**Linux Operation System**

Project 1

106522102 陳品豪

106522066 賴孟昇

106522033 陳韋佑

專題一我們以Fedora 22 Workstation x86\_64進行trace code，之所以要選此系統的原因為，它是我們找到Kernel版本中，版本最接近作業需求— 4.0的系統，Fedora 22的Kernel 版本為**4.0.4**。

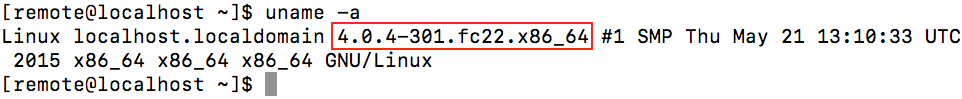


圖 1 – 文件中所找到的內容皆來自於 Linux Kernel 4.0.4

此專題，我們花了非常大量的時間在追朔Linux Kernel source code的邏輯，並且將 Linux Kernel 中的 Segmentation 及 Paging 架構解析出來。由於投影片內容有參考大量的程式碼，故附有大量的參考圖片，以至於不方便全數放上文件中，所以大部分的程式碼參考內容皆存在投影片中。本文件僅以文字配合其中較具有代表性的圖表來解說大致上的架構，所有介紹到的架構流程都有相對應的程式碼佐證，若在此文件中未發現相對應的程式碼區塊，煩請參考投影片部分。

目錄

1. [The segmentation and page unit of x86\_64 2](#_Toc502092507)
2. [**Segmentation：** 2](#_Toc502092508)
3. [**Page：** 7](#_Toc502092509)
4. [Linear Address Arrangement in Linux x86\_64 9](#_Toc502092510)
5. [The size of Linux kernel 4.0 10](#_Toc502092511)

為了本文件整體敘述的流暢性，我們將作業回答的部分做了整合，**目錄1**包含作業1-1, 1-2，**目錄2**包含作業1-3, 1-5，**目錄3**則為作業1-4。

The segmentation and page unit of x86\_64

* **Segmentation：**

Linux x86\_64 (64-bit)下和x86 (32-bit) 間的Segment有些異同，首先**GDT(Global Descriptor Table)** entry數量由32個減少為16個，可由下列的程式碼和圖表中看出差異



圖 2 – Kernel 中所定義的 GDT Entries

GDT表中的 Entry 數量減少許多，因為 x64架構下有很多東西都是直接將 Logical Address 映射至 Linear Address，圖3為Linux 在 x86\_64 架構下的GDT Entry，其中**KERNEL32\_CS**, **USER32\_CS**是因為Linux要相容於x32架構的應用程式（Compatibility Mode），故Code Segment在Kernel及User都各有一個給相容模式用來置換的位置。



圖 3 – x86\_64 架構下的GDT 表

下圖為舊的x32架構的GDT表，主要差異在**PNPBIOS**及**APMBIOS**在x64架構的GDT已經被拿掉了。



圖 4 – x86 架構下的 GDT 表

然而Linux在剛開始booting的時候只有少數幾個GDT entries，booting後才會將16個entries補齊。



圖 5 – Booting 時的 GDT 表定義



圖 6 – Booting 時的 GDT 表

GDT的Data Structure也由原來的x32架構的**8 bytes**擴增為x86\_64 的**16 bytes**，由程式碼與架構圖可以看出每個entry使用了16 bytes來定義，且offset被分為三個部分，大小共為64 bits。

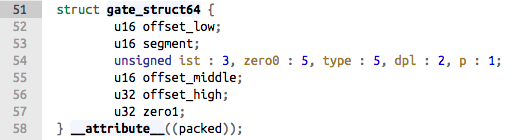


圖 7 – 86\_64 架構的 GDT Entry Structure



圖 8 – x86\_64 架構的 GDT Entry （16 bytes）



圖 9 - Base Address被分為三個部分

**Segment Selector**的部分，x86\_64則與x86相同，仍然由16個bit組成，且由低位到高位分別為RPL (2 bits), TI (1 bit)以及index (13 bits)。



圖 10 – Segment Selector

Linux 使用了遮罩來取得index, TI 及 RPL，如下圖所示，0x3相當於[11]，也就是最後2個位元代表了RPL；0x4相當於[100]，也就是第三個位元代表TI。



圖 11 – Selector 使用遮罩的方式定義各位元的功能

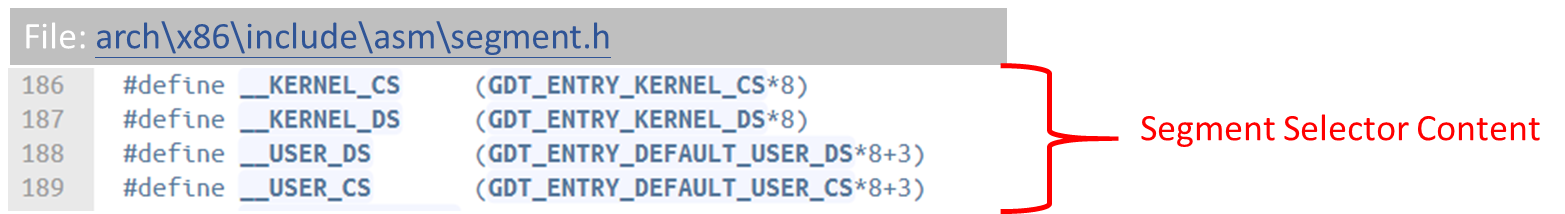


圖 12 - Linux x86\_64 使用四個GDT Entry表示Segment   
(不含32位元相容模式)

Linux 透過**pack\_gate**這個程式產生GDT entries，每個entry分別包含了64位元的 offset (base address) 以及相關 flags。



圖 13 - Create GDT Entry

Linux x86\_64的logical address包含了16-bit的segment selector及64-bit的offset。Kernel先透過TI欄位判斷要透過GDT還是LDT將logical address轉換為linear address，再來由GDTR/LDTR找到GDT/LDT的位置，將Segment Selector內index乘上16 (每個Entry 16 bytes) 加上GDTR/LDTR即可找到Segment Descriptor ，Segment Descriptor內存著64-bit的base address，將此base address加上logical address內的offset轉換為linear address。

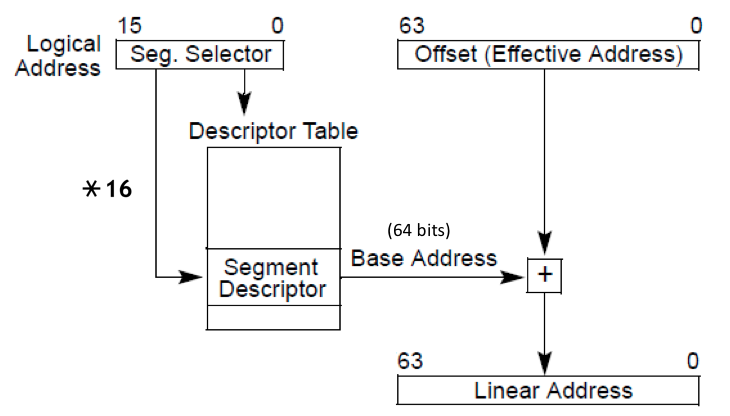


圖 14 - Convert Logical Address to Linear Address

* **Page：**

Linux x86\_64的Page為四層式架構 (4-Level Page Tables)，且僅使用了64 bits中的48 bits進行定址。此48 bits被分為4個9-bit的table及12-bit的index，如下圖所示。



圖 15 – Linear Address 切割成4張表及Index

4個Table分別為**Page Global Directory (PGD)** **Table**, **Page Upper Directory (PUD) Table**, **Page Middle Directory (PMD) Table**以及**Page Table Entry (PTE)**，且每個Table皆由512個8 bytes的Entry組成，

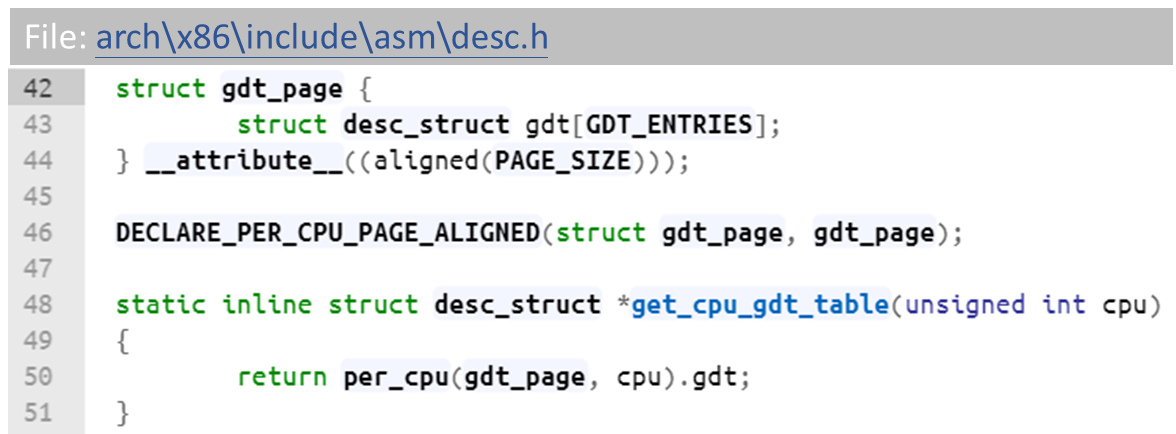


圖 16 - 每個CPU都有自己的GDT



圖 17 – Linux x86\_64 四層表架構

每個 Table Entry 都會記載 **Page Frame Number(PFN)**，相當於一個實體位置 (Physical Address)，如下圖所示，PFN 0就相當於實體記憶體中的第一個4KB page，在該位置可以查詢到一個page或者下一張表的位置，因為一張表剛好會佔用一個4KB Page。從PGD接連一路查詢到PTE後，最後的PFN將指向目標區塊的4KB Page。若PMD直接指向一個Page，該Page將會是一個2MB的Page，PUD也可以直接指向1GB Page，但PGD則無法直接指向更大的Page。

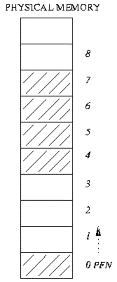


圖 18 - PNF 對應 Physical address

Page Entry佔用了8 bytes，如下圖所示，一個Table包含512個Entry，所以一張Table仍然為4KB，剛好佔用一個4KB Page。PFN佔用了34個bit，也就是可以指向2^34個4KB，因此最多可以指向64TB實體記憶體位置。

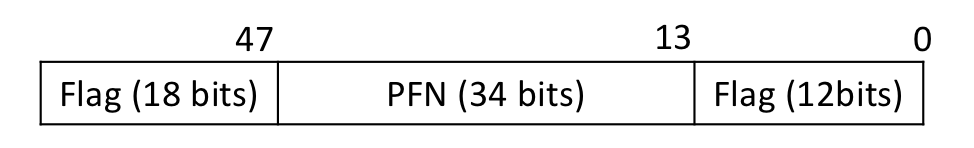


圖 19 - Page Table Entry （8 bytes）

當查詢到PFN就相當於查詢到實體記憶體位置，但若加上 **Virtual memory map**的offset就可以取得一個 **struct page**物件，該物件描述了此Page的描述及狀態，包含被多少程序映射、被多少程序存取中或是否被swap，相當於該Page的Descriptor。

### Linear Address Arrangement in Linux x86\_64

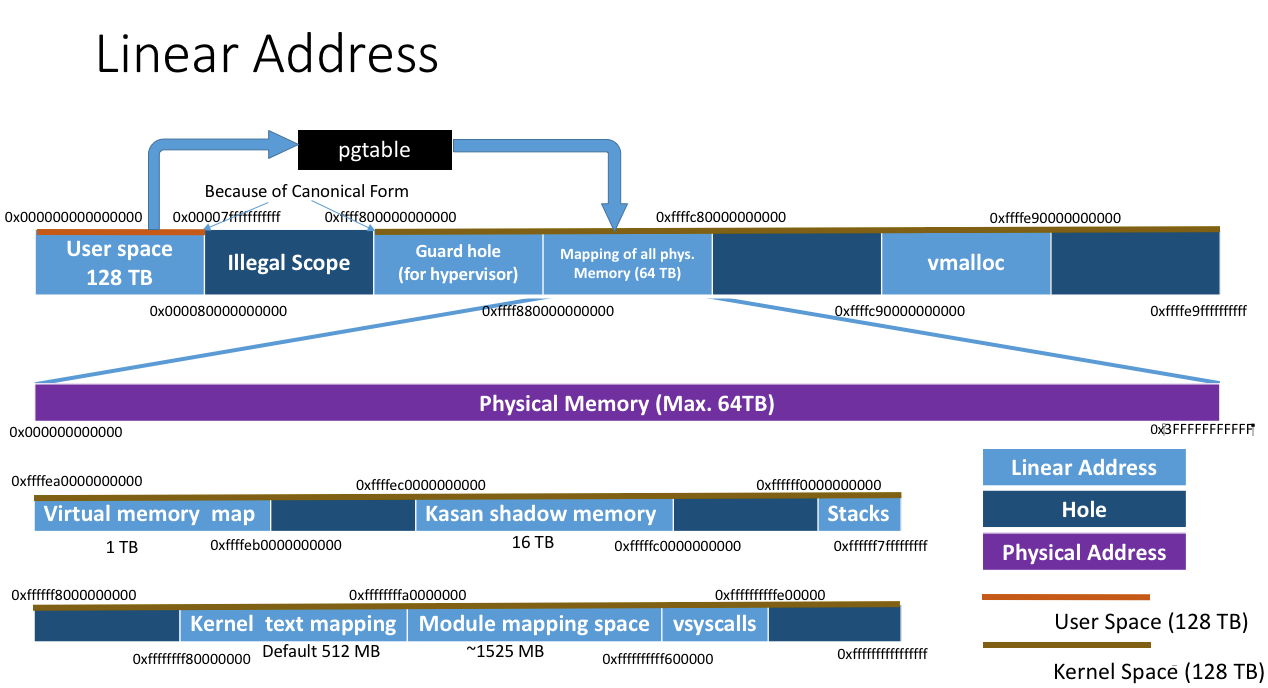


圖 20 - 虛擬位置表

Linux x86\_64 支援了256TB的虛擬位置定址，分成了兩等分，前後各128TB，中間為無效區域，Linux採用了**Canonical Form**，也就是在Linux 4.0中的Linear address指使用了48個bits，剩餘前面的16個bits要重複第48的bit，也就會造成00..007ff..ff會直接跳到ff..ff800..00的原因，若寫成二進為表示為：

* 000000..0**0**1111111..111 (第48bit為0，前面16位元都要為0)
* 111111..1**1**0000000..000 (第 48bit 為 1，前面16位元都要為1)

前半部的128TB為User Space，後面的128TB為Kernel Space。User Space 的128TB為每個Process 個別擁有，也就是 process A 及 process B 各擁有自己的 128TB。

Kernel Space為後半部128TB的位置，包含了64TB的實體記憶體參照(Linux x86\_64記憶體最高支援到64TB)、vmalloc、虛擬記憶體映射、KASAN區塊、Kernel Stack、Kernel Code(Kernel text mapping)、Module映射及system call。

虛擬記憶體映射存放了所有Page的資訊 (請參考投影片struct page)，KASAN區塊用於動態記憶體偵錯，每8 bytes會有1 byte 的KASAN shadow，也就是映射128TB的User Space需要有16TB的 KASAN大小。

### The size of Linux kernel 4.0

以我們使用的Fedora 22為例，在boot資料夾下可以觀察到Kernel檔案為vmlinux-4.0.4-301.fc22.x86\_64，且其檔案大小為5900kB(1k=1000)，所以Linux Kernel 4.0.4 size為5900kB 。

此外，我們發現，一樣都在Kernel 4.x.x這個版號中，不同的小版號所編譯出來的kernel大小略有不同，版本越新檔案越大，我們分別針對Kernel 4.0.4, Kernel 4.4.14, Kernel 4.11.8以及Kernel 4.13.8四種版本進行分析，大小分別為5900kB, 6160kB, 7300kB, 7400kB，如下圖所示。

|  |  |
| --- | --- |
| Kernel Version | Size |
| Vmlinuz-4.0.4-301 | 5900 kB |
| Vmlinuz-4.4.14-200 | 6160 kB |
| Vmlinuz-4.11.8-300 | 7300 kB |
| Vmlinuz-4.13.8-300 | 7400 kB |

表 1 - Linux Kernel Size

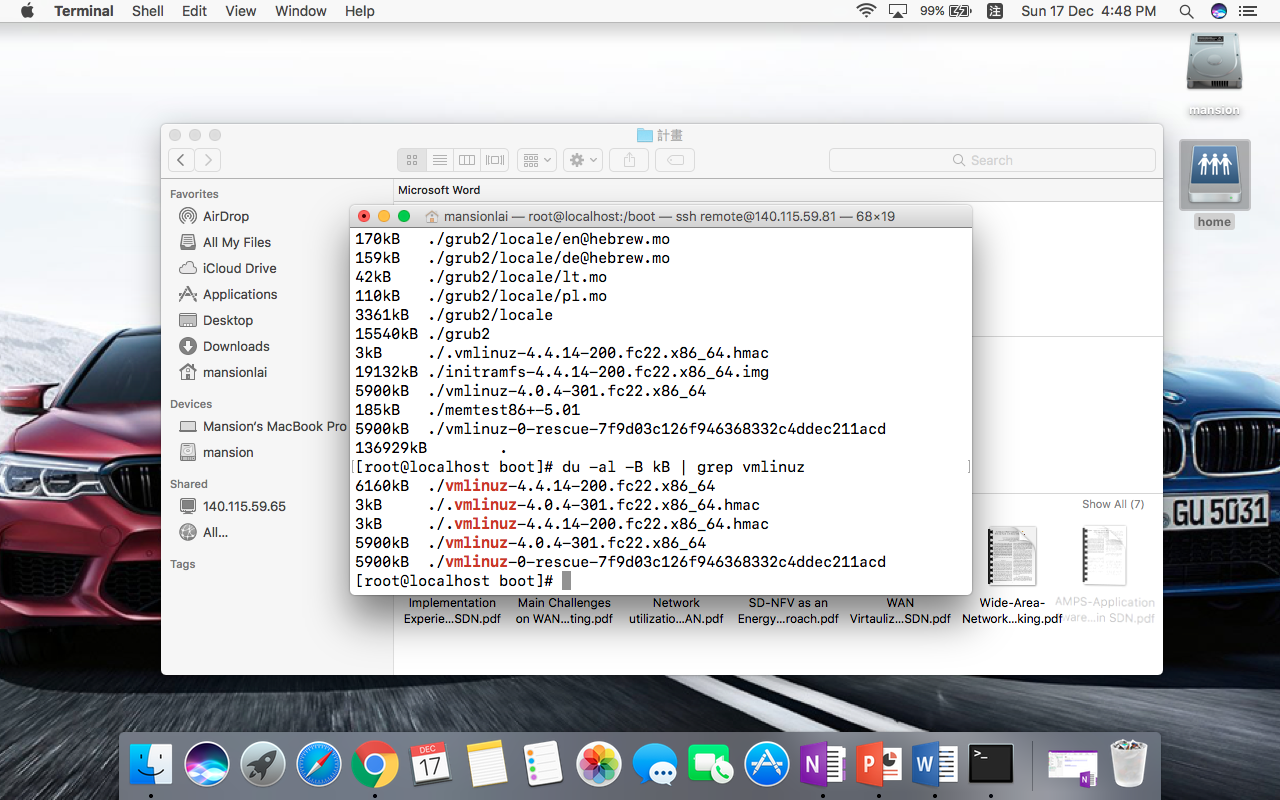


圖 21 - Kernel Size ( version 4.4.14 and 4.0.4 )

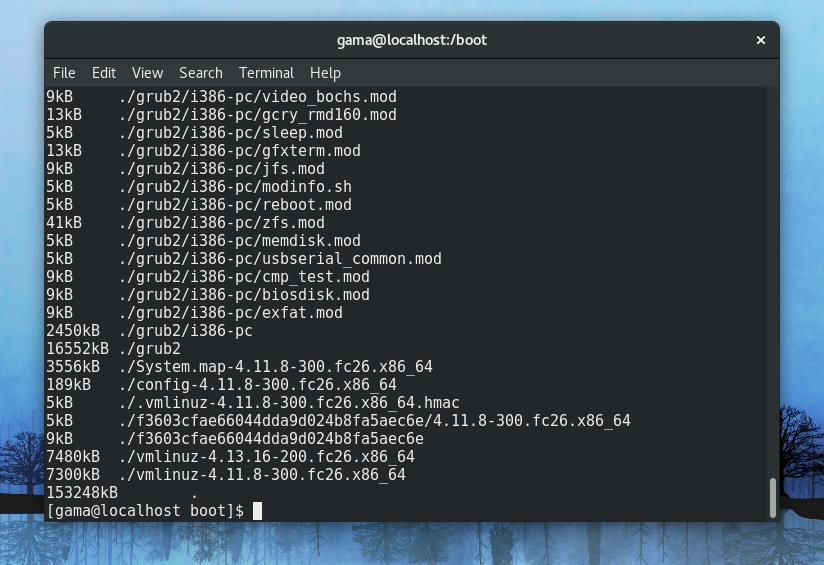


圖 22 - Kernel Size ( version 4.13.16 and 4.11.8 )