IGEL

Introduction to the Linux Device Driver

Takanari Hayama
IGEL Co.,Ltd.
1-3-6 Mirokuji, Fujisawa
Kanagawa 251-0016, Japan
TEL: +81-466-29-3505
FAX: +81-466-29-3506
E-mail: taki@igel.co.jp

IGEL

Linux Kernel Basics

IGEL

History of Linux

- Created by Linus Torvalds, who was a student of University of Helsinki in Finland, based on Minix in 1991.
- Linux 0.01 born in August 1991.
  - Linux ran on the Minix system at this time.
- First official release of the Linux Ver.0.02 was in October 1991.
  - Several GNU software like gcc and bash ran on Linux
- At the time of Ver.1.0 kernel development, approx. 80 developers contributed.
- For the Ver.2.0 kernel, approx.190 developers contributed officially.

IGEL

Versioning Rule

- Minor Version
  - Even number release is a "Stable Release", e.g. 2.2.14.
  - Odd number release is a "Development Release", e.g. 2.3.39.
- Development Release
  - All new features are tested in this release first.
  - Not Stable.
- Stable Release
  - For general use.
  - All new features introduced in the old development release is included.

IGEL

Features of Linux Kernel

- POSIX.1 compliant kernel distributed under GNU GPL
  - Looks like UNIX but slightly different.
  - Some of the POSIX.4 functionalities are supported.
  - Latest kernel can be obtained from the web like http://kernelnotes.org/.
- Variety of the supported platform
  - x86, PowerPC, Alpha, Sparc, ARM, MIPS
  - On-going development on PA-RISC and other platform
- Shows good performance with the less computation resource (CPU, Memory, Disk Space etc.)
- Has lots of nifty functions.
- Neat functions for device driver development:
  - Loadable Kernel Module
  - /proc Filesystem

C 2000 IGEL Co.,Ltd.

Features of Linux Kernel (contd.)

- Multi-process, Multi-processor
- Multi-platform
- Multi-user
- Inter-process Communication
- POSIX Compliant Terminal, PTY(pseudo terminal)
- Variety of Peripheral Devices like SCSI, Sound Card, Graphics Card, NIC and so on
- Buffer Cache for I/O
- Demand Paging
- Shared Library with Dynamic Link Capability
- Variety of Supported File System (ext2fs, Fat16/32, ISO-9660 etc.)
- Network Protocol like TCP/IP (including IPv6)

C 2000 IGEL Co.,Ltd.

IGEL

Compiling Linux Kernel

- Untar the source under `/usr/src`

```
# cd /usr/src
# rm /usr/src/linux
# bzip2 -cd linux-2.2.14.tar.bz2 | tar xvf -
# mv linux linux-2.2.14
# ln -s linux-2.2.14 linux
```

- Configure Kernel

```
# cd linux
# make menuconfig OR make config
```

If you are running X, you can try `make xconfig` .

- Compile

```
# make dep; make clean; make bzImage
# make modules ; make modules_install
```

Installing Linux Kernel

- Copy kernel to `/` or `/boot` depending on your configuration

```
# cd /usr/src/linux
# mv /vmlinuz /vmlinuz.old
# cp arch/i386/boot/zImage /vmlinuz
```

- If you are using lilo, don't forget to execute lilo. Then reboot.

```
# lilo
# reboot or Ctrl+Alt+Del
```

N.B. In the case of misconfiguration of the kernel, you'd better to keep the old kernel. Edit `/etc/lilo.conf`, to make it bootable with the old kernel.

© 2000 IGEL Co.,Ltd.

IGEL

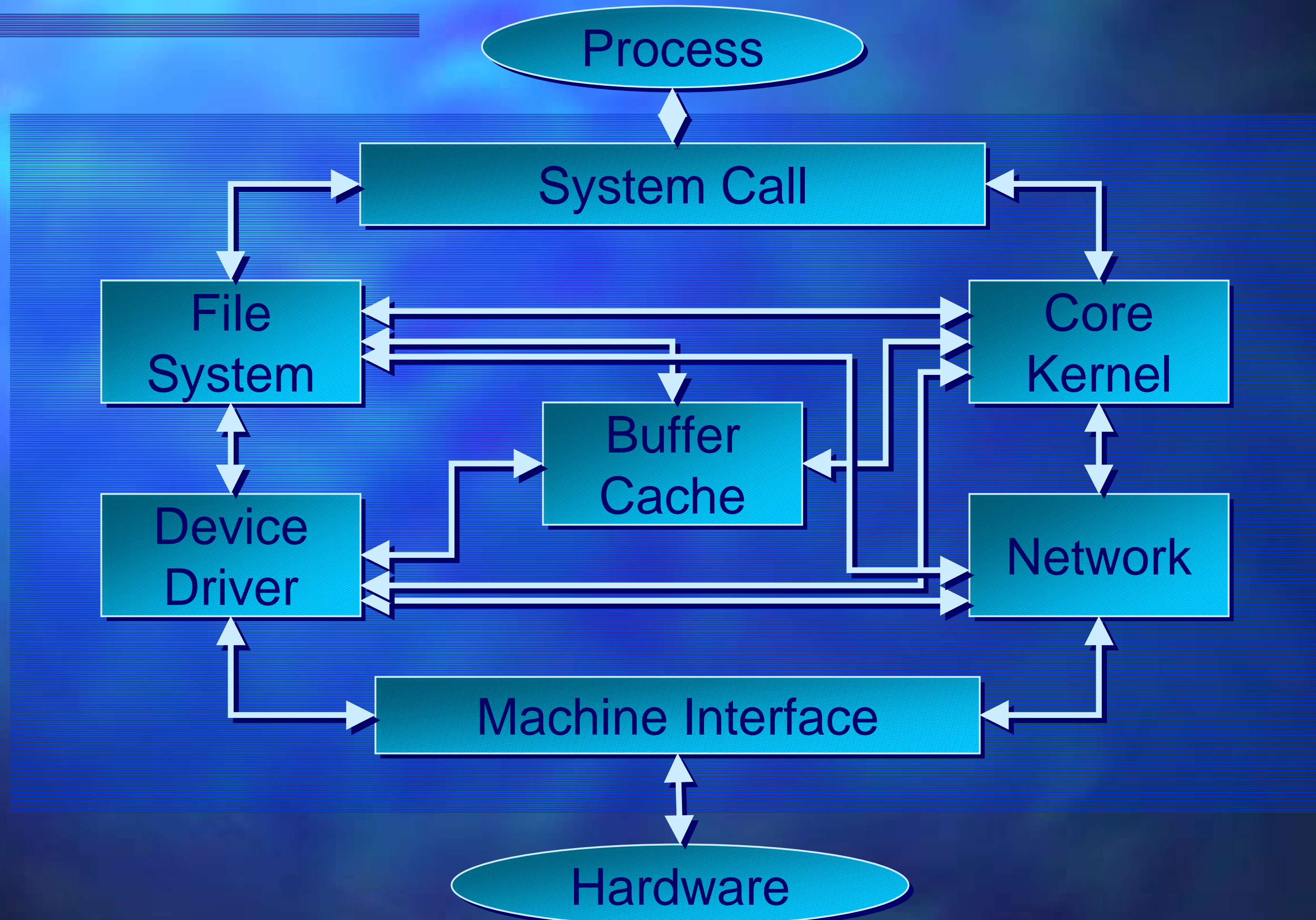
Important Directories

- /usr/src/linux
  - Kernel source code
- /usr/include/linux
  - Platform independent Linux kernel header files.
  - Generally, it is symlink to /usr/src/linux/include/linux.
- /usr/include/asm
  - Platform dependent Linux kernel header files.
  - Generally, it is symlink to /usr/src/linux/include/asm.
- /usr/include/scsi
  - SCSI driver related header files.
  - Generally, Symlink to /usr/src/linux/include/scsi

Linux Kernel Internal

IGEL

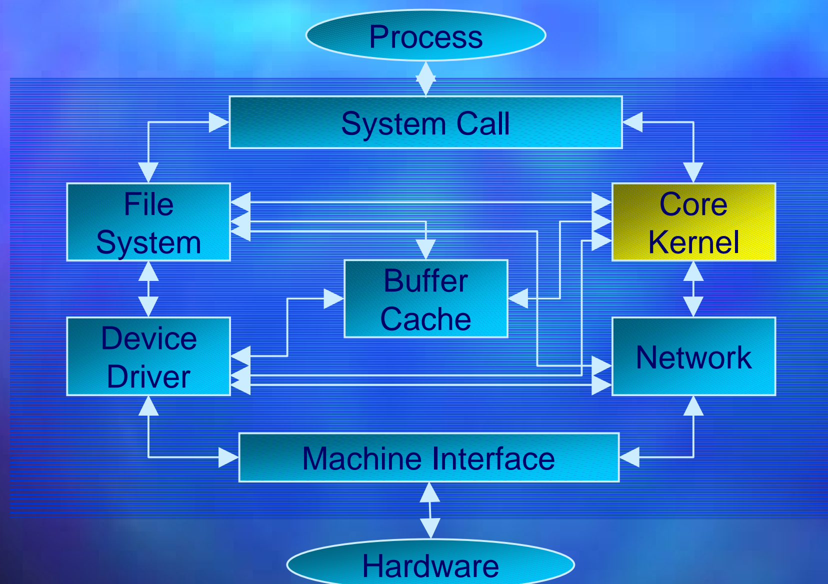
Kernel Structure



C 2000 IGEL Co.,Ltd.

IGEL

Core Kernel

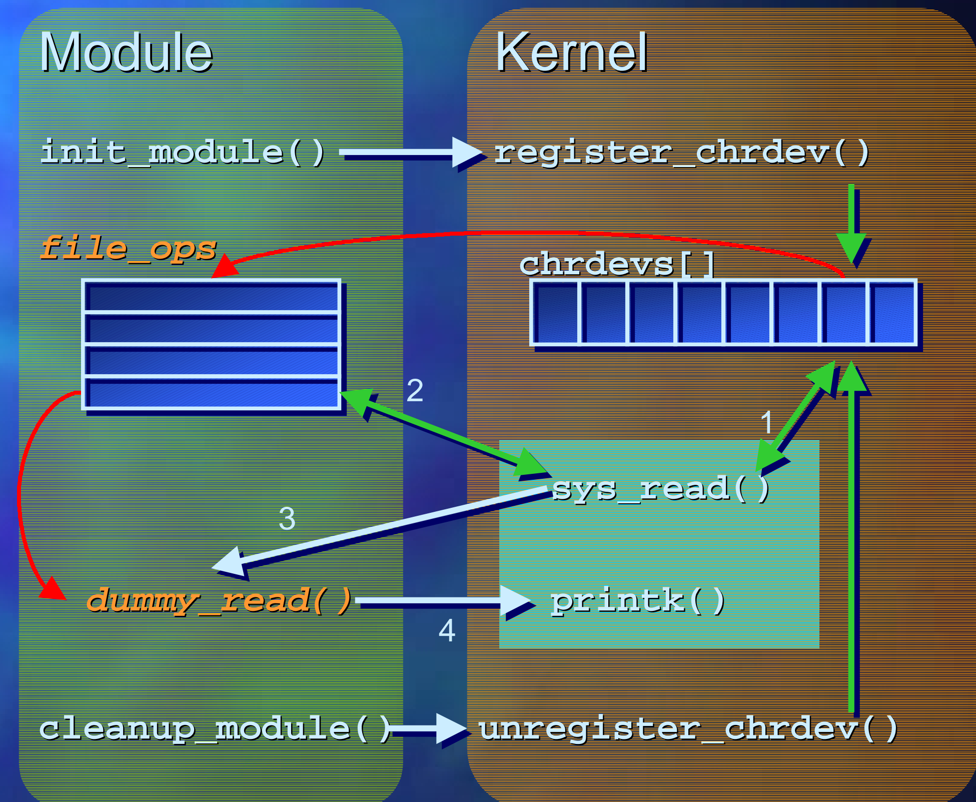


- The subsystem which provides core service of the kernel
  - Scheduler
    - Process Creation and Management
  - Signal
    - Interruption Handling
  - Loadable Module
    - Dynamic Loading of the Kernel Module like Device Driver
  - Memory Management
    - Allocation of the Physical Memory

C 2000 IGEL Co.,Ltd.

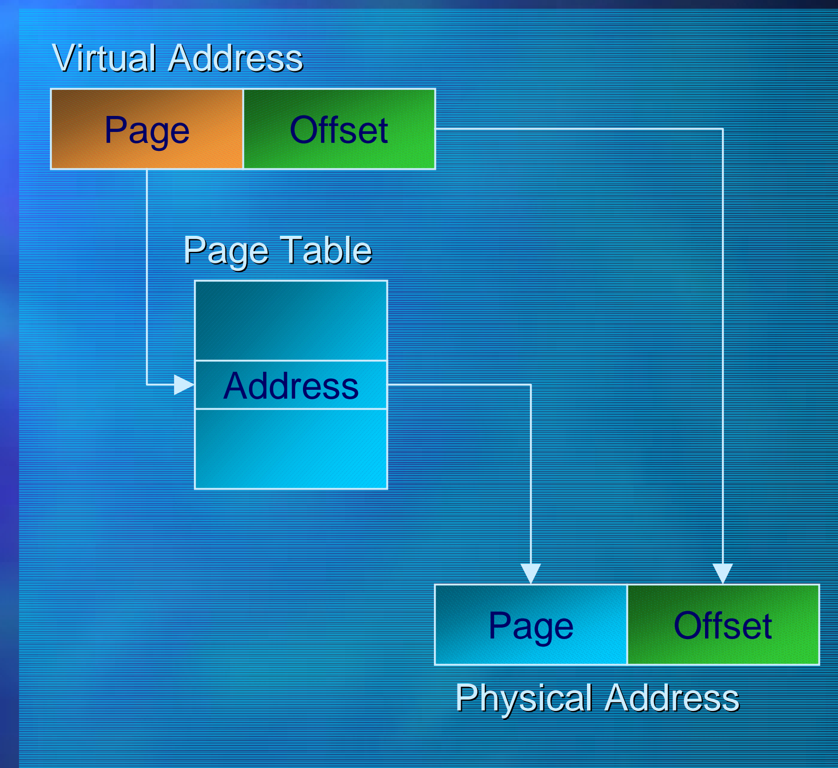
Loadable Module

- Support dynamic load and unload of kernel modules
- Add and update device drivers or file systems without stopping nor rebooting the system
- Make kernel as small as possible by loading the features on demand after boot up



Memory Management

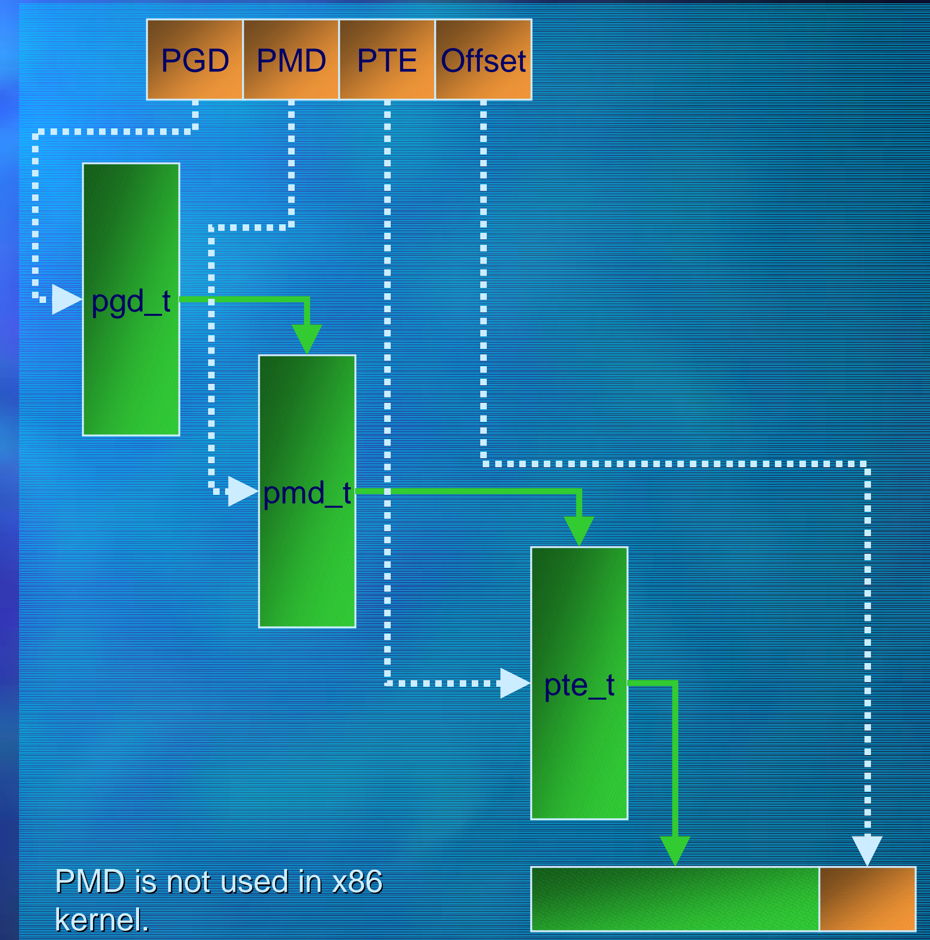
- Generally, a physical memory is less than 4GB
- To use memory efficiently, a mechanism to change from linear address (virtual address) to non-linear address (physical address) is used
- The kernel maintain the lookup table for the virtual address to the physical address conversion



C. 2000 IGEL Co.,Ltd.

Memory Management in Linux

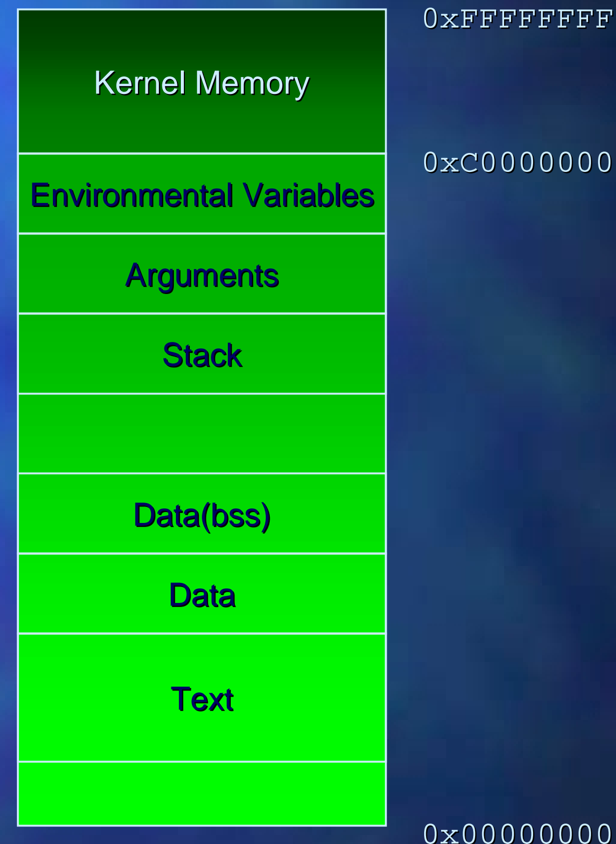
- Linux employs three types of page table for addressing:
  - PGD. PaGe Directory.
  - PMD. Page Mid-level Directory.
  - PTE. Page Table Entry.
- This multi-level page tables reduce the size of page table required at the same time



IGEL

Virtual Memory

- Each process consists of the several memory object (segment):
  - Text Segment
  - Data Segment
    - Including BSS. Block Started by Symbol.
  - Stack Segment
- Programs and Shared Libraries are mapped to the Process. s virtual memory space as memory objects (Memory Map).



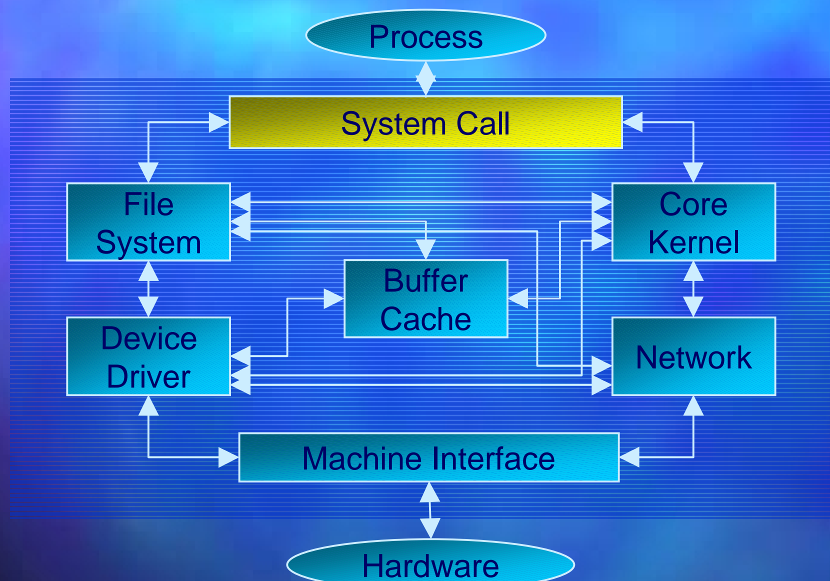
. C. 2000 IGEL Co.,Ltd.

IGEL

Virtual Memory (contd.)

```
Sc.taki[7:28am]% cat /proc/self/maps
08048000-0804d000 r-xp 00000000 03:01 258103      /bin/cat
0804d000-0804e000 rw-p 00004000 03:01 258103      /bin/cat
0804e000-08050000 rwxp 00000000 00:00 0
40000000-40006000 r-xp 00000000 03:01 276549      /lib/ld-linux.so.1.9.9
40006000-40007000 rw-p 00005000 03:01 276549      /lib/ld-linux.so.1.9.9
40007000-40008000 rw-p 00000000 00:00 0
4000b000-40093000 r-xp 00000000 03:01 276503      /lib/libc.so.5.4.46
40093000-40099000 rw-p 00087000 03:01 276503      /lib/libc.so.5.4.46
40099000-400cb000 rw-p 00000000 00:00 0
bfffd000-c0000000 rwxp ffffe000 00:00 0
```

System Call



- Interface to receive services from Kernel.
- Generally, to protect hardware from processes, each process run in User Mode.
- In Linux, software trap is generated when system call is issued.
- After trap, it switches to the Kernel Mode and process according to the system call.

IGEL

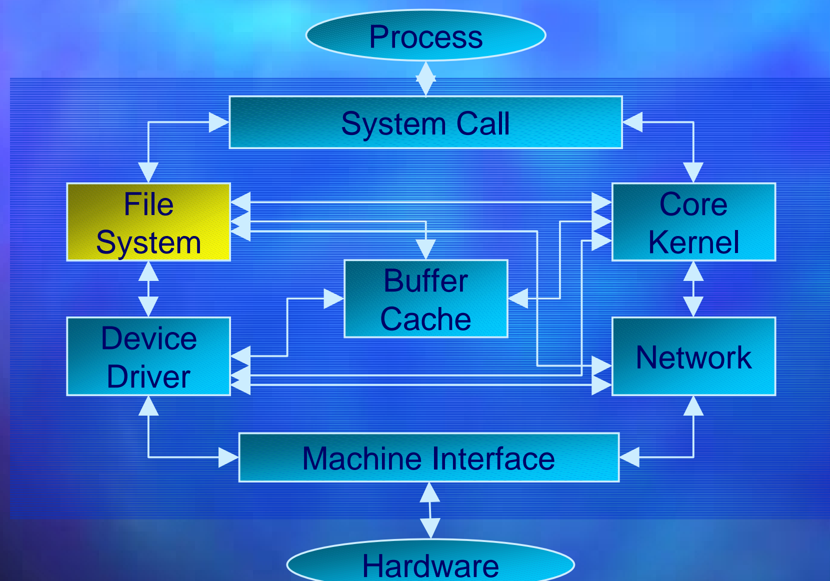
Personality

- Binaries of other UNIX-like OS are executable under Linux
- Depending on the binary, the system call and signal related part needed to be personalized
- Linux allows to change a personality by issuing system call `personality`(2)

- Changeable Personality
  - System Call
    - Different way of issuing system call (trap and segment jump)
    - Different system call numbers
  - Signal from and to Process
    - Different signal number is used for each OS
    - Lookup table is referred for system call like `sigaction` and `sigkill`

IGEL

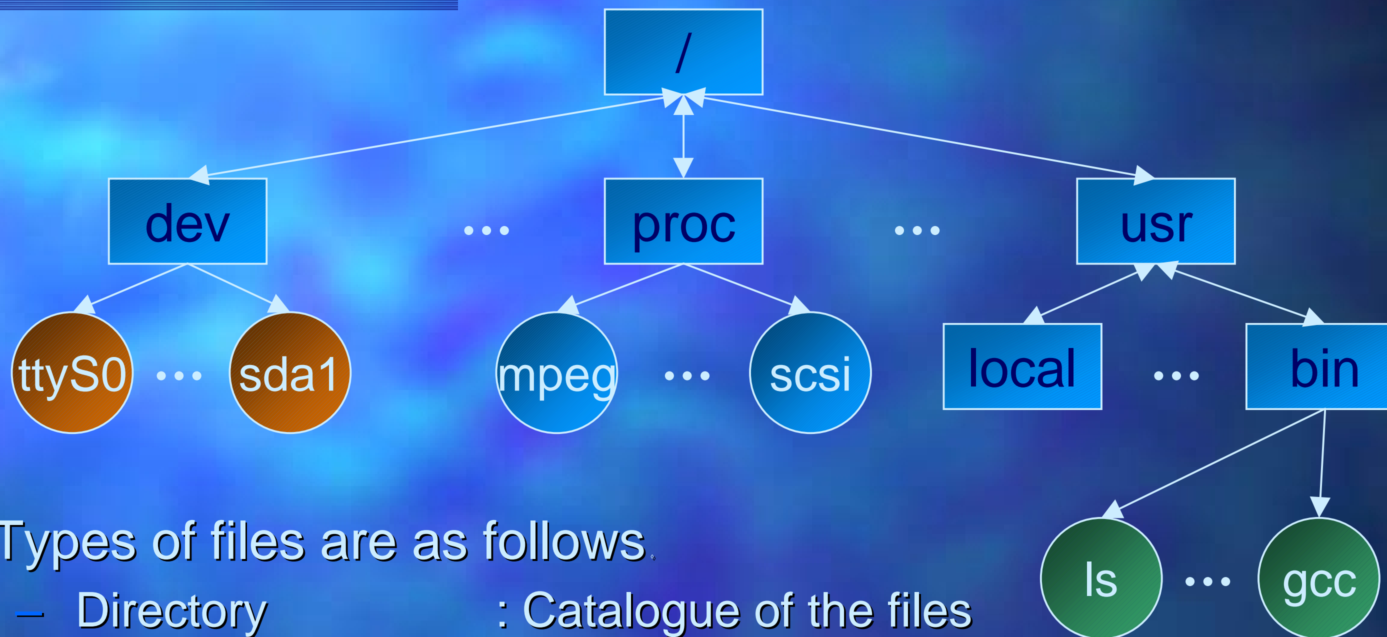
File System



- Linux has tree structured file system like other UNIX-like systems
- Linux supports wide variety of file system:
  - e.g. Ext2 FS, Ext FS, Fat16/32 , VFAT , SMB FS, NTFS, UFS, HPFS, OS/2 , ISO-9660, Umsdos, Minix FS, MS-DOS, NFS, Amiga FS, SysV/Coherent FS, Proc FS, etc.

C 2000 IGEL Co.,Ltd.

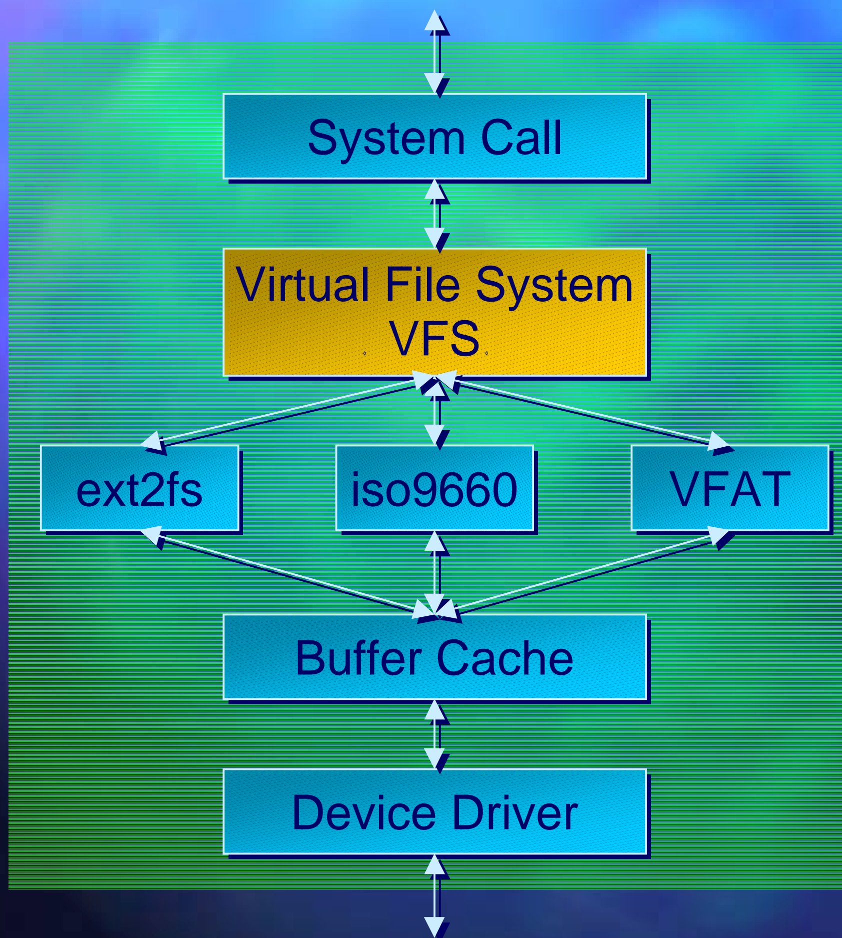
Hierarchical Structure of File System



- Types of files are as follows.
  - Directory : Catalogue of the files
  - Normal File : Data or Binaries
  - Symbolic Link : Pointer to a file or a directory
  - <u>Special File</u> :<u>Entry to a device driver</u>
  - Named Pipe : Inter Process Communication

C. 2000 IGEL Co.,Ltd.

VFS



- Meta level file system
- Independent from real device
- Selects appropriate modules according to user's request
  - All file accesses are dispatched by VFS
- Each file system module access to its device driver thru buffer cache
  - In current kernel, buffer cache cannot be disabled!!

C 2000 IGEL Co.,Ltd.

VFS and inode

- VFS provides
  - Translation of a filename to a pair of device number and inode number
  - Optimization of file access using buffer cache
- VFS works as follows:
  - *Check Arguments*
  - *Translate a filename to a pair of device number and inode number*
  - *Check Permission*
  - *Call appropriate method according to file system*
- Following operations needed to be implemented for each file system:
  - Operations related to File System
  - Operations related to inode
  - Operations related to file
  - Operations related to Quota
- inode is an identifier used to distinguish files in kernel
  - File Attribute Information
    - Owner and permission
  - Pointer to operations for handling file

C 2000 IGEL Co.,Ltd.

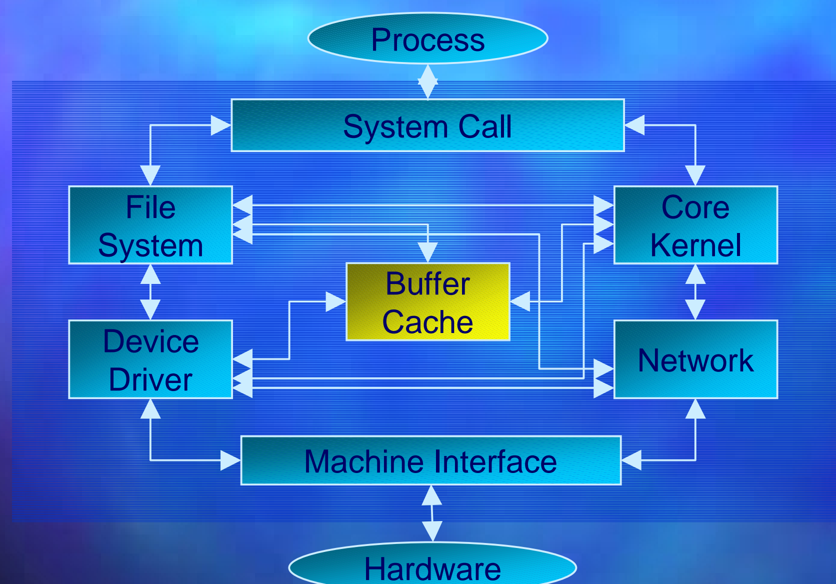
VFS(contd.)

- Operations related to File System
  - Operations related to a physical structure of a file system
  - E.g. Read/write inode, etc
- Operations related to inodes
  - Operations directory manipulate inodes
  - E.g. Delete a file, change permission and so on.
- Operations related to a file
  - Operations related to each file
  - E.g. Read/write file
- Operations related to Quota
  - Operations to realize quota mechanism
  - E.g. Determine whether blocks and inodes are used appropriately

C 2000 IGEL Co.,Ltd.

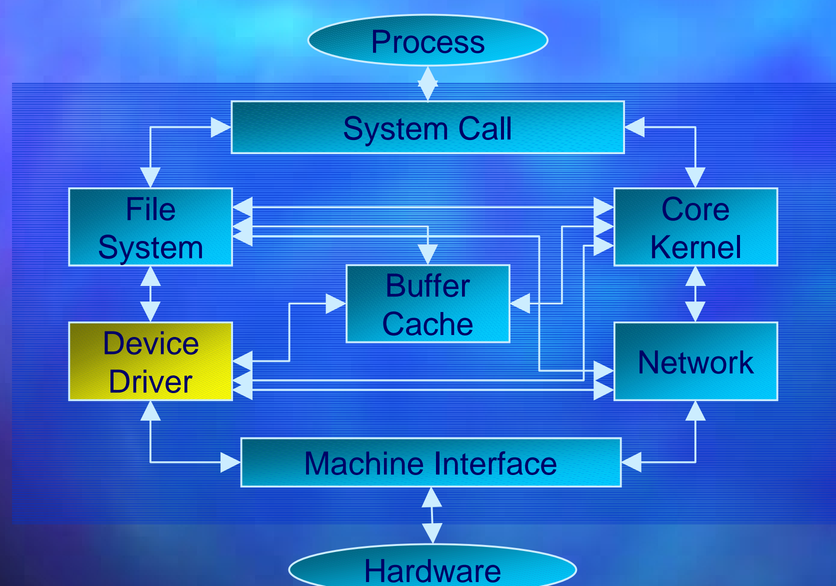
IGEL

Buffer Cache



- The buffer cache caches data from a block device like a hard disk.
- Generally, a disk access is performed through the buffer cache.
- Buffer Cache is flushed every time -
  - Max number of buffer blocks become dirty,
  - Several number of buffers are changed,
  - The `update` process called the system call `bdflush`(2)

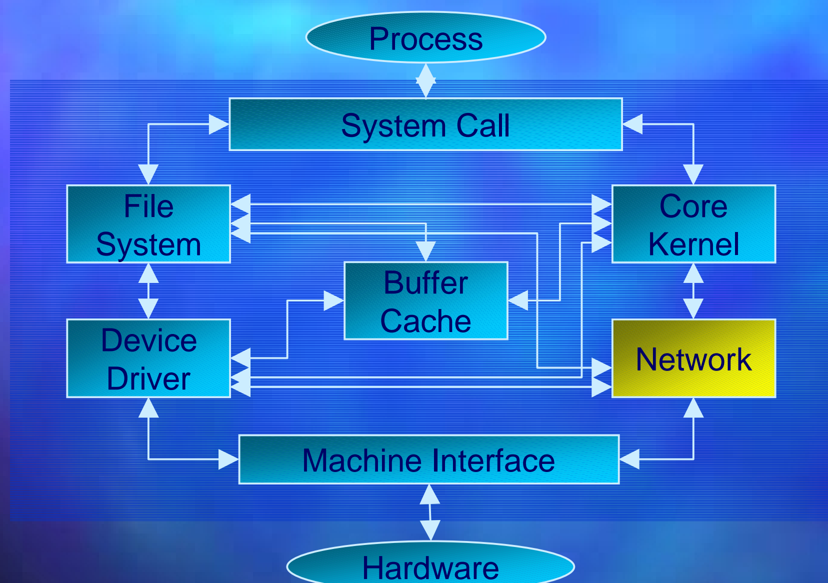
Device Driver



- A device driver is a software module for utilizing a specific hardware
- Linux has four different types of device drivers:
  - ***Character Device***
    A device treatable like a file
  - ***Block Device***
    A device which can host a file system
  - ***Network Interface***
    A physical network device
  - ***SCSI Interface***
    A SCSI interface device

IGEL

Network



- Network subsystem is implemented as an internal service as it is a process independent service
  - Serialize packets which arrive asynchronously and reconstruct a correct data
  - Translate requested data from process into packets and transmit them
- Under Linux, network driver is not visible from /dev directory
  - Data read from the device is not a stream
  - All network facilities are accessed thru a socket

Network Functions Supported by Linux Kernel

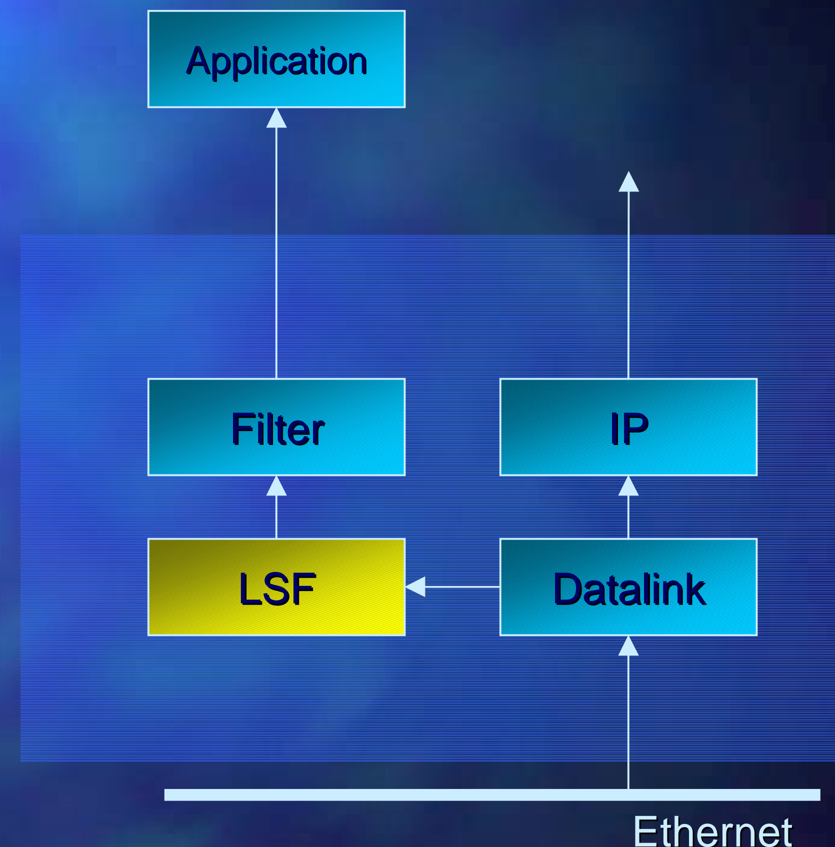
IGEL

- IP Masquerade
  - Translate IP address like NAT
  - http://ipmasq.cjb.net/
- Firewall
  - Support filtering type firewall within kernel (`ipfwadm` is required)
- Many good functions for a router
  - Fast Routing Function
  - QoS
- Variety of supported network protocols:
  - TCP/IP, IPv4, IPv6*,
  - PPP/SLIP
  - IPX
  - AppleTalk
  - X.25
  - Acorn Econet
- Linux Socket Filter
  - For monitoring packet (similar to the BSD Packet Filter)

IGEL

Linux Socket Filter

- BSD Packet Filter like mechanism
- Arbitrary filter written in machine language of pseudo machine can be set to filter packet coming from datalink layer
- Syntax of filter is exactly the same as BSD Packet Filter



IGEL

LSF and BPF Differences

- Use `<linux/filter.h>` instead of `<net/bpf.h>`.
- Use `struct sock_filter` instead of `struct bpf_insn` to define filter.
- To attache and detach filter, create a socket and call `setsockopt`(2) to set filter as follows:
  - `setsockopt(sockfd, SOL_SOCKET, SO_ATTACH_FILTER, &Filter, sizeof(Filter));`
  - `setsockopt(sockfd, SOL_SOCKET, SO_DETACH_FILTER, &value, sizeof(value));`

SOCK_PACKET

- To monitor all packet easily, use `SOCK_PACKET` option to create socket:
  - `socket(AF_INET, SOCK_PACKET, htons(ETH_P_ALL));`
- Types of packets which can be filtered is defined in `<linux/if_ether.h>`.
  - All Packet `ETH_P_ALL`
  - IP Packet `ETH_P_IP`
  - ARP Packet `ETH_P_ARP`
  - etc.

IGEL

Linux Device Driver Basics

C 2000 IGEL Co.,Ltd.

Kernel Space Drivers vs. User Space Drivers

- Kernel Space Drivers
  - Better performance
  - Direct memory access
  - All kind of device driver can be handled
- User Space Drivers
  - The full C library can be linked in
  - Easy to debug
  - Even it hangs, the whole system won t hang
  - Huge memory can be used

Linux Device Driver Basics

- Concept of Kernel Programming
  - Linux Kernel has a kernel thread which allows multi-thread programming
    - SMP configuration is full multi-thread kernel
  - Generally, within the kernel, a scheduling is done in non-preemptive manner
    - Except for a hardware interrupt
  - Kernel runs in event-driven manner
    - System Call
    - IRQ
  - As device drivers have a chance to be used by several processes, it is important to save contexts for each tasks

C 2000 IGEL Co.,Ltd.

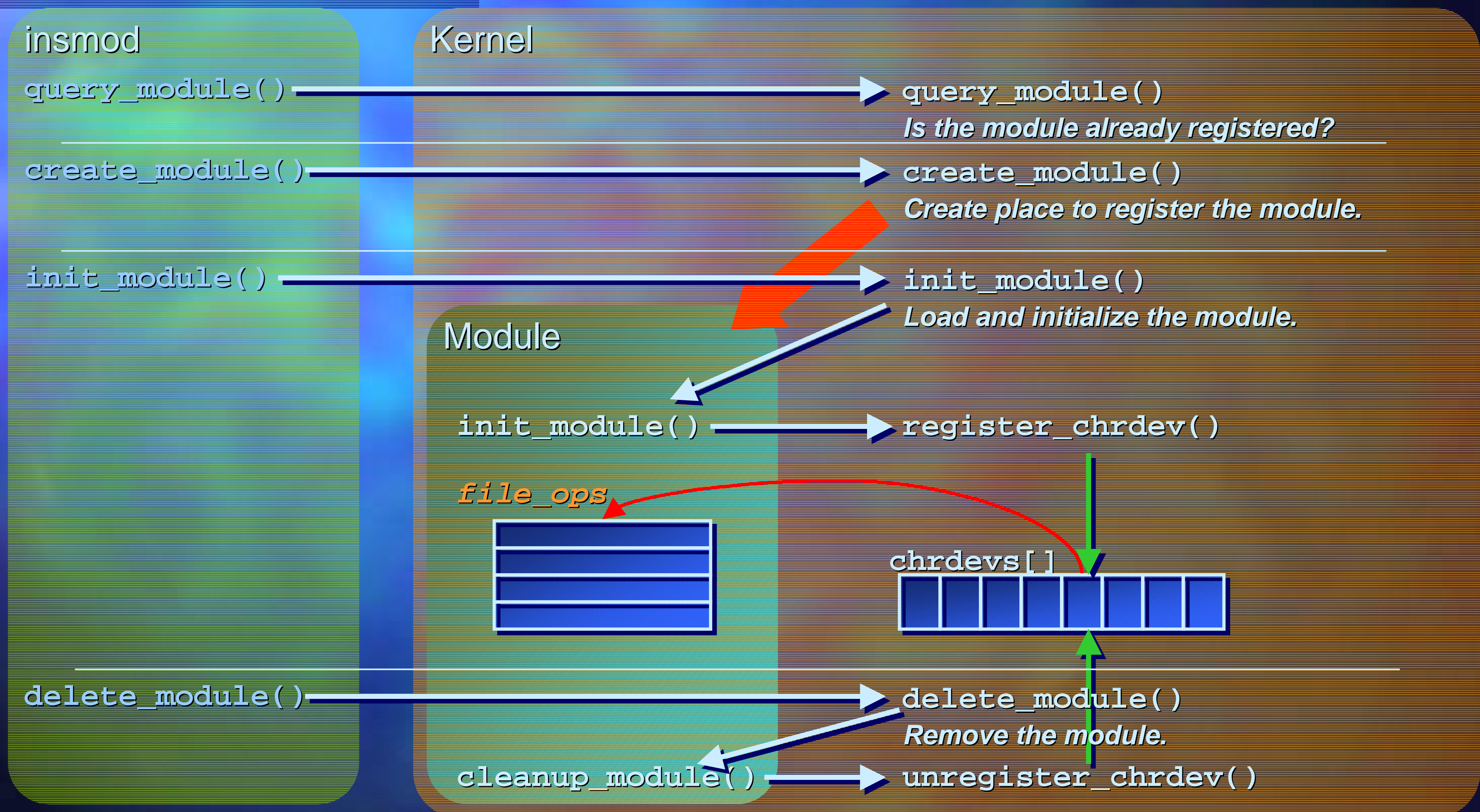
Types of Linux Device Driver

- Character Device
  - A stream-oriented device which can be used like a file
  - E.g. `/dev/tty1`, `/dev/lp1`, etc.
- Block Device
  - A device which can host a file system and can be accessed by units of blocks (generally, 1KB)
  - Under Linux, it is possible to access by arbitrary numbers of bytes
  - E.g. `/dev/hda1`, `/dev/sda1`, etc.
- Network Device
  - A physical layer of a network
  - Cannot be accessed from an ordinal file system, as it is not stream-oriented device
- SCSI Device
  - A driver to control SCSI interface card
  - Implement only routines to drive SCSI controller (Protocols sent thru SCSI controller are the same)

Load and Unload Device Drivers

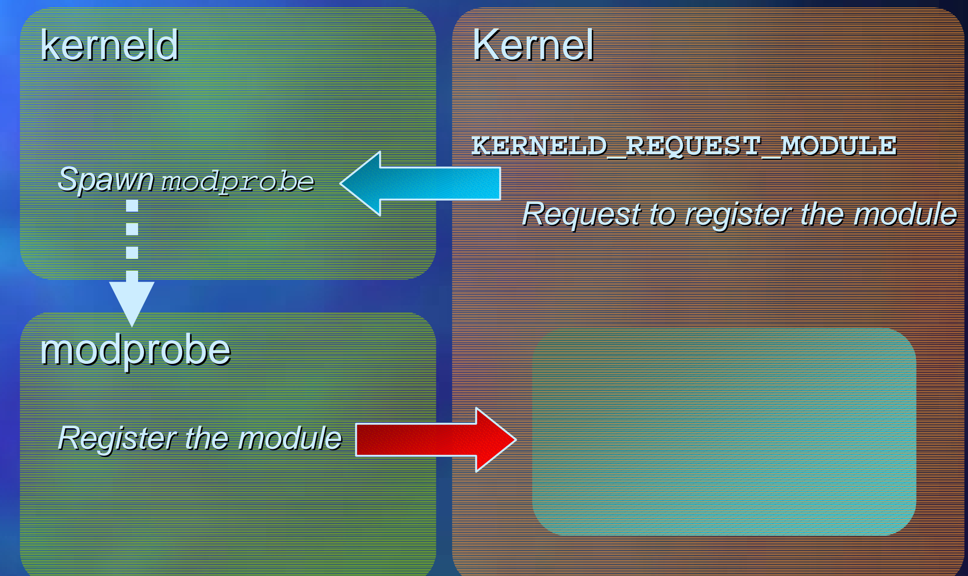
- Under Linux, device drivers can be loaded on demand as loadable modules
- A module can be loaded and unloaded dynamically
  - Use the `modprobe` command to load, and the `rmmod` command to unload
    - `# lsmod` (List Modules)
    - `# modprobe mpegdecode` (Load Modules)
    - `# rmmod mpegdecode` (Unload Modules)
- Global variables may be changed at load time
  - `# modprobe irq=10 sample`
- By using the `kerneld` daemon, modules can be loaded automatically on demand
  - A module is loaded when someone tries to access the device first time
  - SystemV IPC and kerneld options must be turned on to use the `kerneld`
- Device drivers which are not required at boot time may better to be created as modules
  - Much easier to maintain

Load/Unload Modules



Automatic Registration of a Module

- The registration of a module is performed by the user-level process `kerneld`.
- The kernel sends request to the daemon using SystemV IPC message.
- The `kerneld` invokes `modprobe` or `rmmod` commands according to the request.
  - See `<linux/kernel.h>` for more detail.



IGEL

A Template of a Module

```
/*
 * A template of a Module
 */

#define MODULE
#define MODVERSIONS
#include <linux/module.h>
#include <linux/modversions.h>

/*
 * Initialize the module
 */
int init_module(void)
{
  /*
   * Register device driver here.
   */
  printk(. Hello World!\n. );
}
```

```
/*
 * Remove the module
 */
int cleanup_module(void)
{
  /*
   * You must perform clean up
   * in this function. If you
   * don. t do it, then you. ll
   * observe a memory leak and
   * other nightmares.
   */
  printk(. Good-bye World!\n. );
}
```

IGEL

Compiling a Module

- Don't forget to define `__KERNEL__` and `MODULE` to compile module.
- If you decided to set version number to the kernel symbols, don't forget to include `<linux/modversions.h>`
  - This is the kernel compile option
  - By setting this option, you have a chance to load your module without recompiling it
- If the modules consists of several objects, don't forget to set '-r' option for relocatable object.

```
#
# Sample Makefile for compling a module
#
INCS = -I/usr/include
CFLAGS = -D__KERNEL__ -O -Wall $(INCS)

OBJS   = sample1.o sample2.o
TARGET = sample.o

# Multiple objects
all: $(OBJS)
    ld -m elf_i386 -r -o $(TARGET) $(OBJS)

# Single objects
single: single.o
```

Basics of a Module Implementation

IGEL

- Usage count of a module
  - To determine whether the module can be unloaded safely, use the counter:
    - `MOD_INC_USE_COUNT` : Increment the counter
    - `MOD_DEC_USE_COUNT` : Decrement the counter
    - `MOD_IN_USE` : True if the counter is not zero
- Version dependency
  - If the source code depends on the specific kernel version, you may check by evaluating the `LINUX_VERSION_CODE` macro

```
#if LINUX_VERSION_CODE < KERNEL_VERSION(2,2,0)
/* Version dependent code */
#endif
```

Basics of a Module Implementation(contd.)

- If the module needed to be accessed by others, the symbol must be exported as follows:
  - In the case of the 2.0.x kernel

```
static struct symbol_table sample_syms = {
#include <linux/symtab_begin.h>
        X(sample_func),
#include <linux/symtab_end.h>
};
int init_module(void) {
  /* ... */
  register_symtab(&sample_syms);
  /* ... */
}
```

  - In the case of the 2.2.x kernel

```
#define EXPORT_SYMTAB
#include <linux/module.h>
EXPORT_SYMBOL(sample_func);
```

Accessing to the Memory and Mutual Exclusion

IGEL

- Don't forget to declare a pointer as `volatile` when you access to shared data to skip optimization by the compiler
- If you want to make a critical region, use the following steps:
  - ```
    unsigned long flags;
    save_flags(flags);
    cli(); /* Clear IRQ */
    /* Critical region */
    restore_flags(flags);
    ```
- A lock variable using the bit operators
  - ```
    while(set_bit(nr, addr)
                          != 0)
       sleep_for_a_while();
    /* Critical region */
    if (clear_bit(nr, addr)
                          == 0)
       /* error */
    ```
- Atomic operations
  `<asm/atomic.h>`
  - `atomic_add()`
  - `atomic_sub()`
  - `atomic_inc()`
  - `atomic_dec()`
  - `atomic_dec_and_test()`

C 2000 IGEL Co.,Ltd.

Functions for the Memory Management

IGEL

- **`void *kmalloc(unsigned int size, int priority)`**
  - Allocate the physical memory in the kernel. `GFP_KERNEL` or `GFP_ATOMIC` may set for `priority`. `GFP_KERNEL` allows to delay when the number of pages left is less than `min_free_pages`. `GFP_ATOMIC` allocate the memory independent of `min_free_pages`. If the memory is used for DMA, set `GFP_DMA` with `GFP_ATOMIC` or `GFP_KERNEL`(Don. t set this flag for PCI peripheral).
- **`void kfree(void *obj)`**
  - Free allocated kernel memory.
- **`unsigned long get_free_page(int priority)`**
  `unsigned long __get_free_page(int priority)`
  `unsigned long __get_free_pages(int priority, unsigned long order)`
  - Allocate new pages. The function `get_free_page` initialize pages with 0, but the function `__get_free_page` doesn. t. The function `__get_free_pages` allocate order pages of memory(Doesn. t initialize).

Functions for the Memory Management(contd.)

IGEL

- `void *vmalloc(unsigned int size)`
  - Allocate virtual memory in the kernel. CPU can refer the memory properly, but the peripheral devices like PCI cannot access directly. Returned address is the higher than the physical address.
- `void *vremap(unsigned long offset, unsigned long size)`
  `void *ioremap(unsigned long offset, unsigned long size)`
  - Remap the memory segment from `offset` for `size` bytes to the higher memory space, so that the address can be access directory from the kernel. Under the 2.1.x kernel or later, the function `ioremap` is used. This function can be used to make registers on the PCI peripheral directory from the CPU.
- `void vfree(void *obj)`
  `void iounmap(void *obj)`
  - Release allocated memory.

IGEL

Notes on the kmalloc

- The size of memory allocated by the function like `kmalloc` is fixed.
- Under the 2.0.x kernel, the size of memory allocated is slightly smaller than proportional to the power of 2.
  - E.g. If 2048bytes is asked to allocate, 4096bytes is allocated as it includes a header.
- The maximum size of memory allocatable by the kmalloc is 32pages(i.e. 128KB for 32bit CPU)
- The address returned by kmalloc:
  - For 2.0.x kernel the physical address
  - For 2.1.x kernel and later the virtual address
    - `paddr = virt_to_phys(vaddr);`
    - `vaddr = phys_to_virt(paddr);`

IGEL

Allocating Larger Memory

- Tell the kernel that you have less memory than you have actually
  - Under the system with 32MB RAM, write
    `append=" mem=31m"`
    in the `/etc/lilo.conf`. You get the 1MB clean physical memory!
- To access the memory from the user process, open `/dev/mem` and `mmap(2)` from 31MB(31 * 0x100000) for 1MB.
- To access the memory from the kernel, access directory as follows:
  - 2.0.x kernel
    ```
    #define BASE (31*0x100000)
    char *ptr = BASE;
    ```
  - 2.1.x kernel and later
    ```
    char *ptr = __va(BASE);
    ```
- The programmer is responsible for mutual exclusion and other issues as the memory is shared.
- This method can be applied for those device driver needs huge memory for DMA and etc.

Delaying Execution

- How to get the current time
  - The kernel variable `jiffies`
    The time ticks incremented by 100Hz timer interrupt. (1024Hz for Alpha)
  - `do_gettimeofday`
    The entity of the system call `gettimeofday`
- The way to delay execution
  - Busy Wait
  - Rescheduling
  - Task Queue
  - Kernel Timer

- Busy Wait
  - Performs busy wait for 1000usec by `udelay(1000)`.
  - As it wastes CPU, don. t use it for long delay.
- Rescheduling
  - Suspend the current execution and to itself back to the wait queue
  -
```
current->timeout = x;
current->state =
  TASK_INTERRUPTIBLE;
schedule();
current->timeout = 0;
```
  - The variable `x` is `jiffies`

IGEL

Task Queue

- By adding the task to the list, you can execute the task later on.
  - E.g. The bottom half in the interrupt handler.
- The list of queue prepared by the kernel
  - `tq_scheduler`
    The queue used for the normal tasks
  - `tq_timer`
    The queue for the tasks executed by the timer
  - `tq_immediate`
    The queue for the tasks needed to be executed as soon as possible

```
struct tq_struct sample_task;
sample_task.routine = sample_bh;
sample_task.data = (void*)NULL;

int sample_intterupt()
{
  /* Some task here */
  queue_task(&sample_task, &tq_scheduler);

#if 0
  /* for the BH add to the tq_immediate */
  queue_task(&sample_task, &tq_immediate);
  mark_bh(IMMEDIATE_BH);
#endif

  return 0;
}

int sample_bh(void *dummy)
{
  /* Some task here */
  return 0;
}
```

IGEL

Kernel Timer

- If the task needed to be delayed for some definite amount of time, use the kernel timer
- By registering the task and the desired time, the task is invoked when the kernel reaches to the specified time.
  - void init_timer(struct timer_list *timer)
    void add_timer(struct timer_list *timer
  - void del_timer(struct timer_list *timer)

```
struct timer_list sample_timer;
struct wait_queue *wait = NULL;

void sample_timeout(unsigned long x)
{
  wake_up_interruptible(&wait);
}

int sample_timer()
{
  /* Initialize timer */
  init_timer(&sample_timer);

  sample_timer.expires = jiffies + TIME;
  sample_timer.data     = 0;
  sample_timer.function= sample_timeout;

  /* Register timer */
  addtimer(&sample_timer);
  interruptible_sleep_on(&wait);

  return 0;
}
```

Interrupt Handling

- The devices like PCI peripherals send interrupts to tell the arrival of data and etc.
- Linux supports the shared IRQ which is specified in PCI spec
  - Set SH_IRQ when you register IRQ handler
  - Basically, in the case of shared IRQ, the kernel just calls all interrupt handler those sharring the interrupt one by one

```
/*
 * Interrupt handling
 */
void init_module()
{
  /* . . . */
  request_irq(irq, irq_handler,
    SA_INTERRUPT | SA_SHIRQ, . sample. ,
    dev);
  /* . . . */
}

void irq_handler(int irq, void *dev,
    sturct pt_regs *regs)
{
  /* Process here */

  /*
   * If you have the bottom half:
   *   queue_task(&sample_bh,
   *              &tq_immediate);
   *   mark_bh(IMMEDIATE_BH);
   */
  return;
}
```

IGEL

Functions for a Interrupt Handling

- **`int request_irq(unsigned int irq, void (*handler)(int, void*, struct pt_regs*), unsigned long flags, const char *device, void *dev_id)`**
  - Enable interrupt vector `irq` and call `handler` when the kernel receives the `irq`. The `handler` is called with the argument `dev_id`. Specify `SA_INTERRUPT`(disable interrupt while handling interrupt), `SA_SHIRQ`(share IRQ) or `SA_SAMPLE_RANDOM`(the interrupt timestamp can be used to generate system entropy) for `flags`. The `device` is the name of the device appears in `/proc/interrupts`. Returns 0 if success.
- **`void free_irq(unsigned int irq, void *dev_id)`**
  - Release IRQ.

IGEL

Debugging a Device Driver

- Debugging by Printing
  - Insert check point which generate message in the code.
- Debugging by Querying
  - Use a special ioctl or a /proc file system to generate debug information
- Debugging System Faults
  - Use oops message generated by the kernel

IGEL

Debugging by Printing

- The most general debugging method
  - You cannot use stdio library as the kernel has no standard I/O.
- Use the printk which send the kernel output message to the klogd.
- The same arguments as printf can be given.

- The logging level can be defined.
(see <linux/kernel.h>)
  - printk(KERN_DEBUG . ….); is converted to printk(. <7>… .);

Debugging by Querying

- By using a /proc file system, user can access to the kernel information easily.
- No special tools are required for this task.

```
Di.taki[1:04am]% cat /proc/mpeg
Board 0
        RPS0 : no        RPS1 : no        Head0 : 0        Tail0 : 0
        Prg0 : -1  Exec0 : 0x00000000  Prog1 : -1  Exec1 : 0x00000000
        V4L: (704x480)->(704x480)@(0,0), 0bit, 0B/1, 0x00000000, Fmt=00
        DEBI: 0x066b0018 -> 0xffffffff (0B): 0x45430013(1B done)
        Stat0 : 0x00000000      BufSz: 16000B
    RPS Buffer 0
        0:0B(0x066b0018)->0xff  1:0B(0x0669c018)->0xff  2:0B(0x06698018)->0xff
        3:0B(0x06694018)->0xff  4:0B(0x06690018)->0xff  5:0B(0x0668c018)->0xff
        6:0B(0x06688018)->0xff  7:0B(0x06684018)->0xff  8:0B(0x06680018)->0xff
        9:0B(0x0667c018)->0xff
        RPS_I0: 0, RPS_A0: 0, RPS_A1: 0, RPS_I1: 0, FIDB: 0, VGT: 1
          MC1 : 0x00000900        MC2 : 0x00000053        ISR : 0x00000000
          PSR : 0x000008f8        SSR : 0x000000b0       ODD1 : 0xbdf785f7
        EVEN1 : 0x4e844655      PITCH1 : 0xfbf7daaf     HPSCTL : 0x50016000
        HPSVS : 0x5de27746      HPSVG : 0xf3fbf2dc      HPSHPS : 0xf27afa6e
        HPSHS : 0x6b9d35cf      HPSBCS : 0xe25a97e6     HPSCRM : 0xf3cdcb9e
        HPSFMT : 0xc97ffb65
```

Debugging by Querying(contd.)

- System call `ioctl` is used to send the device dependent request to the device driver.
- Advantage (compared to a /proc file system)
  - Output can be larger than a page (4KB)
  - Even the device driver has the debugging code, the user may not notice
- Disadvantage
  - The debugging tool for the device driver is required.
  - If the device driver is freezed in the `ioctl` routine, there is no way to debug

C 2000 IGEL Co.,Ltd.

IGEL

Using a /proc file system

- Register /proc file system handler using the following functions:
  - `proc_register` (for 2.1.x or later)
  - `proc_register_dynamic` (for 2.0.x)
- Register the name, the mode and the callback function
- The callback function generate the output for the read access

```
#include <linux/module.h>
#include <linux/proc_fs.h>

int sample_proc(char *buf, char **start, off_t
    offset, int len, int unused)
{
  len = sprintf(buf, "Hello World\n");
  return len;
}

struct proc_dir_entry sample_proc_entry = {
  0,                    /* inode */
  6, "sample",          /* filename */
  S_IFREG | S_IRUGO,    /* mode(<linux/stat.h>) */
  1, 0, 0, 0,          /* link,owner,group,size */
  NULL,                 /* operations */
  &sample_proc          /* callback function */
};

int init_module(void) {
  proc_register(&proc_root,
    &sample_proc_entry);
}

int clenaup_module(void) {
  proc_unregister(&proc_root,
    sample_proc_entry.low_ino);
}
```

© 2000 IGEL Co.,Ltd.

IGEL

Debugging System Fault

- In the case of segmentation fault in the kernel, the Oops message is generated.
- As the message gives the stack and the register information, it is useful for those who can understand machine language.
- The `ksymoops` command can be used to generate more useful information from the Oops message.

```
Unable to handle kernel NULL pointer dereference at virtual address 00000110
current->tss.cr3 = 0e2f6000, %cr3 = 0e2f6000
*pde = 00000000
Oops: 0000
CPU:    0
EIP:    0010:[<d00287f0>]
Using defaults from ksymoops -t elf32-i386 -a i386
EFLAGS: 00010246
eax: 00000053   ebx: 00000000   ecx: 00000001   edx: 00000000
esi: d0034000   edi: 00000007   ebp: 0000200c   esp: ce2f9ed4
ds: 0018   es: 0018   ss: 0018
Process insmod (pid: 168, process nr: 32, stackpage=ce2f9000)
Stack: d0027e0e d0034000 00000001 00000000 d0034000 00000010 00000005 d0034000
       d0025a3e d0034000 00008000 00000007 0000000f 0000200c cffb16e0 00000000
       d00279a9 d0034000 00000000 00000010 d0025000 00000000 d0025053 ffffffea
Call Trace: [<d0027e0e>] [<d0034000>] [<d0034000>] [<d0034000>] [<d0025a3e>] [<d
0034000>] [<d00279a9>]
       [<d0034000>] [<d0025000>] [<d0025053>] [<d002582f>] [<d002af4f>] [<c0115d
c7>] [<d0025000>] [<d00328a4>]
       [<d0032bd4>] [<d0025048>] [<c0109e84>] [<d0025000>]
Code: a1 10 01 00 00 a9 00 00 08 00 75 d4 5b 5e c3 90 57 56 53 8b
```

IGEL

ksymoops

- The latest version can be obtained from `ftp://ftp.ocs.com.au/pub/ksymoops/`
  - It is also included in the kernel distribution, but you need to compile it manually(`/usr/src/linux/scripts/`)
- See manual and source code to check the usage and the compiling method.
  - `g++ ksymoops.cc` to compile ksymoops in 2.0.36
  - The latest ksymoops can be compiled just by typing `make`
- Getting Oops log
  - `# dmesg > oops.log`
- Running ksymoops
  - For 2.0.36
    `# ksymoops /usr/src/linux/System.map < oops.log`
  - The latest version
    `# ksymoops < oops.log`

IGEL

System Hang

- The kernel may hang while testing the device driver.
- In the worst case, Ctrl+Alt+Del may not work anymore.
- But before pressing the reset button, there is something we can do:
  - Shift + ScrollLock
    - Display the current memory status including buffer cache.
  - Control + ScrollLock
    - Display the current process status.
  - Alt + ScrollLock
    - Display register information.
    - By referring a program counter and the kernel symbol table, you may check where the problem is later on.

Entering to the Device Driver

- Special files are used to access the character device and the block device.
- Special files are required to access the device driver:
  - Making a special file for a character device
    - `# mknod /dev/csample0 c 127 0`
  - Making a special file for a character device
    - `# mknod /dev/bsample0 b 128 0`
- `mknod <filename> <c|b> <major> <minor>`
  - The 2nd argument specifies a character or a block device.
  - The 3rd argument is a major number of the device (i.e. this number is used to identify the device driver.
  - The 4th argument is a minor number used to identify the device within the device driver.

Major and Minor Device Number

- The major device number is used to identify the device driver by the kernel.
- The device driver must register its major number before it starts service.
- The minor device number is used to identify the device in the device driver.

```
Sc.taki[4:02am]% ls -l /dev/fd0*|head
brw-rw----   1 root     floppy     2,   0 May 15  1996 /dev/fd0
brw-rw----   1 root     floppy     2,  36 May 15  1996 /dev/fd0CompaQ
brw-rw----   1 root     floppy     2,   4 May 15  1996 /dev/fd0d360
brw-rw----   1 root     floppy     2,   8 May 15  1996 /dev/fd0h1200
brw-rw----   1 root     floppy     2,  40 May 15  1996 /dev/fd0h1440
brw-rw----   1 root     floppy     2,  56 May 15  1996 /dev/fd0h1476
brw-rw----   1 root     floppy     2,  72 May 15  1996 /dev/fd0h1494
brw-rw----   1 root     floppy     2,  92 May 15  1996 /dev/fd0h1600
brw-rw----   1 root     floppy     2,  20 May 15  1996 /dev/fd0h360
brw-rw----   1 root     floppy     2,  48 May 15  1996 /dev/fd0h410
```

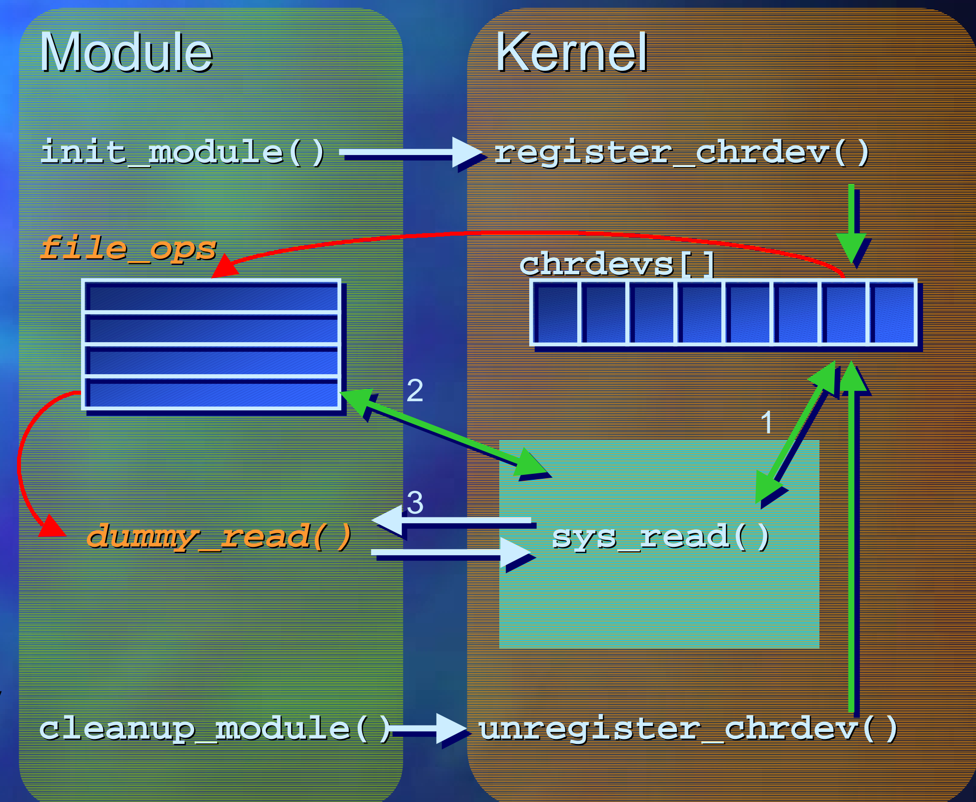
IGEL

Character Device

IGEL

Basics of Character Device

- Register and unregister a character device
  - `register_chrdev`
  - `unregister_chrdev`
- To register a character device, set appropriate callback functions in the structure `file_operations` defined in `<linux/fs.h>`.
- When the user accesses to the device, the kernel call the appropriate callback function by referring the pointer set in the structure `file_operations`.



C. 2000 IGEL Co.,Ltd.

IGEL

Register and Unregister a Character Device

- `int register_chrdev(unsigned int major, const char *name, struct file_operations *fops);`
  - Register a device driver with the name `name` and the major device number `major`. The callback functions which provides services is defined in `fops`.
- `int unregister_chrdev(unsigned int major, const char *name);`
  - Unregister the device driver with the name `name` and the major device number `major`.

IGEL

Structure `file_operations`

- The 2.0.x Kernel

```
struct file_operations {
  int (*lseek) (struct inode*, struct file*, off_t, int);
  int (*read) (struct inode*, struct file*, char *, int);
  int (*write) (struct inode*, struct file*, const char *, int);
  int (*readdir) (struct inode*, struct file*, void *, filldir_t);
  int (*select) (struct inode*, struct file*, int, select_table*);
  int (*ioctl) (struct inode*, struct file*, unsigned int, unsigned long);
  int (*mmap) (struct inode*, struct file*, struct vm_area_struct*);
  int (*open) (struct inode*, struct file*);
  void (*release) (struct inode*, struct file*);
  int (*fsync) (struct inode*, struct file*);
  int (*fasync) (struct inode*, struct file*, int);
  int (*check_media_change) (kdev_t dev);
  int (*revalidate) (kdev_t dev);
};
```

Structure `file_operations`(contd.)

- The 2.1.x kernel and later

```
struct file_operations {
  loff_t (*llseek) (struct file *, loff_t, int);
  ssize_t (*read) (struct file *, char *, size_t, loff_t *);
  ssize_t (*write) (struct file *, const char *, 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 (*fasync) (int, struct file *, int);
  int (*check_media_change) (kdev_t dev);
  int (*revalidate) (kdev_t dev);
  int (*lock) (struct file *, int, struct file_lock *);
};
```

Device Dependent Operations

- `int (*lseek)(struct inode*, struct file*, off_t, int);`
  - The function corresponds to the system call `lseek`.
- `int (*read)(struct inode*, struct file*, char*, int);`
  - The function corresponds to the system call `read`.
- `int (*write)(struct inode*, struct file*, const char*, int)`
  - The function corresponds to the system call `write`.
- `int (*readdir)(struct inode*, struct file*, void*, filldir_t);`
  - This must be set to `NULL` for the device driver.
- `int (*select)(struct inode*, struct file*, int, select_table*);`
  - The function corresponds to the system call `select`. If the given condition become true, then return 1. If not true, then return 0. If this function is set to `NULL`, the I/O to this device must always succeed.

Device Dependent Operations(contd.)

- `int (*ioctl)(struct inode*, struct file*, unsigned int, unsigned long);`
  - The function corresponds to the system call `ioctl`.
- `int (*mmap)(struct inode*, struct file*, struct vm_area_struct*);`
  - The function corresponds to the system call `mmap`.
- `int (*open)(struct inode*, struct file*)`
  - The function corresponds to the system call `open`. If `NULL` is set, the open always succeed.
- `int (*release)(struct inode*, struct file*);`
  - The function corresponds to the system call `close`.
- `int (*fsync)(struct inode*, struct file*);`
  - The function to flush the buffer in the device driver.

Device Dependent Operations(contd.)

- `int (*fasync)(struct inode*, struct file*, int);`
  - The function to set `FASYNC` flag.
- `int (*check_media_change)(kdev_t);`
  - Used only with the block device. Check if the media had been changed since last access. Returns 1 when the media had been changed.
- `int (*revalidate)(kdev_t)`
  - Used only with the block device. The behavior is the device dependent. The function is called every time the media had been changed. Return value should be 0 for safety.

Structure `file`

- `mode_t f_mode`
  - File mode described by `FMODE_READ` bit and `FMODE_WRITE` bit.
  - No need to check, as it is checked by VFS subsystem.
- `loff_t f_pos`
  - The current file position.
  - The `lseek` must update this variable.
- `unsigned short f_flags`
  - The flags like `O_RDONLY` and `O_NONBLOCK(<linux/fcntl.h>)`
  - Need check for Non-blocking I/O.
- `struct inode *f_inode`
  - The inode of the opened file.
- `struct file_operations *f_op`
  - The operations dependent on the file.
  - After the file is opened, the kernel uses these functions as the dispatcher.
  - The dispatcher can be changed according after opening the file(e.g. according to the minor number)
- `void *private_data`
  - The pointer which can be used freely by the device driver.
  - The system call open initializes this variables to `NULL`.
  - This can be used to store the status information, but don. t forget to release the memory at the end.

. C. 2000 IGEL Co.,Ltd.

IGEL

Getting Major and Minor Number

- The major and minor device number can be retrieved from the member of structutre `inode`.
- The 2.0.x kernel (Use the argument `inode`)

```
– int major = MAJOR(inode->i_rdev);
  int minor = MINOR(inode->i_rdev);
```

- The 2.1.x kernel and later (Use the argument `file`)

```
– struct inode *inode = file->f_dentry->d_inode;
  int major = MAJOR(inode->i_rdev);
  int minor = MINOR(inode->i_rdev);
```

C 2000 IGEL Co.,Ltd.

IGEL

Open and Release Methods

```
/*
 * A sample of open method
 */

int sample_open(struct inode *inode,
    struct file *file)
{
   int major = MAJOR(inode->i_rdev);
   int minor = MINOR(inode->i_rdev);

   /*
    * Initialize device
    */

   /* Increment module counter */
   MOD_INC_USE_COUNT;

   return 0;  /* success */
}
```

```
/*
 * A sampe of release method
 */

int sample_release(struct inode *inode,
    struct file *file)
{
   /*
    * Cleanup garbage
    */

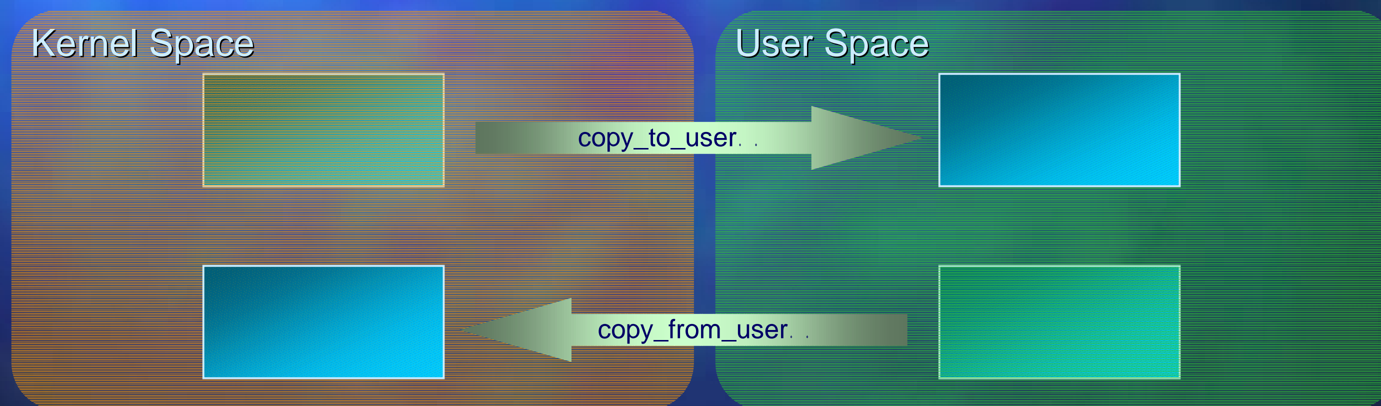
   /* Decrement module counter */
   MOD_DEC_USE_COUNT;

   return 0;  /* success */
}
```

Data Transfer between the Kernel and the User Space

IGEL

- The kernel and the user memory space is the different memory space. No way to use the pointer passed by the user directly.
- To transfer the data between the kernel and the user the following functions are used:
  - The 2.0.x kernel
    ```
    memcpy_tofs(void *to, void *from,  unsigned long count);
    memcpy_fromfs(void *to, void *from,  unsigned long count);
    ```
  - The 2.1.x kernel and later
    ```
    copy_to_user(void *to, void *from, unsigned long len);
    copy_from_user(void *to, void *from, unsigned long len);
    ```



C. 2000 IGEL Co.,Ltd.

IGEL

Verify Memory Space

- It is important to check whether the pointer given the user is valid.
- The 2.0.x kernel
  - `int verify_area(int mode, const void *ptr, unsigned long extent);`
    - Check if the `extent` byte of the memory from `ptr` is valid for the mode `mode`.
    - `VERIFY_READ` or `VERIFY_WRITE` is specified as the `mode`.
- The 2.1.x kernel and later
  - `int access_ok(int type, const void *address, unsigned long size);`
    - This functions is called by `copy_to_user()` and `copy_from_user()` automatically.

Read and Write Methods

```
/*
 * A sample of read method
 */
#define MAX_MINOR   4
#define BUFFER_SIZE 8192
char *buffer[MAX_MINOR];

int sample_read(struct file *file, char
    *buf, int len)
{
  struct inode *inode =
             file->f_dentry->d_inode;
  int major = MAJOR(inode->i_rdev);
  int minor = MINOR(inode->i_rdev);

  /* Data transfer */
  if (len > BUFFER_SIZE)
    len = BUFFER_SIZE;
  copy_from_user(buf, buffer[minor],
    len);

  return len;  /* success */
}
```

```
/*
 * A sample of write method
 */

int sample_write(struct file *file,
    const char *buf, int len)
{
  struct inode *inode =
             file->f_dentry->d_inode;
  int major = MAJOR(inode->i_rdev);
  int minor = MINOR(inode->i_rdev);

  /* Data transfer  */
  if (len > BUFFER_SIZE)
    len = BUFFER_SIZE;
  copy_to_user(buffer[minor], buf,
    len);

  return len;  /* success */
}
```

Implementation Notice on Read and Write

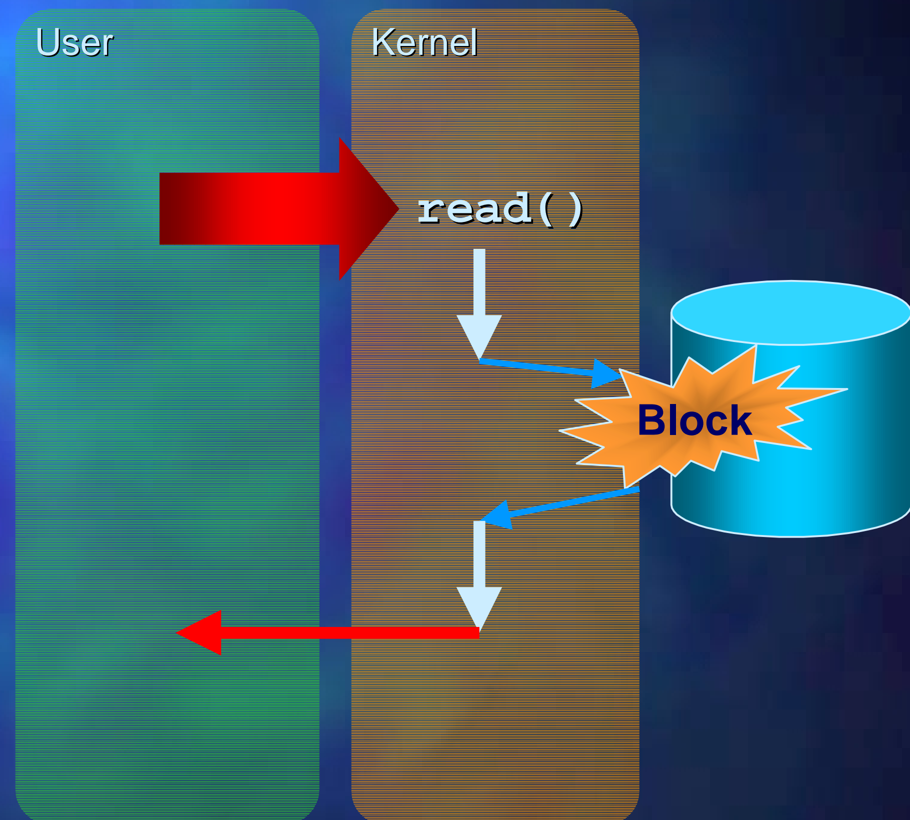
- The read method
  - The return value must be as follows:
    - In the case of EOF, returns always `0`.
    - If there is no data at the moment and the `O_NONBLOCK` flag is set, returns always `-EAGAIN`.
- The write method
  - The return value must be as follows:
    - If the buffer is full and the `O_NONBLOCK` flag is set, returns always `-EAGAIN`.
    - If the device is full, returns `-ENOSPC`.
  - If there is any free space, even 1byte is left, you must copy 1byte to the buffer.
  - Once the data is copied to the internal buffer, you must return immediately without blocking. Checking whether the data is written to the disk must be implemented by the fsync method.

C 2000 IGEL Co.,Ltd.

IGEL

Blocking I/O

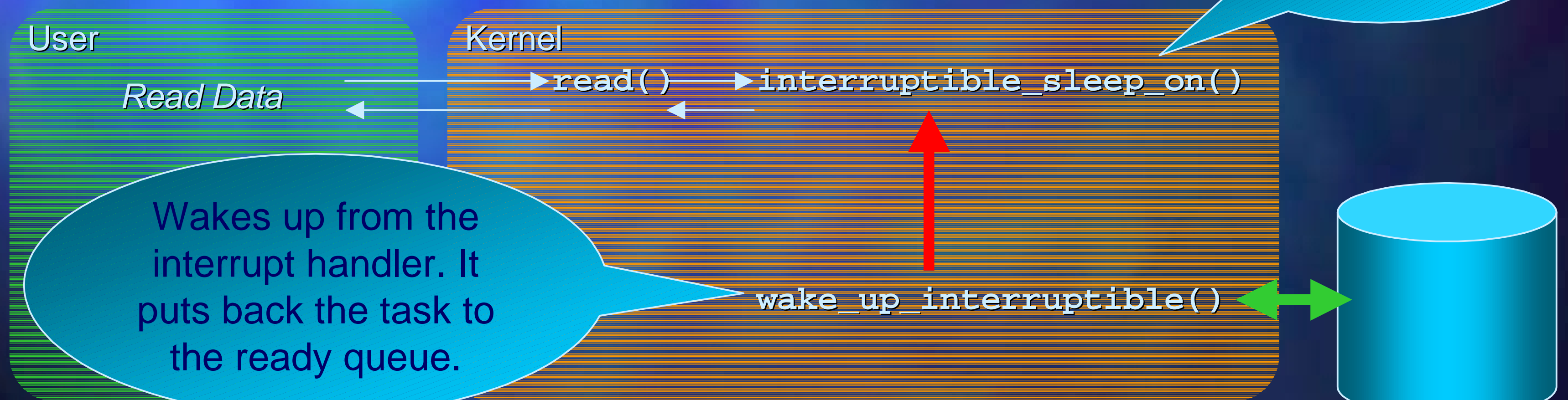
- When the device is not ready for I/O:
  - Block
    Suspend task until the device is ready.
  - Non-block
    Return to caller and tell him that the device is not ready to ready yet.
- In the case of Blocking I/O, we must wait until the device is ready
  - Busy Wait
  - Wait for Event



C 2000 IGEL Co.,Ltd.

Sleep and Wakeup

- To use the limited computation resources effectively:
  - Sleep until the waiting task occurs.
  - When the event occurs, wake up the sleeping task.
- The functions to do it:
  - `void interruptible_sleep_on(struct wait_queue **q);`
    `void sleep_on(struct wait_queue **q);`
  - `void wake_up_interruptible(struct wait_queue **q);`
    `void wake_up(struct wait_queue **q);`



C 2000 IGEL Co.,Ltd.

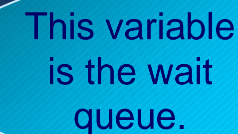
IGEL

select

- The method called by system call `select` to poll if the device is ready.
- The return value of the select method must follow the following rule:
  - If the device is ready, return 1
  - If the device is not ready, return 0
  - If the device is checked for read and the `EOF` condition is true, return 1
  - If the device is checked for write and the device is full, return 0
- To wait until the condition become true, use the `select_wait` for 2.0.x or the `poll_wait` for 2.2.x.

```
int sample_select(struct inode *inode, struct
    file *file, int mode, select_table *table)
{
  sample_dev *dev = file->private_data;
  select(mode) {
  case SEL_IN:
    /* Can read data? */
    if (dev->read_num || dev->eof)
      return 1;

    /* No, put me in the wait queue */
    select_wait(&dev->inq, table);
    /* poll_wait(file, &dev->inq, table); */
    return 0;
  case SEL_OUT:
    /* Can write data? */
  case SEL_EX:
    /* Exception? */
  }
  return 0;
}
```



ioctl

- The user task send the special request as follows:
  - ```
    ioctl(fd, SPL_RESET);
    ioctl(fd, SPL_SET, &var);
    ```
- Old style:
  - ```
    #define SPL_RESET        0
    #define SPL_SET          1
    #define SPL_GET          2
    ```
- New Style:
  - ```
    #define SPL_MAGIC . x.
    #define SPL_RESET        _IO(SPL_MAGIC, 0);
    #define SPL_SET          _IOW(SPL_MAGIC, 1, int);
    #define SPL_GET          _IOR(SPL_MAGIC, 2, int);
    ```

. C. 2000 IGEL Co.,Ltd.
    ```

IGEL

ioctl (contd.)

```
int sample_ioctl(struct inode *inode,
    struct file *file, unsigned int cmd,
    unsigned long arg)
{
  int major = MAJOR(inode->i_rdev);
  int minor = MINOR(inode->i_rdev);
  int ret, size = _IOC_SIZE(cmd);
  int num;

  /* Check memory if needed */
  if (_IOC_DIR(cmd) & _IOC_READ)
    ret = access_ok(VERIFY_WRITE,
      (void*)arg, size);
  if (_IOC_DIR(cmd) & _IOC_WRITE)
    ret = access_ok(VERIFY_READ,
      (void*)arg, size);
  if (ret)
    return ret;

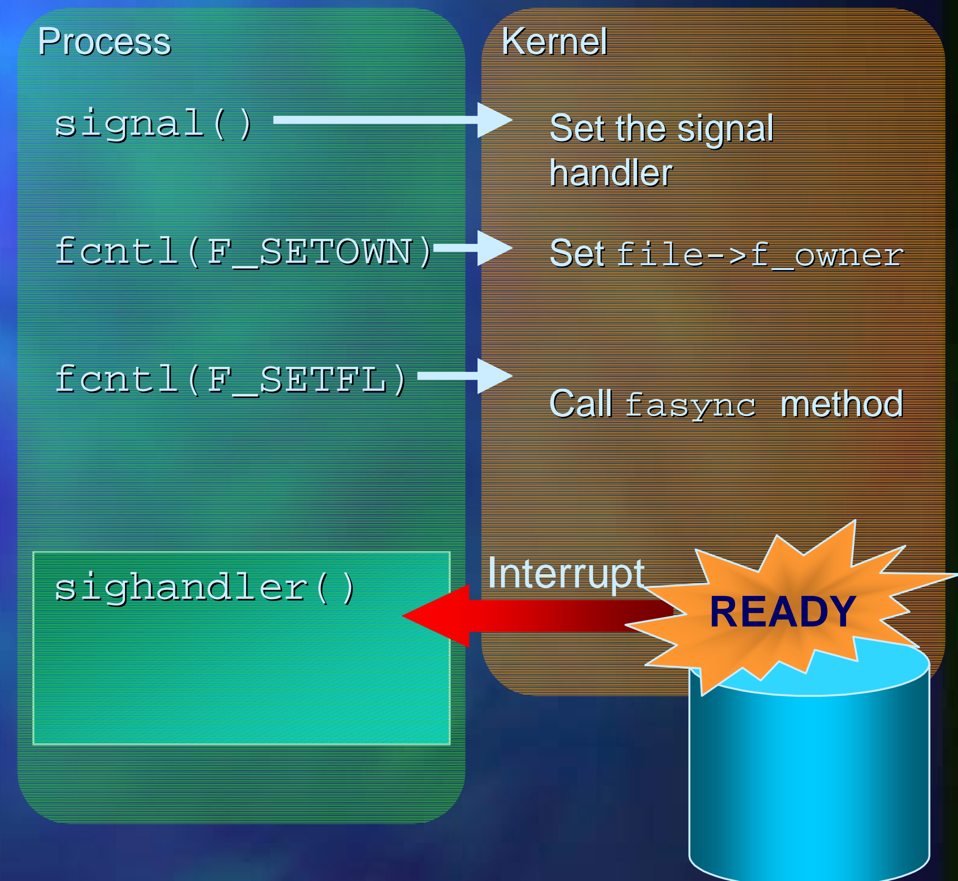
  /* Command */
  swtich(cmd) {
    SPL_RESET:
      /* Reset */
      break;
    SPL_SET:
      /* Set */
      copy_from_user(&num, (void*)arg,
                    size);
      /* Process here */
      break;
    SPL_GET:
      /* Process here */
      copy_to_user((void*)arg, &num,
                  size);
      break;
    default:
      return -EINVAL;
  }
  return 0;
}
```

Signal I/O

- The user process can ask the kernel to send signal when the I/O is ready.

```
signal(SIGIO, &sighandler);
fcntl(0, F_SETOWN, getpid());
oflags = fcntl(0, F_GETFL);
fcntl(0, F_SETFL, oflags |
                      FASYNC);
```

- The method used to realize this mechanism is the `fasync` method.



C 2000 IGEL Co.,Ltd.

Fasync Implementation

```
/* The 2.0.x kernel */
int sample_fasync(struct inode *inode,
    struct file *file, unsigned int
    mode)
{
  sample_dev *dev = file->private_data;
  /*
   * dev->asnc . struct fasync_struct*
   */
  return fasync_helper(inode, file,
    mode, &dev->async);
}

/* The 2.1.x kernel and later */
int sample_fasync(int fd, struct file
    *file, unsigned int mode)
{
  sample_dev *dev = file->private_data;

  return fasync_helper(fd, file, mode
    &dev->async);
}
```

- The function `fasync_helper` is a helper function defined in `drivers/char/tty_io.c`.
- If the I/O become ready, call the function `kill_async` to send SIGIO to the process
  - `kill_fasync(dev->async, SIGIO);`
- Don. t forget to remove from the queue when the file is closed.
  - The 2.0.x kernel `sample_fasync(inode, file, 0);`
  - The 2.1.x kernel and later `sample_fasync(-1, file, 0);`

IGEL

Lseek Implementation

- Use the variable `file->f_pos` to keep the current file position.
- If the file pointer is asked to set invalid position, the return value must be `-EINVAL`.
- Return `-ESPIPE` if the device doesn't allow lseek.

IGEL

Virtual Address

- Memory Map is to map the file or the device to the given virtual address.
- The memory to be mapped is set defined by the structure `vm_area_struct`.
  - `unsigned long vm_start`
    `unsigned long vm_end`
    The destination address. Map memory between `vm_start` and `vm_end`.
  - `struct inode *vm_inode`
    The inode corresponding to the memory.
  - `unsigned long vm_offset`
    The offset of the file or device to map.
  - `struct vm_operations_struct *vm_ops`
    The Pointers to the methods for memory map operations.

Structure vm_operations_struct

- `void (*open)(struct vm_area_struct *vma)`
  - Map memory.
- `void (*close)(struct vm_area_struct *vma)`
  - Close memory map.
- `void (*unmap)(struct vm_area_struct *vma, unsigned long addr, size_t len)`
  - Unmap memory.
- `int (*sync)(struct vm_area_struct *vma, unsigned long addr, size_t len, unsinged int flags)`
  - Write back the dirty pages.
- `unsigned long (*nopage)(sturct vm_area_sturct *vma, unsigned long addr, int write_access)`
  - Called in the case of the page fault.
  - If the `write_access` is not 0, prepare the private pages for the current process.

IGEL

Memory Map

```
/*
 * The simplest example
 */

int sample_mmap(struct inode *inode,
    struct file *file, struct
    vm_area_struct *vma)
{
  size_t size;

  size = vma->vm_end - vma->vm_start;
  if (remap_page_range(vma->vm_start,
    vma-vm_offset, size,
    vma->vm_page_prot))
     return -EAGAIN;

  vma->vm_inode = inode;
  inode->i_count++; // use counter
  return 0;
}
```

- The function `remap_page_range` can be used to map I/O memory of the peripheral device

IGEL

Accessing Peripherals

IGEL

I/O Ports

- Read or writes byte ports
  - `inb(port);`
  - `outb(unsigned char byte, port);`
- Read or write 16-bit ports (word)
  - `inw(port);`
  - `outw(unsigned short word, port);`
- Read or write 32-bit ports (dword)
  - `inl(port);`
  - `outl(unsigned long dword, port);`
- Read or write to the slow device
  - `outb_p()`
  - `outw_p()`
  - `out_p()`
- To slow the I/O explicitly, insert `SLOW_DOWN_IO` in the code.

I/O Ports(contd.)

- Conditions to access I/O ports from user space
  - Compile with `-O` option to force expansion of inline-functions
  - Get permission to access I/O ports by calling the function `ioperm` or the function `iopl`
  - Run as root

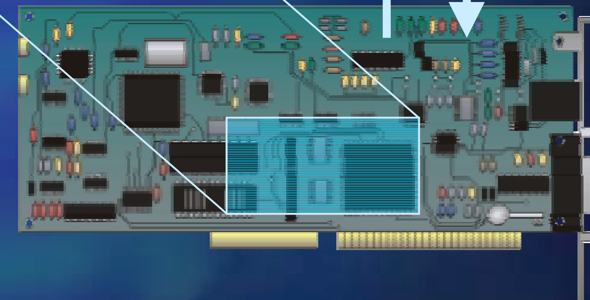
PCI Bus

- IRQs and base addresses are assigned by PCI BIOS.
- The assigned information must be read to access the peripherals.
- Generally, DMA is performed by PCI peripherals PCI peripherals, no need to consider about DMA channel.

Kernel

```
pcibios_read_config_byte()
pcibios_read_config_word()
pcibios_read_config_dword()
pcibios_write_config_byte()
pcibios_write_config_word()
pcibios_write_config_dword()
```

```
ioremap()
```



IGEL

Initialize PCI Peripheral

```
/* The 2.1.x kernel or later */
void init_device()
{
  struct pci_dev *pci_dev = NULL;
  unsigned char pci_bus, pci_device_fn;
  volatile unsigned char *ptr;

  while(pci_dev = pci_find_device(VENDOR_ID, DEVICE_ID, pci_dev)) {
    pci_bus = pci_dev->bus->number;
    pci_device_fn = pci_dev->devfn;

    /* Get IRQ and Base Address */
    pcibios_read_config_byte(pci_bus, pci_device_fn, PCI_INTERRUPT_LINE, &irq);
    pcibios_read_config_dword(pci_bus, pci_device_fn, PCI_BASE_ADDRES_0, &addr);

    /* Access to the Base Address */
    newaddr = ioremap(addr, PAGE_SIZE);
    ptr = newaddr;
    printk(. … %p\n. , *ptr);

    break;
  }
  return;
}
```

. C. 2000 IGEL Co.,Ltd.

IGEL

Initialize PCI Peripheral(contd.)

```
/* The 2.0.x kernel */
void init_device()
{
  unsigned char pci_bus, pci_device_fn;
  volatile unsigned char *ptr;

  if (!pcibios_present()) return;

  for(int i = 0; i < MAX_PCI_DEV; i++) {
    if (pcibios_find_device(VENDOR_ID, DEVICE_ID, i, &pci_bus, &pci_device_fn)
      != 0) continue;

    /* Get IRQ and Base Address */
    pcibios_read_config_byte(pci_bus, pci_device_fn, PCI_INTERRUPT_LINE, &irq);
    pcibios_read_config_dword(pci_bus, pci_device_fn, PCI_BASE_ADDRES_0, &addr);

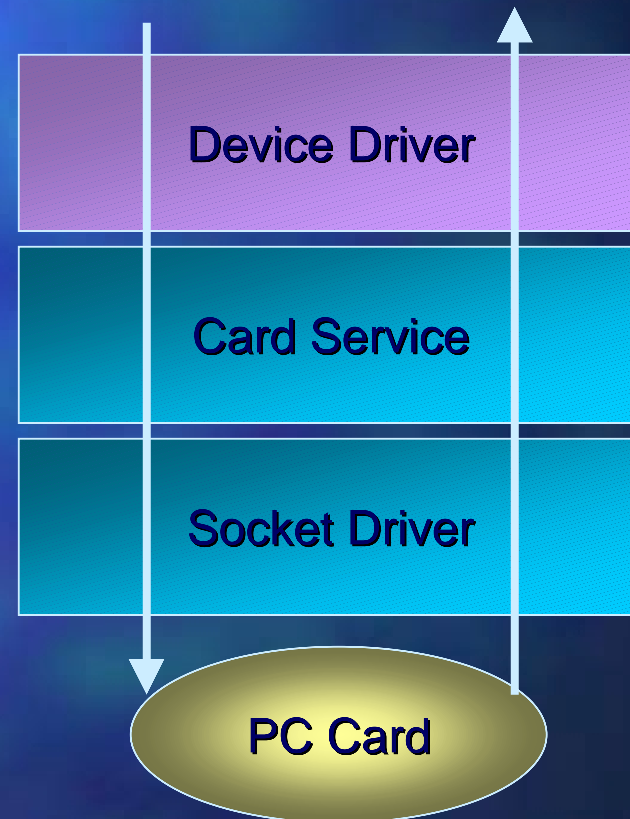
    /* Access to the Base Address */
    newaddr = vremap(addr, PAGE_SIZE);
    ptr = newaddr;
    printk(. … %p\n. , *ptr);

    break;
  }
  return;
}
```

© 2000 IGEL Co.,Ltd.

PCMCIA

- The PCMCIA support is an option under Linux 2.2.x.
  - http://pcmcia.sourceforge.org/
- The PCMCIA in the Linux has the following characteristics:
  - Allows to use the existing device drivers (ISA drivers for 16bit cards and PCI drivers for Card-bus cards)
  - Generally, the PCMCIA driver is not device driver itself, but the enabler.
- Good sample for writing PCMCIA device driver is clients/dummy_cs.c in the PCMCIA distribution.



IGEL

References

- A.Rubini, "Linux Device Driver", O'REILLY, 1998. ISBN-1-56592-292-1.
  - Good reference for writing Linux device drivers. Plenty of examples helps reader to understand how the device driver works. The book is based on the 2.0.x kernel.
- R.Card, E.Dumas, F.Mevel, "The Linux Kernel Book", Wiley, 1998. ISBN-0-471-98141-9.
  - Illustrates the internal of the Linux kernel. Helps to understand how the Linux kernel works. Referring this book with the source code helps you to understand the kernel internals.
- Source code of the Linux 2.2.x kernel
  - There are many changes from the 2.0.x kernel to the 2.2.x kernel. The best reference is still the kernel source code.
- W.R. Stevens, "UNIX Network Programming 2nd Ed.", Prentice Hall, 1997. ISBN 0-13-490012-X & ISBN 0-13-081081-9.
  - The best book describing the networking issues.