

ARMv8 與 Linux的新手筆記

ARMv8 與 Linux的新手筆記

by lod

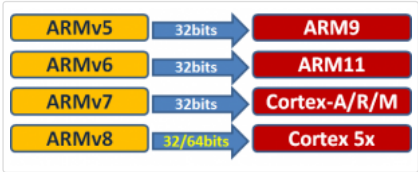
hlchou@gmail.com

從iPhone 5S採用ARMv8處理器架構後,對於ARMv8 64bits的相關討論很多,也受到大家關注,Google也如預期在2014年底推出了Android Lollipop (也就是Android 5.0) 操作環境.(官方網站<http://www.android.com/versions/lollipop-5-0/>) ,這也是目前第一個同時支援32bits與64bits執行環境的Android操作環境,而面對新的ARMv8 64bits技術,目前主要還是以ARM本身所公布的技術資料為主,筆者也會基於ARM與Google Android正是公開的技術資料為主,來跟大家介紹ARMv8相關的軟體開發技術,並會佐以Linux Kernel Source Code做對照,讓大家除了知道技術名詞外,也可以實際的在代碼上有所掌握,,期待這能對產業軟體開發者有一點點幫助.

首先,讓我們簡單的回顧一下ARM處理器的歷史,簡單的區分,像是ARMv5核心架構主要用於ARM9處理器系列,而ARMv6的核心架構,則用於ARM11系列產品,雖然前述處理器架構都是32bits的指令,但為了更節省記憶體的需求,ARM也推出了16bits Thumb指令,用以在記憶體受限的環境下,可以在些微影響效能的前提下,達到兩者的平衡,隨後又延伸這個需求,提供 16/32bits 效率更高的Thumb2指令集架構. 再來就是,近期智慧型手機最主流的Cortex-A系列處理器,則採用的是32bits ARMv7的處理器架構技術,而在導入64bits ARMv8架構前,也開始有一些更大記憶體需求的產品應運而生,ARM提供給32bits架構不超過4GB記憶體限制的解決方案就是LPAE(Large Physical Address Extension)技術,用以支援最大40bit的實體記憶體定址範圍,然在這機制下,每個Process所見的記憶體空間仍受限於32bits 4GB定址.

本文所主要討論的ARMv8架構,同時支援了 32bits 與 64bits兩個模式,並在32bits模式下,也向後相容之前ARMv7架構32bits的軟體產品,藉此可提供目前現有ARM 32bits架構的產品一個平順過渡到ARMv8 32/64bits架構的無縫接軌. ARM並支援在ARMv8 64bits的產品上,同時執行32bits的既有軟體模塊與針對最新64bits軟體架構所設計的軟件,兩個不同指令集與記憶體框架的行程可以同時執行,並透過系統提供的IPC(Inter-Process Communication) 進行協同工作.

參考ARM網頁(<http://www.arm.com/zh/products/processors/cortex-a/cortex-a53-processor.php>),目前已知對外公開的ARMv8架構包含了Cortex A53與A57,這兩者可用於搭配 big.LITTLE的架構,或可單獨採用.



如下表,為大家常見的ARMv7與ARMv8處理器的簡要比較,供參考.

CPU	Core Version	Pipeline	DMIPS/MHz	Physical Memory Addresses
Cortex A7	ARMv7	In-Order	1.9	LPAE 40bits
Cortex A15	ARMv7	Out-of-Order	3.5	LPAE 40bits
Cortex A53	ARMv8	In-Order	2.3	40bits
Cortex A57	ARMv8	Out-of-Order	4.1	44bits

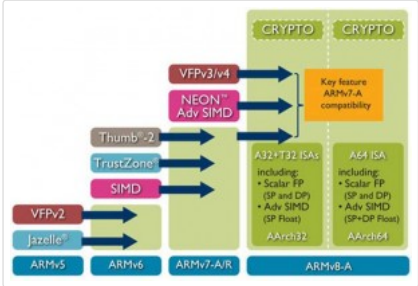
由於本文會有不少ARMv8 與 32/64bits相關技術名詞引用,為了便於訊息的一致性,筆者在此先定義如下

Items	Explain
AArch64	指基於64bits運作的ARMv8 Architecture.(General Purpose Registers X0-X30)
AArch32	指基於32bits運作的ARMv8 Architecture.並且相容於原本的ARMv7 Architecture. (General Purpose RegistersR0-R15)
A64	指在AArch64模式下支援的 ARM 64bits 指令集.
A32	指ARMv7架構下支援的 ARM 32bits 指令集,在ARMv8中也有新加入的A32指令集.
T32	指ARMv7架構下支援的 Thumb2 16/32bits指定集,在ARMv8中也有新加入的T32指令集.

本文所引用的軟體代碼,會以筆者撰寫本文時參考的Linux Kernel 3.16.3 為基礎,線上瀏覽的Source Code網址可以參考 <http://hala01.com/src/linux/linux-3.16.3/HTML/> .最後,撰寫時雖盡力確保資料無誤,若仍有所疏失還請多加包含.

ARMv8的基礎認識

目前的理解,談到ARMv8最多人引用的圖會是ARM網站(<http://www.arm.com/zh/products/processors/instruction-set-architectures/index.php>)所提供如下的架構示意圖 (http://www.arm.com/zh/images/roadmap/V5_to_V8_Architecture.jpg). 簡要來說, ARMv8的架構沿襲以往ARMv7 與之前處理器技術的基礎,除了有既有16/32bits Thumb2指令的支援外,也向前相容現有的ARM 32bits指令集. 基於64bits的AArch64架構,除了新增A64 (ARM 64bits)指令外,也擴充現有的A32 (ARM 32bits) 與T32 (Thumb2 32bits)指令編碼可在ARMv8 AArch32架構中執行,因此若有特別針對ARMv8 32bits指令集新增指令所撰寫的程式碼(例如: 32bits Crypto),這類程式將只能在ARMv8或之後新的處理器架構上執行,而無法向前相容於ARMv8之前的32bits實體處理器執行上.



AArch64 表示為支援64bits Execution State,而AArch32則是支援 ARMv8之前的32bits Execution State,並有因應AArch64新增額外的能力,確保相容於ARMv7-A的架構. AArch32/64也都支援 SIMD (Single-Instruction Multiple-Data)與Floating-Point指令,其中的差異是,AArch32的SIND/Floating-Point使用64bits的暫存器,而AArch64使用的是128bits暫存器.

參考文件 Procedure Call Standard for the ARMv8 Architecture(http://infocenter.arm.com/help/topic/com.arm.doc.ih0042e/IH0042E_aapcs.pdf) 與 Procedure Call Standard for the ARM 64-bit Architecture(AArch64) (http://infocenter.arm.com/help/topic/com.arm.doc.ih0055b/IH0055B_aapcs64.pdf),其中有關General purpose registers and AAPCS64 usage段落有針對這31個General Purpose 64bits暫存器的使用方式,可知64bits提供的額外暫存器,包括在Function Parameter/Return值處理,以及可供函式內優化使用的暫存器數量都增加,善用這些額外暫存器可減少對外部記憶體存取頻率,並讓編譯器優化時,更有空間去改善執行效能.

AArch64 Register	Special	Role in the procedure call standard
x0...x7		Parameter/result registers
x8		Indirect result location register
x9...x15		Temporary registers
x16	IP0	The first intra-procedure-call scratch register (can be used by call veneers and PLT code); at other times may be used as a temporary register.
x17	IP1	The second intra-procedure-call temporary register (can be used by call veneers and PLT code); at other times may be used as a temporary register.
x18		The Platform Register, if needed; otherwise a temporary register.
x19...x28		Callee-saved registers
x29	FP	The Frame Pointer
x30	LR	The Link Register
SP		The Stack Pointer.

AArch32 Register	Special	Role in the procedure call standard
r0...r3		Parameter/result registers
r4...r8 r9 (also as platform register) r10,r11		Temporary registers
r12	IP	The Intra-Procedure-call scratch register.
r13	SP	The second intra-procedure-call temporary register (can be used by call veneers and PLT code); at other times may be as a temporary register.
r14	LR	The Platform Register, if needed; otherwise a temporary register.
r15	PC	Callee-saved registers

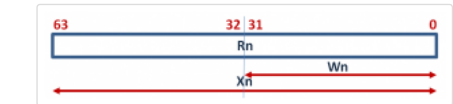
如下表,筆者嘗試從支援的暫存器/指令來分類AArch32 與 AArch64,以供簡要參考.

Execution State	Note
AArch64	1.提供 31個64bits 寬的General-Purpose Registers (from x0~x30,其中 x30亦可作為Procedure Link Registers) 2.提供64bits Program Counter(PC), Stack-Poiner(SP)與Exception-Link-Register (ELR) 3.提供 32個128bits 寬的SIMD Vector 與 Scalar Floating-Point暫存器 4.定義ARMv8 EL0~EL3共4個Execution Privilege 5, 支援64bits Virtual-Addressing 6.定義一組PSTATE用以保存PE(Processing Element)狀態.
AArch32	1.提供 16個32bits 寬的General-Purpose Registers (from r0~r12, 其中r13=SP, r14=LR and r15=PC, 並且r14需同時供ELR與Procedure Link Registers之用) 2.提供一個ELR,用以作為從Hyp-Mode的Exception返回之用. 3.提供 32個64bits 寬的Advanced SIMD Vector 與 Scalar Floating-Point暫存器 4.提供A32與T32兩種指令集的組態 5, 使用32bits Virtual-Addressing 6.只使用CPSR (Current Program State Register)保存PE(Processing Element)狀態.

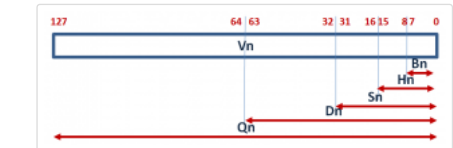
ARMv8共支援以下幾種Data types,

Data Type	Length
Byte (B)	8 bits.
Halfword (H)	16 bits.
Word (S)	32 bits.
Doubleword (D)	64 bits.
Quadword (V)	128 bits.

而ARMv8通用暫存器.可區分32bits (Wn)與 64bits (Xn)兩類,可供程式執行依需求使用.



至於SIMD/浮點數的部分,參考下圖可區分為8bits~128bits(16bits Half-Precision, 32bits Single-Precision and 64bits Double-Precision)不同的Size,並相容於IEEE 754.



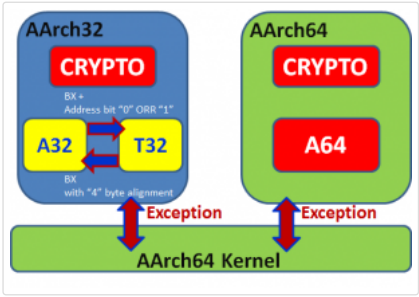
ARMv8跟之前ARM處理器相比,最大的亮點之一就是Crypto加密指令集的支援,目前這部份的支援主要包括基於SIMD指令的 AES,SHA1與SHA2-256硬體加速指令,可參考如下簡表.

Crypto	Explain
--------	---------

New ARMv32 Instruction	
AESD.8	AES single round decryption.
AESE.8	AES single round encryption.
AESIMC.8	AES inverse mix columns.
AESMC.8	AES mix columns.
SHA1C.32	SHA1 hash update accelerator (choose).
SHA1M.32	SHA1 hash update accelerator (majority).
SHA1P.32	SHA1 hash update accelerator (parity).
SHA1H.32	SHA1 hash update accelerator (rotate left by 30).
SHA1SU0.32	SHA1 schedule update accelerator, first part
SHA1SU1.32	SHA1 schedule update accelerator, second part
SHA256H.32	SHA256 hash update accelerator.
SHA256H2.32	SHA256 hash update accelerator upper part.
SHA256SU0.32	SHA256 schedule update accelerator, first part
SHA256SU1.32	SHA256 schedule update accelerator, second part
VMULL.P64	Polynomial multiply long, AES-GCM acceleration 64×64 to 128-bit.

綜觀上述的介紹後,筆者嘗試把AArch32與AArch64兩個執行環境用如下圖加以說明,希望會比較好理解兩者的差異. 紅色部分為這次ARMv8新增的模塊,而黃色部分則為ARMv7既有支援的指令集範圍,可以看到在AArch32的架構下,A32與T32指令集間可以透過BX搭配Address bit 0 為1的方式由ARM Mode切換到Thumb Mode,並可透過BX指令從Thumb Mode切換回ARM Mode,兩者轉換的成本很低. 且ARMv8所支援的Crypto指令也同時在AArch32與AArch64模式下都有支援,可用於加速這兩類指令集模式下加解密指令效率.

而跟過去習知ARM Mode與Thumb Mode指令集切換最大的差異在於,在AArch32與AArch64兩者32bits/64bits執行模式切換時,目前ARMv8架構下只能透過觸發Exception的方式進行切換.也因此這表示一個應用程式撰寫時就必須決定自己是要處於32bits 或64bits的場景.如果希望可以享受到兩個模式下的好處,就必須要同時具備32bits Process與64bits Process,兩者之間再透過IPC(Inter-Process Communication)進行溝通.



ARMv8把Execution Privilege區分為EL0到EL3共4個Level,根據目前的架構,會由下層系統的Execution State決定上層系統所在的模式.若下層系統為32bits,則上層就只能為32bits



反之,若下層系統為64bits,上層就可選擇為32bits或64bits兩者之一,也因此若想要同時讓你的處理器軟體執行環境支援32bits Process與64bits Process,就必須要使用 64bits的Kernel執行環境.

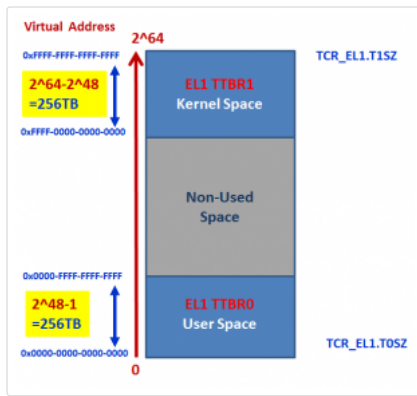


把ARMv8處理器的基礎差異說明後,接下來讓我們進一步嘗試說明ARMv8 AArch32與AArch64在執行時期的Memory體系為何,藉此可對應用程式運作有更進一步的了解.

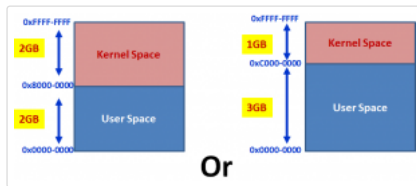
ARMv8 Memory Model

參考Linux Kernel 3.16.3(in /arch/arm/mm/proc-v7-2level.S, <http://hala01.com/src/linux/linux-3.16.3/HTML/S/19381.html>), ARM 32bits下會用TTBR0儲存當下User-Space行程所在的Page Table (也就是0xC0000000以下的記憶體空間),並用TTBR1儲存Kernel Space所在的Page Table (也就是0xC0000000以上的記憶體空間).

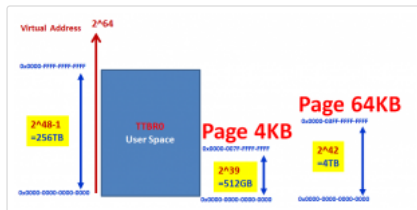
在ARMv8 64bits架構下,會透過EL1的TTBR0 (ttbr0_el1, in /arch/arm64/mm/proc.S, <http://hala01.com/src/linux/linux-3.16.3/HTML/S/22828.html>)儲存當下User-Space行程所在的Page Table,與EL1的TTBR1儲存Kernel Space所在的Page Table,並會依據Page Size與32/64bits行程而有不同的記憶體空間配置. 參考如下圖所示



如下為原本大家習知32bits Linux Kernel記憶體定址方式,依據需求開發者可以配置為 User與Kernel Space各2GB的Layout,或是修改為 User與Kernel Space分別為3GB與1GB的定址方式。



但由於ARM 64bits Kernel的分頁(Page)有4KB 與 64KB兩種大小,參考 TASK_SIZE_64 (/arch/arm64/include/asm/memory.h, <http://hala01.com/src/linux/linux-3.16.3/HTML/S/22730.html#L49>), 且ARM64 Linux Kernel下的high_memory會等於裝置所配置的實體記憶體大小加上PAGE_OFFSET,這塊記憶體可用於線性Memory Mapping實體記憶體與核心虛擬記憶體空間。若有配置Sparse Memory Virtual memmap用以優化 pfn_to_page/page_to_pfn 流程,則vmemmap 起點為Linear Mapping記憶體的起點 virt_to_page(PAGE_OFFSET)終點會等於virt_to_page(high_memory)。

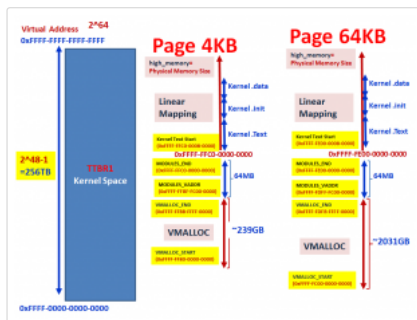


同時, Kernel Space的範圍會透過 $PAGE_OFFSET = (UL(0xffffffff) \ll (VA_BITS - 1))$ 取得(in /arch/arm64/include/asm/memory.h, <http://hala01.com/src/linux/linux-3.16.3/HTML/S/22730.html#L49>), 且ARM64 Linux Kernel下的high_memory會等於裝置所配置的實體記憶體大小加上PAGE_OFFSET,這塊記憶體可用於線性Memory Mapping實體記憶體與核心虛擬記憶體空間。若有配置Sparse Memory Virtual memmap用以優化 pfn_to_page/page_to_pfn 流程,則vmemmap 起點為Linear Mapping記憶體的起點 virt_to_page(PAGE_OFFSET)終點會等於virt_to_page(high_memory)。

當分頁大小為4KB時,決定PAGE_OFFSET大小的VA_BITS會等於 39,因此 $PAGE_OFFSET = (UL(0xffffffff) \ll (39 - 1)) = 0xFFFFFC0000000000$,而Kernel Driver所在的區間 $MODULES_END = PAGE_OFFSET = 0xFFFFFC0000000000$ 與 $MODULES_VADDR = MODULES_END - SZ_64M = 0xFFFFFBFFC0000000$, 並且 $VMALLOC_START = (UL(0xffffffff) \ll VA_BITS) = (UL(0xffffffff) \ll 39) = 0xFFFFF80000000000$,與 $VMALLOC_END = (PAGE_OFFSET - UL(0x400000000) - SZ_64K) = 0xFFFFFBFFBFFF0000$ 。

若分頁大小為64KB時,則PAGE_OFFSET對應的VA_BITS會等於 42,也就是 $PAGE_OFFSET = (UL(0xffffffff) \ll (42 - 1)) = 0xFFFFE00000000000$, 而Kernel Driver所在的區間 $MODULES_END = PAGE_OFFSET = 0xFFFFE00000000000$ 與 $MODULES_VADDR = MODULES_END - SZ_64M = 0xFFFFDFFFC0000000$. 並且 $VMALLOC_START = (UL(0xffffffff) \ll VA_BITS) = (UL(0xffffffff) \ll 42) = 0xFFFFC00000000000$,與 $VMALLOC_END = (PAGE_OFFSET - UL(0x400000000) - SZ_64K) = 0xFFFFDFFBFFF0000$ 。

有關Kernel Memory Mapping的概念,可參考如下示意圖



秉持一貫有Source Code有真相的原則,讓我們從Linux Kernel Source Code的角度來進一步的說明原委,首先從ARM64架構下一個應用程式從執行檔開始貼到記憶體布局的流程來加以說明,

1, 當使用者或應用載入一個新的程式時(例如呼叫load_elf_binary),就會透過函式setup_new_exec(in /fs/exec.c, <http://hala01.com/src/linux/linux-3.16.3/HTML/S/8148.html#L1101>)來為新的Process的記憶體布局進行配置。

2, 在函式setup_new_exec中,則會呼叫arch_pick_mmap_layout(current->mm);進入函式arch_pick_mmap_layout(in /arch/arm64/mm/mmap.c, <http://hala01.com/src/linux/linux-3.16.3/HTML/S/22812.html#L84>),依據不同的處理器差異(ARM 32bits/64bits,MIPS,Parisc,PowerPC,S390,Sparc,Tile,x86,...etc),為目前的Process 記憶體布局進行配置

3, 由於本文著重於ARM 64bits,因此會以ARM64中的函式 arch_pick_mmap_layout(in /arch/arm64/mm/mmap.c, <http://hala01.com/src/linux/linux-3.16.3/HTML/S/22812.html#L84>)為主加以說明,首先這函式可以分為兩個部分

```
void arch_pick_mmap_layout(struct mm_struct *mm)
{
```

```

if (mmap_is_legacy()) {
    mm->mmap_base = TASK_UNMAPPED_BASE;
    mm->get_unmapped_area = arch_get_unmapped_area;
} else {
    mm->mmap_base = mmap_base();
    mm->get_unmapped_area = arch_get_unmapped_area_topdown;
}
}
}

```

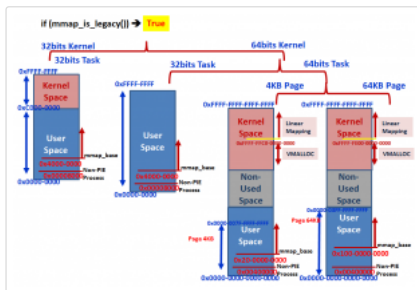
3.1. 若if (mmap_is_legacy())成立è

此時Task的mm->get_unmapped_area為arch_get_unmapped_area(in /mm/mmap.c, <http://hala01.com/src/linux/linux-3.16.3/HTML/S/9939.html#L1873>),表示該Task Process每一個新的Memory Mapping (包括應用程式本身,動態函式庫,MMAP配置...等),都會由低位址往高位址依序配置而上,這比較像是最早對於原本對於MMAP的概念,也就是Legacy的作法。

對應到 32bits Kernel, Task 的mm->mmap_base會等於TASK_UNMAPPED_BASE,若此時Kernel Space起點為0xC0000000,則會從0x40000000為起點。

若是 64bits Kernel, 則 32bits Task的TASK_UNMAPPED_BASE 會等於(PAGE_ALIGN(TASK_SIZE / 4))(in /arch/arm64/include/asm/memory.h, <http://hala01.com/src/linux/linux-3.16.3/HTML/S/22730.html#L65>),其中因為TASK_SIZE_32為0x100000000,因此會跟32bits Kernel下的32bits Task一樣都以0x40000000為起點。

若為64bits Kernel下的64bits Task, 由於 TASK_SIZE_64 等於 (UL(1) << VA_BITS),若該64bits Kernel使用4KB Page為單位,則VA_BITS 等於39 (若使用64KB Page為單位,則VA_BITS 等於42),對應到TASK_SIZE_64 等於(UL(1) << VA_BITS),因此在4KB Page設定下, TASK_UNMAPPED_BASE 透過(PAGE_ALIGN(TASK_SIZE / 4))會等於0x200000000 (若使用64KB Page,則為0x1000000000), 由於這些組合比較多,筆者把上述的組合繪製如下圖所示



3.2. 若if (mmap_is_legacy()) 不成立è

此時Task的mm->get_unmapped_area為arch_get_unmapped_area_topdown(in /mm/mmap.c, <http://hala01.com/src/linux/linux-3.16.3/HTML/S/9939.html#L1909>),表示該Task Process每一個新的Memory Mapping (包括應用程式本身,動態函式庫,MMAP配置...等),都會反過來,由高位址往低位址依序配置而下,這是屬於目前比較新的Android手機環境所要求的作法。在這架構上, Task 的mm->mmap_base會透過呼叫函式 “unsigned long mmap_base(void)” (in /arch/arm64/mm/mmap.c, <http://hala01.com/src/linux/linux-3.16.3/HTML/S/22812.html#L68>) 決定。

如下為函式mmap_base的實作

```

static unsigned long mmap_base(void)
{
    unsigned long gap = rlimit(RLIMIT_STACK);

    if (gap < MIN_GAP)
        gap = MIN_GAP;
    else if (gap > MAX_GAP)
        gap = MAX_GAP;

    return PAGE_ALIGN(STACK_TOP - gap - mmap_rnd());
}

```

其中, gap = rlimit(RLIMIT_STACK) = rlimit(3)會透過ACCESS_ONCE(tsk->signal->rlim[limit].rlim_cur) 取得目前系統的配置數值。

參考如下代碼, 其中, MAX_GAP會等於 (STACK_TOP/6*5),以 ARM 64bits下的4KB Page來說這個值為(TASK_SIZE_64/6*5) = ((UL(1) << VA_BITS)/6*5)=0x6AAAAAAAA9 (in arch/arm64/mm/mmap.c,<http://hala01.com/src/linux/linux-3.16.3/HTML/S/22812.html#L37>),而 MIN_GAP會等於 (SZ_128M + ((STACK_RND_MASK << PAGE_SHIFT) + 1)) 以 ARM 64bits下的4KB Page來說這個值為(0x08000000 + (((0x3fff >> (12 - 12)) << 12) + 1))=0x47FFF001 (in arch/arm64/mm/mmap.c,<http://hala01.com/src/linux/linux-3.16.3/HTML/S/22812.html#L36>)

```

/*
 * Leave enough space between the mmap area and the stack to honour ulimit in
 * the face of randomisation.
 */
#define MIN_GAP (SZ_128M + ((STACK_RND_MASK << PAGE_SHIFT) + 1))
#define MAX_GAP (STACK_TOP/6*5)

```

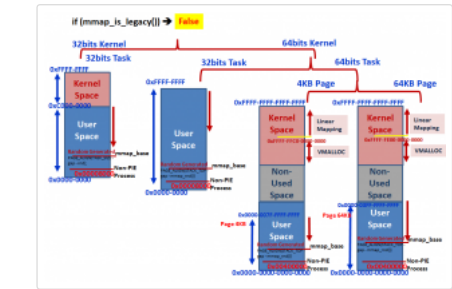
在ARM 64bits搭配4KB Page配置下, 函式mmap_rnd 實作(in arch/arm64/mm/mmap.c, <http://hala01.com/src/linux/linux-3.16.3/HTML/S/22812.html#L58>),其中 rnd = (long)get_random_int() & (STACK_RND_MASK >> 1); 而get_random_int函式會透過MD5 Hash機制回傳一個整數任意值, STACK_RND_MASK會等於 (0x3fff >> (PAGE_SHIFT - 12)) = 0x3fff (in /arch/arm64/include/asm/elf.h, <http://hala01.com/src/linux/linux-3.16.3/HTML/S/22803.html#L157>).

因此, rnd 會等於整數隨機任意值 & 0x3fff,之後再進行rnd << (PAGE_SHIFT + 1)等於 rnd << (12 + 1)= 0x3fff <<13 =0x7ffe000.


```
/*
 * Since get_random_int() returns the same value within a 1 jiffy window, we
 * will almost always get the same randomisation for the stack and mmap
 * region. This will mean the relative distance between stack and mmap will be
 * the same.
 * To avoid this we can shift the randomness by 1 bit.
 */

static unsigned long mmap_rnd(void)
{
    unsigned long rnd = 0;
    if (current->flags & PF_RANDOMIZE)
        rnd = (long)get_random_int() & (STACK_RND_MASK >> 1);
    return rnd << (PAGE_SHIFT + 1);
}
```

函式mmap_base 最後回傳的PAGE_ALIGN(STACK_TOP – gap – mmap_rnd()) = AGE_ALIGN(0x7ffffffffff – gap – 0x7ffe000),對應到實際的產品上,會是如下的Layout



接下來,讓我們從CPRS與Process State來進一步了解ARMv8的系統程式.

從 CPSR 看 PSTATE (Process state)

要了解ARMv7 與ARMv8的System Model最好方式就是從 Current Program Status Register (CPSR) 來看兩者的差異,首先筆者參考 ARMv8 ARM文件 (http://www.cs.utexas.edu/~peterson/arm/DDI0487A_a_armv8_arm_errata.pdf) 與ARMv7 Architecture Reference Manual的文件(<http://liris.cnrs.fr/~mmrissa/lib/exe/fetch.php?media=armv7-a-r-manual.pdf>), 並以此為基礎來加以說明.

根據ARM的文件, ARMv7 , ARMv8 32bits 與 ARMv8 64bits的CPSR相比有如下的差異. 最基礎來看就是ARMv7與ARMv8 32bits的比較,會看到ARMv8 32bits多了Bit 20:1L新的PSR屬性. 而ARMv8 32bits與ARMv8 64bits相比則是少了IT與GE屬性,但卻多了D屬性(取代原本32bits下的E屬性).

ARMv8 AArch64	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	N	Z	C	V	RES0							SS	IL	RES0										D	A	I	F	RES0	0	M[3:0]			
	Condition Flag																						Mask bits					M[4]					
ARMv8 AArch32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	N	Z	C	V	Q	IT[1:0]		J	RES0				IL	GE[3:0]				IT[7:2]					E	A	I	F	T	1	M[3:0]				
	Condition Flag																						Mask bits					M[4]					
ARMv7	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	N	Z	C	V	Q	IT[1:0]		J	Reserved RAZ/SBZP				GE[3:0]				IT[7:2]					E	A	I	F	T	M[4:0]						
	Condition Flag																						Mask bits										



首先, 簡述PSR(Program Status Register) 程式狀態暫存器的意義.這個暫存器主要用來紀錄程序狀態之用,包括反映出目前所處的處理器模式,指令集狀態,以及反應出條件(Cond.)執行指令判斷執行的依據.舉個例子來說,在ARMv7 或ARMv8 32bits下, 當我們從CPSR的4-0bits取出值為b10111就可以知道目前所在的Exception Handler,是發生了Abort,之後再判斷SPSR的4-0bits,若為b10011(SVC Mode)或b10000(User Mode),就可以知道在觸發這個Abort前,處理器是在執行哪一個模式下的程式碼,如果擔心有因為Exception Handle設計不當導致的Abort重入問題,也可以透過CPSR/SPSR前後模式比對,知道是不是Abort重入,可以鎖定潛在的系統問題加以解決. 如下簡述每個欄位的意義

ARMv7			ARMv8 AArch32			ARMv8 AArch64	
功能	說明	位元	功能	說明	位元	功能	說明
	Mode Bits模式位元						Mode Bits† 0b0000 – E 0b0100 – E

Mode[3:0]	b10000(0×0010) -User Mode b10001(0×0011)- FIQ Mode b10010(0×0012)-IRQ Mode b10011(0×0013)-Supervisor Mode b10111(0×0017)-Abort Mode b11011(0×001b)-Undefined Mode b11111(0x001F)-System Mode b10110(0×0016)-Secure Monitor	3-0	Mode[3:0]	the same as ARMv7	3:0	Mode[3:0]	0b0101 – EL0 0b1000 – EL1 0b1001 – EL2 0b1100 – EL3 0b1101 – EL4 M[3:2] holds the current EL. M[1] is unused. M[0] is used to select SP0 and SP1. EL.
Mode[4]	1	4	Mode[4]	= 1 = Exception taken from AArch32	4	Mode[4]	= 0 = Exception taken from EL0
T	Thumb state bit 0=ARM 1=Thumb	5	T	the same as ARMv7	5	RES0	Reserved
F	FIQ Disable 1=禁止 0=允許	6	F	the same as ARMv7	6	F	FIQ Disabled 1=禁止 0=允許
I	IRQ Disable 1=禁止 0=允許	7	I	the same as ARMv7	7	I	IRQ Disabled 1=禁止 0=允許
A	Asynchronous data abort mask bit. 1=禁止 0=允許	8	A	the same as ARMv7	8	A	SError (System Error) are: 1=禁止 0=允許
E	Endianness execution state bit. Controls the load and store endianness for data accesses: 0=Little-endian operation 1=Big-endian operation.	9	E	the same as ARMv7 但由於ARMv8有EL0–EL3,這個Bit可以用來表示目前所在的EL是支援Little or Big-endian	9	D	Process state bit. 0 = 允許 Data Accesses at Software state level. 1 = 禁止 Data Accesses at Software state level. (若除錯Target EL,所以這個Bit,所以Target EL.)
IT[7:2]	If-Then execution state bits for the Thumb IT (If-Then) instruction.	10	c	IT state bits	19:10	RES0	Reserved
		11	b				
		12	a				
		15-13	IT_cond				
GR[3:0]	Greater than or Equal flags	19-16	GR[3:0]	Greater than or Equal flags			
RAZ/SBZP	Reserved	20	IL	Illegal Execution State bit. Shows the value of PSTATE.IL immediately before the exception was taken.	20	IL	Illegal Execution State bit. Shows the value of PSTATE.IL immediately before the exception was taken.
		23:21	RES0	Reserved	21	SS	Software state bit. Shows the value of PSTATE.SS when an exception is taken.
J	Jazelle State Bit	24	J	Jazelle State Bit	27:22	RES0	Reserved
IT[1:0]	If-Then execution state bits for the Thumb IT (If-Then) instruction.	25	e	IT state bits			
		26	d				
Q	Cumulative saturation bit.	27	Q	Sticky Overflow			
V	Overflow condition flag.	28	V	Overflow	28	V	Set to the value of the overflow flag on exception taken to EL1, and flag on exception taken.
C	Carry condition flag	29	C	Carry/Borrow/Extend	29	C	Set to the value of the carry/borrow/extend flag on exception taken to EL1, and flag on exception taken.
							Set to the value of the carry/borrow/extend flag on exception taken to EL1, and flag on exception taken.

Z	Zero condition flag	30	Z	Zero	30	Z	to EL1, and flag on exe
N	Negative condition flag	31	N	Negative/Less than	31	N	Set to the \ to EL1, and flag on exe

EL0	EL1	EL2	EL3	EL4	EL5	EL6	EL7	EL8	EL9	EL10	EL11	EL12	EL13	EL14	EL15	EL16	EL17	EL18	EL19	EL20	EL21	EL22	EL23	EL24	EL25	EL26	EL27	EL28	EL29	EL30	EL31	EL32	EL33	EL34	EL35	EL36	EL37	EL38	EL39	EL40	EL41	EL42	EL43	EL44	EL45	EL46	EL47	EL48	EL49	EL50	EL51	EL52	EL53	EL54	EL55	EL56	EL57	EL58	EL59	EL60	EL61	EL62	EL63	EL64	EL65	EL66	EL67	EL68	EL69	EL70	EL71	EL72	EL73	EL74	EL75	EL76	EL77	EL78	EL79	EL80	EL81	EL82	EL83	EL84	EL85	EL86	EL87	EL88	EL89	EL90	EL91	EL92	EL93	EL94	EL95	EL96	EL97	EL98	EL99	EL100	EL101	EL102	EL103	EL104	EL105	EL106	EL107	EL108	EL109	EL110	EL111	EL112	EL113	EL114	EL115	EL116	EL117	EL118	EL119	EL120	EL121	EL122	EL123	EL124	EL125	EL126	EL127	EL128	EL129	EL130	EL131	EL132	EL133	EL134	EL135	EL136	EL137	EL138	EL139	EL140	EL141	EL142	EL143	EL144	EL145	EL146	EL147	EL148	EL149	EL150	EL151	EL152	EL153	EL154	EL155	EL156	EL157	EL158	EL159	EL160	EL161	EL162	EL163	EL164	EL165	EL166	EL167	EL168	EL169	EL170	EL171	EL172	EL173	EL174	EL175	EL176	EL177	EL178	EL179	EL180	EL181	EL182	EL183	EL184	EL185	EL186	EL187	EL188	EL189	EL190	EL191	EL192	EL193	EL194	EL195	EL196	EL197	EL198	EL199	EL200	EL201	EL202	EL203	EL204	EL205	EL206	EL207	EL208	EL209	EL210	EL211	EL212	EL213	EL214	EL215	EL216	EL217	EL218	EL219	EL220	EL221	EL222	EL223	EL224	EL225	EL226	EL227	EL228	EL229	EL230	EL231	EL232	EL233	EL234	EL235	EL236	EL237	EL238	EL239	EL240	EL241	EL242	EL243	EL244	EL245	EL246	EL247	EL248	EL249	EL250	EL251	EL252	EL253	EL254	EL255	EL256	EL257	EL258	EL259	EL260	EL261	EL262	EL263	EL264	EL265	EL266	EL267	EL268	EL269	EL270	EL271	EL272	EL273	EL274	EL275	EL276	EL277	EL278	EL279	EL280	EL281	EL282	EL283	EL284	EL285	EL286	EL287	EL288	EL289	EL290	EL291	EL292	EL293	EL294	EL295	EL296	EL297	EL298	EL299	EL300	EL301	EL302	EL303	EL304	EL305	EL306	EL307	EL308	EL309	EL310	EL311	EL312	EL313	EL314	EL315	EL316	EL317	EL318	EL319	EL320	EL321	EL322	EL323	EL324	EL325	EL326	EL327	EL328	EL329	EL330	EL331	EL332	EL333	EL334	EL335	EL336	EL337	EL338	EL339	EL340	EL341	EL342	EL343	EL344	EL345	EL346	EL347	EL348	EL349	EL350	EL351	EL352	EL353	EL354	EL355	EL356	EL357	EL358	EL359	EL360	EL361	EL362	EL363	EL364	EL365	EL366	EL367	EL368	EL369	EL370	EL371	EL372	EL373	EL374	EL375	EL376	EL377	EL378	EL379	EL380	EL381	EL382	EL383	EL384	EL385	EL386	EL387	EL388	EL389	EL390	EL391	EL392	EL393	EL394	EL395	EL396	EL397	EL398	EL399	EL400	EL401	EL402	EL403	EL404	EL405	EL406	EL407	EL408	EL409	EL410	EL411	EL412	EL413	EL414	EL415	EL416	EL417	EL418	EL419	EL420	EL421	EL422	EL423	EL424	EL425	EL426	EL427	EL428	EL429	EL430	EL431	EL432	EL433	EL434	EL435	EL436	EL437	EL438	EL439	EL440	EL441	EL442	EL443	EL444	EL445	EL446	EL447	EL448	EL449	EL450	EL451	EL452	EL453	EL454	EL455	EL456	EL457	EL458	EL459	EL460	EL461	EL462	EL463	EL464	EL465	EL466	EL467	EL468	EL469	EL470	EL471	EL472	EL473	EL474	EL475	EL476	EL477	EL478	EL479	EL480	EL481	EL482	EL483	EL484	EL485	EL486	EL487	EL488	EL489	EL490	EL491	EL492	EL493	EL494	EL495	EL496	EL497	EL498	EL499	EL500	EL501	EL502	EL503	EL504	EL505	EL506	EL507	EL508	EL509	EL510	EL511	EL512	EL513	EL514	EL515	EL516	EL517	EL518	EL519	EL520	EL521	EL522	EL523	EL524	EL525	EL526	EL527	EL528	EL529	EL530	EL531	EL532	EL533	EL534	EL535	EL536	EL537	EL538	EL539	EL540	EL541	EL542	EL543	EL544	EL545	EL546	EL547	EL548	EL549	EL550	EL551	EL552	EL553	EL554	EL555	EL556	EL557	EL558	EL559	EL560	EL561	EL562	EL563	EL564	EL565	EL566	EL567	EL568	EL569	EL570	EL571	EL572	EL573	EL574	EL575	EL576	EL577	EL578	EL579	EL580	EL581	EL582	EL583	EL584	EL585	EL586	EL587	EL588	EL589	EL590	EL591	EL592	EL593	EL594	EL595	EL596	EL597	EL598	EL599	EL600	EL601	EL602	EL603	EL604	EL605	EL606	EL607	EL608	EL609	EL610	EL611	EL612	EL613	EL614	EL615	EL616	EL617	EL618	EL619	EL620	EL621	EL622	EL623	EL624	EL625	EL626	EL627	EL628	EL629	EL630	EL631	EL632	EL633	EL634	EL635	EL636	EL637	EL638	EL639	EL640	EL641	EL642	EL643	EL644	EL645	EL646	EL647	EL648	EL649	EL650	EL651	EL652	EL653	EL654	EL655	EL656	EL657	EL658	EL659	EL660	EL661	EL662	EL663	EL664	EL665	EL666	EL667	EL668	EL669	EL670	EL671	EL672	EL673	EL674	EL675	EL676	EL677	EL678	EL679	EL680	EL681	EL682	EL683	EL684	EL685	EL686	EL687	EL688	EL689	EL690	EL691	EL692	EL693	EL694	EL695	EL696	EL697	EL698	EL699	EL700	EL701	EL702	EL703	EL704	EL705	EL706	EL707	EL708	EL709	EL710	EL711	EL712	EL713	EL714	EL715	EL716	EL717	EL718	EL719	EL720	EL721	EL722	EL723	EL724	EL725	EL726	EL727	EL728	EL729	EL730	EL731	EL732	EL733	EL734	EL735	EL736	EL737	EL738	EL739	EL740	EL741	EL742	EL743	EL744	EL745	EL746	EL747	EL748	EL749	EL750	EL751	EL752	EL753	EL754	EL755	EL756	EL757	EL758	EL759	EL760	EL761	EL762	EL763	EL764	EL765	EL766	EL767	EL768	EL769	EL770	EL771	EL772	EL773	EL774	EL775	EL776	EL777	EL778	EL779	EL780	EL781	EL782	EL783	EL784	EL785	EL786	EL787	EL788	EL789	EL790	EL791	EL792	EL793	EL794	EL795	EL796	EL797	EL798	EL799	EL800	EL801	EL802	EL803	EL804	EL805	EL806	EL807	EL808	EL809	EL810	EL811	EL812	EL813	EL814	EL815	EL816	EL817	EL818	EL819	EL820	EL821	EL822	EL823	EL824	EL825	EL826	EL827	EL828	EL829	EL830	EL831	EL832	EL833	EL834	EL835	EL836	EL837	EL838	EL839	EL840	EL841	EL842	EL843	EL844	EL845	EL846	EL847	EL848	EL849	EL850	EL851	EL852	EL853	EL854	EL855	EL856	EL857	EL858	EL859	EL860	EL861	EL862	EL863	EL864	EL865	EL866	EL867	EL868	EL869	EL870	EL871	EL872	EL873	EL874	EL875	EL876	EL877	EL878	EL879	EL880	EL881	EL882	EL883	EL884	EL885	EL886	EL887	EL888	EL889	EL890	EL891	EL892	EL893	EL894	EL895	EL896	EL897	EL898	EL899	EL900	EL901	EL902	EL903	EL904	EL905	EL906	EL907	EL908	EL909	EL910	EL911	EL912	EL913	EL914	EL915	EL916	EL917	EL918	EL919	EL920	EL921	EL922	EL923	EL924	EL925	EL926	EL927	EL928	EL929	EL930	EL931	EL932	EL933	EL934	EL935	EL936	EL937	EL938	EL939	EL940	EL941	EL942	EL943	EL944	EL945	EL946	EL947	EL948	EL949	EL950	EL951	EL952	EL953	EL954	EL955	EL956	EL957	EL958	EL959	EL960	EL961	EL962	EL963	EL964	EL965	EL966	EL967	EL968	EL969	EL970	EL971	EL972	EL973	EL974	EL975	EL976	EL977	EL978	EL979	EL980	EL981	EL982	EL983	EL984	EL985	EL986	EL987	EL988	EL989	EL990	EL991	EL992	EL993	EL994	EL995	EL996	EL997	EL998	EL999	EL1000	EL1001	EL1002	EL1003	EL1004	EL1005	EL1006	EL1007	EL1008	EL1009	EL1010	EL1011	EL1012	EL1013	EL1014	EL1015	EL1016	EL1017	EL1018	EL1019	EL1020	EL1021	EL1022	EL1023	EL1024	EL1025	EL1026	EL1027	EL1028	EL1029	EL1030	EL1031	EL1032	EL1033	EL1034	EL1035	EL1036	EL1037	EL1038	EL1039	EL1040	EL1041	EL1042	EL1043	EL1044	EL1045	EL1046	EL1047	EL1048	EL1049	EL1050	EL1051	EL1052	EL1053	EL1054	EL1055	EL1056	EL1057	EL1058	EL1059	EL1060	EL1061	EL1062	EL1063	EL1064	EL1065	EL1066	EL1067	EL1068	EL1069	EL1070	EL1071	EL1072	EL1073	EL1074	EL1075	EL1076	EL1077	EL1078	EL1079	EL1080	EL1081	EL1082	EL1083	EL1084	EL1085	EL1086	EL1087	EL1088	EL1089	EL1090	EL1091	EL1092	EL1093	EL1094	EL1095	EL1096	EL1097	EL1098	EL1099	EL1100	EL1101	EL1102	EL1103	EL1104	EL1105	EL1106	EL1107	EL1108	EL1109	EL1110	EL1111	EL1112	EL1113	EL1114	EL1115	EL1116	EL1117	EL1118	EL1119	EL1120	EL1121	EL1122	EL1123	EL1124	EL1125	EL1126	EL1127	EL1128	EL1129	EL1130	EL1131	EL1132	EL1133	EL1134	EL1135	EL1136	EL1137	EL1138	EL1139	EL1140	EL1141	EL1142	EL1143	EL1144	EL1145	EL1146	EL1147	EL1148	EL1149	EL1150	EL1151	EL1152	EL1153	EL1154	EL1155	EL1156	EL1157	EL1158	EL1159	EL1160	EL1161	EL1162	EL1163	EL1164	EL1165	EL1166	EL1167	EL1168	EL1169	EL1170	EL1171	EL1172	EL1173	EL1174	EL1175	EL1176	EL1177	EL1178	EL1179	EL1180	EL1181	EL1182	EL1183	EL1184	EL1185	EL1186	EL1187	EL1188	EL1189	EL1190	EL1191	EL1192	EL1193	EL1194	EL1195	EL1196	EL1197	EL1198	EL1199	EL1200	EL1201	EL1202	EL1203	EL1204	EL1205	EL1206	EL1207	EL1208	EL1209	EL1210	EL1211	EL1212	EL1213	EL1214	EL1215	EL1216	EL1217	EL1218	EL1219	EL1220	EL1221	EL1222	EL1223	EL1224	EL1225	EL1226	EL1227	EL1228	EL1229	EL1230	EL1231	EL1232	EL1233	EL1234	EL1235	EL1236	EL1237	EL1238	EL1239	EL1240	EL1241	EL1242	EL1243	EL1244	EL1245	EL1246	EL1247	EL1248	EL1249	EL1250	EL1251	EL1252	EL1253	EL1254	EL1255	EL1256	EL1257	EL1258	EL1259	EL1260	EL1261	EL1262	EL1263	EL1264	EL1265	EL1266	EL1267	EL1268	EL1269	EL1270	EL1271	EL1272	EL1273	EL1274	EL1275	EL1276	EL1277	EL1278	EL1279	EL1280	EL1281	EL1282	EL1283	EL1284	EL1285	EL1286	EL1287	EL1288	EL1289	EL1290	EL1291	EL1292	EL1293	EL1294	EL1295	EL1296	EL1297	EL1298	EL1299	EL1300	EL1301	EL1302	EL1303	EL1304	EL1305	EL1306	EL1307	EL1308	EL1309	EL1310	EL1311	EL1312	EL1313	EL1314	EL1315	EL1316	EL1317	EL1318	EL1319	EL1320	EL1321	EL1322	EL1323	EL1324
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

ESR_ELx是一個AArch64下CPU Mode基礎狀態判定的基礎,依據Exception Level的改動, AArch64有支援ESR_EL1, ESR_EL2, and ESR_EL3用以提供給 EL0->EL1, EL1->EL2與EL2->EL3模式轉換過程的CPU State判斷. 但簡單來說,最直覺的使用就是參與EC與IL欄位,依據目前3.16版本的Linux Kernel Source Code的arch/arm64/include/asm/esr.h (<http://hala01.com/src/linux/linux-3.16.3/HTML/S/22702.html>)來說,會根據EC定義如下的處理器模式組態作為EL0<->EL1的判斷之用.

```
#define ESR_EL1_EC_SHIFT (28)
#define ESR_EL1_EC_UNKNOWN (0x00)
#define ESR_EL1_EC_WFI (0x01)
#define ESR_EL1_EC_CP15_32 (0x03)
#define ESR_EL1_EC_CP15_64 (0x04)
#define ESR_EL1_EC_CP14_MRR (0x05)
#define ESR_EL1_EC_CP14_L5 (0x06)
#define ESR_EL1_EC_FP_ASR0 (0x07)
#define ESR_EL1_EC_CP10_ID (0x08)
#define ESR_EL1_EC_CP14_64 (0x0C)
#define ESR_EL1_EC_LAL_65 (0x0E)
#define ESR_EL1_EC_SVC32 (0x11)
#define ESR_EL1_EC_SVC64 (0x13)
#define ESR_EL1_EC_SPS64 (0x18)
#define ESR_EL1_EC_HBT_EIO (0x20)
#define ESR_EL1_EC_HBT_ELL (0x21)
#define ESR_EL1_EC_PC_ALIGN (0x22)
#define ESR_EL1_EC_DABT_EIO (0x24)
#define ESR_EL1_EC_DABT_ELL (0x25)
#define ESR_EL1_EC_SP_ALIGN (0x26)
#define ESR_EL1_EC_FP_EXC32 (0x28)
#define ESR_EL1_EC_FP_EXC64 (0x2C)
#define ESR_EL1_EC_SERROR (0x2F)
#define ESR_EL1_EC_BREAKPT_EIO (0x30)
#define ESR_EL1_EC_BREAKPT_ELL (0x31)
#define ESR_EL1_EC_SOFTSTP_EIO (0x32)
#define ESR_EL1_EC_SOFTSTP_ELL (0x33)
#define ESR_EL1_EC_WATCHPT_EIO (0x34)
#define ESR_EL1_EC_WATCHPT_ELL (0x35)
#define ESR_EL1_EC_BPTF32 (0x38)
#define ESR_EL1_EC_BPTF64 (0x3C)
```

以Linux Kernel 的EL1 mode handlers來說共支援以下的Exception Class

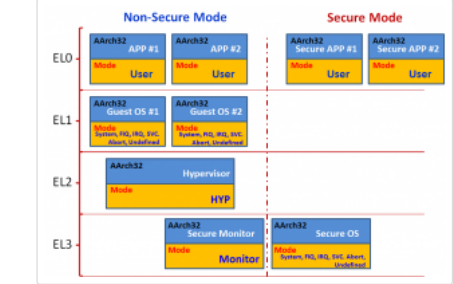
Exception Class in Linux Kernel EL1	Notes
ESR_EL1_EC_UNKNOWN (0x00) (unknown exception in EL1)	Unknown reason (用以包括其它尚未被Exception Class包括到的Exception類型) 例如像是 1, 錯誤的指令集編碼 (像是 Undefined Instruction..etc),不管是在Debug or Non-Debug State 2, Attempted execution of: — An HVC instruction when disabled by HCR_EL2.HCD or SCR_EL3.HCE. — An SMC instruction when disabled by SCR_EL3.SMD. — An HLT instruction when disabled by EDSCR.HDE 3, Attempted execution of an MSR or MRS to SP_EL0 when the value of SPSel.SP is 0. 4, ... (還有許多類型 就不一一敘述.)
ESR_EL1_EC_SYS64 (0x18) (configurable trap)	MSR, MRS, or System instruction execution, that is not reported using EC 0x00, 0x01, or (Exception from MSR, MRS, or System instruction execution in AArch64 state
ESR_EL1_EC_PC_ALIGN (0x22) (pc alignment exception)	Misaligned PC exception
ESR_EL1_EC_DABT_EL1 (0x25) (data abort in EL1)	Data Abort taken without a change in Exception level
ESR_EL1_EC_SP_ALIGN (0x26) (stack alignment exception)	Stack Pointer Alignment exception
ESR_EL1_EC_BREAKPT_EL1(0x31) (debug exception in EL1)	Breakpoint exception taken without a change in Exception level

Exception Class in Linux Kernel EL1	Notes
ESR_EL1_EC_UNKNOWN (0x00)	Unknown reason (用以包括其它尚未被Exception Class包括到的Exception類型) 例如像是 1, 錯誤的指令集編碼 (像是 Undefined Instruction..etc),不管是在Debug or Non-Debug State 2, Attempted execution of: — An HVC instruction when disabled by HCR_EL2.HCD or SCR_EL3.HCE. — An SMC instruction when disabled by SCR_EL3.SMD. — An HLT instruction when disabled by EDSCR.HDE 3, Attempted execution of an MSR or MRS to SP_EL0 when the value of SPSEL.SP is 0. 4, ... (還有許多類型 就不一一敘述.)
ESR_EL1_EC_SYS64 (0x18)	MSR, MRS, or System instruction execution, that is not reported using EC 0x00, 0x01, or (Exception from MSR, MRS, or System instruction execution in AArch64 state
ESR_EL1_EC_PC_ALIGN (0x22)	Misaligned PC exception
ESR_EL1_EC_DABT_EL1 (0x25)	Data Abort taken without a change in Exception level
ESR_EL1_EC_SP_ALIGN (0x26)	Stack Pointer Alignment exception
ESR_EL1_EC_BREAKPT_EL1 (0x31)	Breakpoint exception taken without a change in Exception level

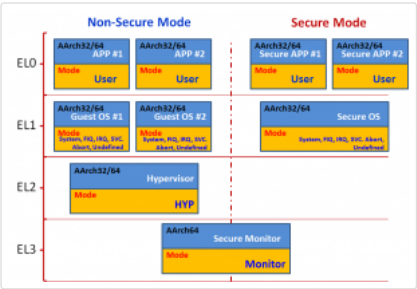
在簡述ARM Process State之後,讓我們進一步說明ARMv8 Security Model.

ARMv8 Security Model

針對Security的需求, ARMv8的系統軟體設計可以提供如下Secure-Mode與Non-Secure Mode的組態,如前面所提到的,若底層EL(Exception Level)為32bits,則上層EL的軟體就只能是32bits.



若底層的EL為64bits,則上層EL就可以依據需求選擇為 32bits或是64bits 的軟體模塊.



結語

在不久的未來,即將有許多配備ARMv8的Android/Linux裝置被普及到市面上,除了更好的執行效率外,64bits更大的記憶體定址能力,也讓原本32bits High/Low-Zone體系被改變,開發者也省去一些力氣在記憶體Zone區的優化上,但對使用者來說,隨著更多搭配64bits優化方案的日趨成熟,像是編譯器, ART執行環境或第三方應用軟體的優化,相信所帶來的使用者體驗也會更佳.

最後,本文謹涉及基礎的ARMv8與部分軟體概念,對有志於進一步探究ARMv8的初學者希望能有所助益.

December 16th, 2014 | Category: [Loda的技術文章分享](#)