QUẢN LÝ PAGE TABLE TRONG LINUX KERNEL

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG

QUẢN LÝ PAGE TABLE TRONG LINUX KERNEL

Nhóm BBU 5G – Phòng phần mềm hệ thống – VHT

Tháng 12 - 2022

Page table là một khái niệm (cấu trúc dữ liệu) liên quan tới quản lý bộ nhớ trong Linux. Quản lý bộ nhớ là một chủ đề rất rộng trong hệ điều hành, do đó bài viết này chỉ đề cập ngắn gọn một số khái niệm liên quan và tập trung mô tả về page table.

1. Nhắc lại về bộ nhớ ảo – virtual memory

Virtual memory là một kỹ thuật (cơ chế) quản lý bộ nhớ phổ biến nhất trong hệ điều hành nói chung và được sử dụng trong hệ điều hành Linux nói riêng. Nhờ có virtual memory mà Linux có thể quản lý bộ nhớ một cách có hiệu quả, đặc biệt trên khía cạnh đa nhiệm của hệ điều hành.

1.1. Định nghĩa, mục đích virtual memory

Virtual memory là cơ chế hệ điều hành sử dụng để tạo ra các không gian địa chỉ ảo (virtual address) – là nơi mà các ứng dụng (process), thậm chí là cả kernel sử dụng để thao tác thay vì truy cập trực tiếp tới không gian địa chỉ vật lý (physical address) của bộ nhớ vật lý.

Trong hệ thống máy tính, bộ nhớ chính (RAM) là nơi lưu trữ các câu lệnh, biến, dữ liệu,.. của các process, dó đó bất kỳ process nào cũng phải thao tác với bộ nhớ chính qua không gian địa chỉ vật lý. Có thể bạn đã biết, physical address được sử dụng để định địa chỉ các cell nhớ trong các chip nhớ, nó tương ứng với các tín hiệu điện được gửi đi từ CPU đến memory bus. Tuy nhiên vì dung lượng của bộ nhớ chính là hữu hạn và duy nhất trong khi số lượng process có thể rất nhiều, việc các process tự quản lý và truy cập trực tiếp đến địa chỉ vật lý sẽ gây ra các vấn đề bất cập như: không gian địa chỉ của các process bị giới hạn và phân mảnh; các process có thể tùy tiện truy cập vùng nhớ của nhau;…

Cơ chế virtual memory của Linux sẽ cung cấp cho mỗi process một không gian địa chỉ ảo riêng biệt, liên tục và có giới hạn không bị phụ thuộc vào dung lượng RAM. Các không gian địa chỉ ảo này sẽ được kernel quản lý và mapping sang địa chỉ vật lý bao gồm địa chỉ trên RAM và địa chỉ của các thiết bị I/O. Ngoài mapping sang địa chỉ vật lý của bộ nhớ chính (RAM), hệ thống có thể tận dụng vùng nhớ trên ổ đĩa (swap space) để lưu trữ dữ liệu của process trong trường hợp cần thiết. Với virtual memory, hệ điều hành đạt được các mục đích sau:

- Các process được sử dụng dải địa chỉ liên tục và không bị giới hạn bởi dung lượng RAM
- Một process bất kỳ không thể truy cập trái phép vào vùng nhớ của một process khác hoặc của hệ điều hành.

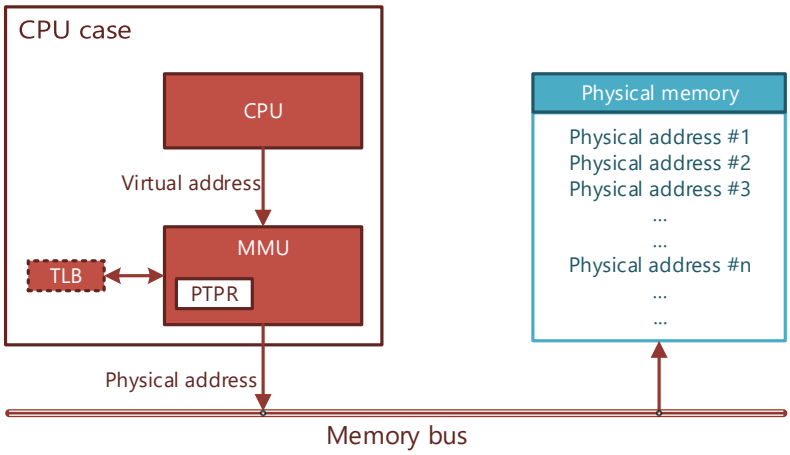
1.2. Triển khai virtual memory và vai trò của page table

Để triển khai được cơ chế virtual memory, cần có sự phối hợp giữa cả phần cứng và phần mềm.

1.2.1. Triển khai virtual memory trên phần cứng

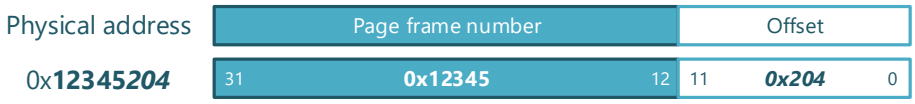
Về mặt phần cứng, trong CPU được thêm vào một khối xử lý là MMU (Memory Management Unit) như mô tả ở Hình 1. Nhiệm vụ của MMU là dịch địa chỉ ảo cho CPU phát ra sang địa chỉ vật lý để truy cập RAM. Ngoài MMU, có một thành phần hardware cũng tham gia vào cơ chế virtual memory là

TLB (Translation Lookaside Buffer). Thành phần này đóng vai trò như một bộ cache cho danh sách mapping. Cơ chế TLB sẽ được mô tả cụ thể trong một tài liệu khác



Hình 1. Các thành phần phần cứng hỗ trợ virtual memory

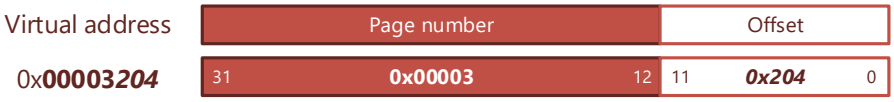
Ngoài ra, physical memory sẽ được tổ chức thành các đơn vị nhớ có kích thước cố định giống nhau để dễ quản lý, gọi là page frame. Kích thước của một page frame là 2^n bytes, n thay đổi tùy theo hệ thống. Mỗi page frame đều page-aligned, có nghĩa rằng địa chỉ bắt đầu của page frame là chia hết 2^n. Như vậy một physical address có thể được chia thành thành phần chính là page frame number và offset như mô tả và ví dụ với một địa chỉ vật lý 32-bit ở Hình 2.



Hình 2. Đánh page frame địa chỉ vật lý

1.2.2. Triển khai virtual memory trên phần mềm

Tương tự như với phần cứng, với phần mềm thì không gian địa chỉ ảo cũng được chia thành các đơn vị nhớ gọi là page, có kích thước bằng với page frame. Kích thước của page và page frame phổ biến nhất là 4KB. Như vậy một virtual address có thể được chia thành thành phần chính là page number và page offset như mô tả và ví dụ với một địa chỉ vật lý 32-bit ở Hình 3.



Hình 3. Đánh page địa chỉ ảo

Danh sách mapping giữa virtual address và physical address như đã đề cập ở phần MMU, được tạo ra và cập nhật bởi hệ điều hành. Tương ứng với mỗi process, hệ điều hành tạo một danh sách mapping như vậy lưu trên RAM, gọi là **page table**. Đây cũng chính là chủ đề được thảo luận chính trong tài liệu này. Cấu trúc của page table sẽ được mô tả chi tiết ở mục 2.1, nhưng có thể đơn giản hóa page table là một bảng mapping giữa page number của địa chỉ ảo và page frame number của địa chỉ vật lý.

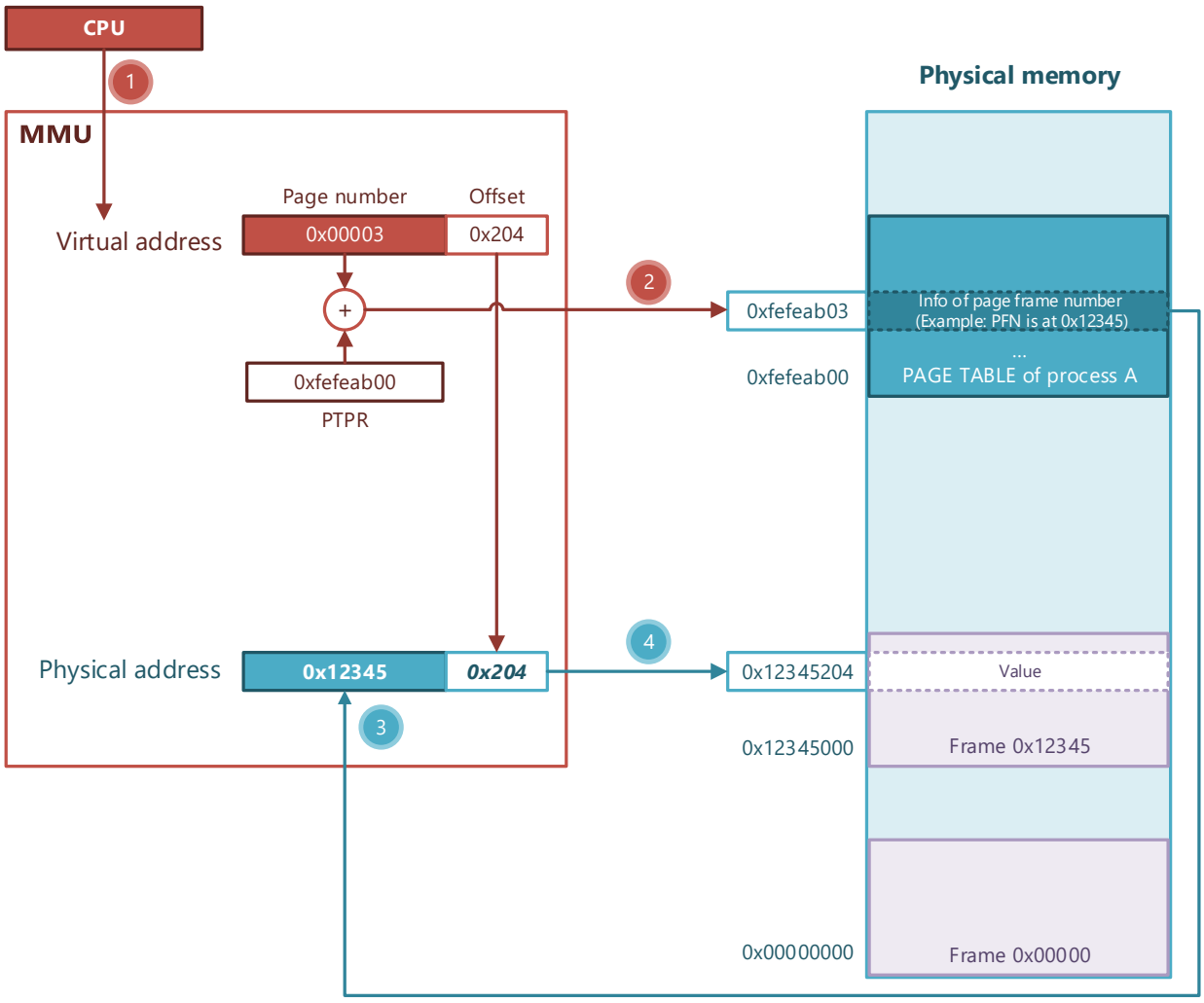
Với input là virtual address, MMU sẽ tra cứu page table để cho ra output là physical address. Tùy thuộc vào kiến trúc, trong bộ MMU sẽ có một thanh ghi có chức năng lưu địa chỉ vật lý của danh sách

mapping, gọi chung là PTPR (Page Table Pointer Register). Với kiến trúc x86 là thanh ghi cr3, với kiến trúc ARM là thanh ghi TTBRx. Khi CPU thực hiện context switch để chạy một process mới, thanh ghi PTPR của MMU sẽ được gán bằng địa chỉ vật lý page table của process mới này để MMU có thể sử dụng.

1.2.3. Phối hợp hoạt động giữa MMU và page table

Luồng phối hợp hoạt động của MMU và page table được mô tả như Hình 4, bao gồm 4 bước:

- 1. CPU phát ra đỉa chỉ virtual address cho MMU (ví dụ 0x00003204).
- 2. Thanh ghi PTPR trong MMU đã được nạp sẵn địa chỉ vật lý của page table của process hiện tại (ví dụ 0xfefeab00). MMU dựa vào thông tin của thanh ghi PTPR và page number (ví dụ 0x00003) để tính ra địa chỉ vật lý của nơi chứa thông tin của page frame number cần tìm.
- 3. Tại vị trí 0xfefeab03, MMU tìm thấy thông tin của page frame number (ví dụ 0x12345), kết hợp với offset của virtual address cho ra địa chỉ physical address mong muốn (ví dụ 0x12345204).
- 4. MMU phát ra địa chỉ physical address (ví dụ 0x12345204) để truy cập tới physical memory.



Hình 4. Luồng cơ chế hoạt động cơ bản của MMU và page table

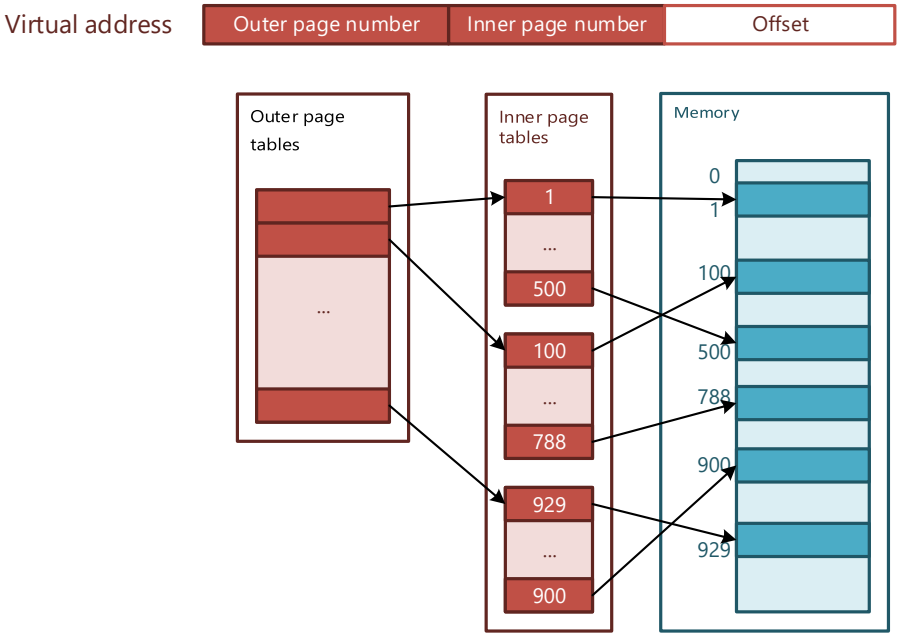
2. Cấu trúc của page table

Trên lý thuyết như đã mô tả ở các mục trước, một địa chỉ ảo được cấu thành từ page number và offset. Còn page table thì hoạt động như một bảng mapping giữa page number và page frame number. Tuy nhiên trên thực tế, cấu trúc page table phức tạp hơn do các nguyên nhân sau:

- Với hệ thống 32 bits và kích thước page 4KB thì mỗi page number có 20 bit để biểu diễn. Để mapping hết dải page number này cần một page table có 2^20 ~ 1048576 entry, trong đó mỗi entry sẽ chiếm 4 bytes. Như vậy sẽ cần quá nhiều bộ nhớ (cụ thể là cần tới 4MB) để lưu trữ page table cho mỗi process. Số lượng entry lớn cũng dẫn tới vấn đề về tốc độ dịch từ địa chỉ ảo sang địa chỉ vật lý. Ngoài ra không phải địa chỉ ảo nào process cũng cần sử dụng tới mà trong thực tế, process chỉ sử dụng các dải địa chỉ nhỏ rời rạc trong không gian địa chỉ ảo của nó. Để giải quyết vấn đề ngày, Linux kernel triển khai **cơ chế multi-level** page table mà trong đó, page table của process được tổ chức thành các page table nhỏ theo cấp bậc. Cơ chế này sẽ được mô tả kỹ ở mục 2.1.
- Ngoài các tập page table cho các process, kernel cũng sở hữu cho riêng nó một tập các page table, gọi là master kernel page table. Tùy theo kiến trúc chip mà cách sử dụng kernel page table sẽ khác nhau. Sự phối hợp giữa kernel page table và process page table sẽ được mô tả ở mục 2.2.

2.1. Cơ chế multi-level

Như đã mô tả ở các phần trước, một virtual address sẽ được tách thành hai thành phần cơ bản là page number và offset. Với kiến trúc 32 bits thì page number thường có độ dài 20 bits trong khi offset có độ dài 12 bits. Với cơ chế multi-level ta sẽ chia dải page number thành các dải page con nhỏ hơn. Ta ví dụ trong trường hợp chia dải page number thành 2 dải page là outer page và inner page. Khi đó thay vì tạo một page table duy nhất để chứa tất cả các entry trỏ thẳng tới bộ nhớ vật lý thì hệ thống sẽ tạo ra một outer page table, trong đó mỗi entry của outer page table sẽ trỏ tới một inner page table. Entry của inner page table mới tiếp tục trỏ đến địa chỉ vật lý. Hình 5 mô tả sự phân cấp của hệ page table 2 cấp.



Hình 5. Mô hình page table hai cấp

Với việc phân cấp page table, giả sử trong trường hợp tất cả các địa chỉ ảo trong không gian đỉa chỉ ảo của proccess đều cần sử dụng thì bộ nhớ dùng để lưu trữ multi-level page table sẽ nhiều hơn one-level page

table. Tuy nhiên như đã nói, do process chỉ sử dụng một số các dải địa chỉ nhỏ rời rạc trong không gian địa chỉ ảo của nó. Với one-level page table, sẽ không có cách nào load/unload một dải địa chỉ ảo nào đó trong khi process đang chạy do page table này là một dải địa chỉ liên tục. Tuy nhiên với multi-level page table, các inner page table là các table rời rạc và linh động, hệ điều hành chỉ cần load/unload những inner page table cần thiết (là những page table mô tả các dải địa chỉ cần dùng) rồi mapping/unmapping tới outer page table, qua đó giảm thiểu được kích thước lưu trữ page table của một process một cách đáng kể.

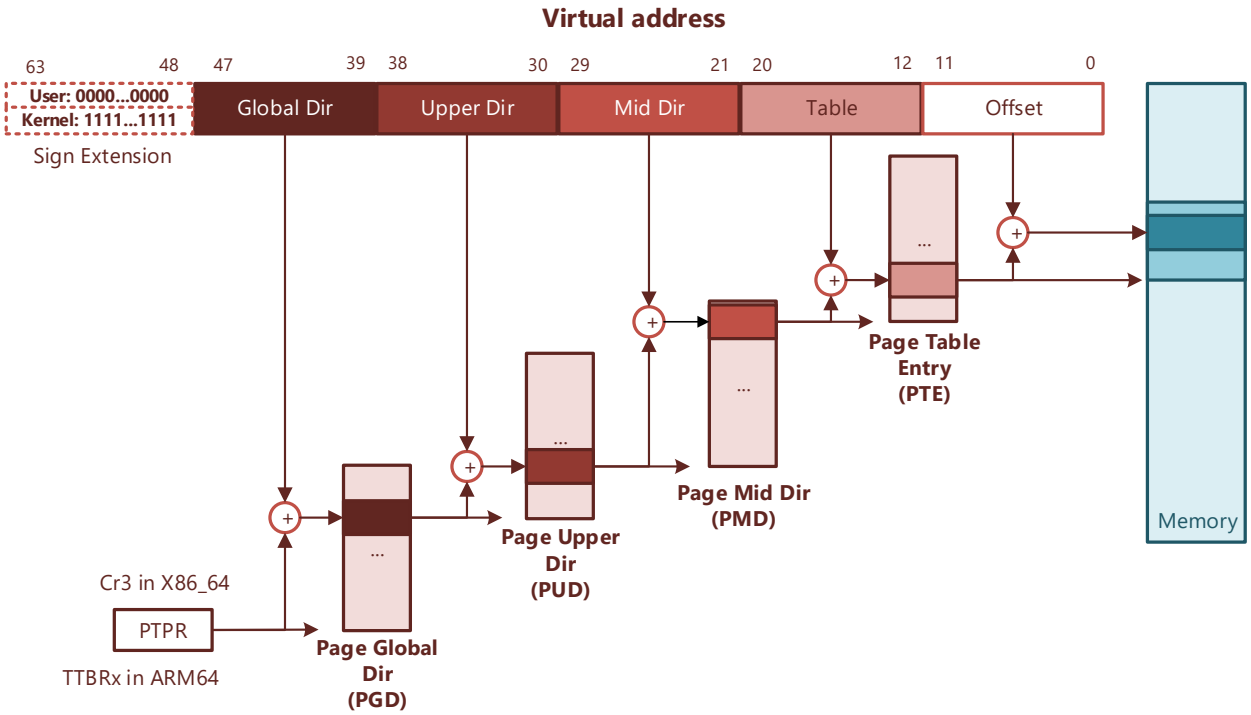
Số lượng level của page table trong hệ thống sẽ phụ thuộc vào loại kiến trúc (X86_32, X86_64, ARM32, ARM64, …). Kiến trúc với nhiều bit hơn thường sẽ sử dụng page table có nhiều level hơn. Hiện tại Linux kernel đã support tới 5-level page table cho kiến trúc X86_64. Bảng 1 là danh sách số lượng level của page table ở một số các kiến trúc phổ biến trong trường hợp kích thước page là 4KB.

Kiến trúc	Số bit đánh địa chỉ virtual	Level của page table
X86_32	32	2
	36 (MMU support feature PAE)	3
X86_64	48	4
	56 (MMU support LA57)	5 (hỗ trợ từ kernel 4.14 trở lên)
ARM32	32	2
	32 (MMU support feature LPAE)	3
ARM64	39	3
	48	4

Bảng 1. Danh sách số lượng level của page table ở một số kiến trúc phổ biến

Do có rất nhiều level của page table mà các kiến trúc cũng như kernel hỗ trợ, bài viết này sẽ lấy trường hợp 4-level page table làm ví dụ. Đối với số lượng các level khác cơ chế cũng sẽ tương tự.

Các chip có kiến trúc 64 bits (X86_64 và ARM64) hiện nay thường chỉ hỗ trợ đánh 48 bits địa chỉ cả physical lẫn virtual, ngoại trừ một số các chip mới (X86_64 hỗ trợ LA57 đánh địa chỉ 57 bits, ARM64 hỗ trợ LPA&LVA đánh địa chỉ 52 bits). Với việc đánh địa chỉ phổ biến nhất 48 bits, địa chỉ virtual sẽ được cấu trúc như Hình 6, bao gồm 4 level là PGD, PUD, PMD, PTE. Mỗi level có độ rộng 9 bits, tức là có thể chưa tối đa 512 entry.



Hình 6. Cấu trúc 4-level page table

- **Page Global Directory (PGD):** Level đầu tiên của page table. Địa chỉ vật lý của PGD được lưu trữ ở thanh ghi PPTR (ở X86 là thanh ghi cr3, ở ARM là TTBRx). Địa chỉ entry PGD của virtual address được xác định bằng PTPR + Global Dir Index. Mỗi entry của PGD chứa thông tin địa chỉ vật lý của một bảng ở level tiếp theo (PUD).
- **Page Upper Directory (PUD):** Level thứ 2 của page table. Địa chỉ entry PUD của virtual address được xác định bằng PTPR + Upper Dir Index. Mỗi entry của PUD chứa thông tin địa chỉ vật lý của một bảng ở level tiếp theo (PMD).
- **Page Mid-level Directory (PMD):** Level thứ 3 của page table. Địa chỉ entry PMD của virtual address được xác định bằng PTPR + Mid Dir Index. Mỗi entry của PMD chứa thông tin địa chỉ vật lý của một bảng ở level tiếp theo (PTE).
- **Page Table Entry (PTE)**: Level thứ cuối của page table. Địa chỉ entry PTE của virtual address được xác định bằng PTPR + Table Index. Mỗi entry của PTE chứa thông tin địa chỉ vật lý của **page frame** nằm trên bộ nhớ. Kết hợp giữa địa chỉ vật lý của page frame và trường offset trong virtual address sẽ được physical address mong muốn.

Ngoài 4 level trong virtual address, hệ điều hành cũng tận dụng nốt 64 – 48 = 16 bits còn lại (gọi là sign extension) phục vụ mục đích phân biệt kernel virtual address và user virtual address. Khi 16 bits đầu tiên của một địa chỉ ảo được đánh toàn 1 thì nó thuộc kernel space, nếu được đánh toàn 0 thì nó thuộc user space. Câu chuyện của sign extension sẽ liên quan đến cả kernel page table và process page table mà sẽ được mô tả ở mục 2.2.

Đối với kiến trúc X86_64 theo cơ chế 4-level paging, không gian địa chỉ ảo bị giới hạn 2^{48} = 256 TiB và không gian địa chỉ vật lý bị giới hạn 2^{46} = 64TiB. Mặc dù 64TiB là một con số lớn nhưng đến thời điểm hiện tại, một số hệ thống đã bắt đầu có nhu cầu vượt qua con số này. Do đó một số chipset x86_64 mới (lần đầu tiên là thế hệ Ice Lake) và kể cả Linux kernel (từ kernel 4.11) đã được update để hỗ trợ không gian địa

chỉ ảo lên tới 2^{57} = 128 PiB, không gian địa chỉ vật lý lên tới 2^{52} = 4 PiB và hệ thống 5-level paging. Bảng 2 mô tả sự khác nhau giữa 4-level và 5-level paging.

Range	4-level paging	5-level paging (Ice Lake)
Virtual Address	256 TiB = 2^{48}	128 PiB = 2^{57}
Userspace	128 TiB	64 PiB
Physical Address	64 TiB = 2^{46}	4 PiB = 2^{52}
Merged to upstream	v2.6.10	v4.11-rc2

Bảng 2. So sánh 4 và 5 level paging

Từ Linux kernel 4.11 hỗ trợ 5-level page table thông qua hai cờ là CONFIG_PGTABLE_LEVELS=5 và CONFIG_X86_5LEVEL=y.

```
$ cat /boot/config-$(uname -r) | grep -E 'X86_5LEVEL|PGTABLE_LEVELS'
CONFIG_PGTABLE_LEVELS=5
CONFIG_X86_5LEVEL=y
```

Kiểm tra hệ thống có thực sự chạy 5-level paging hay không thông qua cờ ***la57*** của lscpu.

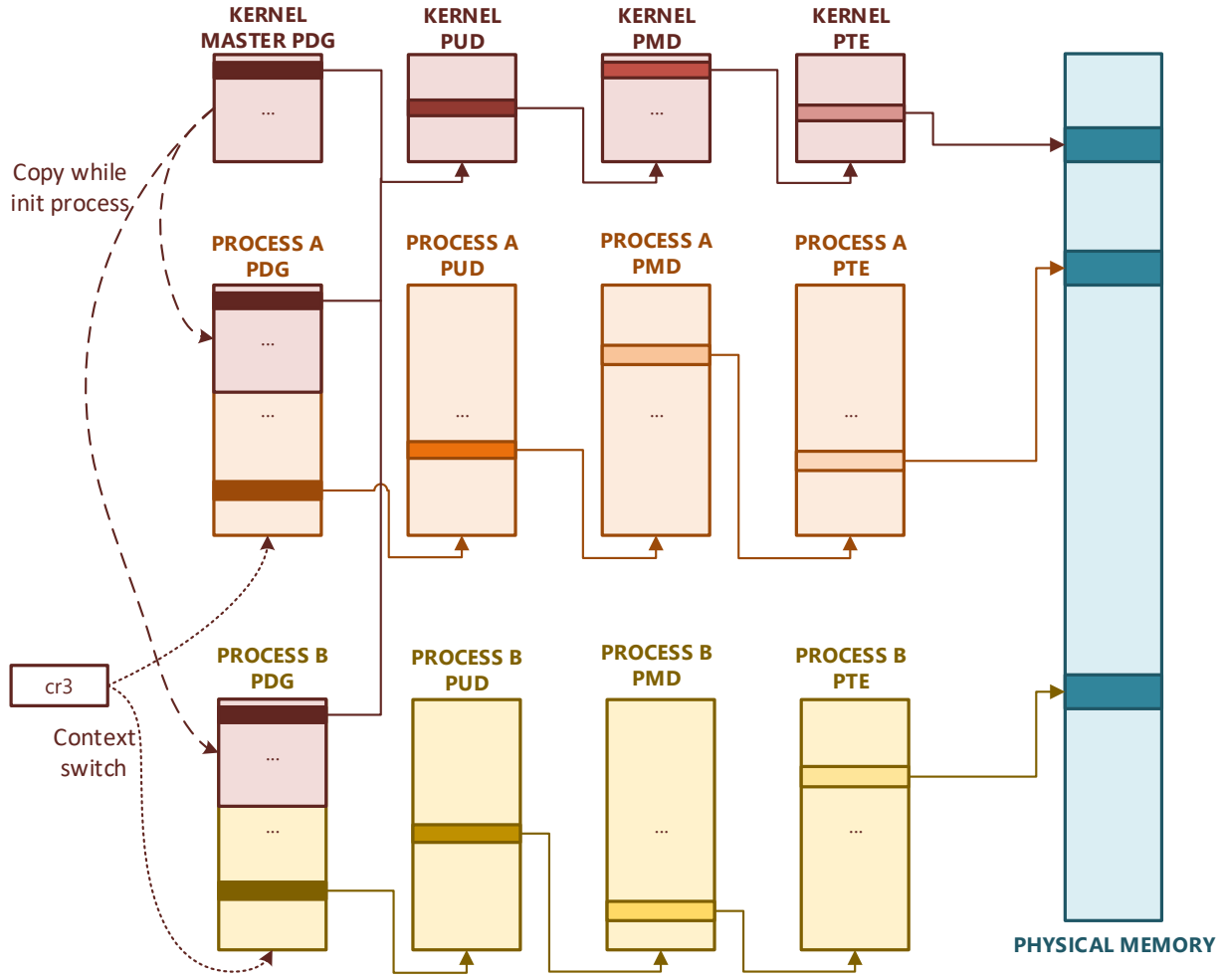
```
$ lscpu | grep -i la57
Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx
fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon … avx512_vnni
avx512_bitalg tme avx512_vpopcntdq la57 rdpid md_clear pconfig flush_l1d arch_capabilities
```

2.2. Kernel page table và process page table

Nhắc lại một chút, như đã biết thì mỗi process chạy trong hệ điều hành Linux đều sẽ được kernel cấp phát một multi-level page table để mô tả ánh xạ dải địa chỉ ảo của process đó với địa chỉ vật lý của bộ nhớ. Ngoài các process thì kernel cũng cần có page table vì bản thân kernel cũng phải thao tác qua địa chỉ ảo một khi MMU đã được kích hoạt. Page table của kernel cũng được cấu trúc thành multi-level và level đầu tiên của page table này còn được gọi là Master Kernel Page Global Directory hoặc Swapper Page Global Directory. Kernel sẽ thực hiện khởi tạo page table của chính nó ở giai đoạn boot hệ thống. Quy trình khởi tạo kernel page table sẽ được mô tả cụ thể ở mục 3.1.1.

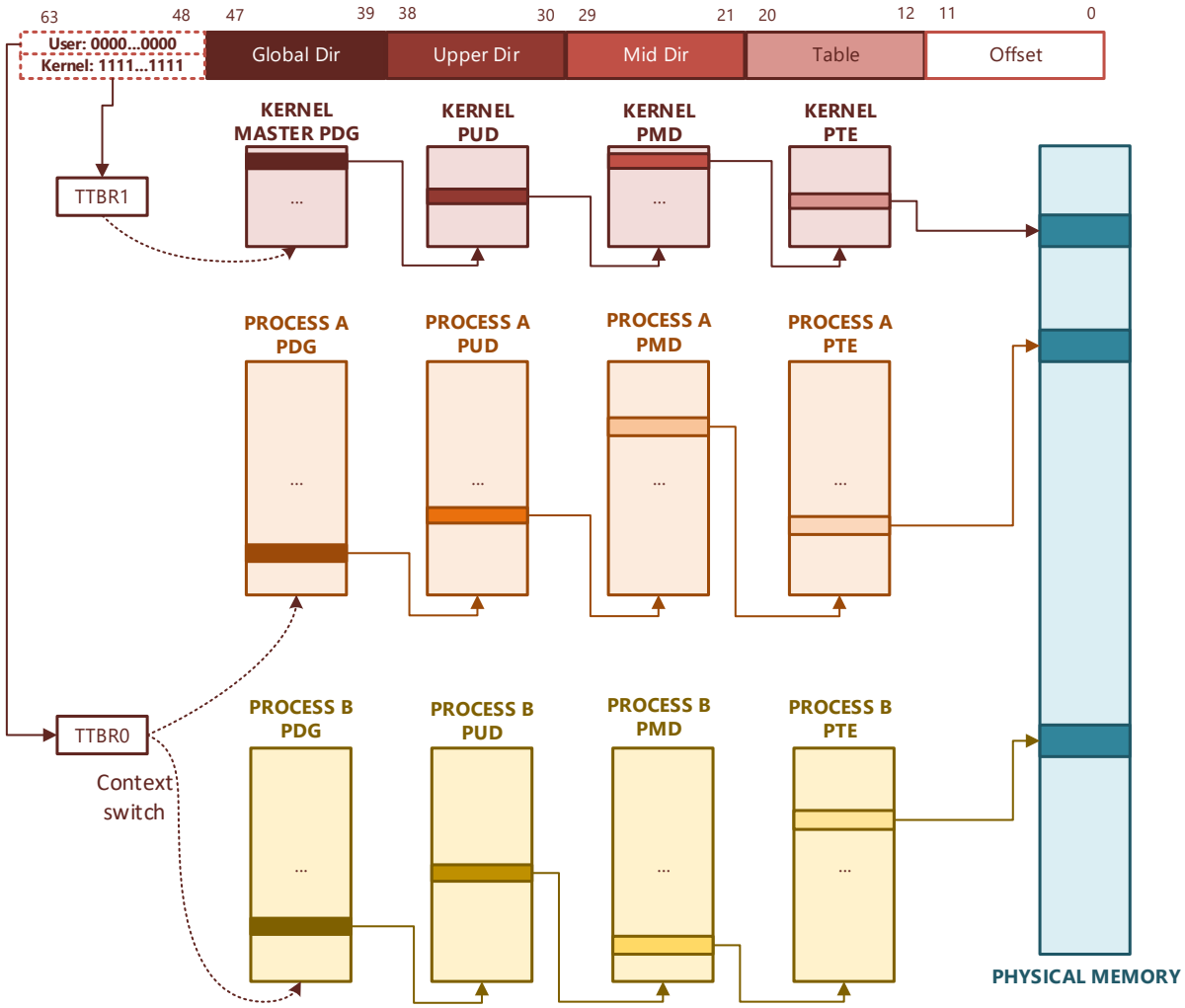
Hệ điều hành sẽ phát sinh nhu cầu phải truy cập cả kernel virtual address lẫn process virtual address ở một giai đoạn nào đó. Ví dụ như một user process switch vào kernel mode, ở kernel mode cần truy cập được địa chỉ ảo của kernel lẫn user. Khi truy cập kernel address sẽ cần truy xuất tới kernel page table, trong khi đó truy cập user address sẽ cần truy xuất tới process page table. Khi đó sẽ cần sự phối hợp giữa MMU, kernel page table và process page table để có thể dịch được một địa chỉ ảo sang địa chỉ vật lý. Sự phối hợp này sẽ là khác nhau giữa các kiến trúc, cụ thể tùy thuộc vào yếu tố "số lượng thanh ghi dùng để lưu trữ địa chỉ page table của MMU":

- Kiến trúc X86_64: MMU của X86_64 chỉ hỗ trợ 1 thanh ghi cr3 lưu địa chỉ của process page table đang chạy. Khi process đang ở user space, sẽ không xảy ra vấn đề gì với process page tale. Tuy nhiên khi process vào kernel space, sẽ không có cách nào truy cập được dải địa chỉ của kernel space nếu không có thông tin gì về kernel page table. Do đó, kernel sẽ sử dụng cơ chế process page table "mượn" nội dung của kernel page table như mô tả qua Hình 7. Trong quá trình khởi tạo page table cho process, hàm pdg_ctor() (https://elixir.bootlin.com/linux/v4.19.267/source/arch/x86/mm/pgtable.c#L132) được gọi có nhiệm vụ clone copy lại nội dung của kernel page table vào process page table.



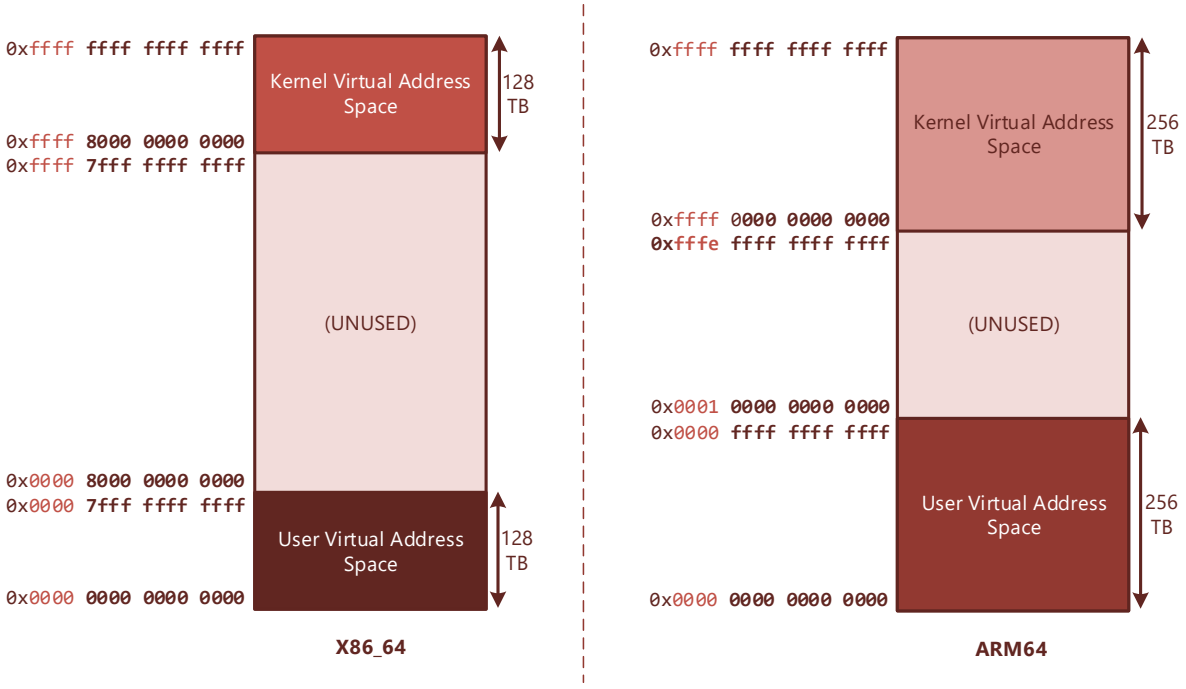
Hình 7. Phối hợp giữa kernel page table và process page table trong kiến trúc X86_64

- Kiến trúc ARM64: MMU của ARM64 hỗ trợ tới 2 thanh ghi lưu địa chỉ của page table là TTBR0 và TTBR1. TTBR1 có nhiệm vụ lưu địa chỉ vật lý của kernel page table, trong khi đó TTBR0 lưu địa chỉ vật lý của process page table hiện hành. Ngoài ra, với sự hỗ trợ của sign extention như đã đề cập ở mục 2.1, MMU có thể phân biệt được một địa chỉ ảo là của kernel space hay user space, do đó có thể chọn truy xuất page table từ thanh ghi TTBR0 hay TTBR1 tương ứng. Nhờ đó mà process ở kernel mode vẫn có thể truy cập đến cả địa chỉ ảo của kernel space lẫn user space. Cơ chế hoạt động này được mô tả như trên Hình 9.



Hình 8. Phối hợp giữa kernel page table và process page table trong kiến trúc ARM64

Cũng có thể thấy, sự khác biết về kiến trúc MMU giữa X86_64 và ARM64 đã dẫn tới hệ quả là, không gian địa chỉ ảo của kernel lẫn user trên X86_64 nhỏ hơn so với với ARM64 khi cả hai đều đánh địa chỉ 48 bits như mô tả trên Hình 9. Điều này là do với X86_64, kernel và user entry đều cùng nằm trên một process page table, khi đó 2 dải địa chỉ 46 bits này sẽ phải tách bạch với nhau, cụ thể với user space là 0x0000 0000 0000 đến 0x7fff ffff ffff và với kernel space là 0x8000 0000 0000 đến 0xffff ffff ffff. Còn với ARM64, do tận dụng được lợi thế của thanh ghi TTBR1 để lưu địa chỉ kernel page table và sign extension, dải địa chỉ 48 bits của user space và kernel space có thể trùng nhau, chỉ cần khác nhau 16 bits sign extension.



Hình 9. Dải địa chỉ ảo của kernel và user trong hệ thống X86_64 và ARM64

5. Tài liệu tham khảo

Billimoria, K. N. (2021). *Linux Kernel Programming.* Packt Publishing.

Daniel P. Bovet, Marco Cesati. (2006). *Understanding the Linux Kernel, Third Edition.* O'Reilly Media.

Docs, L. K. (n.d.). *5-level paging.* Retrieved from docs.kernel.org: https://docs.kernel.org/x86/x86_64/5level-paging.html

Gorman, M. (2004). *Understanding the Linux Virtual Memory Manager.* Bruce Perens' Open source series.

Madieu, J. (2022). *Linux Device Driver Development Second Edition.* Packt Publishing.

Press, L. (n.d.). *Introduction to 5-Level Paging in 3rd Gen Intel Xeon Scalable Processors with Linux.* Retrieved from lenovopress.lenovo.com: https://lenovopress.lenovo.com/lp1468.pdf