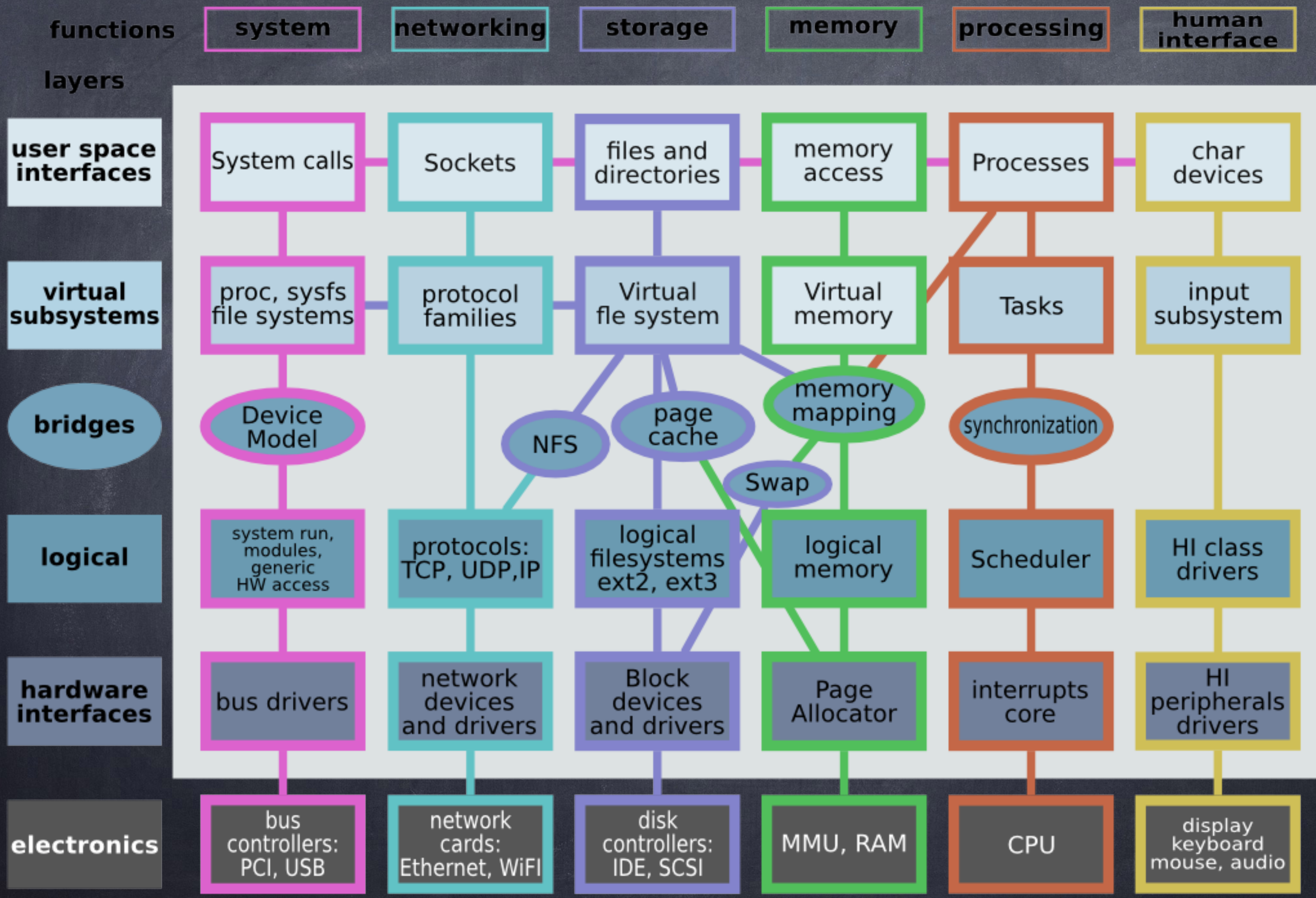
*.o

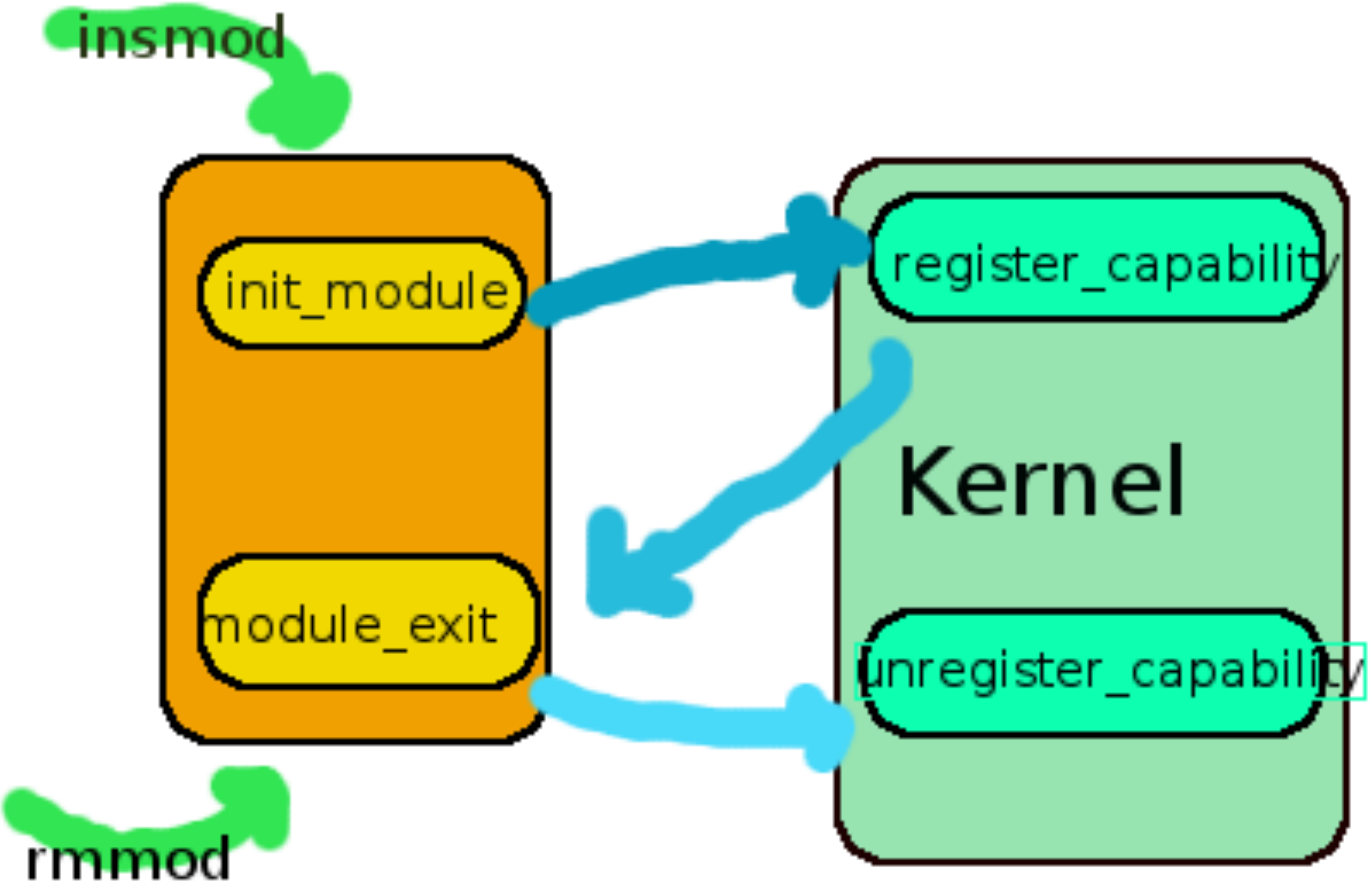
Krerk Piromsopa, Ph. D.

What is Kernel Module?

- Pieces of code that can be loaded and unloaded into the kernel upon demand.
- Extend the functionality of the kernel without rebooting the system.
  - Hardware Drivers
  - Special API
  - etc.

Linux kernel diagram




insmod
init_module
module_exit
rmmod
register_capability
Kernel
unregister_capability

Caution!

- No standard C lib
    - no printf, scanf, gets, etc..
    - no gdb (or any debugging tool)
- Use kernel functions only
    - printk

printk

- Usage
  ```
  printk( [LEVEL] const char *format_string, ... );
  ```

- Example
  ```
  printk(KERN_ERR "something went wrong, return code: %d\n",ret);
  ```

Level:

KERN_EMERG	KERN_NOTICE
KERN_ALERT	KERN_INFO
KERN_CRIT	KERN_DEBUG
KERN_ERR	KERN_DEFAULT
KERN_WARNING	KERN_CONT

Let's build our first Kernel Module

Dummy Kernel Module

```c
#include <linux/module.h>       /* Needed by all modules */
#include <linux/kernel.h>       /* Needed for KERN_INFO */

MODULE_LICENSE("GPL");
MODULE_AUTHOR("KRERK PIROMSOPA, PH.D. <Krerk.P@chula.ac.th>");
MODULE_DESCRIPTION("CP OS API");

int init_module(void)
{
        printk(KERN_INFO "CUCPMOD: init\n");
        /*
         * non 0 - means init_module failed
                 - module can't be loaded.
         */
        return 0;
}

void cleanup_module(void)
{
        printk(KERN_INFO "CUCPMOD: cleanup\n");
}
```

CPMOD.C

Makefile

```
obj-m += cpmod.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
```

About Debian

- Dont forget to install linux-kernel-headers (linux-libc-dev)

Common Commands

- lsmod - Show the status of modules in the Linux Kernel
- insmod - Simple program to insert a module into the Linux Kernel
  insmod [filename] [module options...]
- rmmod - Simple program to remove a module from the Linux Kernel
  rmmod [-f] [-s] [-v] [modulename]
- modprobe - Add and remove modules from the Linux Kernel
  modprobe [-v] [-V] [-C config-file] [-n] [-i] [-q] [-b] [modulename] [module parameters...]
- modinfo - Show information about a Linux Kernel module
  modinfo [-0] [-F field] [-k kernel] [modulename|filename...]
- dmesg - print or control the kernel ring buffer

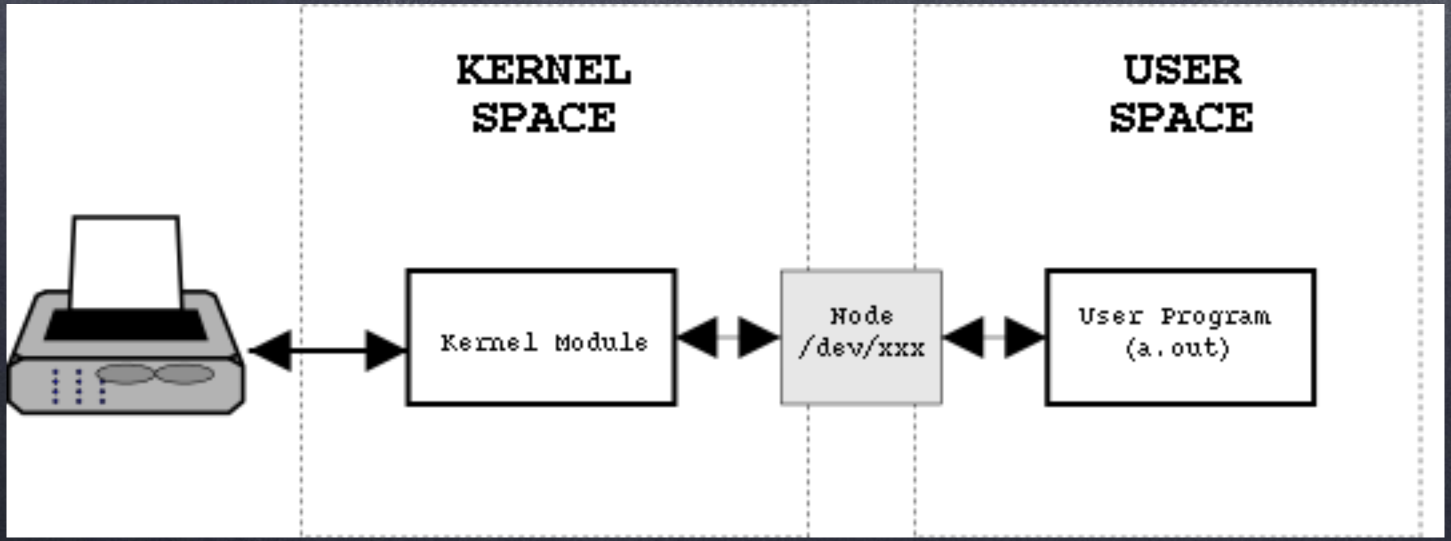
Device Drivers

Linux Devices

- Device File
  - an interface for a device driver in a file system
    (like an ordinary file)
  - Use file permissions for device permissions
  - Use file operations for accessing device

Example

- A write to /dev/lp0 will print to an associated printer



Just for fun..

- Open a terminal.
- Use "who" command to figure out the pseudo terminal device file.
- Try "echo data > /dev/pts/0"

```
krerk@OSBox:~$ who
krerk    pts/0        2016-11-22 15:23 (???.???.??.??)
krerk@OSBox:~$ echo "I am a master." >/dev/pts/0
I am a master.
krerk@OSBox:~$ 
```

Character Devices

- sending and receiving single characters
- Cannot seek
- eg. Serial/Parallel Port

Block Devices

- sending entire blocks of data
- Can seek
- eg. Disk, USB Camera

Major and Minor Device No.

- Devices are divided into sets called major device numbers.
- Same major number means same driver

```
brw-rw---- 1 root disk 8, 0 Nov 12 14:18 /dev/sda
brw-rw---- 1 root disk 8, 1 Nov 12 14:18 /dev/sda1
brw-rw---- 1 root disk 8, 2 Nov 12 14:18 /dev/sda2
brw-rw---- 1 root disk 8, 5 Nov 12 14:18 /dev/sda5
crw------- 1 krerk tty 136, 0 Nov 23 22:32 /dev/pts/0
```

To create a device

- Try
  mknod /dev/osinfo c 250 0
- This will create

```
crw-r--r-- 1 root root 250, 0 Nov 23 06:51 /dev/osinfo
```

File Operations

```
struct file_operations {
  struct module *owner;
   loff_t(*llseek) (struct file *, loff_t, int);
   ssize_t(*read) (struct file *, char __user *, size_t, loff_t *);
   ssize_t(*aio_read) (struct kiocb *, char __user *, size_t, loff_t);
   ssize_t(*write) (struct file *, const char __user *, size_t, loff_t *);
   ssize_t(*aio_write) (struct kiocb *, const char __user *, size_t, loff_t);
  int (*readdir) (struct file *, void *, filldir_t);
  unsigned int (*poll) (struct file *, struct poll_table_struct *);
  int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
  int (*mmap) (struct file *, struct vm_area_struct *);
  int (*open) (struct inode *, struct file *);
  int (*flush) (struct file *);
  int (*release) (struct inode *, struct file *);
  int (*fsync) (struct file *, struct dentry *, int datasync);
  int (*aio_fsync) (struct kiocb *, int datasync);
  int (*fasync) (int, struct file *, int);
  int (*lock) (struct file *, int, struct file_lock *);
   ssize_t(*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
   ssize_t(*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
   ssize_t(*sendfile) (struct file *, loff_t *, size_t, read_actor_t, void __user *);
   ssize_t(*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
  unsigned long (*get_unmapped_area) (struct file *, unsigned long, unsigned long, unsigned long,
unsigned long);

};
```

see linux/fs.h for more details

TODO

- Implement only related operations
- Leave others to NULL

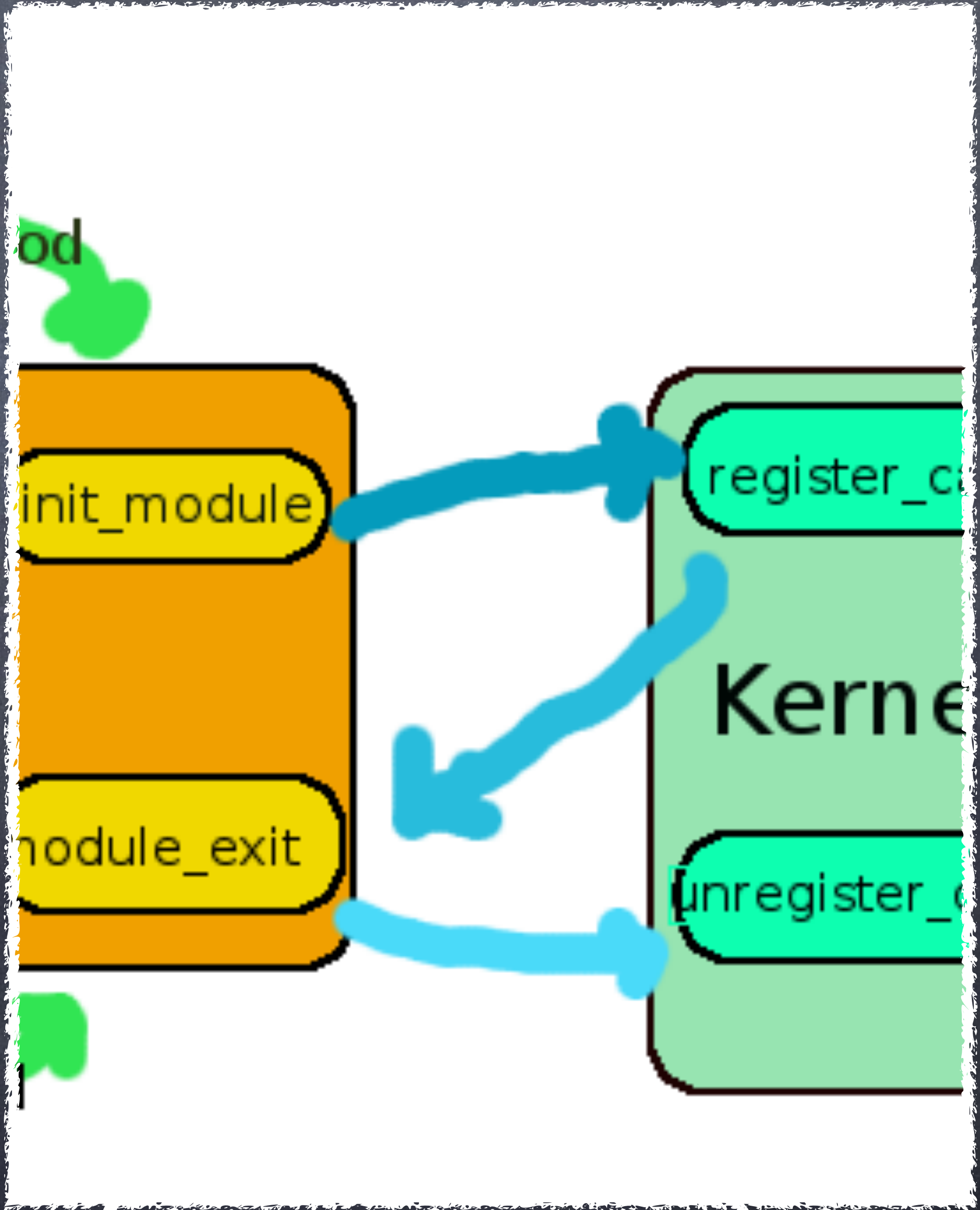
A better way

- Use C99 syntax for convenient

```
struct file_operations fops = {
   .read = device_read,
   .write = device_write,
   .open = device_open,
   .release = device_release
};
```

To register device

- int register_chrdev(unsigned int major, const char *name, struct file_operations *fops);
- unregister_chrdev(unsigned int major, const char *name);



see linux/fs.h for more details

Example:
Read-only char device

```
static int device_open(struct inode *, struct file *);
static int device_release(struct inode *inode, struct file *file);
static ssize_t device_read(struct file *, char *, size_t, loff_t *);

// File operations structor
// Only implement those that will be used.
static struct file_operations dev_fops = {
        .read = device_read,
        .open = device_open,
        .release = device_release
};
```

More about C

- For security, try to use static modifiers for functions, variables in C to make it local to your kernel module.

More about driver

- To avoid module from being unloaded (rmmod) while using,
  - On every access to module (e.g. device_open), please lock the module with try_module_get(THIS_MODULE);
  - On release, release the module with module_put(THIS_MODULE);

Let's have fun.

Q1. Dummy Mod

- Make a dummy driver that will do the following:
  - On initial, display KERN_INFO "CPMOD: init"
  - On cleanup, display KERN_INFO "CPMOD: cleanup\n"

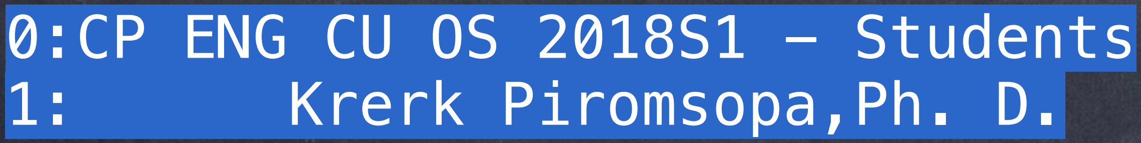
Q2. Char Driver

- Create osinfo driver (minor no. 0) (at /dev/osinfoa).
- Reading from this device file will display:

```
0:CP ENG CU OS 2018S1 - Instructors
1:      Krerk Piromsopa,Ph. D.
2:      Kunwadee Sripanidkulchai, Ph. D.
3:      Thongchai Rojkangsadan
4:      Veera Muangsin, Ph.D.
```

Q3. Adv. Char Driver

- Modify driver in Q2 to display to detect minor no.
- If minor no. == 1, reading from the device will display names of your team members.



```
0:CP ENG CU OS 2018S1 – Students
1:      Krerk Piromsopa,Ph. D.
```

Hint: To detect minor number, use
static inline unsigned iminor(const struct inode *inode)

Q4. CPU model driver

- Use the following function (taken from arch/x86/include/asm/processor.h) to create a device driver for extracting cpuid (vendor ID, features, serial no.).

```
static inline void native_cpuid(unsigned int *eax, unsigned int *ebx,
                                unsigned int *ecx, unsigned int *edx)
{
        /* ecx is often an input as well as an output. */
        asm volatile("cpuid"
            : "=a" (*eax),
              "=b" (*ebx),
              "=c" (*ecx),
              "=d" (*edx)
            : "0" (*eax), "2" (*ecx)
            : "memory");
}
```

```
eax = 3; /* processor serial number */
native_cpuid(&eax, &ebx, &ecx, &edx);
printf("serial number 0x%08x%08x\n", edx, ecx);
```

For more details , see https://en.wikipedia.org/wiki/CPUID

More Hint.

- Kernel supports limited string function.
- For Q4, try
  - snprintf, strncat.
- See Kernel API (https://www.kernel.org/doc/htmldocs/kernel-api/) for more details.

Quick demonstration

References

- https://www.kernel.org/
  - Kernel API https://www.kernel.org/doc/htmldocs/kernel-api/
- http://elixir.free-electrons.com/linux/latest/source