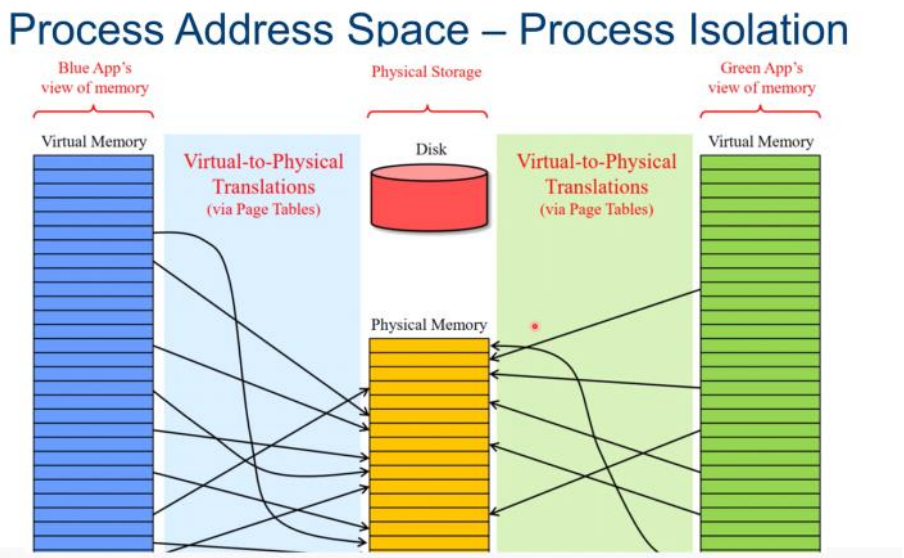
# memory

2023年4月28日 10:09

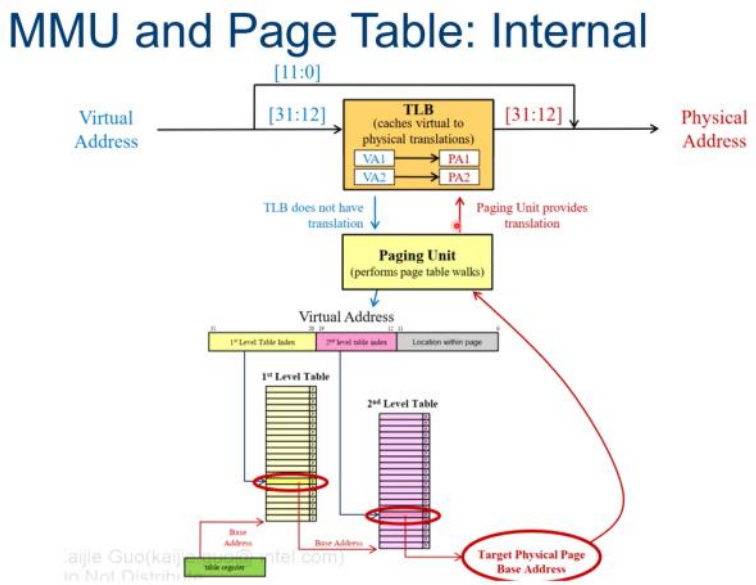
## Agenda

- Process Address Space
- Page
- MMU and Page Table
- TLB
- Virtual Address Space
- Memory allocation in user space
- Memory allocation in kernel space
- Huge Page
- Basic memory tools
- Q&A

80286
地址空间保护隔离
VA到PA的转换
页表




每个进程有一个自己的patge table




MMU（HW）进行page walk

二级页表

# Page Global Directory:

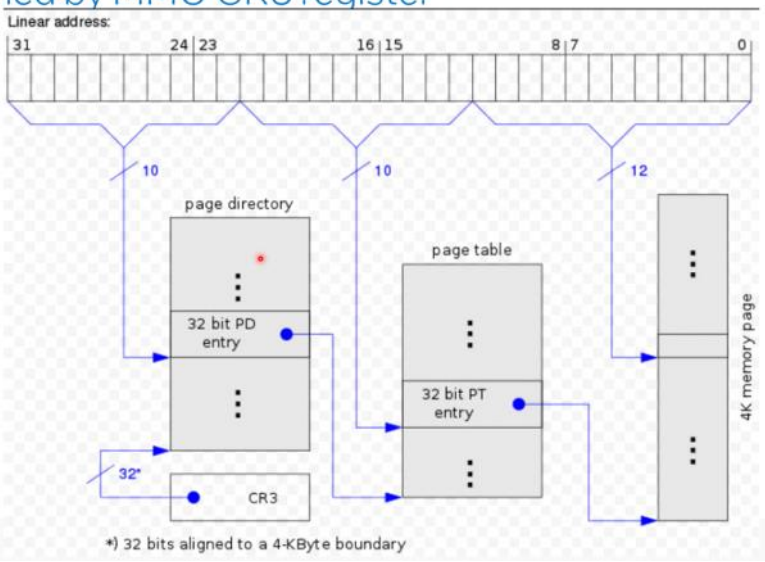
- Consumed by MMU CR3 register

```
struct mm_struct {
        struct {
                struct vm_area_struct *mmap;            /* list of VMAs */
                struct rb_root mm_rb;
                u64 vmacache_seqnum;                   /* per-thread vmacache */
#ifdef CONFIG_MMU
                unsigned long (*get_unmapped_area) (struct file *filp,
                                unsigned long addr, unsigned long len,
                                unsigned long pgoff, unsigned long flags);
#endif
                unsigned long mmap_base;        /* base of mmap area */
                unsigned long mmap_legacy_base; /* base of mmap area in bottom-up allocations
#ifdef CONFIG_HAVE_ARCH_COMPAT_MMAP_BASES
                /* Base adresses for compatible mmap() */
                unsigned long mmap_compat_base;
                unsigned long mmap_compat_legacy_base;
#endif
                unsigned long task_size;        /* size of task vm space */
                unsigned long highest_vm_end;   /* highest vma end address */
                pgd_t * pgd;
```

把进程的pgd读到CR3寄存器

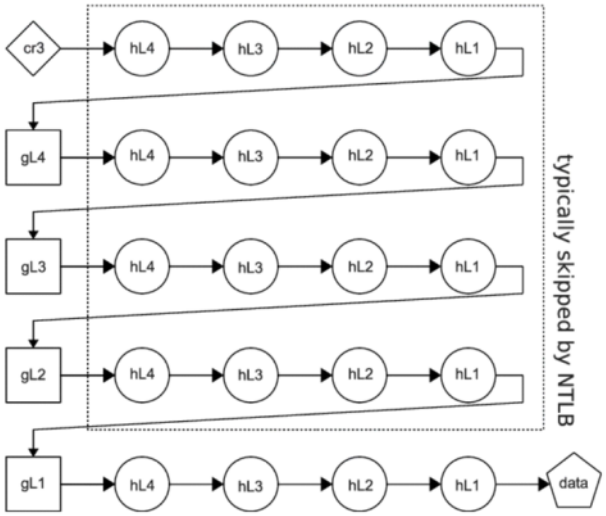
# Page Table Walk in Host (2 Levels)

- Consumed by MMU CR3 register



# Nested Page Table Walk in Guest

- Consumed by MMU CR3 register
- Assisted by EPT (VT-x)
- O(MxN) time complexity
- Meaning of TLB hit rate amplified in cloud



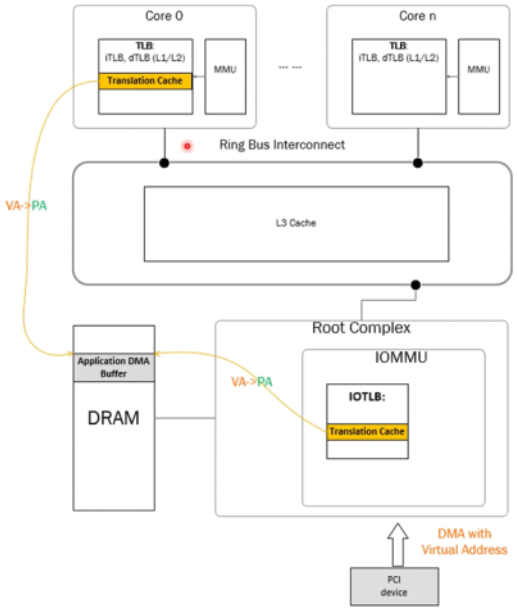

嵌套的页表

适用于虚拟化场景

gva -> gpa -> hva -> hpa

EPT

# MMU and Page Table: Multi-Core



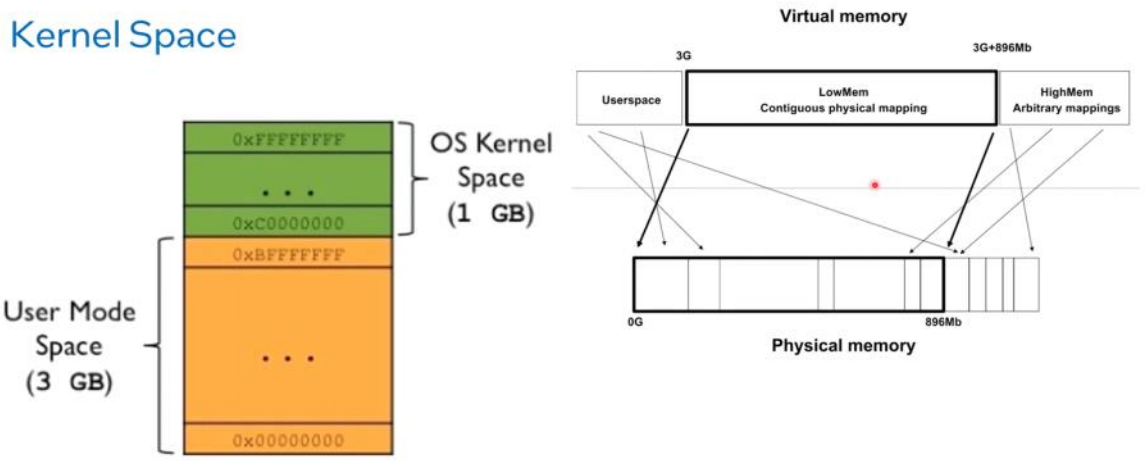

- Each physical core has dedicated TLB
- In Hyperthread mode, threads competitively share TLB
- TLB flushed upon process context switch (Expensive)

SPR

IOMMU IOTLB 基于VA进行DMA

# Virtual Address Space

## Kernel Space

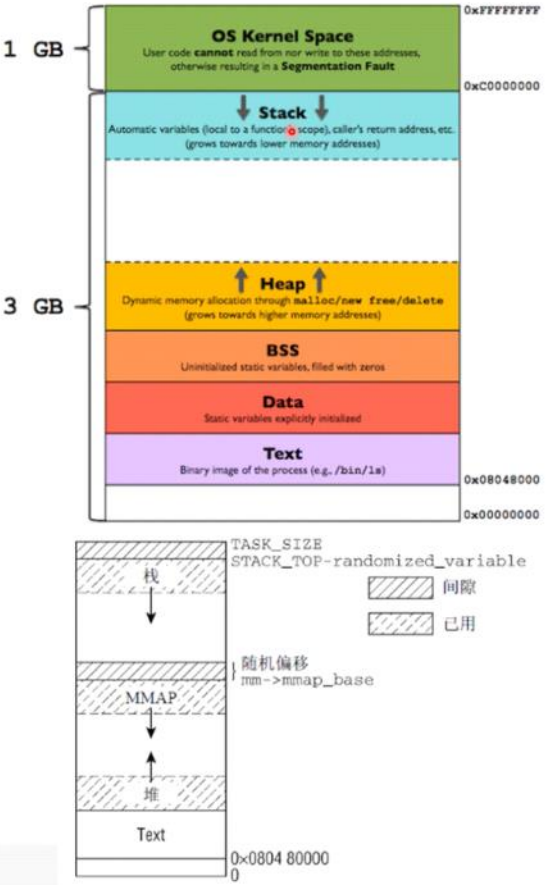



0-896M是连续的

# Virtual Address Space

## User Space

- Text for code segments contain executable instructions
- Data for global or static variant
- BSS uninitialized data segement
- Heap for dynamic memory allocation
- Stack









mmap地址段（mm->mmap_base开始）
/proc/mmap
objdump

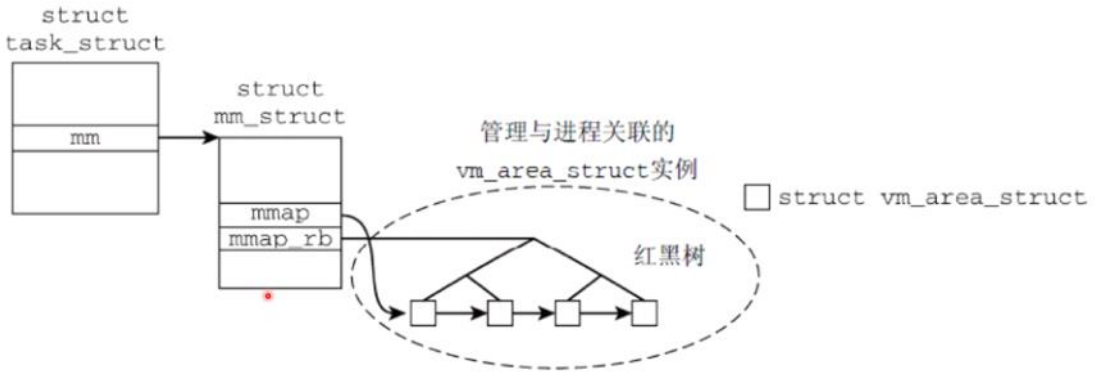
# Process Virtual Space

```
00400000-00401000 r-xp 00000000 fd:02 657816                             /home/kaijie/tmp/sbrk
00600000-00601000 r--p 00000000 fd:02 657816                             /home/kaijie/tmp/sbrk
00601000-00602000 rw-p 00001000 fd:02 657816                             /home/kaijie/tmp/sbrk
00882000-008a3000 rw-p 00000000 00:00 0                                  [heap]
7ff6863cb000-7ff686596000 r-xp 00000000 fd:00 663983                     /usr/lib64/libc-2.25.so
7ff686596000-7ff686796000 ---p 001cb000 fd:00 663983                     /usr/lib64/libc-2.25.so
7ff686796000-7ff68679a000 r--p 001cb000 fd:00 663983                     /usr/lib64/libc-2.25.so
7ff68679a000-7ff68679c000 rw-p 001cf000 fd:00 663983                     /usr/lib64/libc-2.25.so
7ff68679c000-7ff6867a0000 rw-p 00000000 00:00 0
7ff6867a0000-7ff6867c6000 r-xp 00000000 fd:00 667147                     /usr/lib64/ld-2.25.so
7ff6869a8000-7ff6869aa000 rw-p 00000000 00:00 0
7ff6869c5000-7ff6869c6000 r--p 00025000 fd:00 667147                     /usr/lib64/ld-2.25.so
7ff6869c6000-7ff6869c8000 rw-p 00026000 fd:00 667147                     /usr/lib64/ld-2.25.so
7ffeddb67000-7ffeddb88000 rw-p 00000000 00:00 0                          [stack]
7ffeddbbd000-7ffeddbc0000 r--p 00000000 00:00 0                          [vvar]
7ffeddbc0000-7ffeddbc1000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

```
Contents of section .plt:
 400490 ff35720b 2000ff25 740b2000 0f1f4000  .5r. ..%t. ...@.
 4004a0 ff25720b 20006800 000000e9 e0ffffff  .%r. .h.........
 4004b0 ff256a0b 20006801 000000e9 d0ffffff  .%j. .h.........
 4004c0 ff25620b 20006802 000000e9 c0ffffff  .%b. .h.........
 4004d0 ff255a0b 20006803 000000e9 b0ffffff  .%Z. .h.........
Contents of section .text:
 4004e0 31ed4989 d15e4889 e24883e4 f0505449  1.I..^H..H...PTI
 4004f0 c7c09006 400048c7 c1200640 0048c7c7  ....@.H.. .@.H..
 400500 b7054000 ff15e60a 2000f40f 1f440000  ..@..... ....D..
 400510 55b84010 6000483d 40106000 4889e574  U.@.`.H=@.`.H..t
 400520 17b80000 00004885 c0740d5d bf401060  ......H..t.].@.`
 400530 00ffe00f 1f440000 5dc3660f 1f440000  .....D..].f..D..
 400540 be401060 00554881 ee401060 004889e5  .@.`.UH..@.`.H..
 400550 48c1fe03 4889f048 c1e83f48 01c648d1  H...H..H..?H..H.
 400560 fe7415b8 00000000 4885c074 0b5dbf40  .t......H..t.].@
 400570 106000ff e00f1f00 5dc3660f 1f440000  .`......]..f..D..
 400580 803db50a 20000075 17554889 e5e87eff  .=.. ..u.UH...~.
 400590 ffffc685 a30a2000 015dc30f 1f440000  ...... ..]...D..
 4005a0 f3c30f1f 4000662e 0f1f8400 00000000  ....@.f.........
 4005b0 554889e5 5de80955 4889e548 83ec1040  UH..]..UH..H...H
 4005c0 c745f800 000000e8 d4feffff 89c6bfb0  .E..............
 4005d0 064000b8 00000000 e8d3feff ffe8eefe  .@..............
 4005e0 ffffbf00 300000e8 d4feffff 488945f8  ....0.......H.E.
 4005f0 e8abfeff ff89c6bf 10074000 b8000000  ..........@.....
 400600 00e8aafe ffffe8c5 feffffb8 00000000  ................
 400610 c9c3662e 0f1f8400 00000000 0f1f4000  ..f...........@.
 400620 41574156 4989d741 5541544c 8d25de07  AWAVI..AUATL.%..
 400630 20005548 8d2dde07 20005341 89fd4989   .UH.-.. .SA..I.
 400640 f64c29e5 4883ec08 48c1fd03 e827feff  .L).H...H....'..
 400650 ff4885ed 742031db 0f1f8400 00000000  .H..t 1.........
 400660 4c89fa4c 89f64489 ef41ff14 dc4883c3  L..L..D..A...H..
 400670 014839dd 75ea4883 c4085b5d 415c415d  .H9.u.H...[]A\A]
 400680 415e415f c390662e 0f1f8400 00000000  A^A_..f.........
 400690 f3c3                                 ..
Contents of section .fini:
 400694 4883ec08 4883c408 c3                 H...H....
Contents of section .rodata:
 4006a0 01000200 00000000 00000000 00000000  ................
 4006b0 7362726b 20746573 742c2073 65652063  sbrk test, see c
 4006c0 75727265 6e742062 7265616b 206f6620  urrent break of
 4006d0 68656170 20757369 6e672027 63617420  heap using 'cat
 4006e0 2f70726f 632f2564 2f6d6170 73207c20  /proc/%d/maps |
 4006f0 67726570 20686561 70207c20 61776b20  grep heap | awk
 400700 277b7072 696e7420 24317d27 0a000000  '{print $1}'....
 400710 30783333 30302062 79746573 20616c6c  0x3000 bytes all
 400720 6f636174 65642074 6f206865 61702c20  ocated to heap,
 400730 73656520 63757272 656e7420 62726561  see current brea
 400740 6b206f66 20686561 70207573 696e6720  k of heap using
 400750 27636174 202f7072 6f632f25 642f6d61  'cat /proc/%d/ma
 400760 7073207c 20677265 70206865 6170207c  ps | grep heap |
 400770 2061776b 20277b70 72696e74 2024317d   awk '{print $1}
 400780 270a00                               '..
```

动态库mmap, 用户mmap等多个mmap

# Mm_struct




可以找到每个mmap的地址

# Vm_area_struct

```
struct vm_area_struct {
        /* The first cache line has the info for VMA tree walking. */

        unsigned long vm_start;         /* Our start address within vm_mm. */
        unsigned long vm_end;           /* The first byte after our end address
                                           within vm_mm. */

        /* linked list of VM areas per task, sorted by address */
        struct vm_area_struct *vm_next, *vm_prev;

        struct rb_node vm_rb;

        /*
         * Largest free memory gap in bytes to the left of this VMA.
         * Either between this VMA and vma->vm_prev, or between one of the
         * VMAs below us in the VMA rbtree and its ->vm_prev. This helps
         * get_unmapped_area find a free area of the right size.
         */
        unsigned long rb_subtree_gap;

        /* Second cache line starts here. */

        struct mm_struct *vm_mm;        /* The address space we belong to. */
        pgprot_t vm_page_prot;          /* Access permissions of this VMA. */
        unsigned long vm_flags;         /* Flags, see mm.h. */

        /*
         * For areas with an address space and backing store,
         * linkage into the address_space->i_mmap interval tree.
         */
        struct {
                struct rb_node rb;
                unsigned long rb_subtree_last;
        } shared;

        /*
         * A file's MAP_PRIVATE vma can be in both i_mmap tree and anon_vma
         * list, after a COW of one of the file pages.  A MAP_SHARED vma
         * can only be in the i_mmap tree.  An anonymous MAP_PRIVATE, stack
         * or brk vma (with NULL file) can only be in an anon_vma list.
         */
        struct list_head anon_vma_chain; /* Serialized by mmap_sem &
                                          * page_table_lock */
        struct anon_vma *anon_vma;      /* Serialized by page_table_lock */
```

32bit VA->64bit VA
purley(48bits VA) 128TB

# Process Virtual Space: 4 Level Page Table

| Start addr | Offset | End addr | Size | VM area description |
|---|---|---|---|---|
| 0000000000000000 | 0 | 00007fffffffffff | 128 TB | user-space virtual memory, different per mm |
| 0000800000000000 | +128 TB | ffff7fffffffffff | ~16M TB | ... huge, almost 64 bits wide hole of non-canonical virtual memory addresses up to the -128 TB starting offset of kernel mappings. |
| | | | | Kernel-space virtual memory, shared between all processes: |
| ffff800000000000 | -128 TB | ffff87ffffffffff | 8 TB | ... guard hole, also reserved for hypervisor |
| ffff880000000000 | -120 TB | ffff887fffffffff | 0.5 TB | LDT remap for PTI |
| ffff888000000000 | -119.5 TB | ffffc87fffffffff | 64 TB | direct mapping of all physical memory (page_offset_base) |
| ffffc88000000000 | -55.5 TB | ffffc8ffffffffff | 0.5 TB | ... unused hole |
| ffffc90000000000 | -55 TB | ffffe8ffffffffff | 32 TB | vmalloc/ioremap space (vmalloc_base) |
| ffffe90000000000 | -23 TB | ffffe9ffffffffff | 1 TB | ... unused hole |
| ffffea0000000000 | -22 TB | ffffeaffffffffff | 1 TB | virtual memory map (vmemmap_base) |
| ffffeb0000000000 | -21 TB | ffffebffffffffff | 1 TB | ... unused hole |
| ffffec0000000000 | -20 TB | fffffbffffffffff | 16 TB | KASAN shadow memory |
| | | | | Identical layout to the 56-bit one from here on: |
| ffffffc0000000000 | -4 TB | ffffffdfffffffff | 2 TB | ... unused hole<br>vaddr_end for KASLR |
| ffffffe0000000000 | -2 TB | ffffffe7fffffffff | 0.5 TB | cpu_entry_area mapping |
| ffffffe8000000000 | -1.5 TB | ffffffeeffffffff | 0.5 TB | ... unused hole |
| fffffff0000000000 | -1 TB | fffffff7fffffffff | 0.5 TB | %esp fixup stacks |
| fffffff8000000000 | -512 GB | fffffffeefffffff | 444 GB | ... unused hole |
| ffffffff00000000 | -68 GB | ffffffffefffffff | 64 GB | EFI region mapping space |
| ffffffff80000000 | -4 GB | ffffffff7fffffff | 2 GB | ... unused hole |
| ffffffff80000000 | -2 GB | ffffffff9fffffff | 512 MB | kernel text mapping, mapped to physical address 0 |
| ffffffff80000000 | -2048 MB | | | |
| ffffffffa0000000 | -1536 MB | fffffffffeffffff | 1520 MB | module mapping space |
| ffffffffff000000 | -16 MB | | | |
| FIXADDR_START | ~-11 MB | ffffffffff5fffff | ~0.5 MB | kernel-internal fixmap range, variable size and offset |
| ffffffffff600000 | -10 MB | ffffffffff600fff | 4 kB | legacy vsyscall ABI |
| ffffffffffe00000 | -2 MB | ffffffffffffffff | 2 MB | ... unused hole |

- 48 bits virtual address
- Default for 64 system

```
address sizes   : 46 bits physical, 48 bits virtual
```

SPR(57bit VA 64PB 5级页表)

# Process Virtual Space: 5 Level Page Table

| Start addr | Offset | End addr | Size | VM area description |
|---|---|---|---|---|
| 0000000000000000 | 0 | 00ffffffffffffff | 64 PB | user-space virtual memory, different per mm |
| 0100000000000000 | +64 PB | feffffffffffffff | ~16K PB | ... huge, still almost 64 bits wide hole of non-canonical virtual memory addresses up to the -64 PB starting offset of kernel mappings. |
| | | | | Kernel-space virtual memory, shared between all processes: |
| ff00000000000000 | -64 PB | ff0fffffffffffff | 4 PB | ... guard hole, also reserved for hypervisor |
| ff10000000000000 | -60 PB | ff10ffffffffffff | 0.25 PB | LDT remap for PTI |
| ff11000000000000 | -59.75 PB | ff90ffffffffffff | 32 PB | direct mapping of all physical memory (page_offset_base) |
| ff91000000000000 | -27.75 PB | ff9fffffffffffff | 3.75 PB | ... unused hole |
| ffa0000000000000 | -24 PB | ffd1ffffffffffff | 12.5 PB | vmalloc/ioremap space (vmalloc_base) |
| ffd2000000000000 | -11.5 PB | ffd3ffffffffffff | 0.5 PB | ... unused hole |
| ffd4000000000000 | -11 PB | ffd5ffffffffffff | 0.5 PB | virtual memory map (vmemmap_base) |
| ffd6000000000000 | -10.5 PB | ffdeffffffffffff | 2.25 PB | ... unused hole |
| ffdf000000000000 | -8.25 PB | fffffbffffffffff | ~8 PB | KASAN shadow memory |
| | | | | Identical layout to the 47-bit one from here on: |
| fffffc0000000000 | -4 TB | fffffdffffffffff | 2 TB | ... unused hole<br>vaddr_end for KASLR |
| fffffe0000000000 | -2 TB | fffffe7fffffffff | 0.5 TB | cpu_entry_area mapping |
| fffffe8000000000 | -1.5 TB | fffffeffffffffff | 0.5 TB | ... unused hole |
| ffffff0000000000 | -1 TB | ffffff7fffffffff | 0.5 TB | %esp fixup stacks |
| ffffff8000000000 | -512 GB | ffffffeeffffffff | 444 GB | ... unused hole |
| ffffffef00000000 | -68 GB | fffffffeffffffff | 64 GB | EFI region mapping space |
| ffffffff00000000 | -4 GB | ffffffff7fffffff | 2 GB | ... unused hole |
| ffffffff80000000 | -2 GB | ffffffff9fffffff | 512 MB | kernel text mapping, mapped to physical address 0 |
| ffffffff80000000 | -2048 MB | | | |
| ffffffffa0000000 | -1536 MB | fffffffffeffffff | 1520 MB | module mapping space |
| ffffffffff000000 | -16 MB | | | |
| FIXADDR_START | ~-11 MB | ffffffffff5fffff | ~0.5 MB | kernel-internal fixmap range, variable size and offset |
| ffffffffff600000 | -10 MB | ffffffffff600fff | 4 kB | legacy vsyscall ABI |
| ffffffffffe00000 | -2 MB | ffffffffffffffff | 2 MB | ... unused hole |

- 57 bits VA versus 4LPT
- Only supported on latest CPU
- Larger VM range
- Degraded performance
- Must be enabled with kernel compile FLAG

```
address sizes   : 52 bits physical, 57 bits virtual
```

hugepage mmap

# Process Virtual Space: Example

```
7ff8957b1000-7ff8959b0000 ---p 00006000 08:05 3221595625                 /usr/local/lib/libusdm_drv_s.so
7ff8959b0000-7ff8959b1000 r--p 00005000 08:05 3221595625                 /usr/local/lib/libusdm_drv_s.so
7ff8959b1000-7ff8959b2000 rw-p 00006000 08:05 3221595625                 /usr/local/lib/libusdm_drv_s.so
7ff8959b2000-7ff8959c3000 rw-p 00000000 00:00 0
7ff8959c3000-7ff895a48000 r-xp 00000000 08:05 3221563146                 /usr/local/lib/libqat_s.so
7ff895a48000-7ff895c47000 ---p 00085000 08:05 3221563146                 /usr/local/lib/libqat_s.so
7ff895c47000-7ff895c49000 r--p 00084000 08:05 3221563146                 /usr/local/lib/libqat_s.so
7ff895c49000-7ff895c4a000 rw-p 00086000 08:05 3221563146                 /usr/local/lib/libqat_s.so
7ff895c4a000-7ff895c98000 rw-p 00000000 00:00 0
7ff895c98000-7ff895cbe000 r-xp 00000000 08:05 1073742939                 /usr/lib64/ld-2.25.so
7ff895e82000-7ff895e88000 rw-p 00000000 00:00 0
7ff895e88000-7ff895ea8000 r-xp 00000000 08:05 1075445448                 /usr/lib64/libudev.so.1.6.6
7ff895ea8000-7ff895ea9000 r--p 0001f000 08:05 1075445448                 /usr/lib64/libudev.so.1.6.6
7ff895ea9000-7ff895eaa000 rw-p 00020000 08:05 1075445448                 /usr/lib64/libudev.so.1.6.6
7ff895eaa000-7ff895ead000 rw-p 00000000 00:00 0
7ff895eb0000-7ff895ebc000 rw-p 00000000 00:00 0
7ff895ebc000-7ff895ebd000 rw-s 00000000 00:06 343252                     /dev/uio0
7ff895ebd000-7ff895ebe000 r--p 00025000 08:05 1073742939                 /usr/lib64/ld-2.25.so
7ff895ebe000-7ff895ec0000 rw-p 00026000 08:05 1073742939                 /usr/lib64/ld-2.25.so
7fff70e56000-7fff70e77000 rw-p 00000000 00:00 0                          [stack]
7fff70e79000-7fff70e7c000 r--p 00000000 00:00 0                          [vvar]
7fff70e7c000-7fff70e7e000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

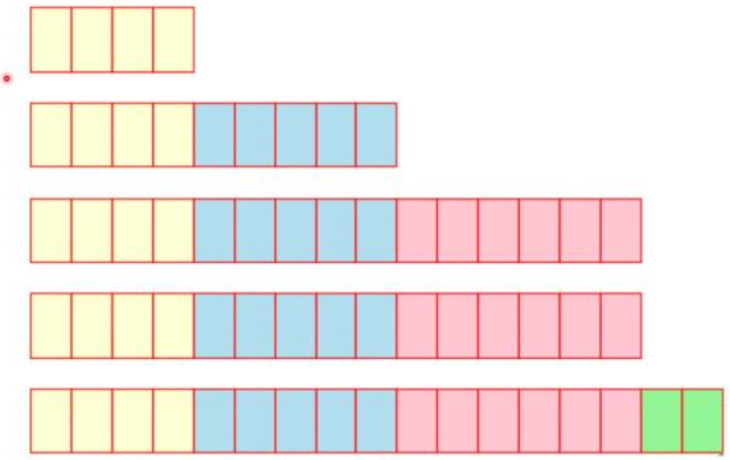
map到userspace

# Memory Allocation Methods in User Space

- Static Memory: BSS
- Static Memory: Data
- Brk/sbrk
- Malloc
- Mmap
- Posix_memalign

# BRK/SBRK: Simple and Naïve Heap Manipulation

p1 = sbrk(4)
p2 = sbrk(5)
p3 = sbrk(5)
free(p2)
p3 = sbrk(2)



sbrk:单向增长：p2被释放,p3不能重复利用，造成内存碎片

# SBRK DEMO: Simple Memory Allocation

```c
int main()
{
        int *p = NULL;

        printf("sbrk test, see current break of heap using 'cat /proc/%d/maps | grep heap | awk \'{print $1}\'\n", getpid());
        getchar();

        p = sbrk(0x3000);

        printf("0x3000 bytes allocated to heap, see current break of heap using 'cat /proc/%d/maps | grep heap | awk \'{print $1}\'\n", getpid());
        getchar();

        return 0;
}
```

```
[kaijie@localhost tmp]$ ./sbrk
sbrk test, see current break of heap using 'cat /proc/6357/maps | grep heap | awk '{print $1}'
```

```
[kaijie@localhost ~]$ cat /proc/6357/maps | grep heap | awk '{print $1}'
017a1000-017c2000
```

```
0x3000 bytes allocated to heap, see current break of heap using 'cat /proc/6357/maps | grep heap | awk '{print $1}'
```

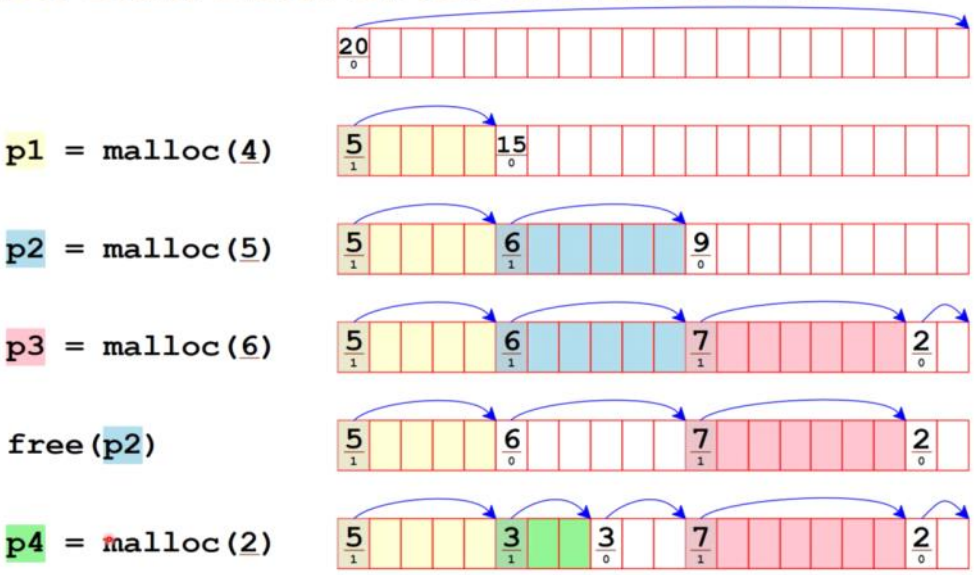
```
[kaijie@localhost ~]$ cat /proc/6357/maps | grep heap | awk '{print $1}'
017a1000-017c5000
```

# SBRK DEMO: Heap Capacity Monitor

```c
int main()
{
        void *brk_init = NULL;
        brk_init = sbrk(0);
        int heap_size = 0;
        printf("brk_init: 0x%x\n", brk_init);

        char *p1 = malloc(0x1000);
        char *p2 = malloc(0x20000);

        printf("brk_current: 0x%x\n", sbrk(0));
        heap_size = sbrk(0) - brk_init;
        printf("Heap consumed: 0x%x\n", heap_size);

        getchar();
        return 0;
}
```

```
[kaijie@localhost tmp]$ gcc -o heap_size heap_size.c
[kaijie@localhost tmp]$ ./heap_size
brk_init: 0x7e4000
brk_current: 0x805000
Heap consumed: 0x21000
```

# Malloc: Build Block on top of BRK/SBRK



# MMAP: Prototype

```
SYNOPSIS
       #include <sys/mman.h>

       void *mmap(void *addr, size_t length, int prot, int flags,
                  int fd, off_t offset);
       int munmap(void *addr, size_t length);

       See NOTES for information on feature test macro requirements.

DESCRIPTION
       mmap() creates a new mapping in the virtual address space of the calling process.  The starting address for the new mapping
       is specified in addr.  The length argument specifies the length of the mapping.

       If addr is NULL, then the kernel chooses the address at which to create the mapping; this is the most  portable  method  of
       creating  a  new  mapping.   If  addr  is not NULL, then the kernel takes it as a hint about where to place the mapping; on
       Linux, the mapping will be created at a nearby page boundary.  The address of the new mapping is returned as the result  of
       the call.

       The  contents of a file mapping (as opposed to an anonymous mapping; see MAP_ANONYMOUS below), are initialized using length
       bytes starting at offset offset in the file (or other object) referred to by the file descriptor fd.  offset must be a mul-
       tiple of the page size as returned by sysconf(_SC_PAGE_SIZE).

       The  prot  argument describes the desired memory protection of the mapping (and must not conflict with the open mode of the
       file).  It is either PROT_NONE or the bitwise OR of one or more of the following flags:
```

userspace和kernel可以share同一段内存

？ mmap到kernel space还是user space

# MMAP: Usage

- Allocate real DRAM (page aligned)
- Allocate File backed virtual memory
- Map MMIO registers to user space
- Allocate Huge Page

private：只能是kernel和进程之间使用

map_hugetlb:把hugepage 映射到userspace?

# Madvise : Advise to Kernel of a Buffer

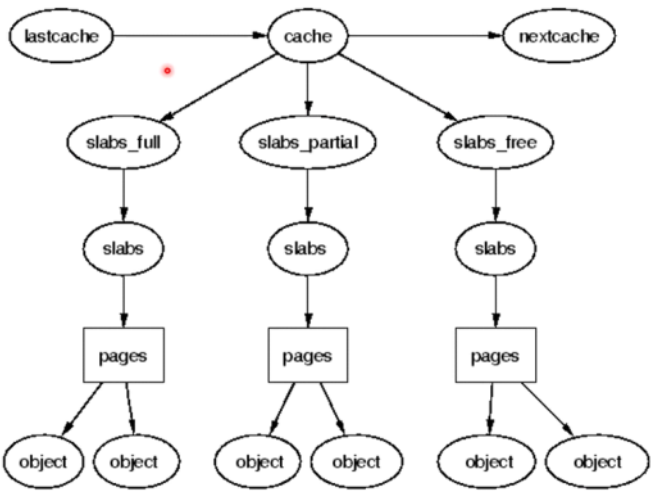
```
NAME
       madvise - give advice about use of memory

SYNOPSIS
       #include <sys/mman.h>

       int madvise(void *addr, size_t length, int advice);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       madvise():
           Since glibc 2.19:
               _DEFAULT_SOURCE
           Up to and including glibc 2.19:
               _BSD_SOURCE

DESCRIPTION
       The  madvise() system call is used to give advice or directions to the kernel about the address range
       beginning at address addr and with size length bytes.  Initially, the system call supported a set  of
       "conventional"  advice  values,  which  are  also available on several other implementations.  (Note,
       though, that madvise() is not specified in POSIX.)  Subsequently, a number of  Linux-specific  advice
       values have been added.

   Conventional advice values
       The  advice  values  listed  below allow an application to tell the kernel how it expects to use some
       mapped or shared memory areas, so that the kernel can choose appropriate read-ahead and caching tech-
       niques.  These advice values do not influence the semantics of the application (except in the case of
       MADV_DONTNEED), but may influence its performance.  All of the advice values listed here have analogs
       in  the  POSIX-specified  posix_madvise(3)  function, and the values have the same meanings, with the
       exception of MADV_DONTNEED.
```

# Memory Allocation Methods in Kernel Space

- kmalloc / zalloc
- vmalloc
- Alloc_pages
- Kmem_cache_create (SLAB)

# Kmalloc: Back Engines

- Just an Interface
- Could be backed by SLAB/SLUB/SLOB
- SLAB: (mm/slab.c)
  - Default allocator until Linux kernel 2.6.23
  - sets up a pool of pre-allocated objects of various sizes
  - Large overhead in meta data
- SLUB: (mm/slub.c)
  - Default allocator in modern Linux kernel
  - Removed meta data compared to SLAB
- SLOB: Cost saving and little overhead, designed for embedded systems




避免过多申请/释放内存对象

slab full: 每个slab上的page都满了

每个slab包含多少page?

lastcache 和nextcache管理slab

kmalloc

cache里包含inode信息等meta data（page上有bitmap）

# Kmalloc: Flags (gfp.h)

```
#define GFP_ATOMIC          (__GFP_
#define GFP_KERNEL          (__GFP_
#define GFP_KERNEL_ACCOUNT  (GFP
#define GFP_NOWAIT          (__GFP_
#define GFP_NOIO            (__GFP_
#define GFP_NOFS            (__GFP_
#define GFP_USER            (__GFP_
#define GFP_DMA             __GFP_D
#define GFP_DMA32           __GFP_D
#define GFP_HIGHUSER        (GFP_US
#define GFP_HIGHUSER_MOVABLE
#define GFP_TRANSHUGE_LIGHT
                            __GFP_
#define GFP_TRANSHUGE       (GFP_TR
```

```
* Useful GFP flag combinations that are commonly used. It is recommended
* that subsystems start with one of these combinations and then set/clear
* %__GFP_FOO flags as necessary.
*
* %GFP_ATOMIC users can not sleep and need the allocation to succeed. A lower
* watermark is applied to allow access to "atomic reserves"
*
* %GFP_KERNEL is typical for kernel-internal allocations. The caller requires
* %ZONE_NORMAL or a lower zone for direct access but can direct reclaim.
*
* %GFP_KERNEL_ACCOUNT is the same as GFP_KERNEL, except the allocation is
* accounted to kmemcg.
*
* %GFP_NOWAIT is for kernel allocations that should not stall for direct
* reclaim, start physical IO or use any filesystem callback.
*
* %GFP_NOIO will use direct reclaim to discard clean pages or slab pages
* that do not require the starting of any physical IO.
* Please try to avoid using this flag directly and instead use
* memalloc_noio_{save,restore} to mark the whole scope which cannot
* perform any IO with a short explanation why. All allocation requests
* will inherit GFP_NOIO implicitly.
*
* %GFP_NOFS will use direct reclaim but will not use any filesystem interfaces.
* Please try to avoid using this flag directly and instead use
* memalloc_nofs_{save,restore} to mark the whole scope which cannot/shouldn't
* recurse into the FS layer with a short explanation why. All allocation
* requests will inherit GFP_NOFS implicitly.
*
* %GFP_USER is for userspace allocations that also need to be directly
* accessibly by the kernel or hardware. It is typically used by hardware
* for buffers that are mapped to userspace (e.g. graphics) that hardware
* still must DMA to. cpuset limits are enforced for these allocations.
*
* %GFP_DMA exists for historical reasons and should be avoided where possible.
* The flags indicates that the caller requires that the lowest zone be
* used (%ZONE_DMA or 16M on x86-64). Ideally, this would be removed but
* it would require careful auditing as some users really require it and
* others use the flag to avoid lowmem reserves in %ZONE_DMA and treat the
* lowest zone as a type of emergency reserve.
*
* %GFP_DMA32 is similar to %GFP_DMA except that the caller requires a 32-bit
* address.
*
* %GFP_HIGHUSER is for userspace allocations that may be mapped to userspace,
* do not need to be directly accessible by the kernel but that cannot
* move once in use. An example may be a hardware allocation that maps
* data directly into userspace but has no addressing limitations.
*
* %GFP_HIGHUSER_MOVABLE is for userspace allocations that the kernel does not
* need direct access to but can use kmap() when access is required. They
* are expected to be movable via page reclaim or page migration. Typically,
* pages on the LRU would also be allocated with %GFP_HIGHUSER_MOVABLE.
*
* %GFP_TRANSHUGE and %GFP_TRANSHUGE_LIGHT are used for THP allocations. They
```



# Kmalloc: Core Affinity Memory

```
static __always_inline void *kmalloc_node(size_t size, gfp_t flags, int node)
{
#ifndef CONFIG_SLOB
        if (__builtin_constant_p(size) &&
                size <= KMALLOC_MAX_CACHE_SIZE) {
                unsigned int i = kmalloc_index(size);

                if (!i)
                        return ZERO_SIZE_PTR;

                return kmem_cache_alloc_node_trace(
                                kmalloc_caches[kmalloc_type(flags)][i],
                                                flags, node, size);
        }
#endif
        return __kmalloc_node(size, flags, node);
}
```

kmalloc：指定在那个numa node上分配内存，避免跨numa node

# Kmem_cache_create: Talk with SLAB Directly

- High speed allocation for commonly used objects with known size
- Widely used by Linux internal structures, e.g. task_struct, mm_struct

```
/**
 * kmem_cache_create - Create a cache.
 * @name: A string which is used in /proc/slabinfo to identify this cache.
 * @size: The size of objects to be created in this cache.
 * @align: The required alignment for the objects.
 * @flags: SLAB flags
 * @ctor: A constructor for the objects.
 *
 * Cannot be called within a interrupt, but can be interrupted.
 * The @ctor is run when new pages are allocated by the cache.
 *
 * The flags are
 *
 * %SLAB_POISON - Poison the slab with a known test pattern (a5a5a5a5)
 * to catch references to uninitialised memory.
 *
 * %SLAB_RED_ZONE - Insert `Red` zones around the allocated memory to check
 * for buffer overruns.
 *
 * %SLAB_HWCACHE_ALIGN - Align the objects in this cache to a hardware
 * cacheline.  This can be beneficial if you're counting cycles as closely
 * as davem.
 *
 * Return: a pointer to the cache on success, NULL on failure.
 */
struct kmem_cache *
kmem_cache_create(const char *name, unsigned int size, unsigned int align,
                  slab_flags_t flags, void (*ctor)(void *))
{
        return kmem_cache_create_usercopy(name, size, align, flags, 0, 0,
                                          ctor);
}
EXPORT_SYMBOL(kmem_cache_create);
```

# Slab debug (proc/slabinfo)

```
[kaijie@localhost ~]$ sudo cat /proc/slabinfo
slabinfo - version: 2.1
# name            <active_objs> <num_objs> <objsize> <objperslab> <pagesperslab> : tunables <limit> <batchcount> <s
isofs_inode_cache     50     50    640   25    4 : tunables    0    0    0 : slabdata      2      2      0
fuse_request         120    120    136   30    1 : tunables    0    0    0 : slabdata      4      4      0
fuse_inode            19     19    832   19    4 : tunables    0    0    0 : slabdata      1      1      0
nf_conntrack         425    425    320   25    2 : tunables    0    0    0 : slabdata     17     17      0
rpc_inode_cache       25     25    640   25    4 : tunables    0    0    0 : slabdata      1      1      0
ext4_groupinfo_4k    532    532    144   28    1 : tunables    0    0    0 : slabdata     19     19      0
ip6-frags              0      0    184   22    1 : tunables    0    0    0 : slabdata      0      0      0
PINGv6               156    156   1216   26    8 : tunables    0    0    0 : slabdata      6      6      0
RAWv6                286    286   1216   26    8 : tunables    0    0    0 : slabdata     11     11      0
UDPv6                300    300   1280   25    8 : tunables    0    0    0 : slabdata     12     12      0
tw_sock_TCPv6          0      0    248   16    1 : tunables    0    0    0 : slabdata      0      0      0
request_sock_TCPv6     0      0    304   26    2 : tunables    0    0    0 : slabdata      0      0      0
TCPv6                156    156   2368   13    8 : tunables    0    0    0 : slabdata     12     12      0
kcopyd_job             0      0   3312    9    8 : tunables    0    0    0 : slabdata      0      0      0
dm_uevent              0      0   2632   12    8 : tunables    0    0    0 : slabdata      0      0      0
scsi_sense_cache    2656   2656    128   32    1 : tunables    0    0    0 : slabdata     83     83      0
mqueue_inode_cache     17     17    960   17    4 : tunables    0    0    0 : slabdata      1      1      0
jbd2_transaction_s     96     96    256   16    1 : tunables    0    0    0 : slabdata      6      6      0
jbd2_journal_handle   340    340     48   85    1 : tunables    0    0    0 : slabdata      4      4      0
jbd2_journal_head    1224   1224    120   34    1 : tunables    0    0    0 : slabdata     36     36      0
jbd2_revoke_table_s   512    512     16  256    1 : tunables    0    0    0 : slabdata      2      2      0
ext4_inode_cache  140810 160487   1048   31    8 : tunables    0    0    0 : slabdata   5177   5177      0
ext4_allocation_context   128    128    128   32    1 : tunables    0    0    0 : slabdata      4      4      0
ext4_system_zone     204    204     40  102    1 : tunables    0    0    0 : slabdata      2      2      0
ext4_io_end          768    768     64   64    1 : tunables    0    0    0 : slabdata     12     12      0
ext4_pending_reservation   512    512     32  128    1 : tunables    0    0    0 : slabdata      4      4      0
ext4_extent_status  26214  26214     40  102    1 : tunables    0    0    0 : slabdata    257    257      0
mbcache             4380   4380     56   73    1 : tunables    0    0    0 : slabdata     60     60      0
userfaultfd_ctx_cache     0      0    192   21    1 : tunables    0    0    0 : slabdata      0      0      0
dnotify_struct         0      0     32  128    1 : tunables    0    0    0 : slabdata      0      0      0
pid_namespace          0      0    208   19    1 : tunables    0    0    0 : slabdata      0      0      0
UNIX                1488   1488   1024   16    4 : tunables    0    0    0 : slabdata     93     93      0
```

# Slab debug (proc/slabinfo)

```
dma-kmalloc-8k         0      0   8192    4    8 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-4k         0      0   4096    8    8 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-2k         0      0   2048   16    8 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-1k         0      0   1024   16    4 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-512       32     32    512   16    2 : tunables    0    0    0 : slabdata      2      2      0
dma-kmalloc-256        0      0    256   16    1 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-128        0      0    128   32    1 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-64         0      0     64   64    1 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-32         0      0     32  128    1 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-16         0      0     16  256    1 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-8          0      0      8  512    1 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-192        0      0    192   21    1 : tunables    0    0    0 : slabdata      0      0      0
dma-kmalloc-96         0      0     96   42    1 : tunables    0    0    0 : slabdata      0      0      0
kmalloc-rcl-8k         0      0   8192    4    8 : tunables    0    0    0 : slabdata      0      0      0
kmalloc-rcl-4k         0      0   4096    8    8 : tunables    0    0    0 : slabdata      0      0      0
kmalloc-rcl-2k         0      0   2048   16    8 : tunables    0    0    0 : slabdata      0      0      0
kmalloc-rcl-1k         0      0   1024   16    4 : tunables    0    0    0 : slabdata      0      0      0
kmalloc-rcl-512        0      0    512   16    2 : tunables    0    0    0 : slabdata      0      0      0
kmalloc-rcl-256        0      0    256   16    1 : tunables    0    0    0 : slabdata      0      0      0
kmalloc-rcl-192       84     84    192   21    1 : tunables    0    0    0 : slabdata      4      4      0
kmalloc-rcl-128     1024   1024    128   32    1 : tunables    0    0    0 : slabdata     32     32      0
kmalloc-rcl-96      5628   5628     96   42    1 : tunables    0    0    0 : slabdata    134    134      0
kmalloc-rcl-64     27174  28352     64   64    1 : tunables    0    0    0 : slabdata    443    443      0
kmalloc-rcl-32         0      0     32  128    1 : tunables    0    0    0 : slabdata      0      0      0
kmalloc-rcl-16         0      0     16  256    1 : tunables    0    0    0 : slabdata      0      0      0
kmalloc-rcl-8          0      0      8  512    1 : tunables    0    0    0 : slabdata      0      0      0
kmalloc-8k            84     84   8192    4    8 : tunables    0    0    0 : slabdata     21     21      0
kmalloc-4k           313    384   4096    8    8 : tunables    0    0    0 : slabdata     48     48      0
kmalloc-2k          1594   1696   2048   16    8 : tunables    0    0    0 : slabdata    106    106      0
kmalloc-1k          1454   1552   1024   16    4 : tunables    0    0    0 : slabdata     97     97      0
kmalloc-512         2004   2080    512   16    2 : tunables    0    0    0 : slabdata    130    130      0
kmalloc-256         1861   1952    256   16    1 : tunables    0    0    0 : slabdata    122    122      0
kmalloc-192         1722   1722    192   21    1 : tunables    0    0    0 : slabdata     82     82      0
kmalloc-128         1634   1824    128   32    1 : tunables    0    0    0 : slabdata     57     57      0
kmalloc-96          4569   4746     96   42    1 : tunables    0    0    0 : slabdata    113    113      0
kmalloc-64          8184   8960     64   64    1 : tunables    0    0    0 : slabdata    140    140      0
kmalloc-32         11136  11136     32  128    1 : tunables    0    0    0 : slabdata     87     87      0
kmalloc-16         11520  11520     16  256    1 : tunables    0    0    0 : slabdata     45     45      0
kmalloc-8          10752  10752      8  512    1 : tunables    0    0    0 : slabdata     21     21      0
kmem_cache_node      640    640     64   64    1 : tunables    0    0    0 : slabdata     10     10      0
kmem_cache           630    630    448   18    2 : tunables    0    0    0 : slabdata     35     35      0
```

SLAB中可DMA的地址

## Huge Page

- 2M or 1G
- MMU/TLB friendly
- Always PINNED and contiguous
- Designed for performance sensitive application / device

方便userspace内存管理

降低tlb miss

## Basic Linux Memory Inspection

- /proc/meminfo
- /sys/kernel/debug/memcg_slabinfo
- /proc/sys/vm/
  - https://www.kernel.org/doc/Documentation/sysctl/vm.txt
- /sys/kernel/debug/tracing/events/pagemap/
- /sys/kernel/debug/tracing/events/tlb/
- Valgrind
- mtrace
- Dmalloc
- memwatch