Linux Kernel Development
week 1

工海四 b06501018 朱紹勳

如何使用

- Understand Linux Kernel 細節較多，偏向瞭解內核本身
- Linux Device Driver 偏向系統程式實作
- Linux Kernel Development 趨向兩者之間，編排上也有利於新手

→ 以 Linux Kernel Dev 為主，穿插 Understand Linux Kernel

- Focus on OS design discussion – Get Linux 2.6 version source code

Linux Kernel Development Week 1

	Linux Kernel Dev 3rd	Understand Linux Kernel 3rd	Extra
Intro of Linux & Kernel	Chapter 1	Chapter 1	
Kernel Dev & Programming	Chapter 2 & 6	Chapter 1	System Programming Linux Source Code
Interrupt	Chapter 7	Chapter 4	
System Call (optional)	Chapter 5	Chapter 10	

INTRO of LINUX

Chapter 1: Intro of Linux

- Multics(failed) → Unix(successful, open source)
- Unix characteristic:
  - *Simple enough*
  - *Everything is file*
  - *C programming language/ a little Assembly*
  - *Fast process creation*
  - *Virtual memory, file system, LWP, signal, IPC(SVR4), support SMP system*
- Linux:
  - Follow POSIX standard (就算內部不同，介面也要保持相同以保障可移植性)
  - Non-Commercial, open source under GNU

- Kernel, not the interface of OS
- OS ↔ Kernel
  - Extended machine (好用, 封裝: user kernel mode)
  - Resource manager
    - 多用戶:安全, UID & User Group ID, root & superuser
    - Linux is multiprocessing & preemptable

Exception: MS-DOS…

system call => mode switch

- Kernel mode & Kernel space
  - Memory, Access Right, Interrupt…
- System Call
  - How user process access kernel service
- Interrupt
  - signal → find specific ISR

- Kernel structure:
  - linux is **monolithic kernel**

    microkernel(not so efficient because communication, but has *modulation*)

    linux provide *module*, which makes object file link to kernel in runtime

    → better use of RAM, portable in devices, performance…

  https://stackoverflow.com/questions/1806585/why-is-linux-called-a-monolithic-kernel

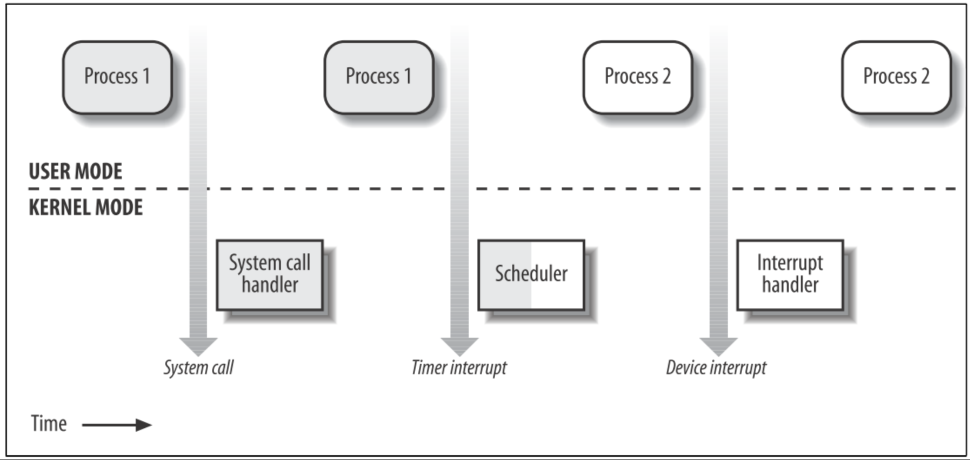
- Difference between Linux & Unix Kernel
  - dynamic loading of kernel modules
  - preemptive
  - thread is more like a light weighted process

Kernel

- General
- Synchronization
- Process Communication
- Process Management
- Memory Management
- Device Driver

File System

Kernel Overview



- Process/ kernel mode
  - Kernel is a process manager, which exists all the time
  - kernel thread is a privileged process run in Kernel Mode and Space

  - (1) System call (2) CPU exception
(3) External Interrupt (4) Kernel thread may activate kernel mode

- Process
  - Descriptor will store info of processes from registers
    - PC, Stack Pointer Reg, Processor Control Reg, Memory Management Reg

  - Want to resume process: Register → Descriptor

- Reentrant(可重進) Kernels
  - Kernel thread 可以釋放CPU， 由此避免阻止其他想進入Kernel 的 process
  - Allow multiple processes run in Kernel Mode

- Process Address Space
  - While entering Kernel → access Kernel data, code area, private stack
  - Linux support mmap() system call

Kernel Overview
- Synchronization

- Reentrant Kernel → Multi-Process → Race Condition → Synchronization
- Non-preemptive(Unix), disable Interrupt is not efficient

- Linux use:
  - Semaphores
    - An interger, waiting process list, down()&up() (atomic)
    - Integer < 0, cannot access
  - Spin lock
    - Only Use in multi-processor

Kernel Overview
- Process Communication

- Shared memory / Message Passing
- POSIX has 20 signals, with following responses:

  1. Ignore 2. Terminate 3. Core dump 4. Suspend 5. Resume

- System Calls…
  - `shmget(key, size, shmflg)   // create share memory`
  - `semget(key, size, semflg)   //`
  - `msgget(key, msgflg)         //`

  - `msgsnd(), msgget()          // put and get from message queue`

  All the resource has to release manually

Kernel Overview
- Process Management

- fork() / exit() / exec()
    - fork() has “Copy-On-Write” mechanism
    - exit() use system call to terminate process, and signal “`SIGCHILD`”

- Zombie Process
    - waitpid() (wait4())
    - Avoid zombie process → *`init`* is the parent of all zombie

Kernel Overview
- Memory Management

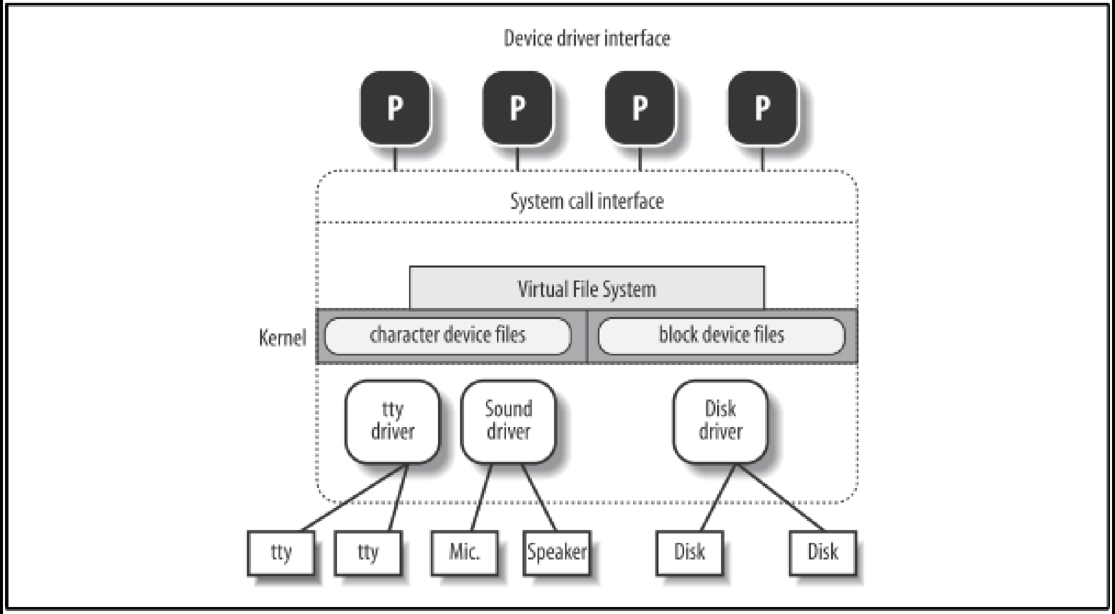
- Virtual Memory
  - MMU : Map memory address from virtual to physical
  - Relocatable,   large memory requirement, exec partly loaded program
- RAM
  - 1. kernel image   2. virtual memory system
  - *Page–frame reclaim algorithm*
- Handle process virtual addr space

- Caching
  - First check if the data is already in RAM
  - sync() sys call : make RAM and disk data consistent

Kernel Overview
- Device Driver

- Device-specific code is encapsulated in modules
- Develop new compatible device by only knowledge interface spec
- Reduce Kernel Image in RAM

```
chu@chu-M32AA1  ~  df                                         127 ↵  217  18:23:09
Filesystem     1K-blocks     Used Available Use% Mounted on
udev             1957120        0   1957120   0% /dev
tmpfs             397060     1804    395256   1% /run
/dev/sda8      253043440 27676428 212443416  12% /
tmpfs            1985280    74448   1910832   4% /dev/shm
tmpfs               5120        4      5116   1% /run/lock
tmpfs            1985280        0   1985280   0% /sys/fs/cgroup
/dev/loop1         56704    56704         0 100% /snap/core18/1885
/dev/loop3         56320    56320         0 100% /snap/core18/1880
/dev/loop4         30720    30720         0 100% /snap/snapd/8790
/dev/loop6        261760   261760         0 100% /snap/gnome-3-34-1804/36
/dev/loop5         51072    51072         0 100% /snap/snap-store/467
/dev/loop2         99456    99456         0 100% /snap/core/9993
/dev/loop8         98944    98944         0 100% /snap/core/9804
/dev/loop7         63616    63616         0 100% /snap/gtk-common-themes/1506
/dev/loop0        165376   165376         0 100% /snap/gnome-3-28-1804/128
/dev/loop11          256      256         0 100% /snap/gtk2-common-themes/13
/dev/loop9         58880    58880         0 100% /snap/discord/115
/dev/loop13        31104    31104         0 100% /snap/snapd/9279
/dev/loop12        58752    58752         0 100% /snap/discord/112
/dev/loop10       114048   114048         0 100% /snap/shutter/27
/dev/sda2         262144    57476    204668  22% /boot/efi
tmpfs             397056       28    397028   1% /run/user/1000
```



I. Linux Basic Concept
File System(1)

- Linux File system:
  - in tree-structured namespace, file name < 255 characters
  - pathname

    absolute (/usr/fold/file), relative(usr/fold/file)

    current directory(./), parent directory(../)
- Link
  - File system is a hard link
  - Hard Link (`$ ln P1 P2`)

    → May cause directory tree to cycle… Harmful

    → Can only link when files are in same file system… Limited
  - soft link or symbolic link (`$ ln -s P1 P2`)

    To overcome the short of hard link
- File type
  - Regular file, Directory, Symbolic link, pipeline, socket...File type

- File description (POSIX Standard)
  - File type, # hard link, file length, UID, UGID, Access right, time…
  - suid(set user ID), sgid(set group ID), sticky(keep file in memory after terminated)

- File Handling system call: (system programming course)
  - fd = open(path, flag, mode); *// flag: operation, mode: access right*

- Ext2 & Ext3, jfs
  - Block, Superblock, Inode
  - file recover
  - Journal File System

Chapter 2: Linux Kernel Dev

- /arch
  - OS design is heavily dependent on hardware
    arm, i386, ia64, mips,x86_64…
- /driver
- /firmware
- /ipc
- /usr
- /kernel
- /script

Chapter 2: Linux Kernel
Compile The Kernel

- Kernel Development:
  - coded in GNU C (no linked to the standard C lib!!!)
  - NO memory protection
  - avoid float computing
  - small stack
  - watch out for the synchronization problem

System Programming & Data structure

System Programming

- man –f prinft
  ```
  printf (1)          - format and print data
  printf (3)          - formatted output conversion
  ```
  - man 2 (system calls) man 3 (library calls)
- gcc (compiler)
  - `gcc abc.c          // must contain main func, abc.out`
  - `gcc abc.c –O3      // 1, 2, 3, g, fast, s`
- gdb (officical debugger)
  - b: breakpoint, bt: current call stack, s: step in, n: step over, r: execute
- Make

Data Structure

- C is not O-O language (use struct)
- Linked List:

General C form Doubly-LL (handcraft)

```
struct fox {
        unsigned long  tail_length;  /* length in centimeters of tail */
        unsigned long  weight;       /* weight in kilograms */
        bool           is_fantastic; /* is this fox fantastic? */
        struct fox     *next;        /* next fox in linked list */
        struct fox     *prev;        /* previous fox in linked list */
};
```

Linux Development Doubly-LL, use <linux/list.h>

```
struct fox {
        unsigned long    tail_length;  /* length in centimeters of tail */
        unsigned long    weight;       /* weight in kilograms */
        bool             is_fantastic; /* is this fox fantastic? */
        struct list_head list;         /* list of all fox structures */
};
```

```
struct fox *red_fox;
red_fox = kmalloc(sizeof(*red_fox), GFP_KERNEL);
red_fox->tail_length = 40;
red_fox->weight = 6;
red_fox->is_fantastic = false;
INIT_LIST_HEAD(&red_fox->list);
```

```
struct fox red_fox = {
  .tail_length = 40,
  .weight = 6,
  .list  = LIST_HEAD_INIT(red_fox.list),
};
```

```
list_add(&f->list, &fox_list);
list_add_tail(&f->list, &fox_list);
```

```
list_move(struct list_head *list, struct list_head *head)
list_move_tail(struct list_head *list, struct list_head *head)
```

Queue

- FIFO structure, <linux/kfifo.h>

CREATE:

```
int kfifo_alloc(struct kfifo *fifo, unsigned int size, gfp_t gfp_mask);    // return 0 when success
```

ENQUEUE, DEQUEUE:

```
unsigned int kfifo_in(struct kfifo *fifo, const void *from, unsigned int len);
unsigned int kfifo_out(struct kfifo *fifo, void *to, unsigned int len);
```

GET SIZE OF QUEUE:

```
static inline unsigned int kfifo_size(struct kfifo *fifo);
static inline unsigned int kfifo_avail(struct kfifo *fifo);
```

RESET, & DESTORY:

```
static inline void kfifo_reset(struct kfifo *fifo);
void kfifo_free(struct kfifo *fifo);
```

Maps

- Asscociative array, key-value pair, often use for **UID**
- Data structure: hash table, self-balanced tree

INITIALIZE idr:

```
void idr_init(struct idr *idp);
```

ALLOCATE NEW UID:

```
int idr_pre_get(struct idr *idp, gfp_t gfp_mask);
int idr_get_new(struct idr *idp, void *ptr, int *id);
```

FIND UID:

```
void *idr_find(struct idr *idp, int id);
```

GET SIZE OF QUEUE:

```
void idr_remove(struct idr *idp, int id);
```

RESET, & DESTORY:

```
void idr_destroy(struct idr *idp);
void idr_remove_all(struct idr *idp);
```

BST & self-balanced

- Red-Black Trees <linux/rbtree.h>

Data Structure:

Traverse :	linked list (O(n))
Producer Consumer Pattern:	queue
Map UID :	map
Data Storage & Access:	RBTree

Interrupt

Interrupt

- Synchronous vs. Asynchronous
  → Exceptions(1. error 2. int/sysenter) vs. hardware I/O
- Interrupt ReQuest (IRQ)

- When receive an Interrupt signal:
  1. CPU halt, save Reg → Kernel Mode Stack
  2. Put address to PC

- Context switch vs Interrupt

Interrupt vs. Exception

- Interrupt:

  Maskable (可屏蔽) : Gerneral
  Nonmaskable (不可屏蔽): Emergency, ex: hardware failure

- Exception - processor detected error

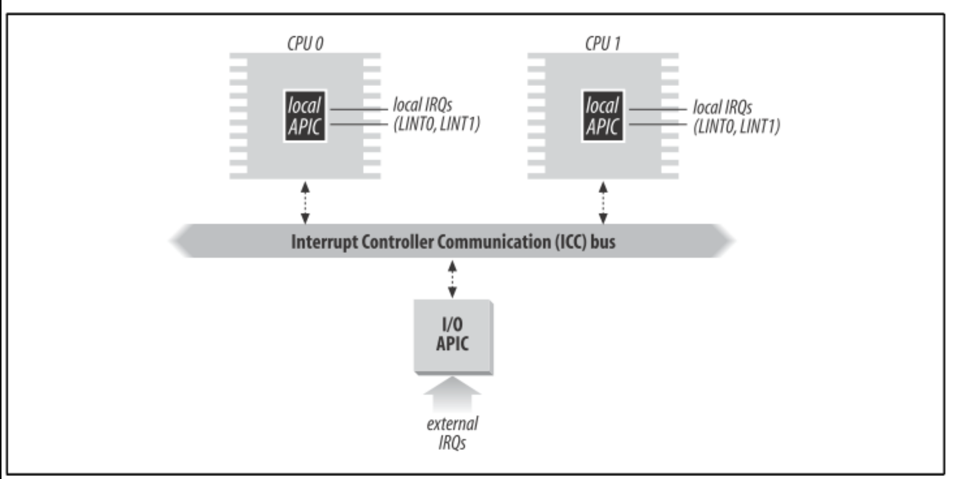
  Fault : error addr → eip, fault can be fixed and resume
  Trap : next exec addr → eip, usually for debug
  Abort : *Fatal Error*, no info in eip,
  Programmed Exception: by int or int 3, also called software interrupt

- Use 0-255 to identify Interrupt or Exception

Interrupt
- IRQ, PIC & APIC

- Programmable Interrupt Controller
  - Monitors IRQ line and wait for signals. If YES:
    a. Signal → vector → Interrupt Controller(CPU read Vector via the data bus)
    b. Signal to the processor INTR pin
    c. CPU write signal to Programmable Interrupt Controllers (PIC), clear INTR
- IRQ line can be enable/disable by PIC, total 2 “8259A” & 15 lines.

- For Multi-Processor, use A (advance) PIC
  - APIC I/O, Interrupt Control Comm bus



Exception

- 80x86 has 20 exceptions →hardware error code → Kernel Mode Stack
  - Div 0 Debug Overflow Device not available…

#	Exception	Exception handler	Signal
0	Divide error	divide_error()	SIGFPE

Interrrupt
- Interrupt Descriptor Table

- Associated with interrupt (exception) vector with address
- Total 2048 bytes (256*8 byte each table)
- Contain 3 types of Gate Descriptor:
  - **Task Gate** Descriptor : TSS selector of process to replace interrupted ones
  - **Interrupt Gate** Descriptor : Handle Interrupt
  - **Trap Gate** Descriptor : Handle Exception

Interrrupt
- IDT – Hardware Handle

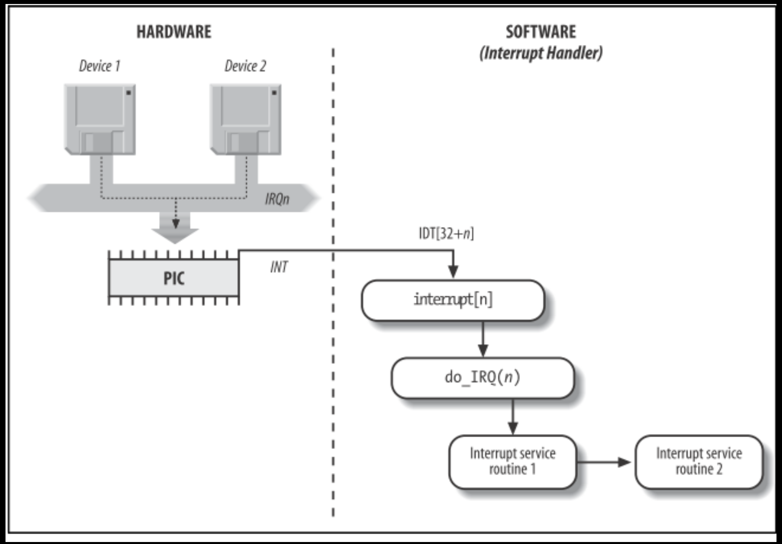
- Detailed description in <Understanding Linux Kernel>
- Contain:
  1. Initialize Table
  2. Different operation under Interrupt & Exception, and others
     - 2.1 Exception
     - 2.2 Interrupt
     - 2.3 Softirqs & Tasklets
     - 2.4 Work Queue
  3. Return from Interrupt & Exception

Interrrupt
- IDT - Exception

- Register → Exception Handler
  - Assembly code
  - push address to stack using do_xxx()
- Enter Exception Handler
  - exec do_xxx()
- Exit from Exception Handler
  - keep tracking signals (user/kernel)
  - ret_from_exception()

```
current->thread.error_code = error_code;
current->thread.trap_no = vector;
force_sig(sig_number, current);
```

Interrrupt
- IDT - Interrupt



- Types: (1) I/O, (2) Timer, (3) Interprocess

- Save IRQ and register value in Kernel Stack
- Sending signal to corresponding PIC for futher interrupt
- exec ISR
  do_IRQ()
- Exit from interrupt
  ret_from_intr()

Interrupt Service Routine

- ISR is normal C function, but run in **interrupt context**
  - Interrupt context is not like Process context, the former cannot sleep
  - Interrupt context interrupt other code, so should be quick and simple
  - The context is divided in to Top and Bottom section, which reduce work load
  - It share the stack with process that interrupted

- Registering an Interrupt Handler
  - Use `request_irq(irq, handler, flag, name, dev) (<linux/interrupt.h>)`
    `// return 0 if success`

irq	: Interrupt number for hardware, or defined
handler	: point to actual interrupt handle program
flag	: IRQF_DISABLED, IRQF_SAMPLE_RANDOM, IRQF_TIMER, IRQF_SHARED
name	: defined in /proc/interrupts file, interrupt line, counter, device name…
dev	: Use for shared interrupt lines, set NULL if not

Interrupt Service Routine

- Freeing an Interrupt Handler

```
void free_irq(unsigned int irq, void *dev)
```

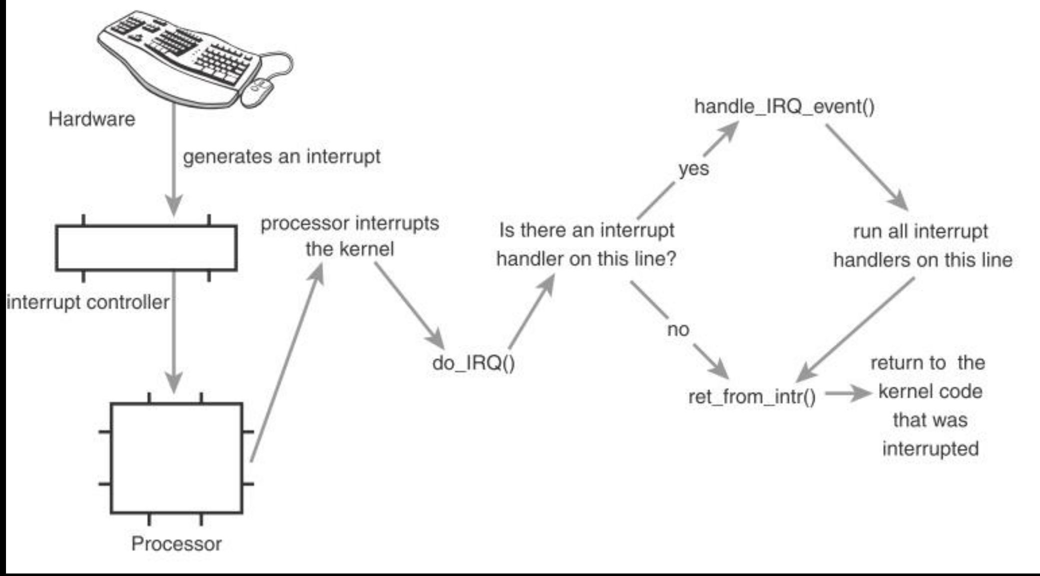
- Writing an Interrupt Handler

```
static irqreturn_t intr_handler(int irq, void *dev)
// corresponded with request_irq()
```

- Shared handler

1. Set flag = IRQT_SHARED in request_irq() (old version)
2. Must define parameter dev
3. Hardware support

IRQ



- `unsigned int do_IRQ(struct pt_regs regs)`
- After calculate the interrupt number and disable others

```
do_IRQ → handle_IRQ_event()(kernel/irq/handler.c) → ISR
```

```
arch/x86/kernel/entry_64.S
arch/x86/kernel/irq.c
```

Interrupt Control

- <asm/system.h> <asm/irq.h>
- The need for synchronization → guarantee that an interrupt handler will not preempt your current code, even disable interrupt by kernel

Table 7.2 **Interrupt Control Methods**

Function	Description
`local_irq_disable()`	Disables local interrupt delivery
`local_irq_enable()`	Enables local interrupt delivery
`local_irq_save()`	Saves the current state of local interrupt delivery and then disables it
`local_irq_restore()`	Restores local interrupt delivery to the given state
`disable_irq()`	Disables the given interrupt line and ensures no handler on the line is executing before returning
`disable_irq_nosync()`	Disables the given interrupt line
`enable_irq()`	Enables the given interrupt line
`irqs_disabled()`	Returns nonzero if local interrupt delivery is disabled; otherwise returns zero
`in_interrupt()`	Returns nonzero if in interrupt context and zero if in process context
`in_irq()`	Returns nonzero if currently executing an interrupt handler and zero otherwise

Vector range	Use
0–19 (0x0–0x13)	Nonmaskable interrupts and exceptions
20–31 (0x14–0x1f)	Intel-reserved
32–127 (0x20–0x7f)	External interrupts (IRQs)
128 (0x80)	Programmed exception for system calls (see Chapter 10)
129–238 (0x81–0xee)	External interrupts (IRQs)
239 (0xef)	Local APIC timer interrupt (see Chapter 6)
240 (0xf0)	Local APIC thermal interrupt (introduced in the Pentium 4 models)
241–250 (0xf1–0xfa)	Reserved by Linux for future use
251–253 (0xfb–0xfd)	Interprocessor interrupts (see the section "Interprocessor Interrupt Handling" later in this chapter)
254 (0xfe)	Local APIC error interrupt (generated when the local APIC detects an erroneous condition)
255 (0xff)	Local APIC spurious interrupt (generated if the CPU masks an interrupt while the hardware device raises it)

IRQ	INT	Hardware device
0	32	Timer
1	33	Keyboard
2	34	PIC cascading
3	35	Second serial port
4	36	First serial port
6	38	Floppy disk
8	40	System clock
10	42	Network interface
11	43	USB port, sound card
12	44	PS/2 mouse
13	45	Mathematical coprocessor
14	46	EIDE disk controller's first chain
15	47	EIDE disk controller's second chain

Exit from interrupt

Consideration:

1. Number of kernel control paths
2. Pending process switch requests
3. Pending signals
4. Single-step mode
5. Virtual-8086 mode

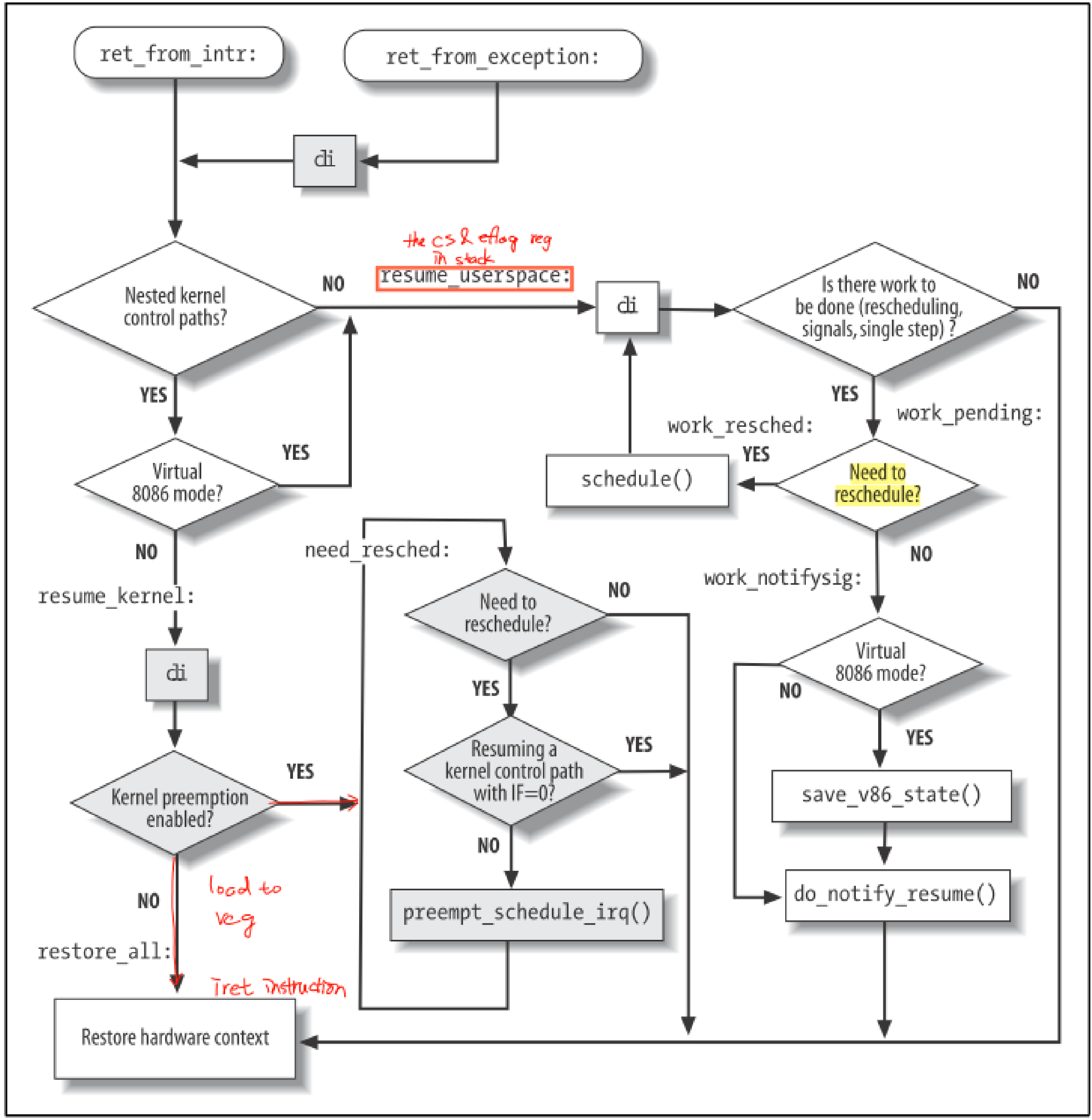


Figure 4-6. Returning from interrupts and exceptions

System Call

System Call

- A layer between user process and hardware
  - Make programming easier (API) - POSIX
  - improve system security : Access Right, Encapsulation
  - Other ways to enter Kernel : exception, external interrupt

- API vs. Sys Call → A function definition / A request to kernel by **trap**

  trap **:** the software interrupt defined #interrupt 128 in X86

  Trigger by **int $0x80** instruction

- Return of Sys Call:
  - usually (-) means error
  - or defined in errno.h ex : *include/asm-i386/errno.h*

System Call
- Handler & Service Routine

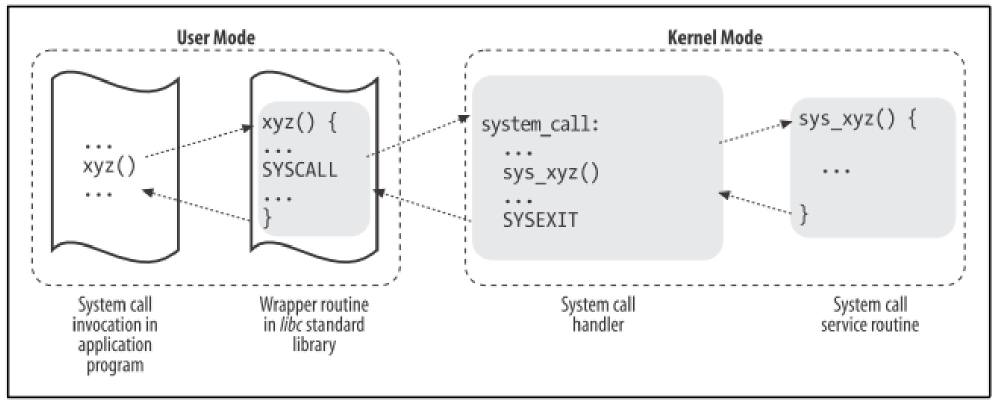


Figure 10-1. Invoking a system call

- Routine
  1. Save register to Kernel Mode Stack
  2. → C func() → System call handler(assembly) → System Call Service Routine
  3. Exit from handler, Stack → register, Switch to User Mode

- System Call Identification
  - We will pass system call number to help us, EAX register → Kernel
  - find #system call in dispatch table(store in an array), there are 289 in 2.6 ver
  - NR_syscalls macro 檢查 system call 是否合法
    - if so, call *sys_call_table(,%rax,8);

- Parameter of func() : ebx, ecx, edx, esi, edi register in X86_32
- Name of Service Routine : sys_xyz() ← xyz()

System Call
- Ways to invoke Sys Call

- There are two ways to invoke and exit Sys Call:

1.	invoke :	用 int $0x80	assembly	(old)
	exit :	用 iret	assembly	(old)
2.	invoke :	用 sysenter	assembly	
	exit :	用 sysexit	assembly	

The following are the descriptions: