作業三：
瞭解作業系統的初始化

中正大學 作業系統實驗室

指導教授：羅習五

作業目標及負責助教

- 作業目標：
  - 了解QEMU的好處及缺點
  - 了解Linux kernel如何設定system call的進入點
  - 了解Linux kernel如何設定中斷向量表
  - 約略了解x86-64的特殊暫存器
  - 玩Linux kernel
- 負責助教：曹芳駿
  - Email：squidt@csie.io

下載已經做好的vmware檔案

- http://lonux.cs.ccu.edu.tw/os-2018/hw2.tar.bz2
- vmware的帳號密碼：shiwulo, 12345678
- QEMU/kvm的帳號密碼：os2018， 12345678

目錄內容介紹

- 所有相關檔案都放在/home/shiwulo/qemu底下
- 『linux-source-code』放可以除錯的Linux的原始碼
- 『 busy-box-source-code 』放busybox (initramfs用的shell)
- 『kvm.sh』是可以執行的檔案，快速執行Linux，但不支援除錯
- 『qemu.sh』是可以執行的檔案，可以執行Linux並除錯
- 『 bzImage』Linux的核心
- 『 qemu.dsk 』一個完整的Linux虛擬硬碟
- 上述虛擬硬碟可以直接用kvm執行『sudo kvm -m 512 qemu.dsk』

Linux開機流程介紹

- 在電腦系統裡面，開機的時候，首先執行ROM裡面的資料，對PC而言，就是BIOS，隨後BIOS讀取MBR，在讀取開機磁碟的boot sector
  - MBR（Master Boot Record）位於block device的第一個block，裡面記載partition table（最多4個），及少許程式碼。MBR支援的block device容量只到2TB。目前使用GPT代替MBR
  - 依照MBR指定的boot partition到該partition讀取boot sector內的boot loader
  - bootloadery載入Linux kernel
- 常見的bootloader
  - 於PC上最常見的是GRUB
  - 於嵌入式系統最常見的是U-Boot

http://linux.vbird.org/linux_basic/0510osloader.php

Linux初步

- 到/boot下載入Linux kernel（vmlinuz-XXX）及initramfs-XXX
  - 因此/boot所使用的檔案系統一定要是bootloader認得的檔案系統
  - 有時候/boot會額外分配一個partition，就是因為要讓bootloader認得
- vmlinuz-XXX的前半部分是解壓縮程式碼，後半部分是Linux kernel的壓縮檔
  - 因此vmlinuz-XXX可以將自己解開，並放到記憶體的適當位置
- 隨後Linux kernel解開initramfs-XXX
  - initramfs-XXX裡面放著一個完整的檔案系統，在Linux還未驅動主要的儲存體（如硬碟），Linux使用initramfs-XXX內的檔案
  - initramfs-XXX內部包含了驅動程式
  - Linux kernel預設會執行initramfs-XXX內的/init，因此/init必須是執行檔

initramfs-XXX與busybox

- initramfs-XXX裡面放足夠多的東西，讓Linux kernel可以驅動周邊即可（尤其是可以驅動開機用的block device）
- Busybox是專為嵌入式系統設計的一組套件，能夠產生常見的系統軟體，因此我們可以用busybox產生一個initramfs-XXX要用的檔案系統雛形
- 在這份作業中，initramfs-XXX的檔案系統雛形放在/usr/src/initrams
- 將busybox產生的檔案複製到/usr/src/initrams
  - 在busybox下，makemenuconfig，make install
  - cp busybox/_install /usr/src/initrams

initramfs-XXX與busybox

- Busybox的重點在於選擇static link（除非我們將動態函數庫全部包入initramfs）
  - settings -> Build static binary (no shared libs)
- 使用mknod在/dev底下製造「裝置檔案」
  - 請自行參閱/usr/src/initrams/dev底下有哪些裝置檔案
  - 請自行參閱mknod的使用說明

在Linux source code中指定initrams的位置

- make menuconfig
  - General setup -> Initramfs source file(s)
  - 指定到/usr/src/initramfs
- 編譯核心
  - make –j8

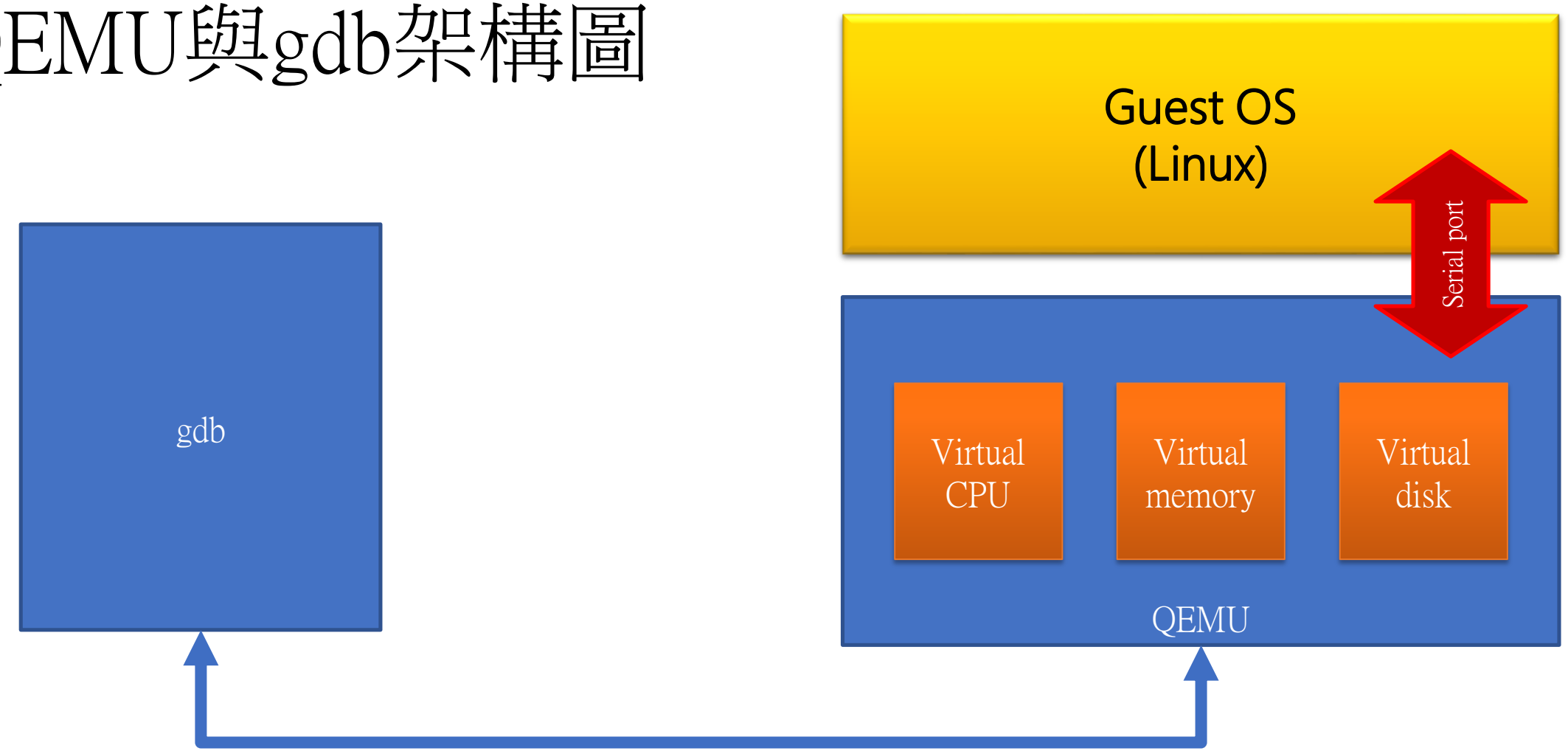
在kvm或者QEMU指定kernel及initramfs

- 分別使用參數-kernel -initrd設定
  - bzImage在/linux_source/arch/arch/x86/boot/bzImage
  - Initramfs在/linux_source/usr/initramfs_data.cpio.gz
- 還可以設定kvm或QEMU在開機後要mount的虛擬硬碟（optional）
  - -hda ./qemu.dsk
  - qemu.dsk的產生方式是：
    - qemu-img create -f qemu.dsk
    - kvm -boot d -cdrom image.iso -m 512 -hda qemu.dsk
    - 然後按照一般的流程安裝Linux
    - 建議選用沒有GUI的Linux，因為QEMU或KVM的執行速度不快（如：ubuntu server version）

KVM與QEMU的差別

- QEMU使用dynamic binary translation技術，因此速度慢，但可以模擬任何硬體（例如：ARM、MIPS、RISC-V）
- 由於QEMU使用純軟體模擬，因此對debugger（如：gdb）的支援非常良好
- 如果QEMU的執行平台和被模擬平台，屬於同一個指令集（instruction set），那麼可以使用KVM加速
- KVM執行速度較快，但是對gdb的支援較差
- 在本次作業僅使用KVM製造虛擬硬碟，其他部分使用QEMU

QEMU與gdb架構圖



很特別，這個地方是走TCP/IP，因此可以（如果需要的話）
可以遠端除錯

開始本次作業

啟動debuggee

- shiwulo@vm:~/qemu$ ./qemu.sh
- 啟動以後，QEMU會停住，等待gdb的連線

啟動debugger

- 另外開一個terminal，輸入
- shiwulo@vm:~/qemu$ gdb ./vmlinux
- gdb會讀取vmlinux的symbol table，進入gdb後請接下一頁

debugger - 追蹤初始化

```
(gdb) target remote localhost:666
Remote debugging using localhost:666
0x000000000000fff0 in cpu_hw_events ()
(gdb) b start_kernel
Breakpoint 1 at 0xffffffff82939cc9: file init/main.c, line 532.
(gdb) bt
#0  0x000000000000fff0 in cpu_hw_events ()
#1  0x0000000000000000 in ?? ()
(gdb)
```

接上一頁

(gdb) b trap_init
Breakpoint 2 at 0xffffffff82949959: file arch/x86/kernel/traps.c, line 950.
(gdb) b idt_setup_ist_traps
Breakpoint 3 at 0xffffffff82949a55: file arch/x86/kernel/idt.c, line 291.
(gdb) b syscall_init
Breakpoint 4 at 0xffffffff810a5750: file arch/x86/kernel/cpu/common.c, line 1530.
(gdb) b entry_SYSCALL_64
Breakpoint 6 at 0xffffffff81c00020: file arch/x86/entry/entry_64.S, line 214.
(gdb) b entry_SYSCALL_compat
Breakpoint 7 at 0xffffffff81c016b0: file arch/x86/entry/entry_64_compat.S, line 201.

接上一頁

(gdb) b do_IRQ

Breakpoint 9 at 0xffffffff81c01d20: file arch/x86/kernel/irq.c, line 233.

(gdb) b switch_mm

Breakpoint 13 at 0xffffffff810def80: file arch/x86/mm/tlb.c, line 147.

追蹤system call

啟動qemu以後，裡面有一隻程式叫做「hello」，同學們可以追蹤這支程式做了哪些system call，在kernel內部發生了什麼事情

學習目標

- 作業系統對於system call和interrupt的初始化動作的目的，主要是要讓這些事件發生時，讓硬體知道該到哪裡執行程式（設定program counter）及使用哪一個堆疊（stack）
- Intel的指令集非常的老舊，因此殘餘了很多『相容於歷史的設計』，比方說call gate、TSS、LDT、GDT、IDT，這些相容性的設計讓簡單的事情變得很複雜。
- 上述的x86系統架構於這份投影片附錄的地方，約略的做了講解

學習目標

- 同學們只需要『大約』看懂 C 語言的部分，知道在哪些函數設定了系統的進入點 (例如：system call、interrupt)
- 能夠交互比對上述的系統進入點，真的在相對應的事件發生的時候，CPU自動地切換內部的暫存器，到達kernel的特定位置開始對應的程序

評分方式

- 請任選一個你覺得有趣的地方，列出該函數的呼叫者是誰，該函數又呼叫了哪些函數，並且說明該函數中各個程式區塊的約略功能
- 請善用google
- 作業繳交期限11/05 23:59
- 繳交方式：助教將於10/24前公布

如何「玩」Linux

- 在我們啟動之後，Linux只開機到initramfs，已經將虛擬硬碟mount到/mnt/root中，同學們可以試著切換根目錄到/mnt/root模擬一下開機流程，了解Linux kernel開機完成後，如何執行user space的初始化
- 使用gdb對於大家覺得很奇怪的地方進行trace，例如：中斷怎樣處理、system call如何處理。於第二次作業中，無法對kernel進行完整除錯，但本次作業可以
- 本次作業的局限性：由於kernel是經過gcc最佳化過的，因此追蹤有些程式碼時「會亂跳」或者「追蹤不到」

參考文件

http://events17.linuxfoundation.org/sites/events/files/slides/entry-lce.pdf

https://www.kernel.org/doc/html/v4.14/process/adding-syscalls.html

附錄：關於x86硬體簡介

認識x86

- x86處理器，起始於1978年，8086的定址空間只有16bit（65K），大部分的電腦只有4～16K的記憶體
- 程式主要使用組合語言，例如：著名的DOS（Microsoft的作業系統）
- 為了方便程式設計師撰寫程式，組合語言的功能非常的強大，例如：有專門給字串處理用的組語
- 在記憶體方面也給程式設計師很大的方便，例如：
  - cs:0x800　　;代表程式區段第800道指令
  - ds:0x700　　;代表資料區段第800個位置的地方
  - ss:0x600　　;代表堆疊裡面第600號的位置

認識x86

- 下面位置中，cs、ds、ss稱之為區段，segment
  - cs:0x800　　;代表程式區段第800道指令
  - ds:0x700　　;代表資料區段第800個位置的地方
  - ss:0x600　　;代表堆疊裡面第600號的位置

segment

- segment是一個古老的設計，但x86-64傳承了自8086起的這個古老設計
- 在x86-64原則上將segment剔除，但是還是會看到
  - Segment selector（一個數字，用來選擇目前要用哪一個Segment descriptor）
  - Segment descriptor（一個資料結構，主要記載segment的開始位置、大小、存取的屬性）

X86-64的定址模式

注意名稱：叫做segment selector

後面會介紹怎樣用selector挑出一個真正的區段

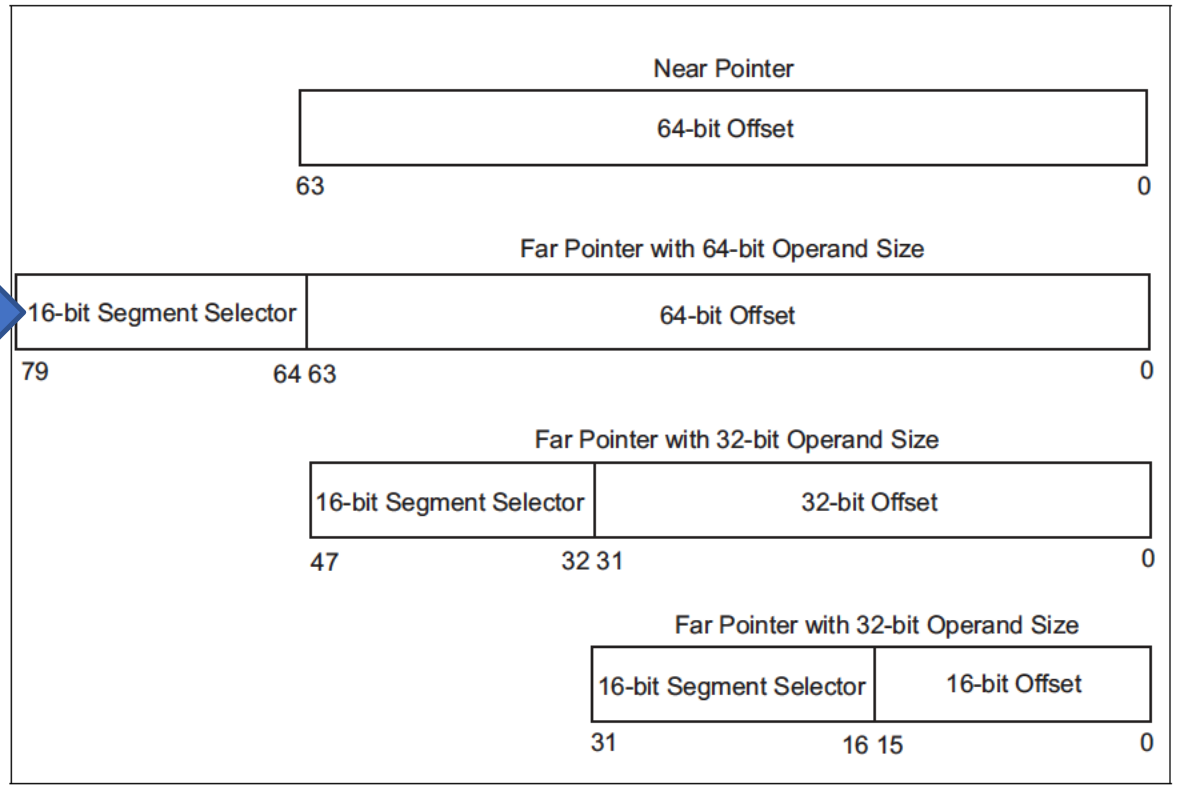


Figure 4-5. Pointers in 64-Bit Mode

x86-64中斷向量表

IA-32e system level registers

紅色部分是我們會用到的部分

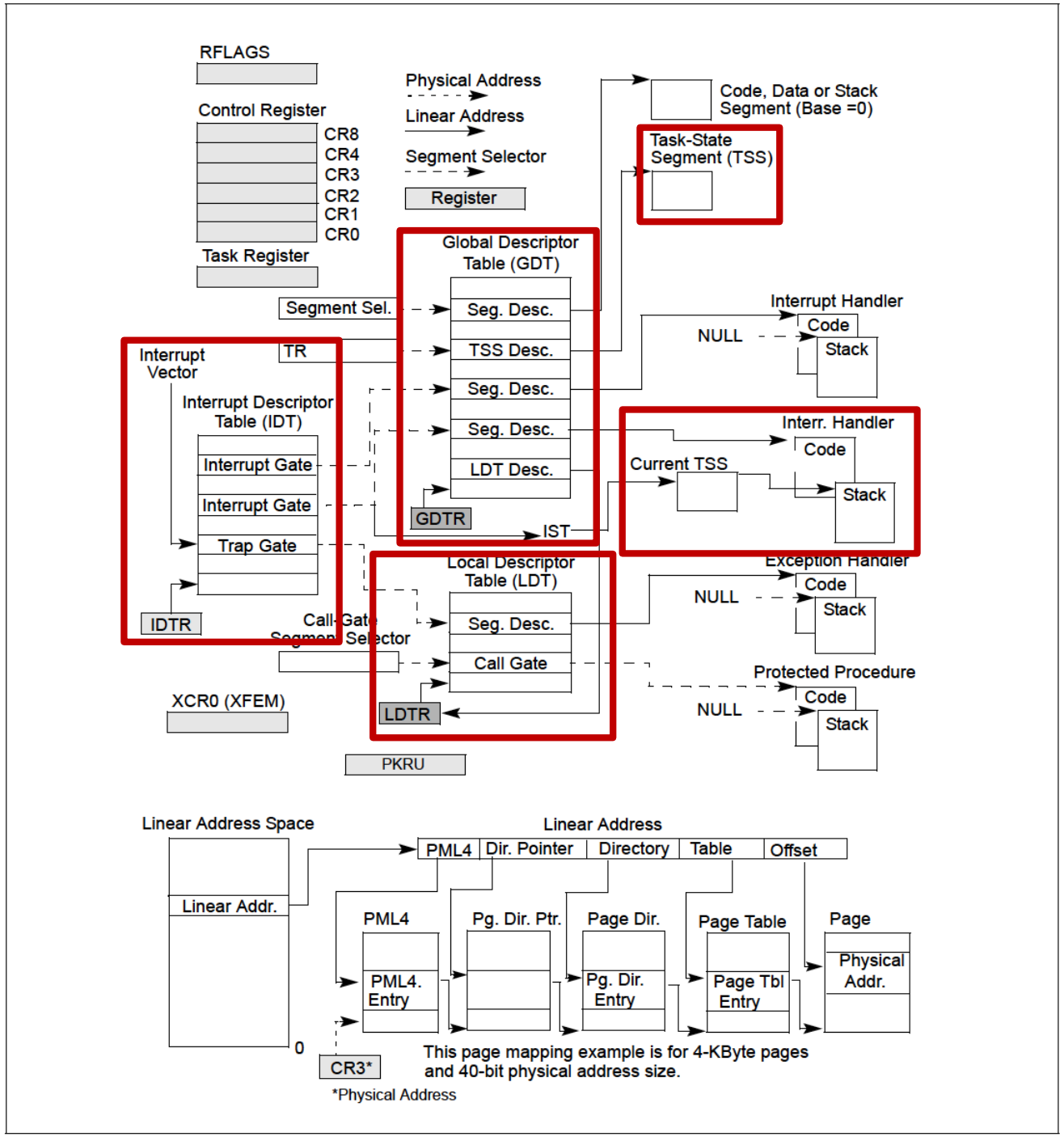


Figure 2-2. System-Level Registers and Data Structures in IA-32e Mode

IA-32e system level registers

- 藍色部分就是segment descriptor，系統有很多segment descriptor ，以陣列的方式存在
- 在x86-64中有三個segment descriptor「陣列」，分別用IDTR、LDTR GDTR指向這三個陣列
- segment selector就是index用來指向LDTR、GDTR，而IDTR陣列，就是中斷向量表

嚴格來說IDT內部放的東西應該要叫做gate，但其樣式跟seg. descriptor很像

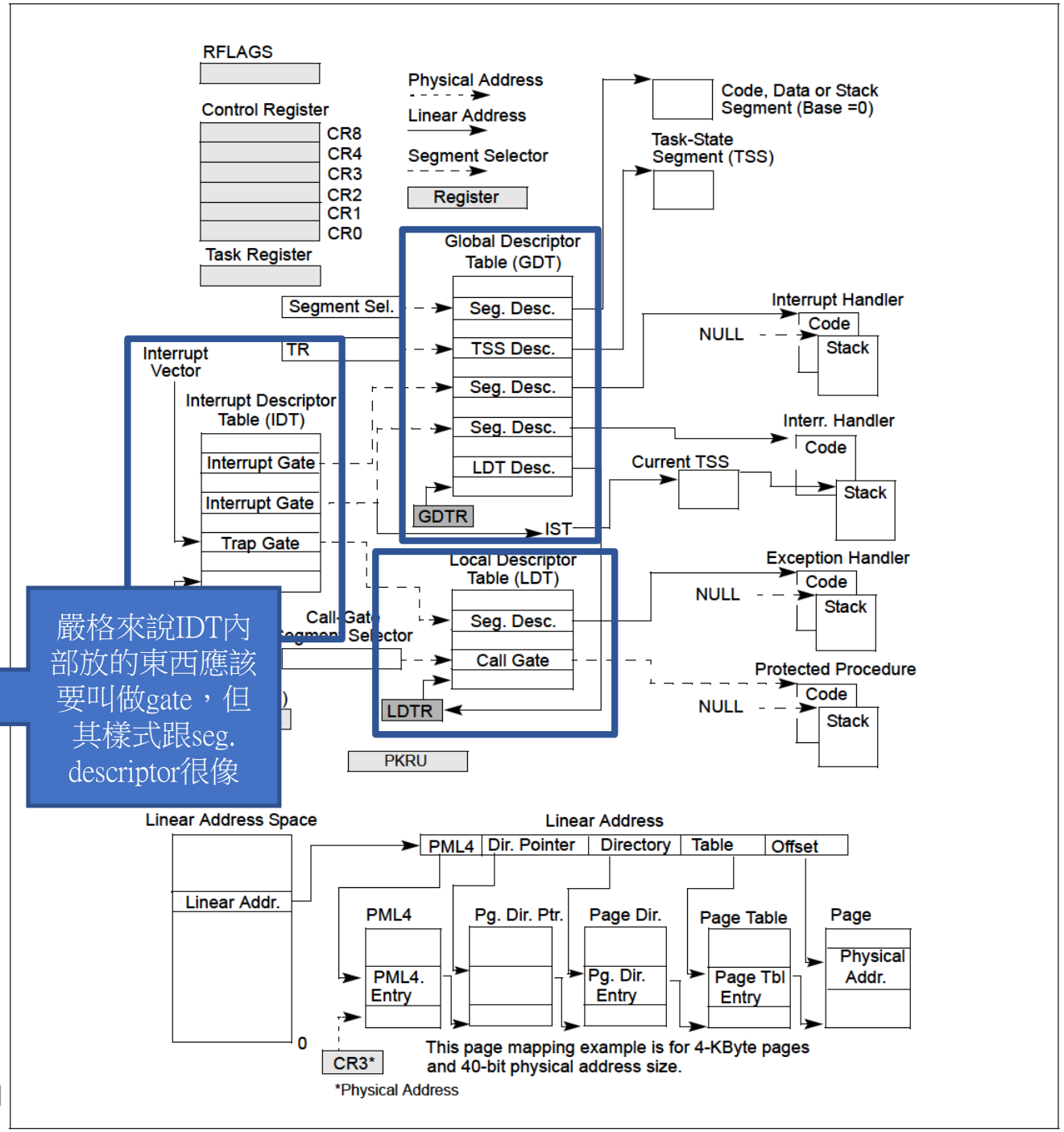


Figure 2-2. System-Level Registers and Data Structures in IA-32e Mode

IDT Gate Descriptor

中斷向量表的其中一個entry，大小為128 bits，紅色的部分共64 bits指向ISR

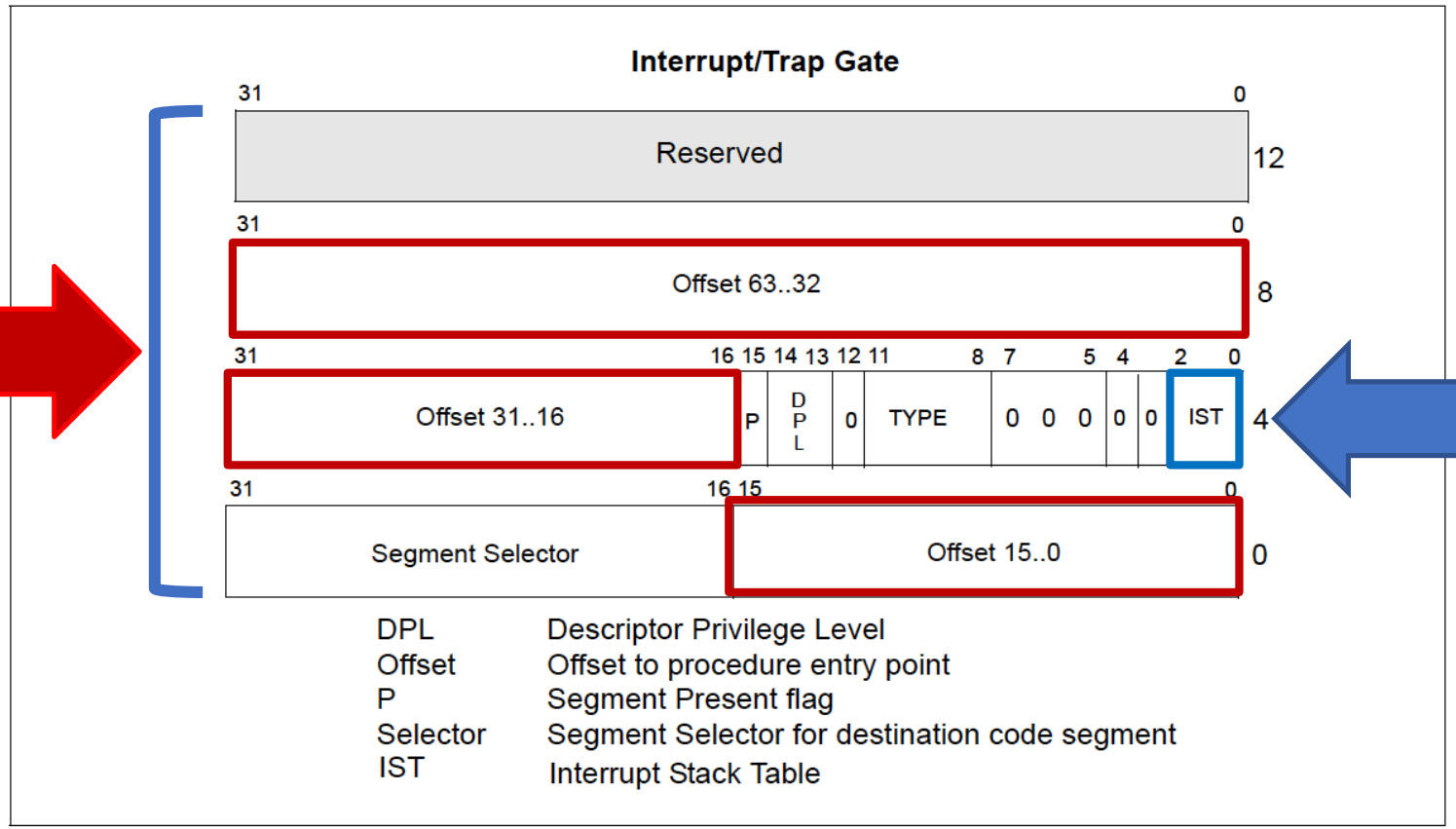


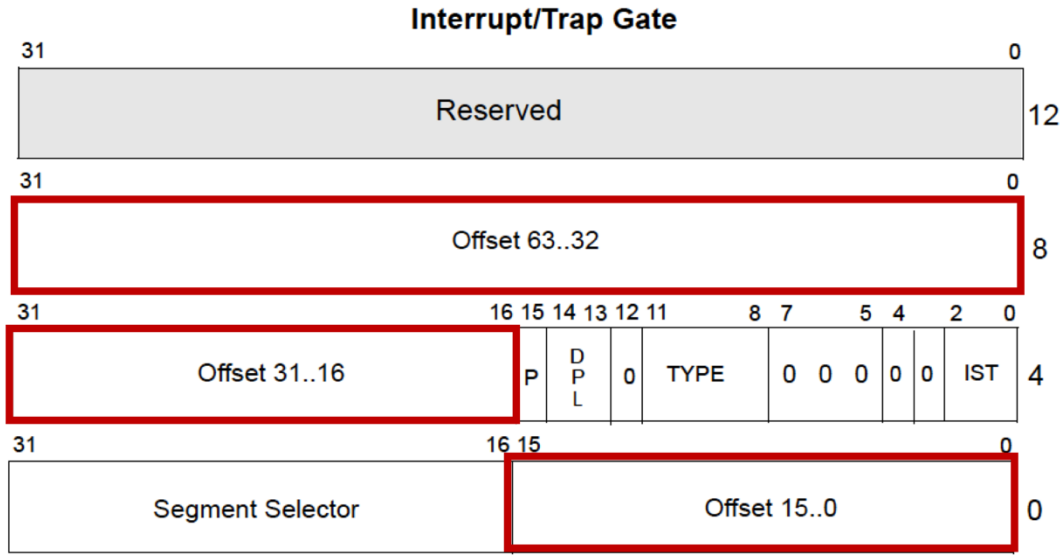
Figure 6-7. 64-Bit IDT Gate Descriptors

IST，值介於1~7，可以讓OS選擇發生中斷的時候要切換到哪一個stack

ISR：interrupt server routine

Segment descriptor：
以Interrupt vector table為例

- 右圖的重點在於紅色的部分，形成一個64位元的位址，將會指向ISR



Segment descriptor：
以Interrupt vector table為例

- 藍色的部分是一個不太重要的欄位，segment selector，於下一頁解釋

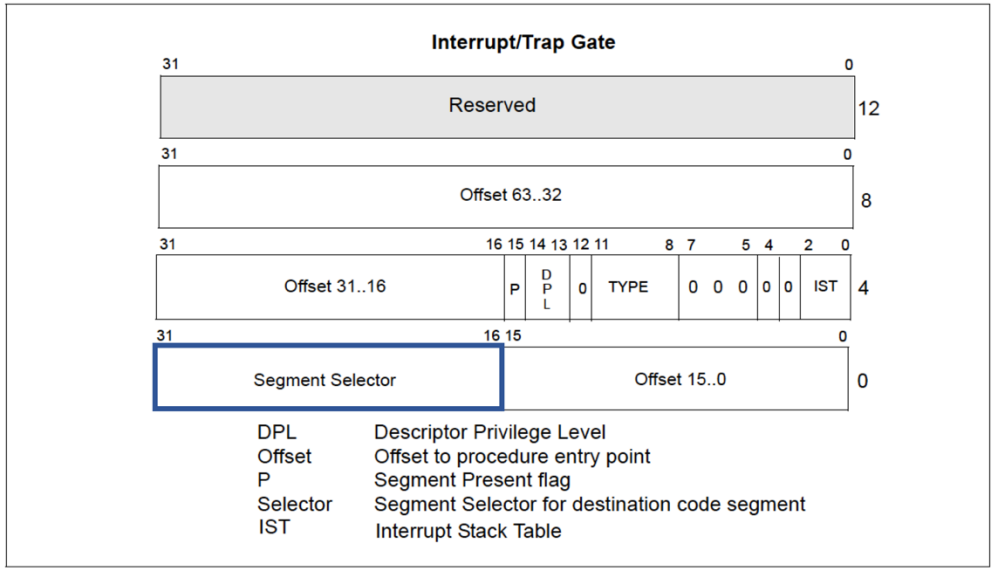
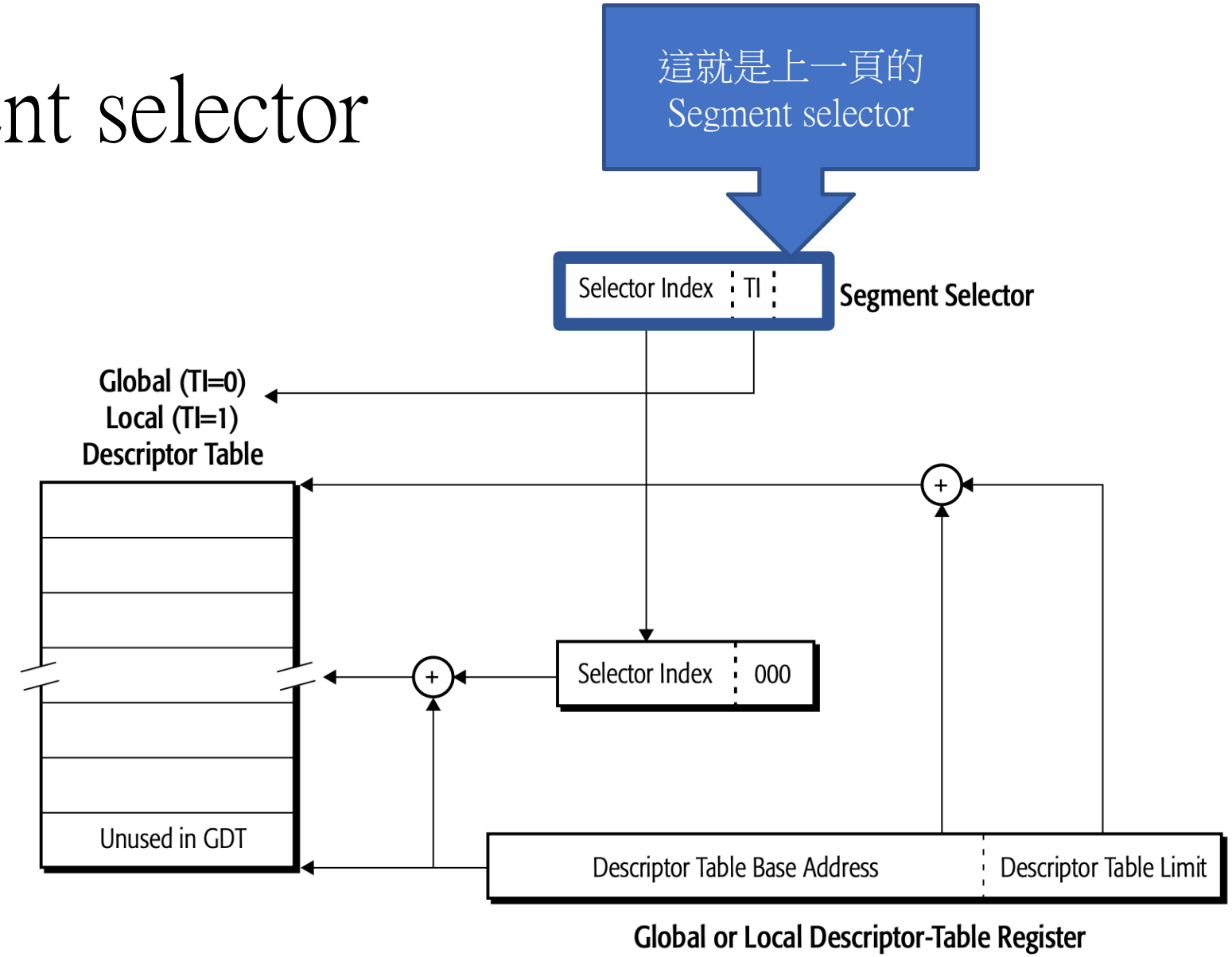


Figure 6-7. 64-Bit IDT Gate Descriptors

Segment selector



segment selector的細節

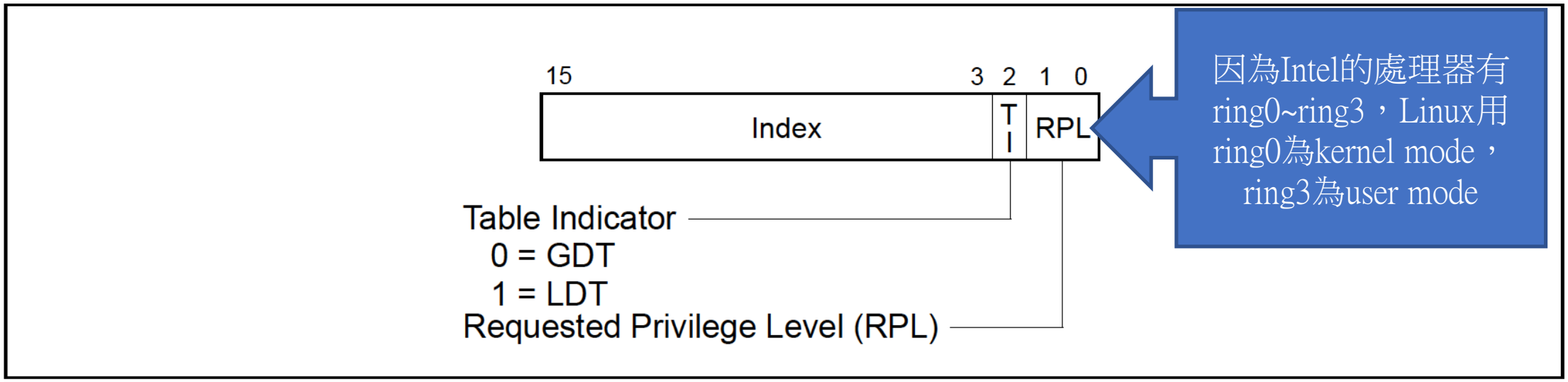
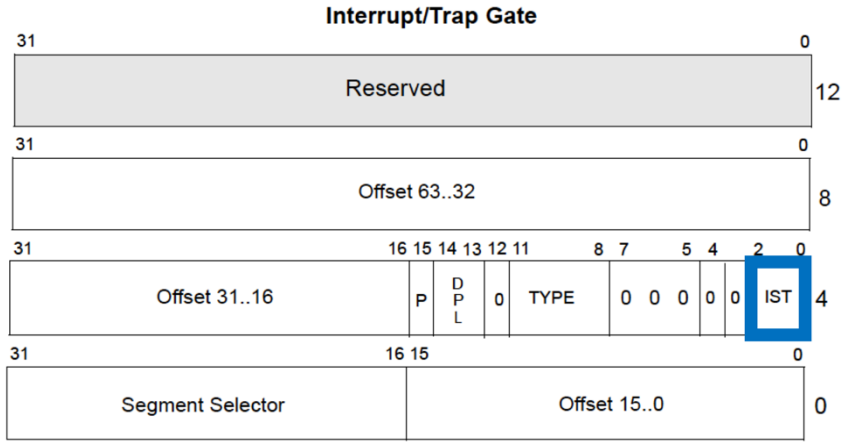


Figure 3-6. Segment Selector

Task state segment（tss）



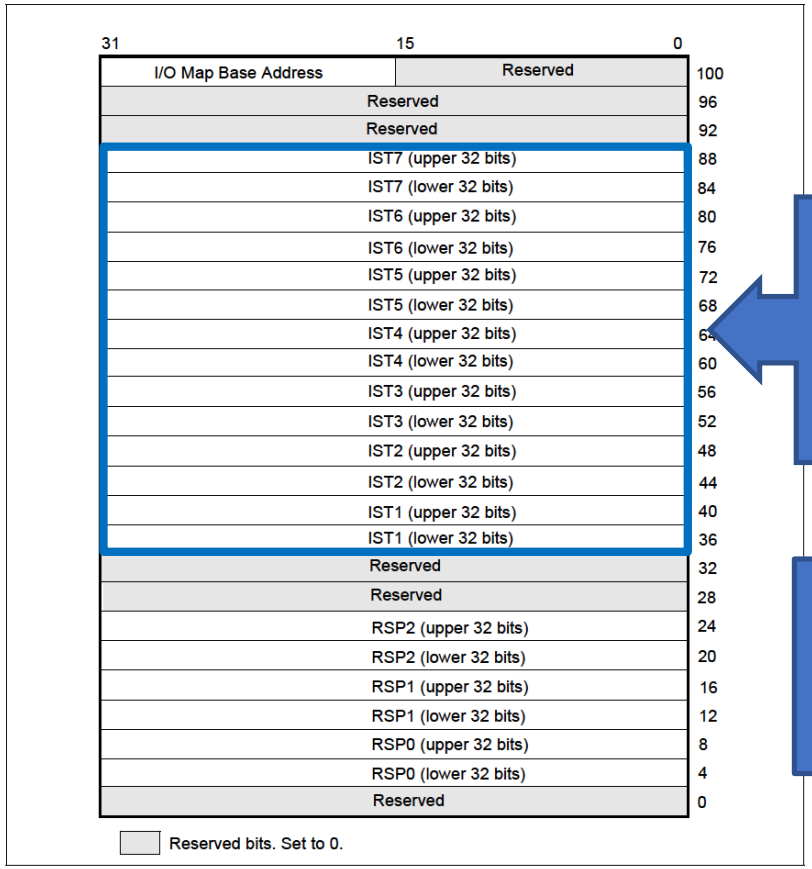


Figure 7-11. 64-Bit TSS Format

「IST」記錄在TSS七種stack的選擇

藉此，Linux給中斷一個獨立stack

system call

syscall & sysret

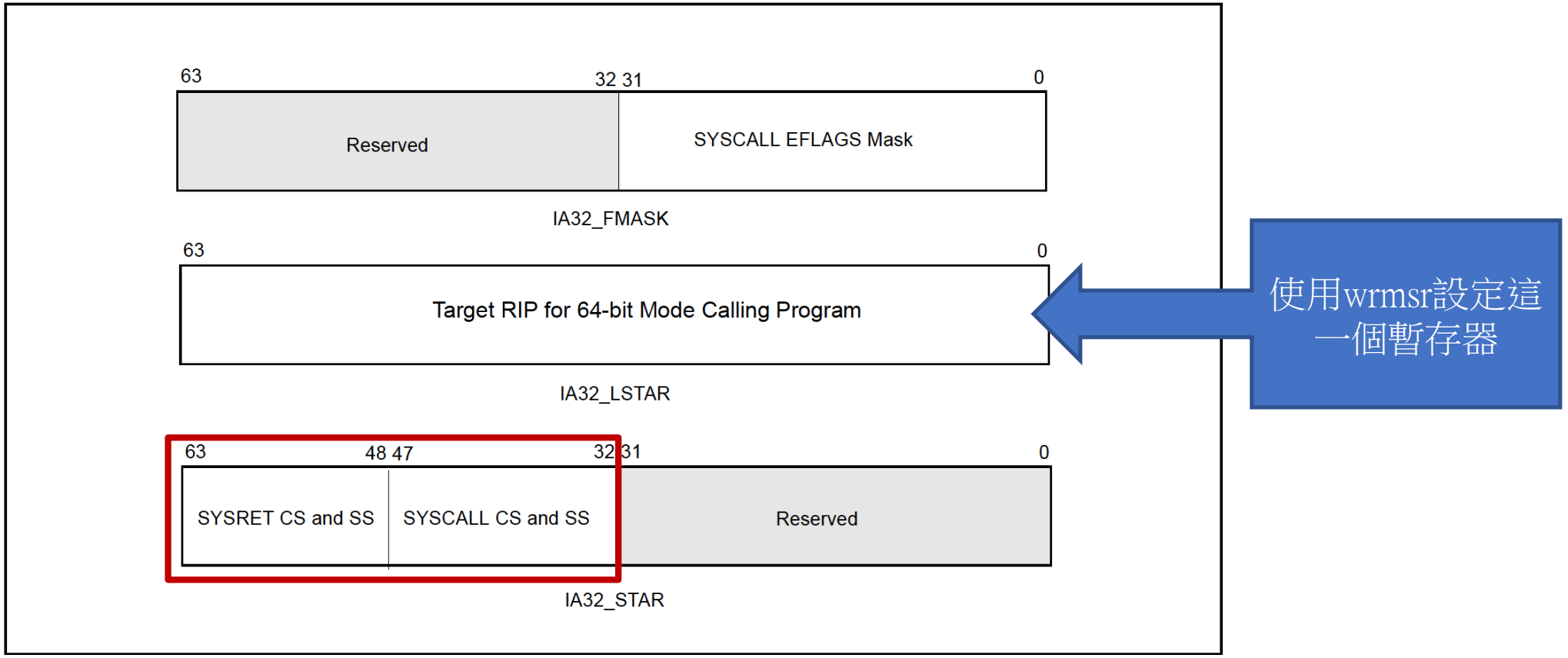


Figure 5-14. MSRs Used by SYSCALL and SYSRET