# ZONE Type

2016-06-13 (http://jake.dothome.co.kr/zone-types/) Moon Young-il (http://jake.dothome.co.kr/author/admin/) Linux Kernel (http://jake.dothome.co.kr/category/linux/)

<kernel v5.0>

In Linux, the physical memory address area is managed by dividing it into ZONES. Each ZONE is as follows:

- ZONE_DMA
- ZONE_DMA32
- ZONE_NORMAL
- ZONE_HIGHMEM
- ZONE_MOVABLE
- ZONE_DEVICE

## ZONE_DMA & ZONE_DMA32

In a system that uses devices that directly access memory using DMA (Direct Memory Access), if the DMA-enabled device covers all of the address buses used by the system, memory can be allocated and used using the default ZONE_NORMAL without having to configure a separate ZONE_DMA. However, if the DMA-enabled device is smaller than the physical memory address supported by the system, you must configure a separate ZONE_DMA to make the memory allocation for the DMA device less than the restricted address so that the DMA device can access it. In the case of x86 and ARM, the following conditions are used to determine whether or not to use CONFIG_ZONE_DMA and CONFIG_ZONE_DMA32.

- DMA-enabled devices do not need to set up CONFIG_DMA if the device's address bus is designed to be accessible up to the system's maximum physical memory access address. However, a device that uses DMA can set a CONFIG_DMA if the device's address bus is less accessible than the system's maximum physical memory access address.
  - 32bit x86:
    - Using CONFIG_ZONE_DMA
  - 64bit x86:
    - Using CONFIG_ZONE_DMA and CONFIG_ZONE_DMA32
  - arm:
    - Don't use ZONE_DMA as default. However, in the case of a specific system, a ZONE_DMA is used and the size is specified to operate.
  - arm64:
    - By default, use ZONE_DMA and ZONE_DMA32.

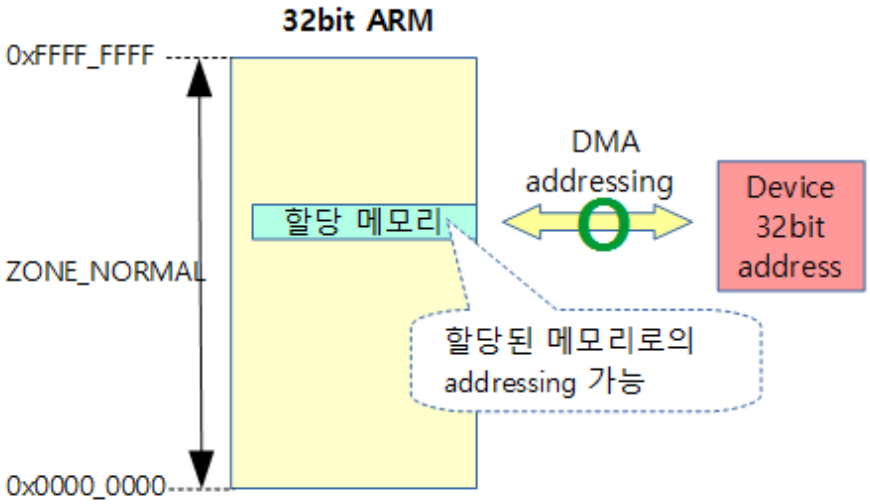
        - Note: mm: generalize ZONE_[DMA|DMA32] (https://github.com/torvalds/linux/commit/63703f37aa09e2c12c0ff25afbf5c460b21bfe4c#diff-6a0166b1d8bbb576287048e4de42eb7e8a1f118e357f407d3bdf7da9fc26d94d) (2021, v5.14-rc1)
    - Previously, ZONE_DMA was used in kernel v3.15-rc1, but from v4.16-rc1 it was changed back to ZONE_DMA32.
        - Note: arm64: Replace ZONE_DMA32 with ZONE_DMA (https://github.com/torvalds/linux/commit/19e7640d1f2302c20df2733e3e3df49acb17189e)
        - Note: arm64: replace ZONE_DMA with ZONE_DMA32 (https://github.com/torvalds/linux/commit/ad67f5a6545f7fda8ec11d7a81e325a398e1a90f)
- DMA Controllers and Channels Reference
    - DMA (Cycle Stealing) (http://www.jidum.com/jidums/view.do?jidumId=470) | Ojidum
    - DMA (Direct Memory Access) – CPU Secretly Zero (http://recipes.egloos.com/5152867) Car | Embedded Recipes
    - STM32F103 ADC – Read with Multi Channel DMA (http://blog.naver.com/PostView.nhn?blogId=gauya&logNo=220232257331&parentCategoryNo=&categoryNo=22&viewDate=&isShowPopularPosts=true&from=search) | frost

### DMA Size Setting

- 32bit x86
    - It is fixed at 16MB and is automatically configured.
    - It is used to make devices running on the ISA bus used in the AT (24) system using the 80286-bit address bus backward compatible
- 64bit x86
    - It is fixed at 4GB and is automatically configured.
    - Existing buses with only 32-bit addresses (32bit EISA, PCI, 24-bit ISA, ...) It is used to make devices compatible with the
- 32bit ARM
    - If you are using CONFIG_DMA, the DMA size is configured to be flexible.
    - If the embedded system circuit configuration uses only a restricted address bus pin on the DMA device device, limit the area to the addresses that can be specified.
        - e.g. if the DMA device device on arm32 supports 28 bit address bus
            - The DMA size should be set to 256MB.
- 64bit ARM
    - Based on the physical memory starting address, it is automatically configured with a physical memory address up to 4GB.

The following figure shows when a device that uses DMA should set its CONFIG_DMA.

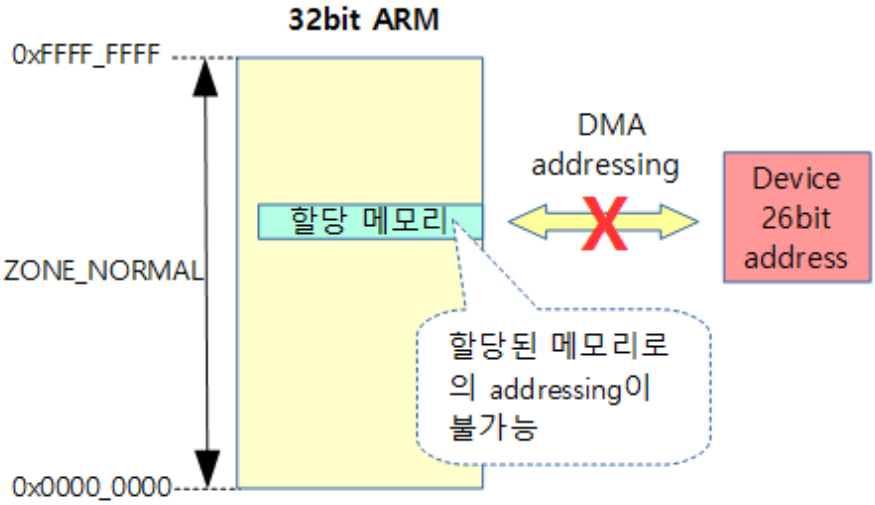
**a) ZONE_DMA 를 사용하지 않아도 되는 경우**




(http://jake.dothome.co.kr/wp-content/uploads/2016/06/zone-dma-1.png)

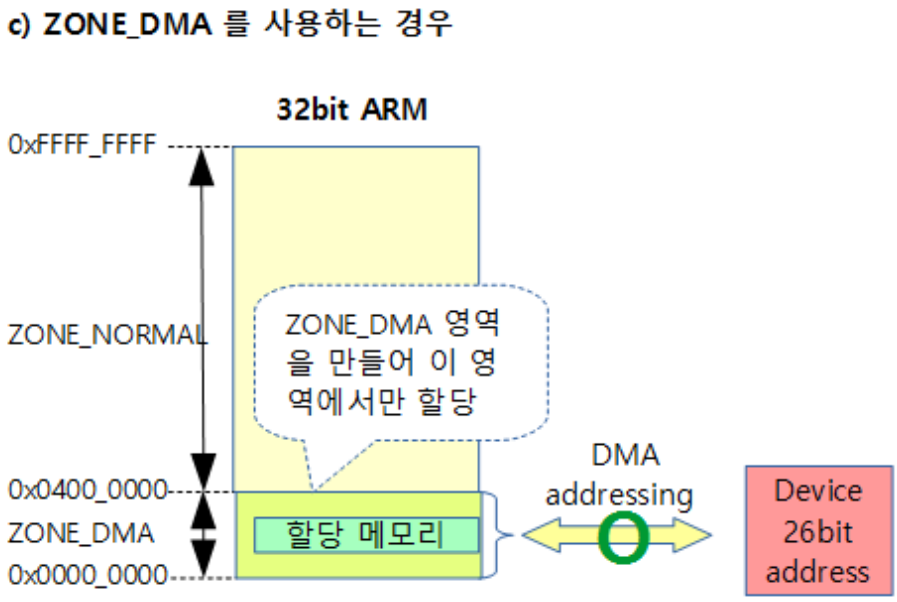
- DMA-enabled devices support 32-bit addresses, so you don't need to set up a special ZONE_DMA.

**b) ZONE_DMA를 사용하지 않은 경우**




(http://jake.dothome.co.kr/wp-content/uploads/2016/06/zone-dma-2.png)

- Device devices that use DMA only support limited 24-bit addresses, and memory allocated to 32-bit addresses is not accessible by the device device.




(http://jake.dothome.co.kr/wp-content/uploads/2016/06/zone-dma-3.png)

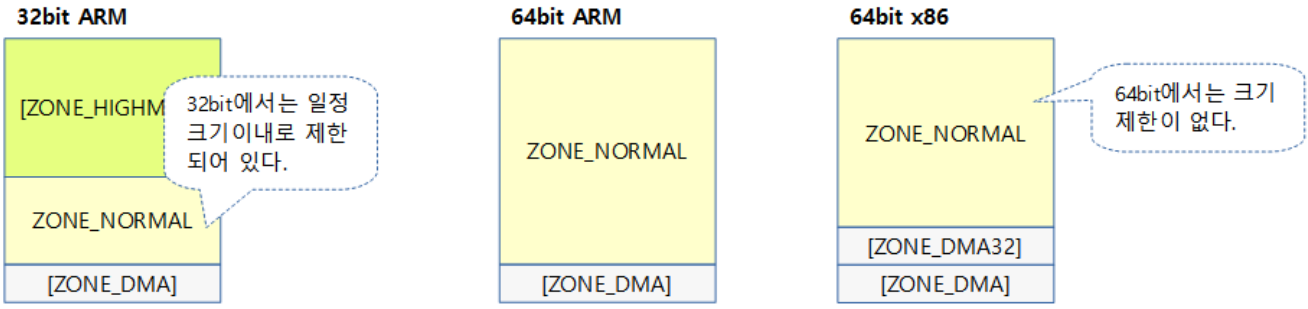
- Because DMA-enabled device devices only support limited 24-bit addresses, they receive and use the relevant memory allocation in ZONE_DMA zone.
  - ZONE_DMA size is set to 64MB (2^26).

## ZONE_NORMAL

- This is the extent of the physical memory supported by the system that can be used by pre-mapped 1:1 to virtual addresses in the architecture. Therefore, when allocating kernel memory, it is the fastest accessible memory area along with the ZONE_DMA area.
  - On a 32-bit system, the 4G space is divided into user space and kernel space, so the user space is left empty and a part of the kernel space is used to deploy physical memory, which is ZONE_NORMAL.
    - 32bit arm
      - Kernel size depends on configuration.
        - CONFIG_VMSPLIT_3G
          - Maximum zone size for normal zone = 760 MB
          - 3G user space, 1G kernel space
          - It is used by most PCs, Linux and embedded kernels.
        - CONFIG_VMSPLIT_2G
          - Maximum zone size for normal zone = 1 GB + 760 MB
          - 2G user space, 2G kernel space
          - This is the setting used by the Raspberry Pi 2 and some embedded systems.
        - CONFIG_VMSPLIT_1G
          - Maximum zone size for normal zone = 2 GB + 760 MB
          - 1G user space, 3G kernel space
          - It's an extreme setting, so it's rarely used.
    - 32bit x86
      - Maximum zone size for normal zone = 896 MB

- Of course, ZONE_DMA and ZONE_DMA32 are also mapped 1:1 to virtual addresses.
- On a 32-bit system, the normal zone is limited in size and is therefore the most expensive area used for kernel allocation.
- In the ARMv64 architecture, which is a 8-bit ARM, the maximum kernel address space is 256T, of which half (128T) can be used as a physical memory placement. In other words, it's a huge normal zone, and since this space is so large, there's no reason to create a separate highmem zone.
  - Starting with the ARMv8.2 architecture, it supports up to 52-bit physical addresses, supporting 4 Peta spaces.

The following figure shows that the ZONE_NORMAL area on a 32-bit system is limited.




(http://jake.dothome.co.kr/wp-content/uploads/2016/06/zone-2a.png)

## ZONE_HIGHMEM

The areas of memory that the ZONE_NORMAL cannot handle are all made up of ZONE_HIGHMEM.

- On 32-bit systems, only partial 1:1 mapping is possible, so more than ZONE_NORMAL of memory will use this area.
- On 64-bit systems, all physical memory can be mapped 1:1, so ZONE_HIGHMEM is not used.
- When allocating user memory, the memory of the highmem area is consumed first, and the normal zone area is used only when it is consumed.

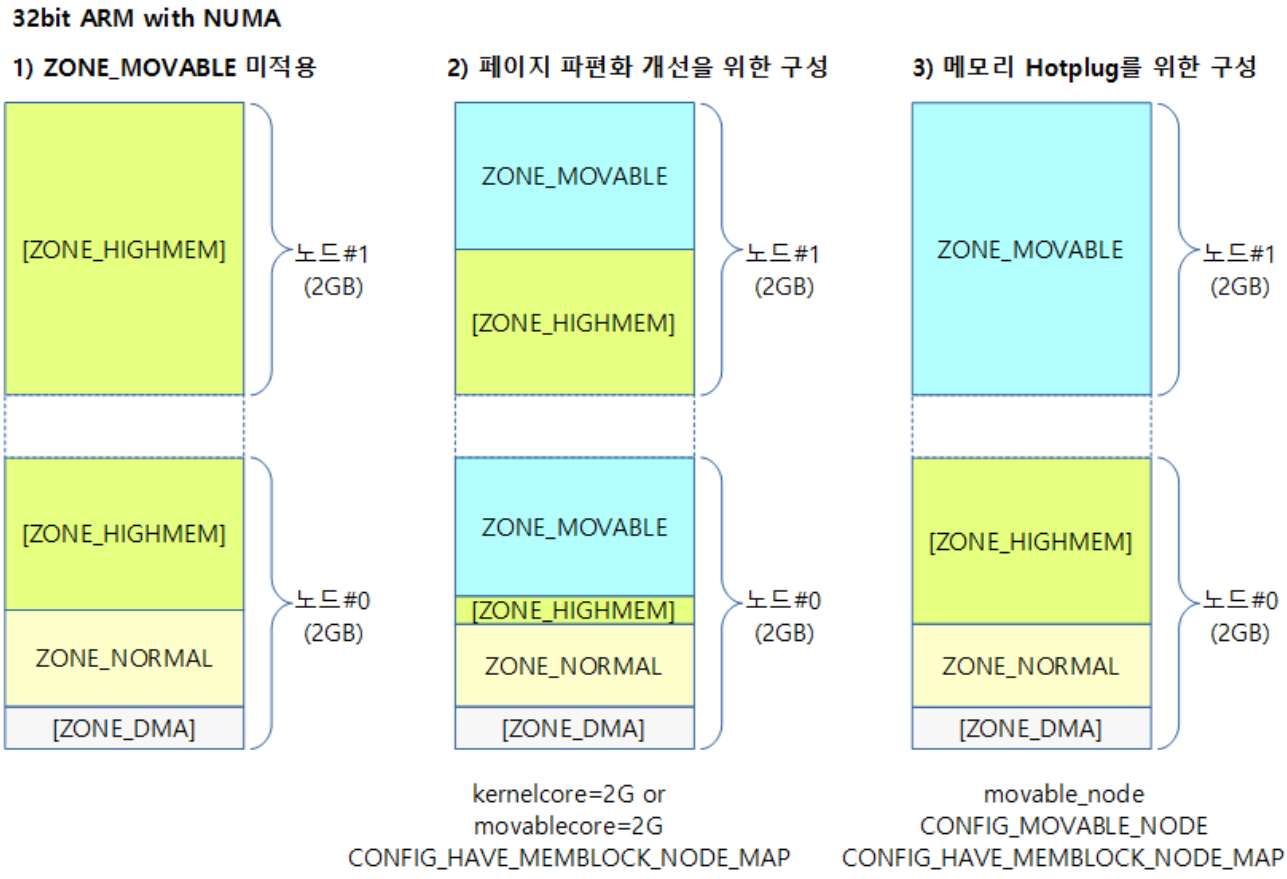
## ZONE_MOVABLE

This area is used for two purposes:

- Anti-fragmentation
  - For the last zone, the page allocator, which is implemented as a buddy system by allocating a portion of each node, uses this area exclusively to prevent memory fragmentation.
  - Even if this area is not configured, the buddy system classifies it as a migration type to prevent fragmentation as much as possible, and if this area is specified, the entire area will be composed of movable pages, which can show greater efficiency in preventing fragmentation.

    - The last zone is the one that exists at the top of the system and is the most different depending on the system, and it is listed in order from highest to lowest as shown below.
        - ZONE_HIGHMEM -> ZONE_NORMAL -> ZONE_DMA (or ZONE_DMA32)
        - On a 64-bit system without ZONE_HIGHMEM, it can be a ZONE_NORMAL, and without it, it can be a ZONE_DMA (or ZONE_DMA32).
- Hot-plug memory support
    - In order to prevent memory fragmentation of the Buddy system and to support the recently implemented "memory hotplugs", the memory hotplugs are configured to be used as ZONE_MOVABLE for all nodes.

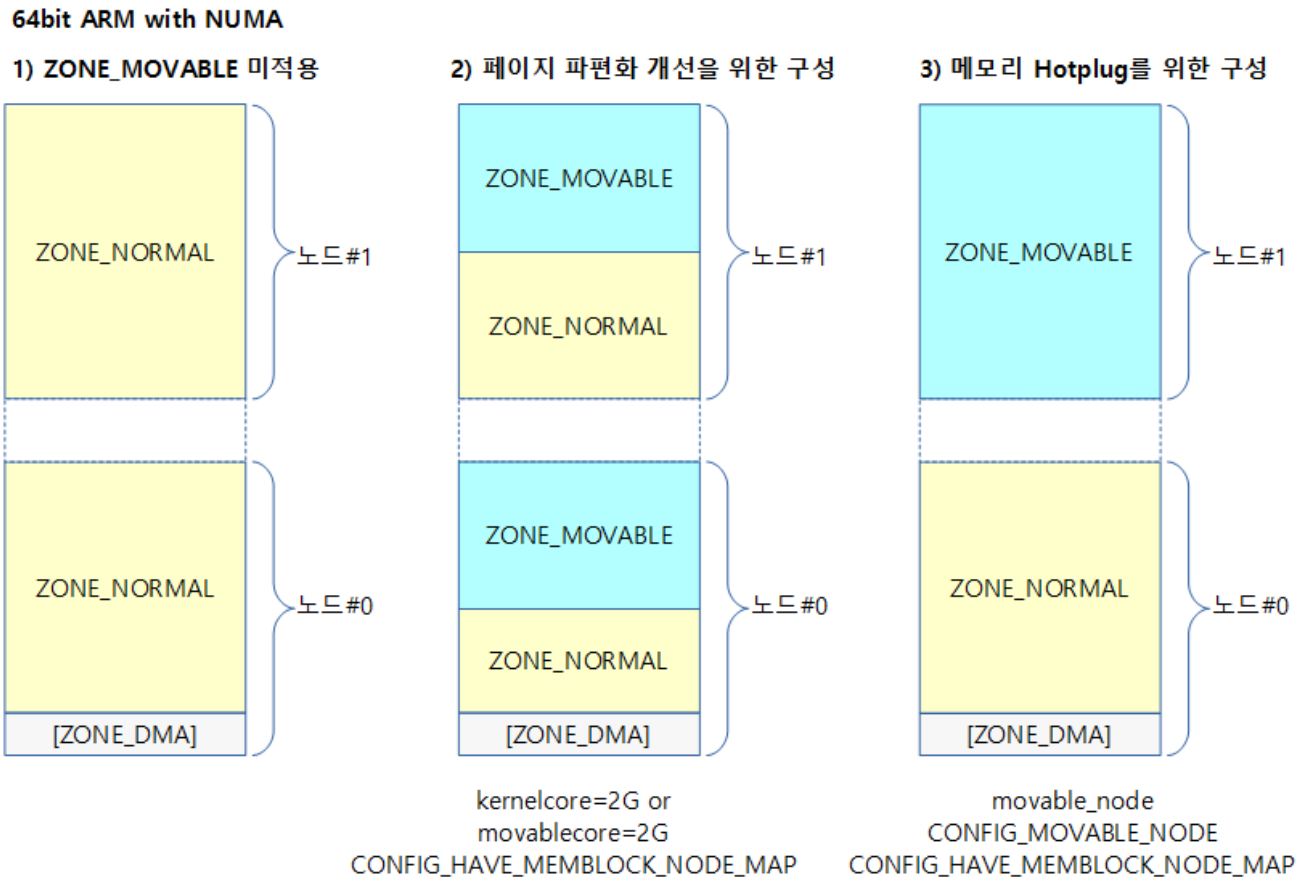
The following figure shows the ZONE_MOVABLE configuration on a NUMA 32-bit ARM system.




(http://jake.dothome.co.kr/wp-content/uploads/2016/06/zone-3.png)

The following figure shows the ZONE_MOVABLE configuration on a 64-bit ARM system.

- As of kernel v4.16-rc1, ZONE_DMA32 is used instead of ZONE_DMA.




(http://jake.dothome.co.kr/wp-content/uploads/2016/06/zone-4.png)

## ZONE_DEVICE

- It is newly added in kernel v4.3 and is intended for use in large persistent memory devices and heterogeneous memory devices.
  - Compared to ZONE_DMA device's desire to access the CPU's main memory, ZONE_DEVICE seeks to:
    - The CPU wants access to a large amount of device memory.
      - Persistent memrory e.g. 8G main memory, 512G high-speed device memory
      - heterogeneous memory e.g. 2G main memory, 6G GPU device memory
    - You want to use cache on the CPU and device side for high-speed processing.
    - simultaneity
      - On both sides, h/w cache coherent is running.
      - On the CPU side, we want to use atomic operations.
- Requirement
  - Memory hot-plug and hot-remove must be supported, and SPARSEMEM_VMEMMAP settings are required to be used only in 64-bit.
- It is being implemented from the server-level x86_64 and PPC_64 architectures.
- You can also use DAX mappings that use userspace for persistent memory devices.
  - Note: mm: ZONE_DEVICE for "device memory" (https://github.com/torvalds/linux/commit/033fbae988fcb67e507720351218189084 8b8e90)

- Heterogeneous Memory The memory accessed by the device can also be accessed by the CPU. (CONFIG_HMM)
    - Added in kernel v4.14.
    - Specify all or some areas.
    - It can be used as a memory swap device via ZONE_DEVICE.
    - HMM memory set to private (CONFIG_DEVICE_PRIVATE) can only be accessed by the device, and the CPU cannot access it using the kernel's normal memory-related APIs, but only through the HMM-specific API.
    - HMM memory set to public (CONFIG_DEVICE_PUBLIC) can be accessed by both the device and the CPU.
        - It can be accessed using all of the kernel's memory APIs.
        - H/W should support solving synchronization and cache coherent issues for two-way mapping.
- High-speed interface
    - The fastest is when the device memory is embedded in the SoC and integrated into the internal high-speed bus.
        - GDDR memory used by embedded GPUs
    - You can also use the DRAM memory bus, which is the fastest external bus
        - Intel's Optane memory (expensive and CPU must support)
    - It is the most popular external (external to the chip, not the system) high-speed interface using PCIe.
        - High-speed SSDs and Intel's Optane memory (low-cost when your CPU can't support it)
            - For reference, the interfaces to which traditional SSDs are connected started with SATA, but now there are AHCI and NVMe (a faster protocol than AHCI) that use PCIe.
        - PC-mounted GPUs also use PCIe.
        - Since the PCIe bus is tens of times slower than the memory access speed, there is a slow limitation when accessing from the CPU.
        - Atomic operations and cache coherent for device memory have not been supported for a long time, but have recently become possible.
- Consultation:
    - Heterogeneous Memory Management (HMM) (https://www.kernel.org/doc/html/v4.18/vm/hmm.html) | Kernel.org
    - ZONE_DEVICE and the future of struct page (https://lwn.net/Articles/717555/) | LWN.net

# consultation

- Memory Model -1- (Basic) (http://jake.dothome.co.kr/mm-1) | Qc
- Memory Model -2- (mem_map) (http://jake.dothome.co.kr/mem_map) | Qc
- Memory Model -3- (Sparse Memory) (http://jake.dothome.co.kr/sparsemem/) | Qc
- Memory Model -4- (APIs) (http://jake.dothome.co.kr/mem_map_page) | Qc
- ZONE Type (http://jake.dothome.co.kr/zone-types) | Sentence C – Current post

- bootmem_init (http://jake.dothome.co.kr/bootmem_init-64) | Qc
- zone_sizes_init() (http://jake.dothome.co.kr/free_area_init_node/) | Qc
- NUMA -1- (ARM64 initialization) (http://jake.dothome.co.kr/numa-1) | Qc
- build_all_zonelists() (http://jake.dothome.co.kr/build_all_zonelists) | Qc

- Memory Hotplug (http://www.kernel.org/doc/Documentation/memory-hotplug.txt) | kernel.org
- The Memory Hot-plug Framework and its Updates | Fujitsu.com – Download (http://events.linuxfoundation.org/sites/events/files/slides/The%20Memory%20Hot-plug%20Framework%20and%20its%20Updates.pdf)

## 21 thoughts to "ZONE Type"

**MOON YOUNG-IL (HTTP://JAKE.DOTHOME.CO.KR)**
2017-04-06 20:02 (http://jake.dothome.co.kr/zone-types/#comment-151850)

Add ZONE_DEVICE content

RESPONSE (/ZONE-TYPES/?REPLYTOCOM=151850#RESPOND)

**CURRYN**
2018-01-25 09:31 (http://jake.dothome.co.kr/zone-types/#comment-175428)

When I was studying about the Linux kernel, I didn't understand the ZONE type, but I understood it after reading this article. Thank you for the good article.

RESPONSE (/ZONE-TYPES/?REPLYTOCOM=175428#RESPOND)

**MOON YOUNG-IL (HTTP://JAKE.DOTHOME.CO.KR)**
2018-01-26 10:55 (http://jake.dothome.co.kr/zone-types/#comment-175439)

Since I just (?) write an article without considering the reader's level of understanding, most people say that my writing is difficult. ^^;
I'm glad you understood.

RESPONSE (/ZONE-TYPES/?REPLYTOCOM=175439#RESPOND)

**DAI NGUYEN**

2018-06-13 10:55 (http://jake.dothome.co.kr/zone-types/#comment-178216)

Dear Mr.Moon Young-IL,
why does the kernel need HIGHMEM zone on 32 bits system? and why doesn't the kernel use 1:1 map directly as in 64 bits system?

RESPONSE (/ZONE-TYPES/?REPLYTOCOM=178216#RESPOND)

**MOON YOUNG-IL (HTTP://JAKE.DOTHOME.CO.KR)**

2018-06-13 23:33 (http://jake.dothome.co.kr/zone-types/#comment-178228)

First, let's look at the biggest features of normal and highmem zones.

1) normal zone
– Use a portion of kernel space to map the physical memory in advance so that the kernel can quickly allocate memory.
– The normal zone has limited space and is used first for kernel memory allocation.

2) highmem zone
– Physical memory is larger than the kernel virtual address space and can not be pre-mapped. The area exceeding this is called highmem zone.
– Unlike kernel memory allocation, when user memory allocation is used, always use physical memory in user space. At this time, the physical memory located in the highmem zone is used first. If this highmem zone is exhausted, the normal zone is used.

Let's look at the difference between a 64-bit system and a 32-bit system for the normal zone.

64bit
– A very large virtual address space is given on 64-bit systems.
– ARMv8 architecture with 64 bit ARM is given a maximum of 256T virtual address space for kernel purposes.
You can use half 128T space as a normal zone.
– Since all this huge physical memory can be accessed from kernel memory, there is no need to separate them by separate highmem zone.

– 32bit ARM
– The maximum virtual address space available on a 32-bit system is 4G.
– Use 1G, which is 1/4 of the kernel size (adjustable kernel configuration) as the kernel space.
– Use a space of several hundreds mega bytes as a normal zone, excluding some in 1G space.
– All memories exceeding normal zone are separated into highmem zone.

Thanks,

————————————————————————————————

먼저 normal zone과 highmem zone의 가장 큰 특징을 알아보겠습니다.

1) normal zone
– 커널에서 메모리를 빠르게 할당할 수 있도록 커널 공간의 일부를 사용하여 미리 물리 메모리를 매핑하여 사용한다.
– normal zone은 공간이 제한되어 있어 커널 메모리 할당에 우선 사용됩니다.

2) highmem zone
– 물리 메모리가 커널 가상 주소 공간보다 더 커 미리 매핑되지 못하는 공간이다. 이 초과하는 영역을 highmem이라고 한다.
– 커널 메모리 할당과 다르게 유저 메모리 할당을 이용하는 경우 항상 유저 공간에 물리 메모리를 매핑하여 사용합니다. 이 때 우선적으로 highmem zone에 위치한 물리 메모리를 사용합니다. 이 highmem zone을 모두 소모하는 경우 normal zone이 사용됩니다.

그 다음 normal zone에 대한 64비트 시스템과 32비트 시스템의 차이를 알아보겠습니다.

64bit
– 64bit 시스템에서는 아주 큰 가상 주소 공간이 주어집니다.
– 64 bit ARM인 ARMv8 아키텍처는 커널 용도로 최대 256T 가상 주소 공간이 주어집니다.
그 중 절반 128T 공간을 normal zone으로 사용할 수 있습니다.
– 이렇게 아주 큰 물리 메모리를 모두 커널 메모리에서 접근가능하므로 별도의 highmem 영역을 두어 따로 구분할 필요 없습니다.

– 32bit ARM
– 32bit 시스템에서 사용할 수 있는 최대 가상 주소 공간의 크기는 4G 입니다.
– 그 중 1/4(대부분 커널이 사용하는 커널 설정)인 1G를 커널 공간으로 사용합니다.
– 1G 공간에서 일부를 제외한 수백M의 공간을 normal zone으로 사용합니다.
– normal zone을 초과하는 메모리들은 모두 highmem zone으로 분리합니다.

(끝)

응답 (/ZONE-TYPES/?REPLYTOCOM=178228#RESPOND)

**DAI NGUYEN**
2018-06-15 16:01 (http://jake.dothome.co.kr/zone-types/#comment-178284)

Thank you for your quick response. I have a recommendation for you that texts and message in the pictures should be commented in english. Because I can not read and understand Korea language well.

응답 (/ZONE-TYPES/?REPLYTOCOM=178284#RESPOND)

**DAI NGUYEN**
2018-06-15 16:17 (http://jake.dothome.co.kr/zone-types/#comment-178285)

Thank you for your quick response. I have a good idea for you. Because I can not read and understand Korea language well. so you can write articles in english. if true, thank you so much.

응답 (/ZONE-TYPES/?REPLYTOCOM=178285#RESPOND)

**문영일 (HTTP://JAKE.DOTHOME.CO.KR)**
2018-06-15 21:47 (http://jake.dothome.co.kr/zone-types/#comment-178292)

Thank you for your feedback. When I first tried to write a blog, I thought I would write it in English. I'll try it someday.

응답 (/ZONE-TYPES/?REPLYTOCOM=178292#RESPOND)

**이호영**
2019-03-07 21:02 (http://jake.dothome.co.kr/zone-types/#comment-203297)

글들 볼때마다 깊이감과 이해감에 존경스럽습니다 글 잘 보고 커널공부 하고 있습니다 !!! 좋은글 감사합니다.

응답 (/ZONE-TYPES/?REPLYTOCOM=203297#RESPOND)

**문영일 (HTTP://JAKE.DOTHOME.CO.KR)**
2019-03-08 07:49 (http://jake.dothome.co.kr/zone-types/#comment-203384)

감사하고 응원합니다. ^^

응답 (/ZONE-TYPES/?REPLYTOCOM=203384#RESPOND)

**SAMURO**
2020-06-30 11:38 (http://jake.dothome.co.kr/zone-types/#comment-258924)

안녕하세요 zone에 대해서 고질적으로 이해가 안되는 부분이 있어 질문드립니다.

1. 물리 메모리공간의 크기가 DMA 디바이스가 접근할 수 있는 주소공간이면 ZONE_DMA또는 ZONE_DMA32를 별도로 지정하지 않는다는 것으로 이해했습니다. 그런데 예를 들어 x86-64bit 환경에서 6GB의 물리 메모리 공간이 있다면, ZONE_DMA32가 필요하게 되고, 이 ZONE_DMA32는 4GB크기로 고정되는 것으로 보이는데, 이런 경우에 ZONE_NORMAL로 사용할 수 있는 영역은 나머지 2GB인가요? 그럼 전체 6GB 메모리 중에서 커널이 kmalloc()하여 사용할 수 있는 영역의 크기는 2GB밖에 되지 않는게 맞나요?

2. ZONE_DMA, ZONE_NORMAL, ZONE_HIGHMEM 모두 가상주소공간에서 커널 주소 영역이 물리메모리에 어떻게 매핑되는가를 설명하고 있는데, 유저 프로세스 주소 영역(예를 들어 4GB 가상주소공간에서 커널이 상위 1GB, 유저프로세스가 하위 3GB를 사용한다면)은 어느 zone에 매핑되는 것인가요?

너무 기본적인 질문 같기도 한데 존의 개념자체가 와닿지 않아 아무리봐도 이해가 되지않네요.
그리고 항상 좋은 블로그 글들 감사드립니다!

응답 (/ZONE-TYPES/?REPLYTOCOM=258924#RESPOND)

**문영일 (HTTP://JAKE.DOTHOME.CO.KR)**
2020-06-30 12:35 (http://jake.dothome.co.kr/zone-types/#comment-258929)

안녕하세요?

1. 6G 메모리 중 ZONE_NORMAL에 사용되는 것이 2G 맞습니다.
ZONE_DMA, ZONE_DMA32, ZONE_NORMAL 모두 kmalloc() 같은 커널 메모리 할당자를 사용하여 사용할 수 있습니다.
다만 가급적 normal -> dma32 -> dma 순으로 메모리를 할당합니다.

2. 가상 공간은 프로세스 수 만큼 많습니다. 10개의 프로세스를 동작시키면 가상 공간이 4G(32bit 기준) x 10개가 되겠죠.
이 가상 공간들의 3G 영역 중 프로세스가 사용한 만큼만 물리 메모리를 할당하여 사용합니다.
그 중 highmem을 우선 할당하여 사용합니다.
부족한 경우 normal, dma 순으로 할당합니다.
또한 가상 공간 중 커널이 사용하는 1G 영역은 모든 프로세스와 공유되는 공간으로 설계되고,
커널이 normal, dma 순으로 메모리를 할당하여 사용합니다. ( kmalloc() 등으로 highmem을 사용하지 않습니다.)

참고로 zone은 물리 메모리 영역에 대해 속도 및 접근 제한 등의 이유로 나누어 관리합니다.

감사합니다.

응답 (/ZONE-TYPES/?REPLYTOCOM=258929#RESPOND)

**SAMURO**
2020-07-02 10:59 (http://jake.dothome.co.kr/zone-types/#comment-259368)

감사합니다!!! 이제 이해가 되었습니다.

응답 (/ZONE-TYPES/?REPLYTOCOM=259368#RESPOND)

**메모리바보**
2021-01-22 22:12 (http://jake.dothome.co.kr/zone-types/#comment-303298)

안녕하세요,
좋은 정보 감사합니다.

여태까지 메모리 Zone을 나누는 것은 커널 메모리에만 국한된 건줄 알았는데, 댓글 문의 중 유저 스페이스영역에 대한 메모리 할당은 highmem, normal, dma 순으로 할당된다고 나와 있네요.
그렇다면 highmem이 없는 64bit의 경우, 유저 스페이스에서 사용하는 메모리는 normal 존에서 할당되나요? normal zone은 virtual memory와 physical memory가 directly mapping되어 있는데 각 프로세스마다 개별 주소공간(0x0000 — 0x max)을 어찌 제공하는지 궁금합니다.

응답 (/ZONE-TYPES/?REPLYTOCOM=303298#RESPOND)

**문영일 (HTTP://JAKE.DOTHOME.CO.KR)**
2021-01-23 14:03 (http://jake.dothome.co.kr/zone-types/#comment-303484)

안녕하세요?

highmem이 없는 64bit OS의 경우 normal -> dma32 -> dma 순으로 할당합니다.
arm64의 경우 메모리가 4G 이하인 경우 커널 config 옵션에 따라 normal 또는 dma32 하나를 사용할 것입니다.

normal을 포함한 그 아래의 존들은 커널 가상 주소 영역에 1:1로 프리 매핑되어 있어, 메모리 할당 시 곧바로 액세스가 가능합니다.
그리고 이 영역들의 free 메모리 영역들은 각 유저 영역에서 메모리 요청을 할 때 vma 형태로 영역을 구분해 놓고,
실제 해당 영역에 접근할 때 fault가 발생한 이 후 lazy allocation을 하며 이 때 해당 유저 가상 주소 영역에 매핑합니다.
따라서 유저 영역에 매핑된 물리 메모리의 경우 커널 영역에도 매핑되어 있으므로 이중 매핑된 상태입니다.

그 뿐 아니라 유저 영역에서 사용하는 glibc 같은 공유 라이브러리들은 여러 개의 유저 영역 수 만큼 추가 매핑되어 사용됩니다.

감사합니다.

응답 (/ZONE-TYPES/?REPLYTOCOM=303484#RESPOND)

**메모리바보**
2021-01-23 19:23 (http://jake.dothome.co.kr/zone-types/#comment-303535)

와... 지식의 깊이에 감탄하고 갑니다....
저도 열심히 찾아봤지만 유저 메모리에 이중 매핑이 되어 있다는 얘기는 못본 것 같아요...
그렇다면 정리하면 커널 가상메모리는 모든 물리메모리에 1:1 맵핑이 되어 있고 유저 가상메모리에서 page fault를 통해 할당하는 물리메모리의 경우, 실제로는 유저 가상메모리와 커널 가상메모리 두 개다에 맵핑이 되어 있다는 말씀이시네요.
감사합니다.!

응답 (/ZONE-TYPES/?REPLYTOCOM=303535#RESPOND)

**문영일 (HTTP://JAKE.DOTHOME.CO.KR)**
2021-01-23 22:24 (http://jake.dothome.co.kr/zone-types/#comment-303572)

네. 정확히 보셨습니다. ^^

응답 (/ZONE-TYPES/?REPLYTOCOM=303572#RESPOND)

**길을잃었다**
2022-09-21 03:16 (http://jake.dothome.co.kr/zone-types/#comment-307067)

안녕하십니까,

ZONE_NORMAL에 대하여 본문에서 다음과 같이 말씀을 해주셨는데,
'시스템에서 지원하는 물리 메모리가 아키텍처의 가상 주소에 '
1:1로 미리 매핑되어 있다는 부분이 명확히 이해가 안되어 질문을 드립니다..

질문:
'1:1로 미리 매핑' 이라는 것은 ZONE_NORMAL 물리 메모리에 대한 페이지 테이블을 미리 전부 구성 해놓은 것이라고 보면 될까요??
예를 들어, 커널 가상 주소 공간이 0에서 시작을 한다고 가정을 한다면.. arm64 아키텍처 / 8GB 메모리를 전부 ZONE_NORMAL로 구성한 경우
가상 주소 공간 0x00_00000000 ~ 0x01_FFFFFFFF 에 해당하는 페이지 테이블이 미리 구성되어 있는 것이 맞을까요??

응답 (/ZONE-TYPES/?REPLYTOCOM=307067#RESPOND)

**문영일 (HTTP://JAKE.DOTHOME.CO.KR)**
2022-09-21 07:47 (http://jake.dothome.co.kr/zone-types/#comment-307069)

네. 맞습니다. 말씀하신것 처럼 가상 주소가 0부터 시작한다고 보면, 8G 메모리 전부가 NORMAL_ZONE이고
이를 0x00_00000000 ~ 0x01_FFFFFFFF 가상 주소 공간에 모두 매핑합니다.

물론 실제 매핑되는 커널 주소 공간은 다음과 같습니다.
(예: VA_BITS=48을 사용하는 커널 주소 공간 – 0xFFFF0000_00000000 ~ 0xFFFF0001_FFFFFFFF)

감사합니다.

응답 (/ZONE-TYPES/?REPLYTOCOM=307069#RESPOND)

**길을잃었다**
2022-09-21 13:10 (http://jake.dothome.co.kr/zone-types/#comment-307070)

답변 정말 감사드립니다!!

응답 (/ZONE-TYPES/?REPLYTOCOM=307070#RESPOND)

**문영일 (HTTP://JAKE.DOTHOME.CO.KR)**
2022-09-26 09:24 (http://jake.dothome.co.kr/zone-types/#comment-307086)

별말씀을요~~ 좋은 하루 되세요.

응답 (/ZONE-TYPES/?REPLYTOCOM=307086#RESPOND)

### 댓글 남기기

이메일은 공개되지 않습니다. 필수 입력창은 * 로 표시되어 있습니다

댓글

이름 *

이메일 *

웹사이트

댓글 작성

‹ Slub Memory Allocator -9- (캐시 Shrink) (http://jake.dothome.co.kr/slub-cache-shrink/)

자유게시판을 열었습니다. › (http://jake.dothome.co.kr/notice-1/)

문c 블로그 (2015 ~ 2023)