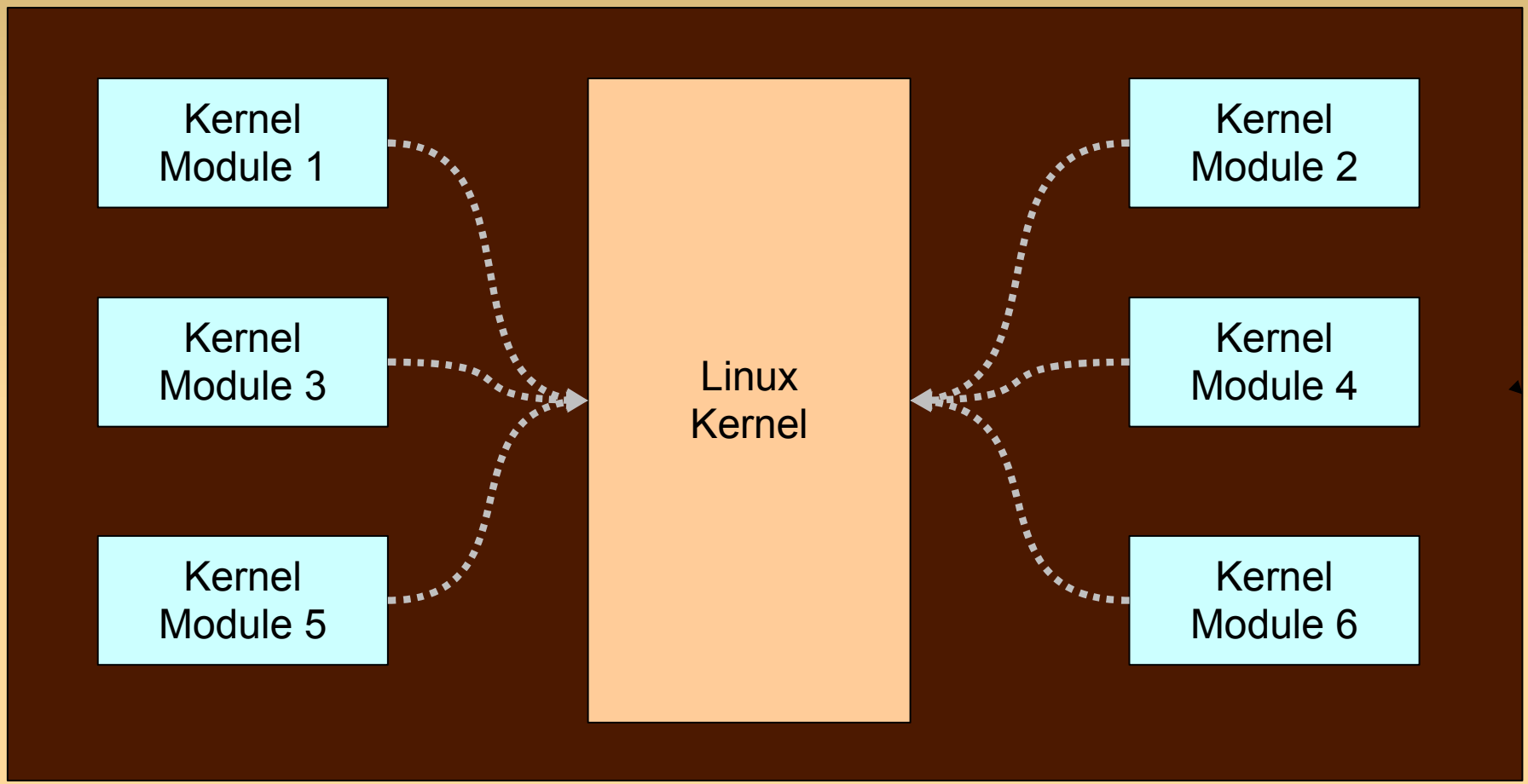
Device Drivers in Linux

Pre-requisites

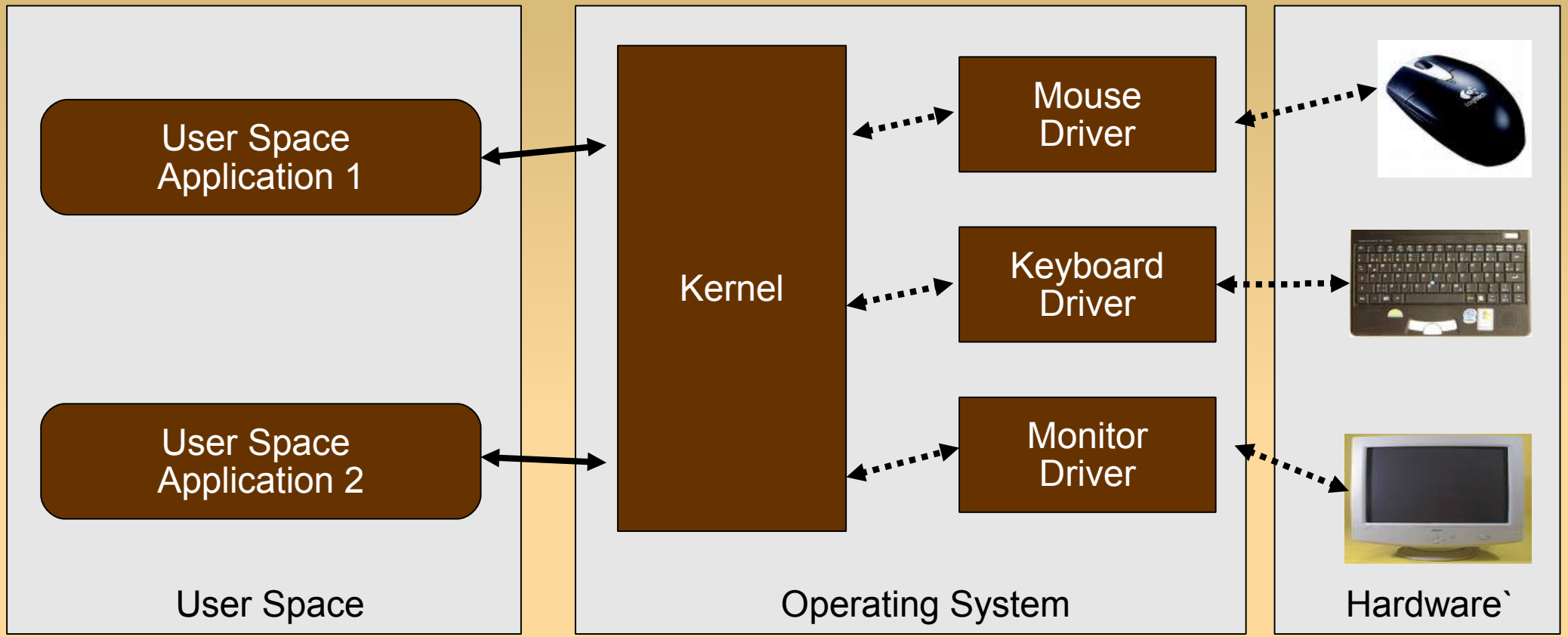
In order to develop Linux device drivers, it is necessary to have an understanding of the following:

* C programming. Some in-depth knowledge of C programming is needed, like pointer usage, bit manipulating functions, etc.
* Microprocessor programming. It is necessary to know how microcomputers work internally: memory addressing, interrupts, etc. All of these concepts should be familiar to an assembler programmer.

The Linux Kernel

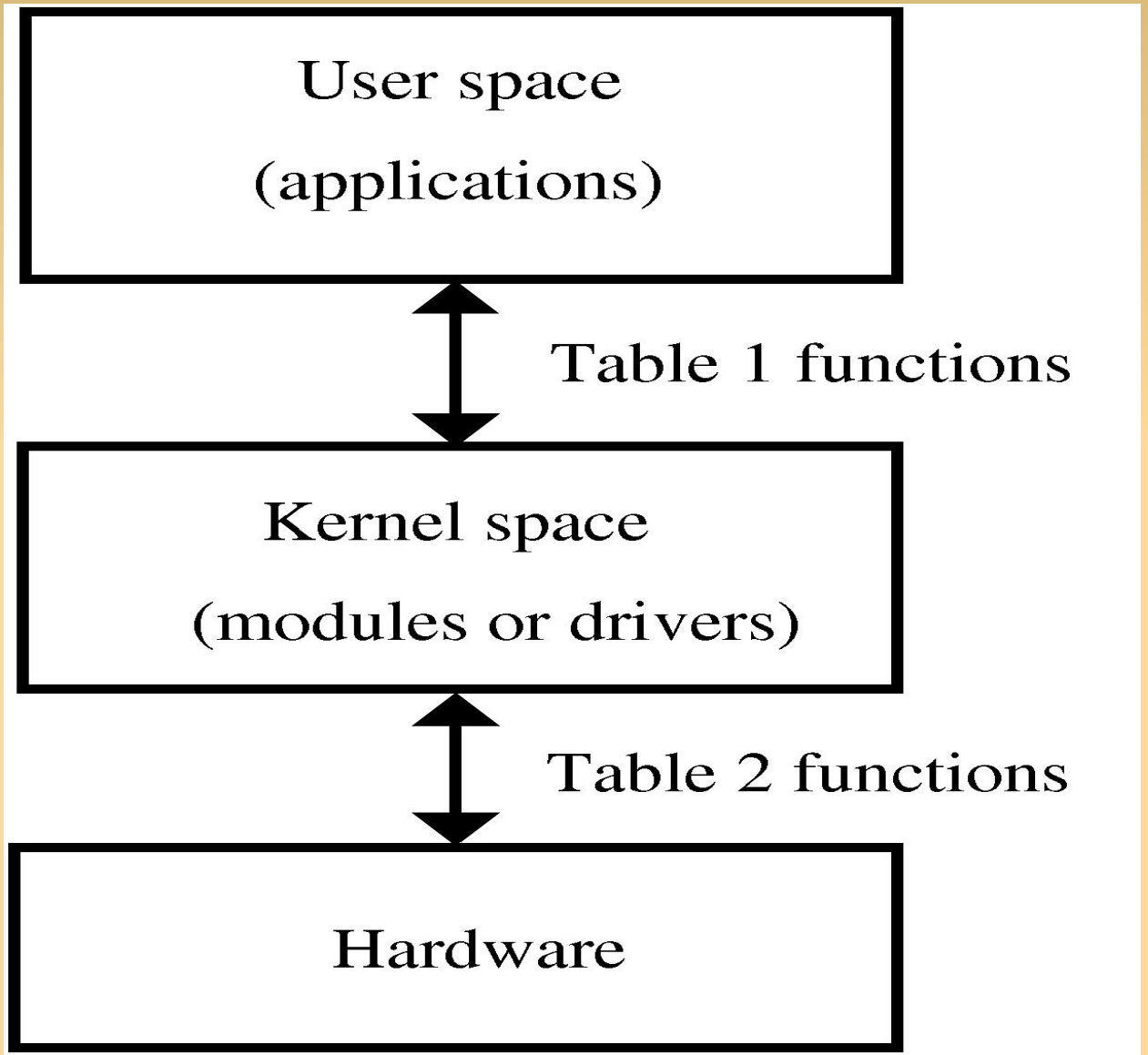


Device Drivers



ws a user space application interaction with a hardware, without need for the knowledge o

Device Drivers



Interfacing functions between user space and kernel space

Events	User functions	Kernel functions
Load module		
Open device		
Read device		
Write device		
Close device		
Remove module		

Interfacing functions between kernel space and the hardware device

Events	Kernel functions
Read data	
Write data	

We live in a **DRIVER** crazy world...**WHY**?????

Large Linux User Base....therefore, more potential buyers for device manufacturers

To have control access to the PC

To ease the programmability for Application Developers, without having to learn the electronics of hardware

The Role of a Device Driver

Mechanisms not Policies

Concurrent Accesses

Case I: One Device, many applications concurrently

Case II: Many Devices, one application

Case III: Many Devices, Many Applications

Platform Portability should be considered

System Modularity should be considered

Some Examples.....

sk as a contiguous array of blocks. Accessibility restrictions are implem

vithout implementing restrictions on the application

Classification of Drivers

Character Devices

Examples: Serial Devices, Text consoles....

Block Devices

Examples: Hard disk drivers, CDROMs....

Network Devices

Examples: Ethernet, WLAN.....

Exclusions from the above (USB, SCSI etc..,)

Module Programming

Loadable kernel modules

Modules: add a given functionality to the kernel (drivers, filesystem support, and many others).

Can be loaded and unloaded at any time, only when their functionality is need. Once loaded, have full access to the whole kernel address space. No particular protection.

Modules make it easy to develop drivers without rebooting: load, test, unload, rebuild, load...

Module Programming

- Module Initialization
- Module De-Initialization
- Compilation & Loading a Module
- The Makefile
- Module Parameters
- Kernel Symbol Table

The first driver: loading and removing the driver in user space

<nothing.c> =

```
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");
```

<Makefile1> =

```
obj-m := nothing.o
```

To Compile:

```
$ make -C /lib/modules/2.6.28-11-generic/build
M=/home/raajita/vector/presentation/Class3/nothing modules
```

```
$insmod nothing.ko
```

```
$lsmod
```

```
$rmmod nothing
```

The first driver: loading and removing the driver in user space

Events	User functions	Kernel functions
Load module	insmod	
Open device		
Read device		
Write device		
Close device		
Remove module	rmmod	

The “Hello world” driver: loading and removing the driver in kernel space

<hello.c> =

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
MODULE_LICENSE("Dual BSD/GPL");
static int hello_init(void) {
  printk("<1> Hello world!\n");
  return 0;
}
static void hello_exit(void) {
  printk("<1> Bye, cruel world\n");
}
module_init(hello_init);
module_exit(hello_exit);
```

The Makefile

```
obj-m := hello.o

KERNELDIR = /lib/modules/$(shell uname -r)/build
PWD :=$(shell pwd)

default:
$(MAKE) -C $(KERNELDIR) M=$(PWD) modules
```

The “Hello world” driver: loading and removing the driver in kernel space

Events	User functions	Kernel functions
Load module	insmod	module_init()
Open device		
Read device		
Write device		
Close device		
Remove module	rmmod	module_exit()

Compiling & Loading the Module

```
linux:~ $ make
linux:~ $ su
linux:~ # insmod hello.ko
linux:~ # dmesg
linux:~ # lsmod
linux:~ #  rmmod hello
```

Module Parameters

Parameters may be passed to the modules during module insertion

The macro ***"module_param(name, type,perm)"*** is used to initialize the parameter at runtime.
It takes 3 arguments which are the parameter name, parameter type and the permissions associated with the parameter

This macro may be defined anywhere in the module

The Kernel Symbol Table

gh the use of kernel macros *"EXPORT_SYMBOL(name)"* which take

es in requirement of similar functionality as done by the exported funct

User Space Drivers

Advantages

Extensive Library Support

Debugging Support

Crash independent

Memory optimization when driver is not used

Disadvantages

Interrupts & Port non-availability

Response time due to context switches and page swapping

Direct access to memory is possible my mmap and only by privileged users

References

ubini and Greg Kroah-Hartman,"*Linux Device Drivers*",3rd Edition, O'R

cock, "*Interfacing the Serial/ RS-232 port,V 5.0*"

nux_kernel

permail/ucc/2003-June/009997.html