

The Aricent logo is located in the top right corner. It features the word "Aricent" in a stylized orange font, with a registered trademark symbol (®) to its upper right. The background of the slide is a dark, abstract image with swirling, glowing lines in shades of blue and yellow, creating a sense of motion and energy.

Aricent®

Engineering excellence. **Sourced.**

Linux Device Driver Basics

By

-HEMAKUMAR

# Session-4

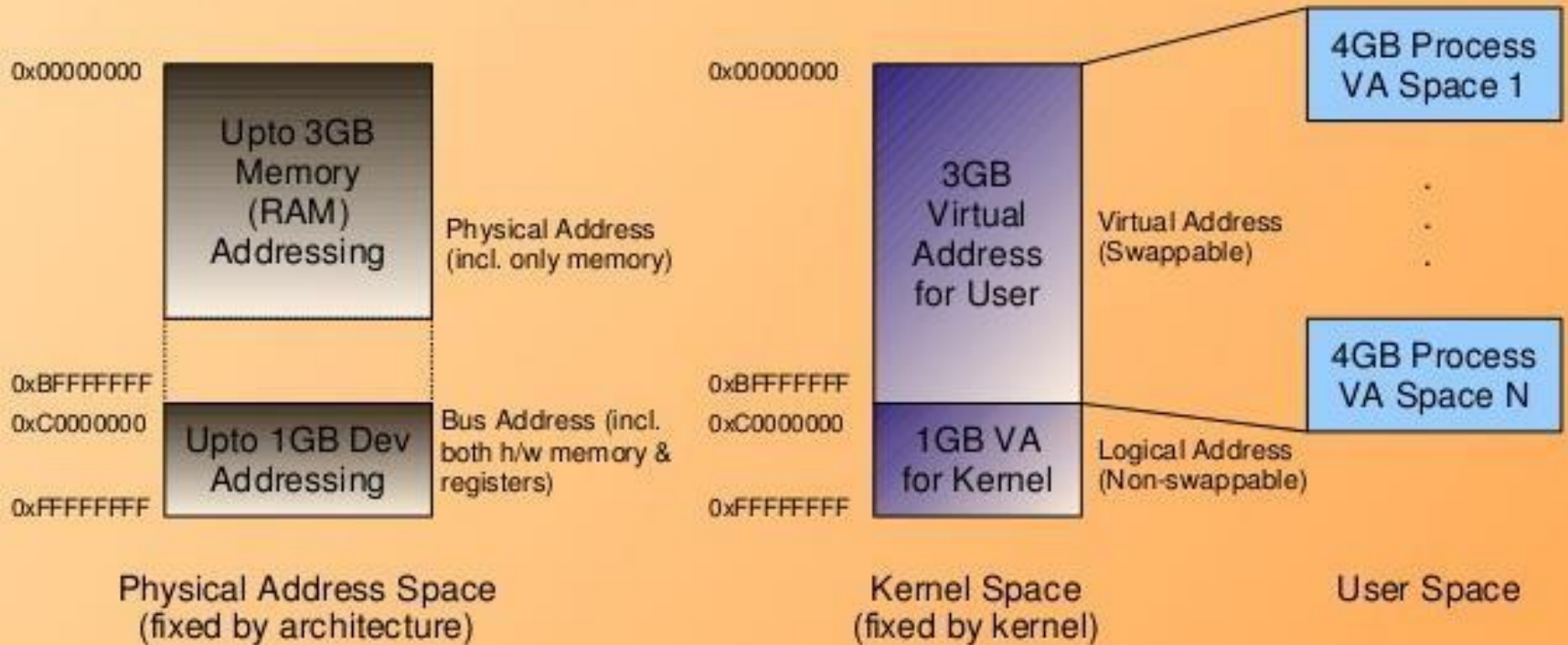
## Low-level Accesses

## What to Expect?

- ✧ After this session, you would know
  - Various Address Spaces in Linux
  - Role of Memory Manager in Linux
  - Accessing the Memory in Kernel Space
  - Accessing the Device or Hardware
    - Memory
    - Registers
  - Low-level Access in Drivers

# Address Spaces in Linux

★ An Example assuming 32-bit architecture



# Linux Memory Manager

- ☆ Provides Access Control to h/w & memory resources
- ☆ Provides Dynamic Memory to kernel sub-system
  - Drivers
  - File Systems
  - Stacks
- ☆ Provides Virtual Memory to Kernel & User space
  - Kernel & User Processes run in their own virtual address spaces
  - Providing the various features of a Linux system
    - System reliability, Security
    - Communication
    - Program Execution Support



# Kernel Space Memory Access

## ★ Virtual Address for Physical Address

- Header: <linux/gfp.h>
  - unsigned long \_\_get\_free\_pages(flags, order); etc
  - void free\_pages(addr, order); etc
- Header: <linux/slab.h>
  - void \*kmalloc(size\_t size, gfp\_t flags);
    - ➔ GFP\_USER, GFP\_KERNEL, GFP\_DMA
  - void kfree(void \*obj);
- Header: <linux/vmalloc.h>
  - void \*vmalloc(unsigned long size);
  - void vfree(void \*addr);

# Memory allocations in Kernel

## kmalloc and kfree:

- Basic allocators, kernel equivalents of glibc's malloc and free.

- #include <[linux/slab.h](#)>

- static inline void \*[kmalloc](#)([size\\_t](#) size, int flags);

size: number of bytes to allocate

flags: priority (see next page)

- void [kfree](#) (const void \*objp);

- Example: struct mem \*ptr;

ptr = [kmalloc](#)(sizeof(struct mem), GFP\_KERNEL);

if (!ptr)

/\* handle error ... \*/

...

[kfree](#)(ptr);

# Memory allocations in Kernel.

## Kmalloc features:

- ❑ Quick (unless it's blocked waiting for memory to be freed).
- ❑ Doesn't initialize the allocated area.  
You can use [kcalloc](#) or [kzalloc](#) to get zeroed memory.
- ❑ The allocated area is contiguous in physical RAM.
- ❑ Allocates by  $2^n$  sizes, and uses a few management bytes.  
So, don't ask for 1024 when you need 1000! You'd get 2048!
- ❑ Caution: drivers shouldn't try to [kmalloc](#) more than 128 KB (upper limit in some architectures).
- ❑ Minimum allocation: 32 or 64 bytes (page size dependent).



# Memory allocations in Kernel..

## Kmalloc main flags:

Defined in [include/linux/gfp.h](#) (GFP: [\\_\\_get\\_free\\_pages](#))

### ☐ [GFP\\_KERNEL](#)

Standard kernel memory allocation. May block. Fine for most needs.

### ☐ [GFP\\_ATOMIC](#)

Allocated RAM from interrupt handlers or code not triggered by user processes. Never blocks.

### ☐ [GFP\\_USER](#)

Allocates memory for user processes. May block. Lowest priority.

# Memory allocations in Kernel...

- [vmalloc](#) can be used to obtain big chunk of memory in virtual address space (that means pages allocated are not contiguous in RAM, but the kernel sees them as one block).
- May block if free pages are not available.
  - void [vmalloc](#)(unsigned long size);
  - void [vfree](#)(void \*addr);
- Example:
  - char \*buf;
  - buf = [vmalloc](#)(16 \* PAGE\_SIZE); /\* get 16 pages \*/
  - if (!buf)
  - /\* error! failed to allocate memory \*/
  - /\* buf now points to at least a 16\*PAGE\_SIZE bytes \* of virtually contiguous block of memory
  - \*/
  - [vfree](#)(buf);

# Kernel Space Device Access

## ★ Virtual Address for Bus/IO Address

### ➤ Header: <asm/io.h>

- void \*ioremap(phys\_addr\_t bus\_addr, unsigned long size);
- void iounmap(void \*addr);

## ★ I/O Memory Access

### ➤ Header: <asm/io.h>

- u[8|16|32] ioread[8|16|32](void \*addr);
- void iowrite[8|16|32](u[8|16|32] value, void \*addr);

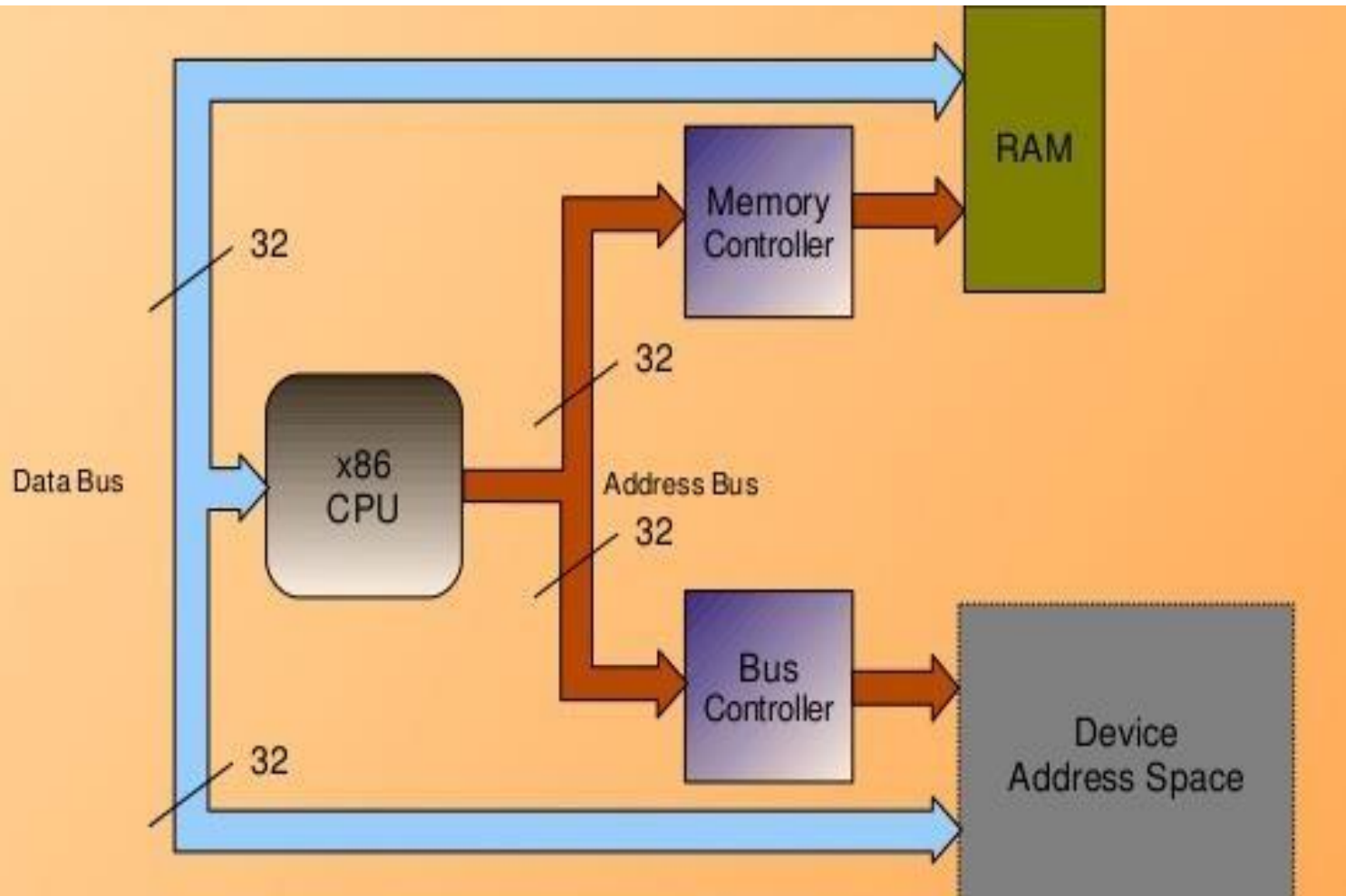
## ★ Kernel Window: /proc/iomem

## ★ Access Permissions

### ➤ Header: <linux/ioport.h>

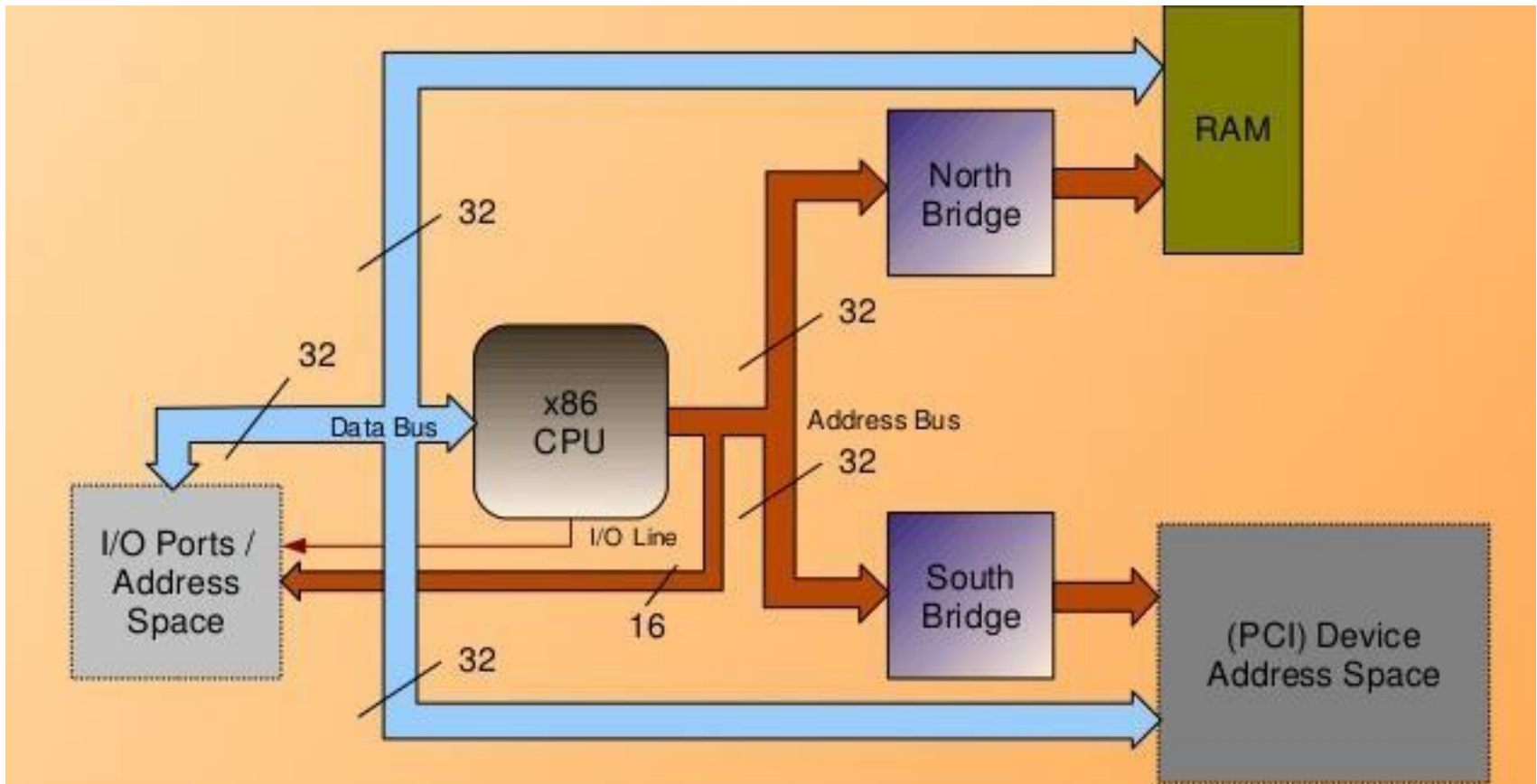
- struct resource \*request\_mem\_region(resource\_size\_t start, resource\_size\_t size, label);
- void release\_mem\_region(resource\_size\_t start, resource\_size\_t size);

# x86 Memory & Device Access



# x86 Hardware Architecture

complete



# I/O Access (x86\* specific)

## \* I/O Port Access

- ➔ u8 inb(unsigned long port);
- ➔ u16 inw(unsigned long port);
- ➔ u32 inl(unsigned long port);
- ➔ void outb(u8 value, unsigned long port);
- ➔ void outw(u16 value, unsigned long port);
- ➔ void outl(u32 value, unsigned long port);

## \* Header: <asm/io.h>

## \* Kernel Window: /proc/ioports

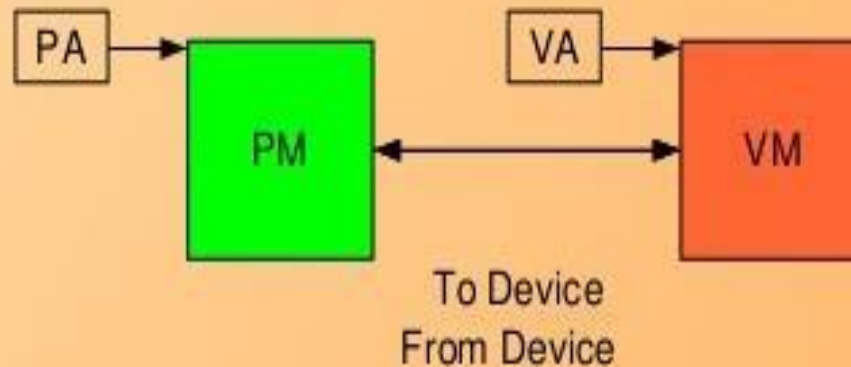
## \* Access Permissions

### ➔ Header: <linux/ioport.h>

- struct resource \*request\_region(resource\_size\_t start, resource\_size\_t size, label);
- void release\_region(resource\_size\_t start, resource\_size\_t size);



# DMA Mapping

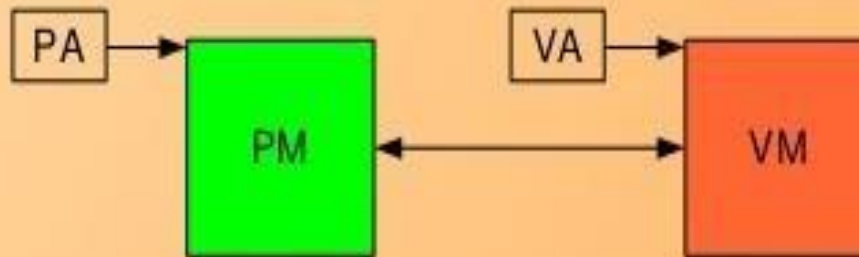


## ✧ APIs

- ➔ `dma_addr_t dma_map_single(struct device *, void *, size_t, enum dma_data_direction);`
- ➔ `void dma_unmap_single(struct device *, dma_addr_t, size_t, enum dma_data_direction);`
- ➔ Directions
  - `DMA_BIDIRECTIONAL`
  - `DMA_TO_DEVICE`
  - `DMA_FROM_DEVICE`
  - `DMA_NONE`

✧ Header: `<linux/dma-mapping.h>`

# DMA Allocation



## \* APIs

- `void *dma_alloc_coherent(struct device *, size_t, dma_addr_t *, gfp_t);`
- `void dma_free_coherent(struct device *, size_t, void *, dma_addr_t);`
- `int dma_set_mask(struct device *, u64 mask);`

\* Header: `<linux/dma-mapping.h>`

## Barriers

- ☆ Heard about Processor Optimization?
- ☆ `void barrier(void);`
  - For surrounding instructions
  - Header: `<linux/kernel.h>`
- ☆ `void [r|w|]mb(void);`
  - For surrounding read/write instructions
  - Header: `<asm/system.h>`

# Memory & Character Driver

## ★ Dynamic Memory Experiments

- Preserve latest write in /dev/memory
- Control the preserve size using ioctl
- Implement seek

## Hardware & Character Driver

- ★ Digital/Analog I/O Control on the Board
- ★ Figure out
  - Operation Relevant Registers
  - Hardware Access Addresses
  - Relevant low-level access APIs to be used
- ★ Driver for I/O access over /dev/io



## What all have we learnt?

- ✧ Various Address Spaces in Linux
- ✧ Role of Memory Manager in Linux
- ✧ Accessing the Memory in Kernel Space
- ✧ Accessing the Device or Hardware
  - ✧ Memory
  - ✧ Registers
- ✧ Barriers
- ✧ Low-level Access in Drivers



Any Queries?

## Optional Slides for Session-4 Low-Level Access

# Linux Kernel Interrupt handling

## Interrupts:

- Asynchronous events triggered by devices to notify some change.
- ISR routine will be executed when interrupt occurs.
- Driver should implement ISR routine and register with Kernel Interrupt subsystem.
- Routines to register&unregister the ISR:

```
int request_irq(unsigned int irq, irq_handler_t handler,  
unsigned long flags, const char *name, void *dev);  
void free_irq(unsigned int irq, void *dev_id);
```

# Linux Kernel Interrupt handling.

## Example: For Keyboard ISR registration.

```
static unsigned int upcode; //global var to update the scancode.
request_irq(KB_INT, /1 * The number of the keyboard IRQ on PCs*/
            irq_handler, /* our handler */
            SA_SHIRQ, "test_keyboard_irq_handler", (void *)(&data));
free_irq(KB_INT, &data);
irq_handler(int irq, void *dev_id, struct pt_regs *regs)
{
    unsigned char scancode;
    scancode = read_kbd_input();
    printk("scancode = %d\n", scancode);
    upcode = scancode;
    wake_up_interruptable(&my_kb_queue);
    return 0;
}
```

# Device Control Interfaces

## Device drivers Basic Interface functions:

While writing a drivers for particular device, we should consider below

- Actual functionality implementation.
- Implementation to control the device.

### 1. Functionality Interface routines

- Implements actual functionality of device

- Ex: Camera device

- Enable --- Open()

- Disable --- close()

- Capture --- Read()

- Display --- write()

### 2. Control Interface routines

- Implements control of device

- Ex: resolution , zoom , brightness setting etc --- ioctl()

# Device Control Interfaces

## ioctl() - IO control

- ❑ All the Device control operations should be implemented using `ioctl()`
- ❑ `ioctl()` implementation is unique for each device.
- ❑ `ioctl()` function can support 255 control interfaces.
- ❑ `ioctl()` is a part of file operation structure.
- ❑ Each Driver `ioctl()` function should export `ioctl` request commands to userspace.
- ❑ U-space apps can use these request commands & u-space `ioctl()` API to control the device.



# Device Control Interfaces

## ioctl() - IO control - from u-space

- U-space App can make use of ioctl() API to initiate device specific functionality.

Header file :

```
#include <sys/ioctl.h>
```

API:

```
int ioctl(int fd, int request, ...)
```

arg1: file descriptor

arg2: ioctl request command.

arg3: untied pointer – depends of request command type

Ex: `ret_val = ioctl(fd, TICAM_BRIGHTNESS_SET, data)`

`ret_val = ioctl(fd, TEST_FILL_CHAR, a)`

# Device Control Interfaces

## ioctl() - IO control - K-space implementation

- Identify Control operations offered by the device
- Declare a unique ioctl request command for each control operation. Take one header file generate ioctl commands & export the header file to u-spce.
- Implement an ioctl routine in a driver to process ioctl requests.

# Device Control Interfaces

## ioctl() - IO control - K-space implementation

Kernel provides Macro's to generate unique ioctl request commands in <linux/ioctl.h>.

- ☐ \_IO(type, n)
- ☐ \_IOW(type, n, dataitem)
- ☐ \_IOR(type, n, dataitem)
- ☐ \_IOWR(type, n, dataitem)

Which one to use depends on

- ☐ Type or magic num
- ☐ Sequence num – 8-bit wide
- ☐ Direction – reading or writing
- ☐ Size – of user data (int size or char size)

# Device Control Interfaces

## ioctl() - IO control - K-space implementation

Assume we are writing driver – assume 1024 bytes of RAM as a device for us.

Now we will implement driver ioctl() to fill the data in that 1024bytes.

In header file:

```
#define TEST_IOCTL_MAGIC  SP
//commands
#define TEST_FILL_ZERO    _IO(TEST_IOCTL_MAGIC, 1)
#define TEST_FILL_CHAR    _IOW(TEST_IOCTL_MAGIC, 2, char)
#define TEST_SET_SIZE    _IOW(TEST_IOCTL_MAGIC, 3, uint)
#define TEST_GET_SIZE _IOR(TEST_IOCTL_MAGIC, 4, uint)

#define TEST_MAX_COMMANDS  5
#define TEST_MAX_LEN 1024
```

# Device Control Interfaces

## ioctl() - IO control - K-space implementation

Implementation of drivers\_ioctl():

```
Int test_char_dev_ioctl(struct inode *inode, struct file *filp
                        unsigned int cmd, unsigned long arg)
{
    unsigned int i, size;
    char c;
    int bytes;
    if(_IOC_TYPE(cmd) != TEST_IOCTL_MAGIC)//to validate magic num
        return -EINVAL;
    if(_IOC_NR(cmd) > TEST_MAX_COMMANDS) //to validate seq num
        return -EINVAL;
    switch(cmd){
        case TEST_FILL_ZERO:
            for(i=0;i<TEST_MAX_LENGTH;i++)
                char_dev_buf[i] = 0;
            break;
    }
}
```

# Kernel source management tools.

## 1.Cscope:

<http://cscope.sourceforge.net/>

- ☐ Tool to browse source code (mainly C, but also C++ or Java).
- ☐ Can be used from editors like vim and emacs.

## 2.Kscope:

<http://kscope.sourceforge.net>

- ☐ A graphical front-end to Cscope
- ☐ Makes it easy to browse and edit the Linux kernel sources
- ☐ Can display a function call tree
- ☐ Nice editing features: symbol completion, spelling checker, automatic indentation...

## 3.Source navigator:

## 4.Source insight:

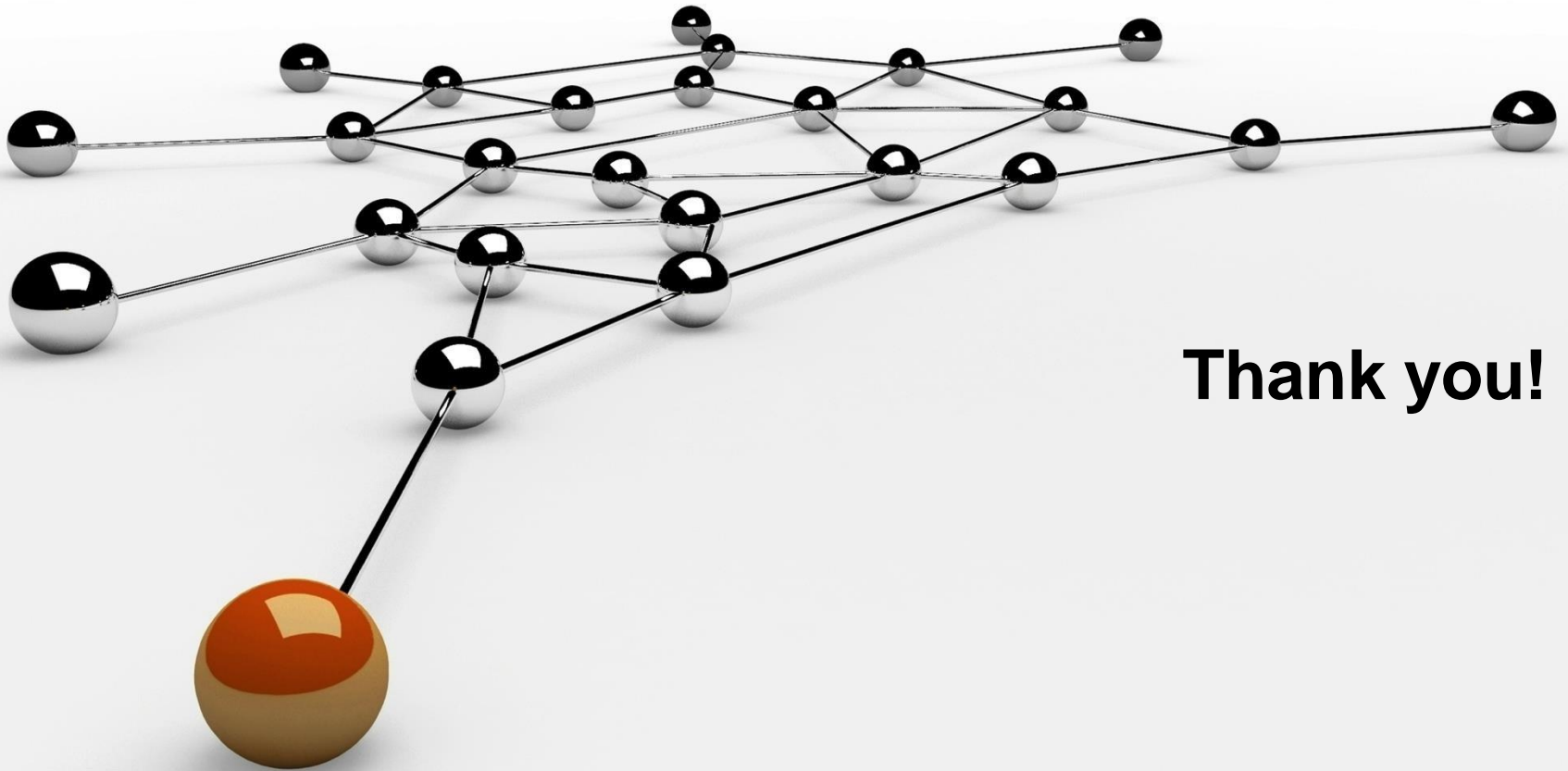


# Books for Ref..

## Books:

- Understanding the Linux Kernel, D. P. Bovet and M. Cesati, O'Reilly & Associates, 2000.
- Linux Core Kernel – Commentary, In-Depth Code Annotation, S. Maxwell, Coriolis Open Press, 1999.
- TheLinux Kernel, Version 0.8-3, D. ARusling, 1998.
- Linux Kernel Internals, 2<sup>nd</sup> edition, M. Beck et al., Addison-Wesley, 1998.
- Linux Kernel, R. Card et al., John Wiley & Sons, 1998.
- Linux Device Drivers, 3rd Edition, Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman Published by O'Reilly Media, Inc., 1005

# Questions ??



**Thank you!**