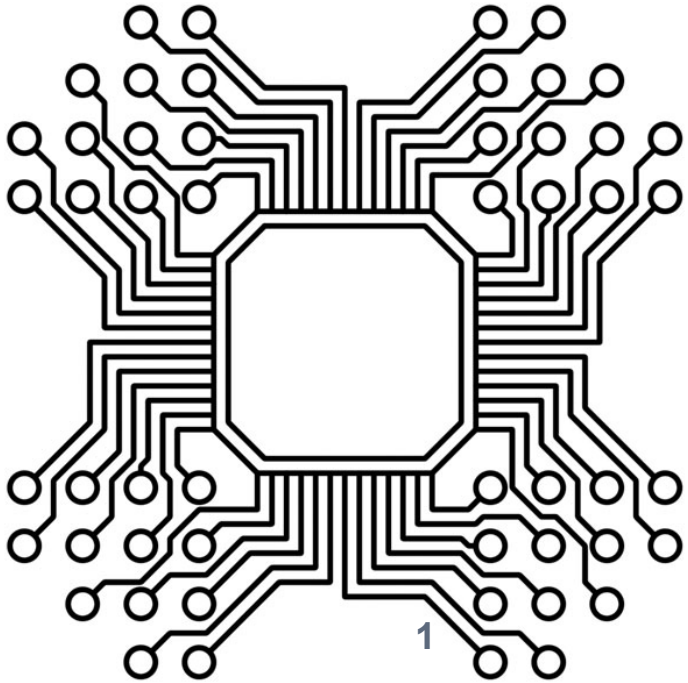
ADVANCED EMBEDDED PROGRAMMING

Lecturer: Dr. Bui Ha Duc
Dept. of Mechatronics
Email: ducbh@hcmute.edu.vn



C Library

- Library: bunch of files that contain just functions and declaration Write once use many
- C Library is a collection of object files
- Types of C library:
  - *Static libraries*: linked into the program during the linking phase of compilation and are not relevant during runtime
  - *Dynamic libraries*: linked by multiple program at the same time during running time

Create a C library

Create static libraries

- Step 1: create C source files containing any functions that will be used.
- Step 2: Compile these files into objects

```
gcc -c libraryCode.c -o object.o
```

- Step 3: Create library

```
ar rc libname.a object.o
```

- Step 4: Linking your program to the libraries, make sure you specify where the library can be found

```
gcc file.c -L. -lname -o newfile
```

Create a C library

Create dynamic libraries

- Step 1: create C source files containing any functions that will be used.
- Step 2: Compile these files into Position Independent Code

```
gcc -c -fPIC libraryCode.c -o object.o
```

- Step 3: Create library

```
gcc -shared -o libname.so objfile.o
```

- Step 4: Linking your program to the libraries, make sure you specify where the library can be found

```
gcc file.c -L. -lname -o newfile
```

Create a C library

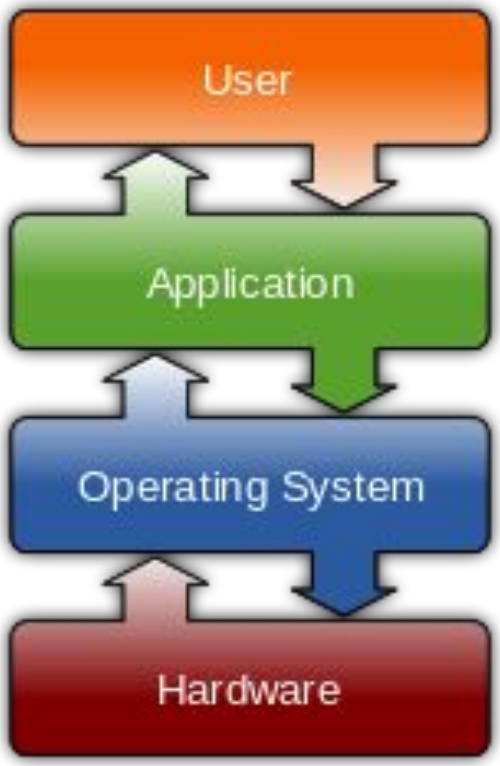
Step 5: Copy file *library.h* to *usr/include*
And file *library.so* to *usr/lib*

Or declare the List of Dynamic Dependency:

export LD_LIBRARY_PATH=:/path/to/library.so

Device Driver

- Device driver is a computer program that **operates or controls a particular type of device** that is attached to a computer
- Driver provides a **software interface** to hardware devices, enabling operating systems and other computer programs to access hardware functions.



Functions of Driver

- Isolate the user program from the complexity of the hardware device
  - E.g. open, copy a file in hard disk
- Provides a consistent user interface to a large variety of hardware device

Device Driver Architecture

- Linux lets you add and remove kernel components at runtime
  - Provide flexibility
  - Enhance upgrade capability
  - Module can be stored on media other than root
- Linux device driver are broadly classified into two basic categories:
  - **Character devices** can be thought of as serial streams of sequential data.

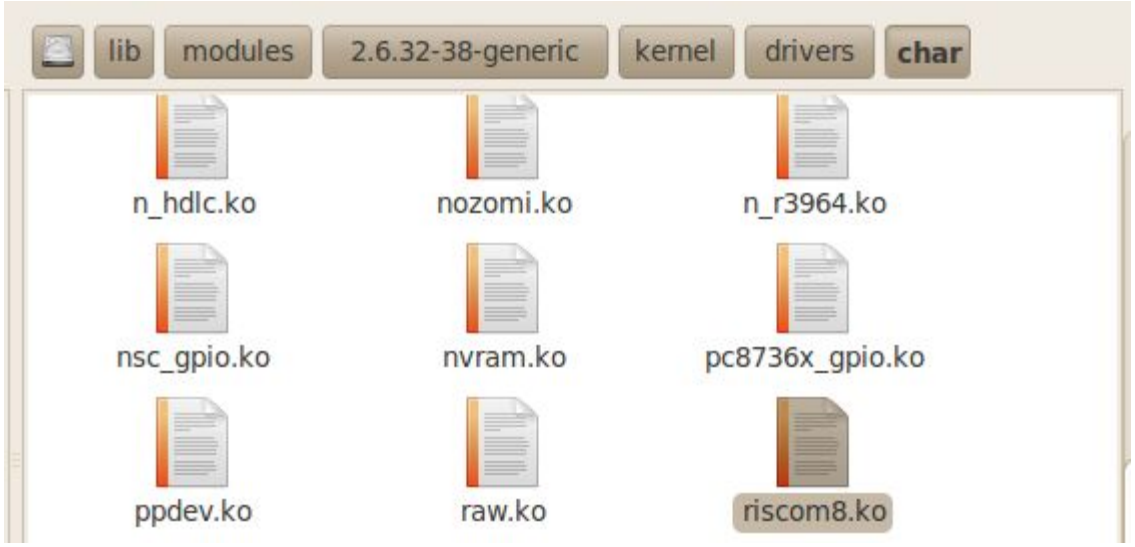
    e.g. serial ports and keyboards.
  - **Block devices** are characterized by the capability to read and write blocks of data to and from random locations on an addressable medium

    e.g. hard drives and USB Flash drives.

Driver location

- Generally, a driver is stored in folder **/lib/modules/<kernel_version>/kernel**

- Driver is stored as ***.ko** file



Minimal Device Driver

```
#include <linux/kernel.h>
#include <linux/module.h>
static int __init mini2440_hello_module_init(void)
{
      printk("Hello, Mini2440 module is installed !\n");
      return 0;
}
      static void __exit mini2440_hello_module_cleanup(void)
{
      printk("Good-bye, Mini2440 module was removed!\n");
}
module_init(mini2440_hello_module_init);
module_exit(mini2440_hello_module_cleanup);
MODULE_LICENSE("GPL");
```

Notes:

- No main() function
- No built-in C function
- 2 fundamental function
  - module_init()
  - module_exit()
- printk : print messeage to kernel log files

Compile a Driver

- A device driver must be **compiled against the kernel** on which it will execute
- **Method 1**: Create a Make file and put it in the same folder with file hello.c

```
obj-m := hello1.o
KDIR  := /lib/modules/$(shell uname -r)/build
PWD   := $(shell pwd)
default:
	$(MAKE) -C $(KDIR) M=$(PWD) modules
```

- Run the Make file
- Load the module

Compile a Driver

- **Method 2:** Add code to the kernel source tree, and do the appropriate configuration
- **Step 1**. Edit the configuration file Kconfig (linux-2.6.32.2/drivers/char/Kconfig)

```
config MINI2440_HELLO_MODULE
    tristate "Mini2440 module sample"
    depends on MACH_MINI2440
    default m if MACH_MINI2440
    help
      Mini2440 module sample.

config LEDS_MINI2440
    tristate "LED Support for Mini2440 GPIO LEDs"
    depends on MACH_MINI2440
    default y if MACH_MINI2440
    help
      This option enables support for LEDs connected to GPIO lines
      on Mini2440 boards.

config MINI2440_BUTTONS
    tristate "Buttons driver for FriendlyARM Mini2440 development boards"
    depends on MACH_MINI2440
    default y if MACH_MINI2440
    help
      this is buttons driver for FriendlyARM Mini2440 development boards
```

Compile a Driver

- **Step 2**. edit the Makefile in the kernel configuration (linux-2.6.32.2/drivers/char/Makefile)

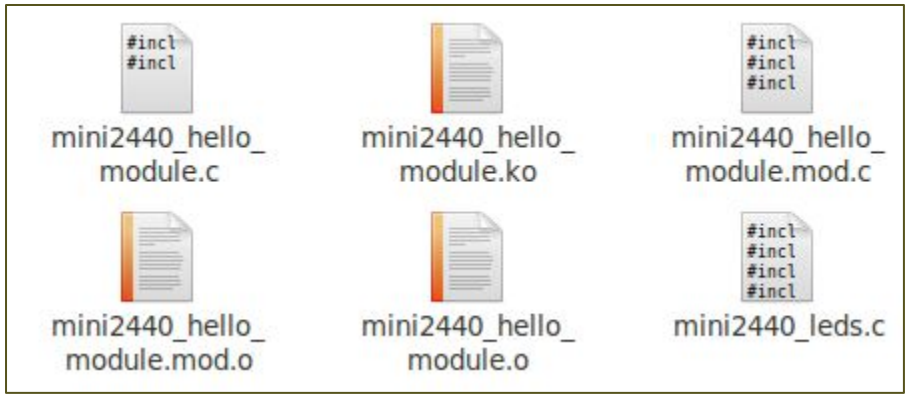
Makefile

```
110 obj-$(CONFIG_PS3_FLASH)      += ps3flash.o
111
112 obj-$(CONFIG_JS_RTC)         += js-rtc.o
113 js-rtc-y = rtc.o
114
115 obj-$(CONFIG_LEDS_MINI2440) += mini2440_leds.o
116 obj-$(CONFIG_MINI2440_HELLO_MODULE) += mini2440_hello_module.o
117 obj-$(CONFIG_MINI2440_BUTTONS)  += mini2440_buttons.o
118 obj-$(CONFIG_MINI2440_BUZZER)   += mini2440_pwm.o
119 obj-$(CONFIG_MINI2440_ADC)  += mini2440_adc.o
120
121 # Files generated that shall be removed upon make clean
122 clean-files := consolemap_deftbl.c defkeymap.c
```

Compile a Driver

- **Step 3.** back to linux-2.6.32.2 root directory, run the makefile

```
make modules
```



Then copy the **.ko** file to /lib/modules/2.6.32.4-FriendlyARM folder on the FriendlyARM board

Loading a driver

- **modprobe** : a utility used to insert a driver into a running kernel.

```
modprobe mini2440_hello_module
```

  - Similar to **insmod**

- **modprobe –r** : remove a driver
  - Similar to **rmmod**

```
rmmod mini2440_hello_module
```

- **lsmod** : list of driver which is inserted in to the kernel

Practical Device Driver

- Beside **module_init** and **module_exit**, we need other functions to interface the device with the program
- ***open()*** *:* prepare the driver for subsequent operations after the device driver is loaded into a live kernel

```
static int hello_open(struct inode *inode, struct file *file)
```

- ***release()*** : provided to clean up after operations are complete

```
static int hello_release(struct inode *inode, struct file *file)
```

- ***ioctl()*** : a special system call for nonstandard communication with the driver

```
static int hello_ioctl(struct inode *inode, struct file *file,
                       unsigned int cmd, unsigned long arg)
```

Device Nodes and mknod

- A device node is a special file type in Linux that represents a device
- Linux keep device nodes in a directory called **/dev**.
- A dedicated utility is used to create a device node on a file
- System is called **mknod**.

mknod /dev/hello1 c 234 0

c means that a char device is to be created

234 major number registered with the kernel

0 minor number, not registered with the kernel

How to use device driver

1. Call **open()** function

```
/* Open the device */
fd = open("/dev/hello1", O_RDWR);
if ( fd == -1 ) {
    perror("open failed");
    rc = fd;
    exit(-1);
}
```

2. Call desired command

```
rc = read(fd, rd_buf, 0);
```