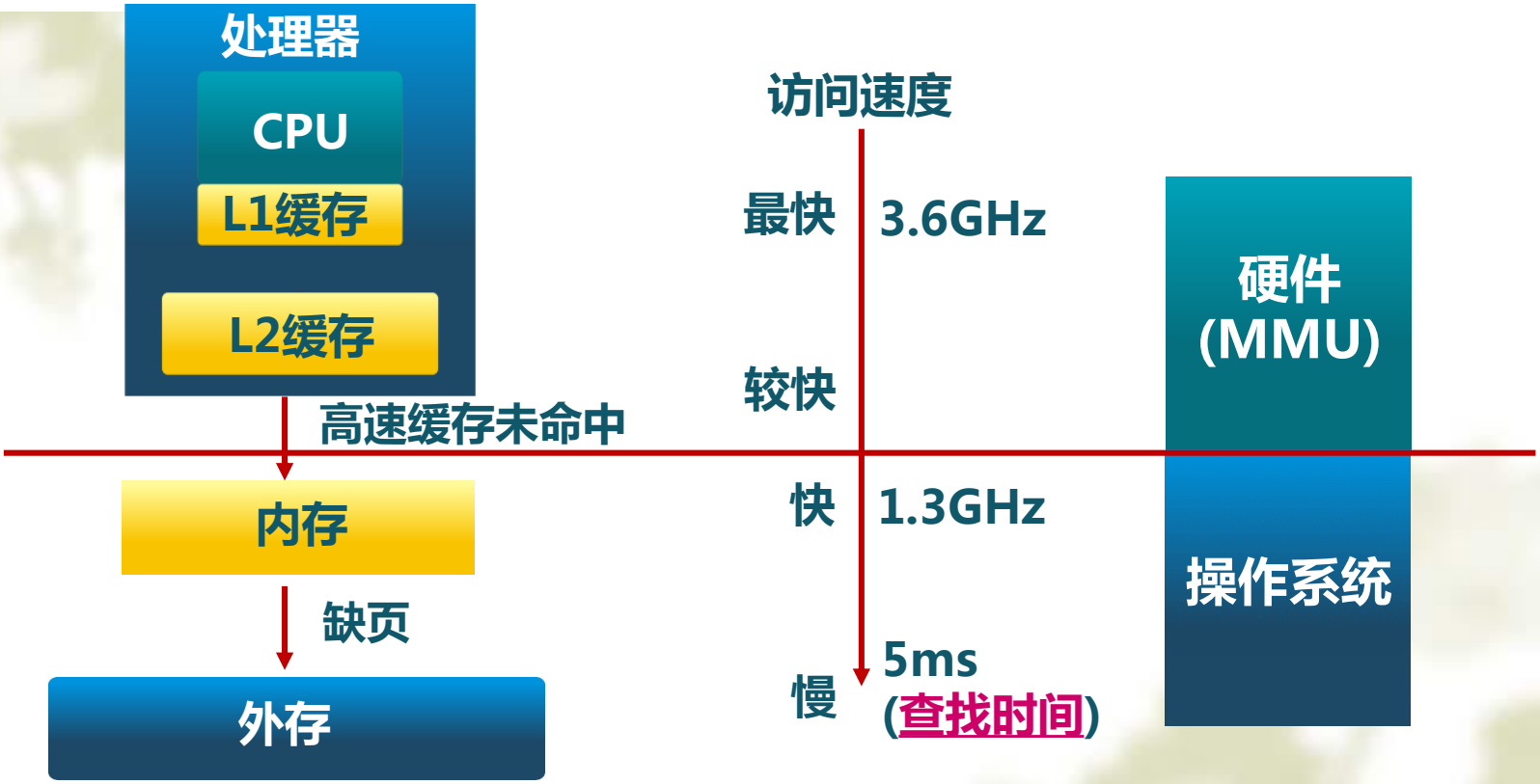
4.1 Linux内存管理机制

西安邮电大学

内存层次



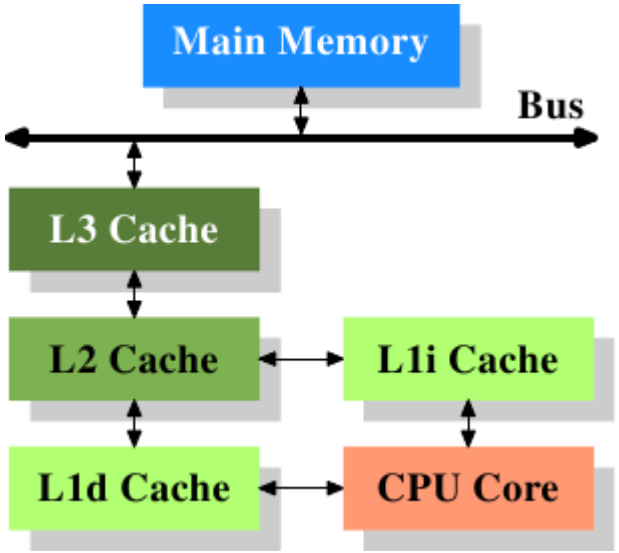
大家所熟知的是内存（RAM）和外存，尽管内存比外存速度快很多，但还是无法与CPU的速度匹配，因此CPU内部就需要更快的存储装置，这就是高速缓存（Cache）。

CPU中的Cache

我们可以通过lscpu命令查看内存的层次结构

$lscpu

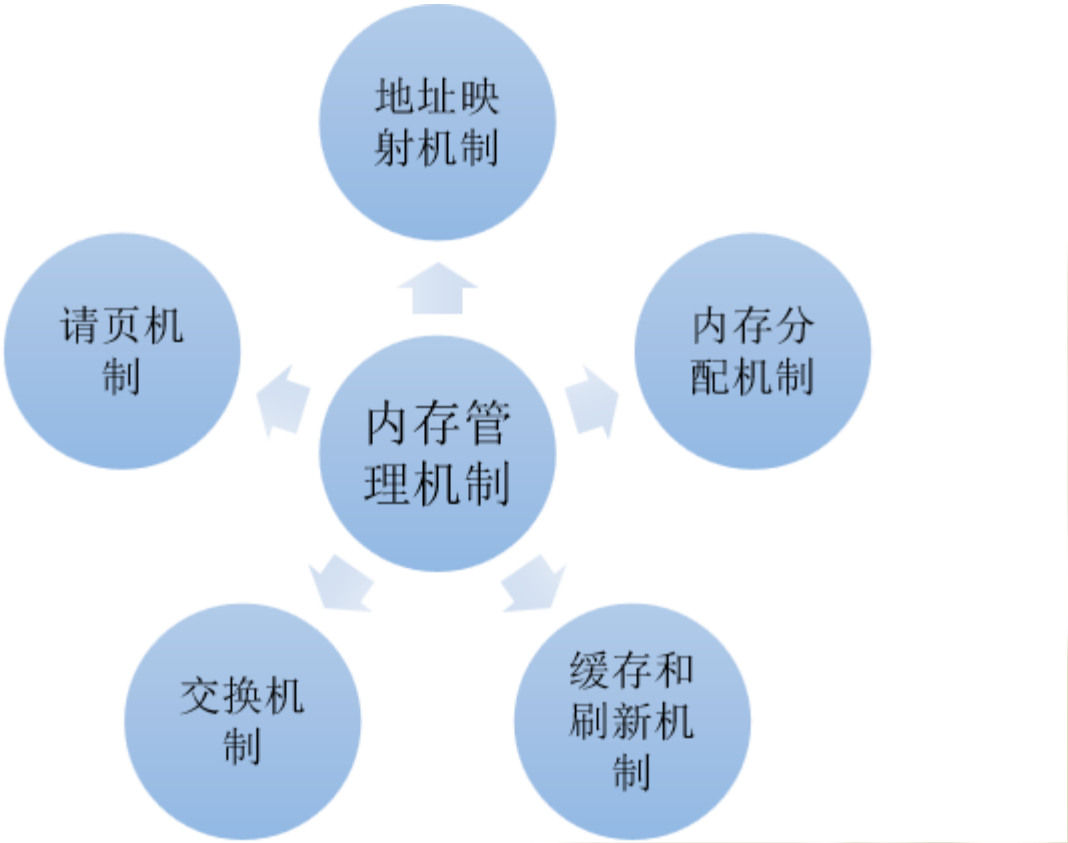
```
L1d cache:                     32K
L1i cache:                     32K
L2 cache:                      256K
L3 cache:                      3072K
NUMA node0 CPU(s):             0
```



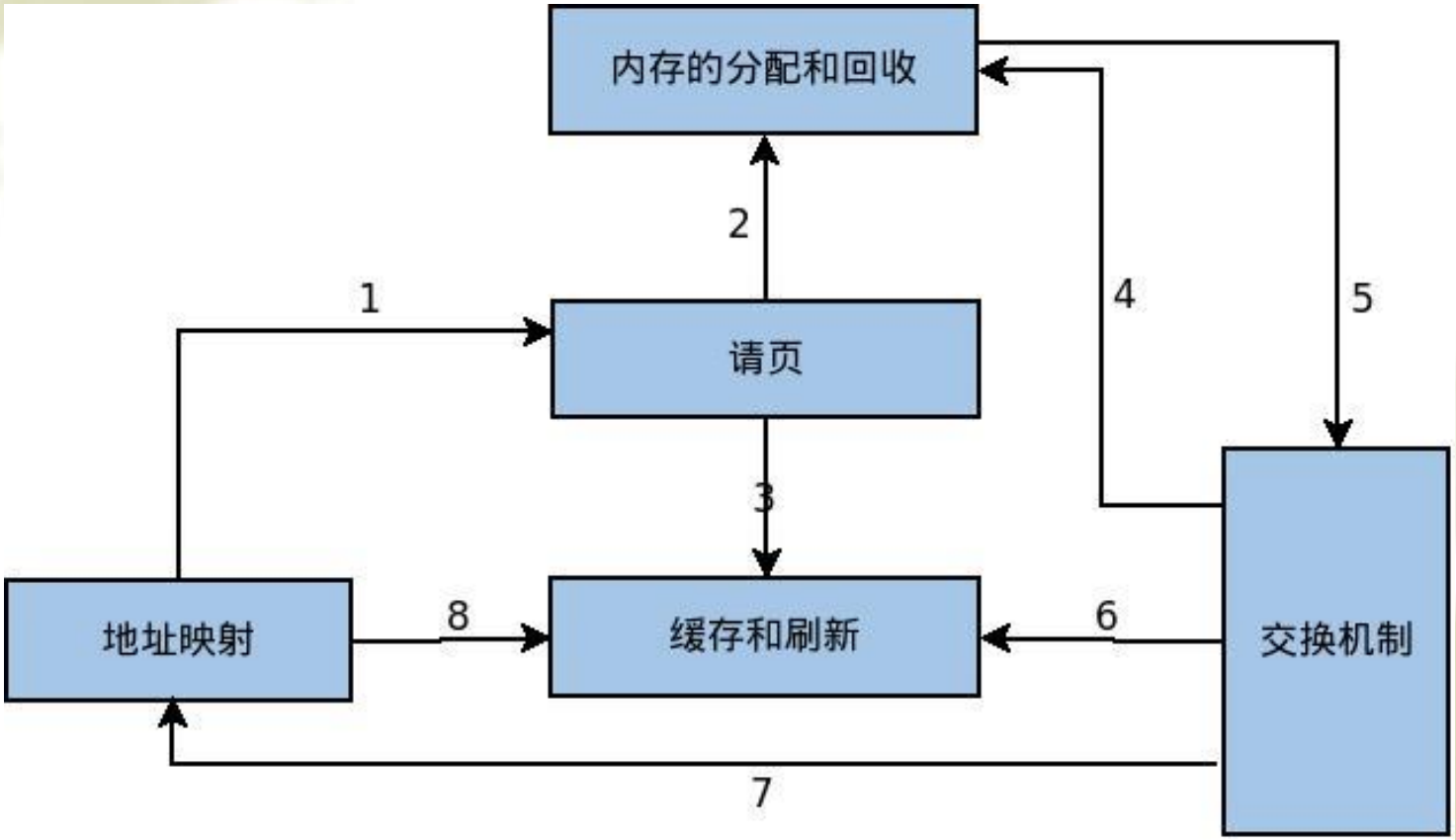
从输出结果看，在x86机器上，L1d 和L1i cache，为一级数据和指令Cache，L2,L3Cache为二级和三级Cache,大小各不同。

虚拟内存实现机制

那么Linux如何对虚拟内存进行管理，我们总结出五种机制，它们分别是：



虚拟内存实现机制关系图



虚拟内存实现机制

这个图如何体现这几种机制之间的相关配合和作用？

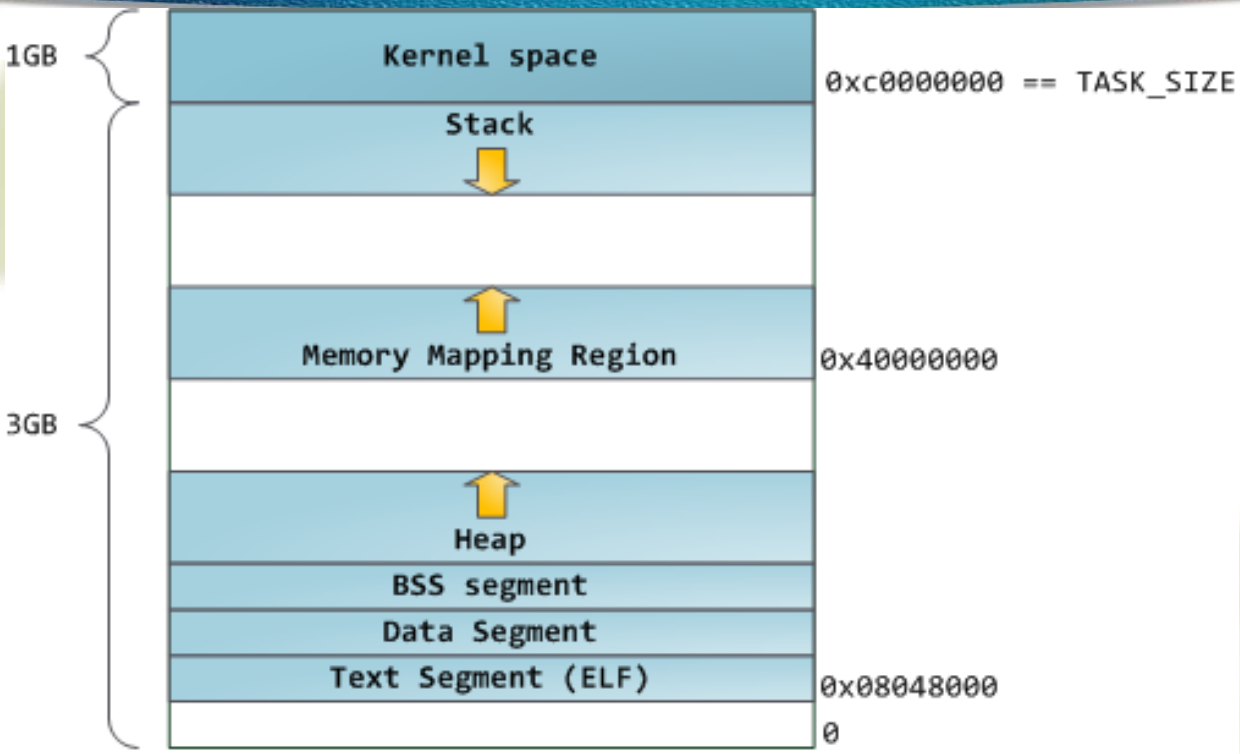
首先内核通过映射机制把进程从磁盘映射到虚拟地址空间，当进程执行时，如果发现要访问的页没有在物理内存时，就发出了页请求如图①；如果有空闲的内存可供分配，就请求分配内存如图②（于是用到了内存的分配和回收机制），并把正在使用的页记录在页缓存中如图③（于是就使用了缓存机制）。如果没有足够的内存可供分配，那么就调用交换机制，腾出一部分内存如图④⑤。另外在地址映射中要通过TLB来加速物理页的寻找如图⑧；交换机制中也要用到交换缓存如图⑥，并且把物理页内容交换到交换文件中后也要通过修改页表来映射文件地址如图⑦。

进程虚拟地址空间

每个进程通过系统调用进入内核，Linux内核空间由系统内的所有进程共享。从进程的角度看，每个进程拥有4GB的虚拟地址空间。每个进程有各自的私有用户空间（0-3GB，这个空间对系统中的其他进程是不可见的。最高的1GB内核空间为所有进程以及内核所共享。

程序一旦被执行就成为一个进程，内核就会为每个运行的进程提供了大小相同的虚拟地址空间，这使得多个进程可以同时运行而又不会互相干扰。具体来说一个进程对某个地址的访问，绝不会干扰其他进程对同一地址的访问，如图是x86 的32位地址空间示意图。

进程地址空间是如何布局的？



每个程序编译链接后形成的的二进制映像文件有一个代码段（Text）和数据段（BSS和Data）。进程运行时须有独占的堆（Heap）和栈(Stack)空间。

连接器和函数库都有自己的代码段（Text）和数据段（BSS和Data）。

进程要映射的文件被映射到内存映射区（Memory Mapping Region）。

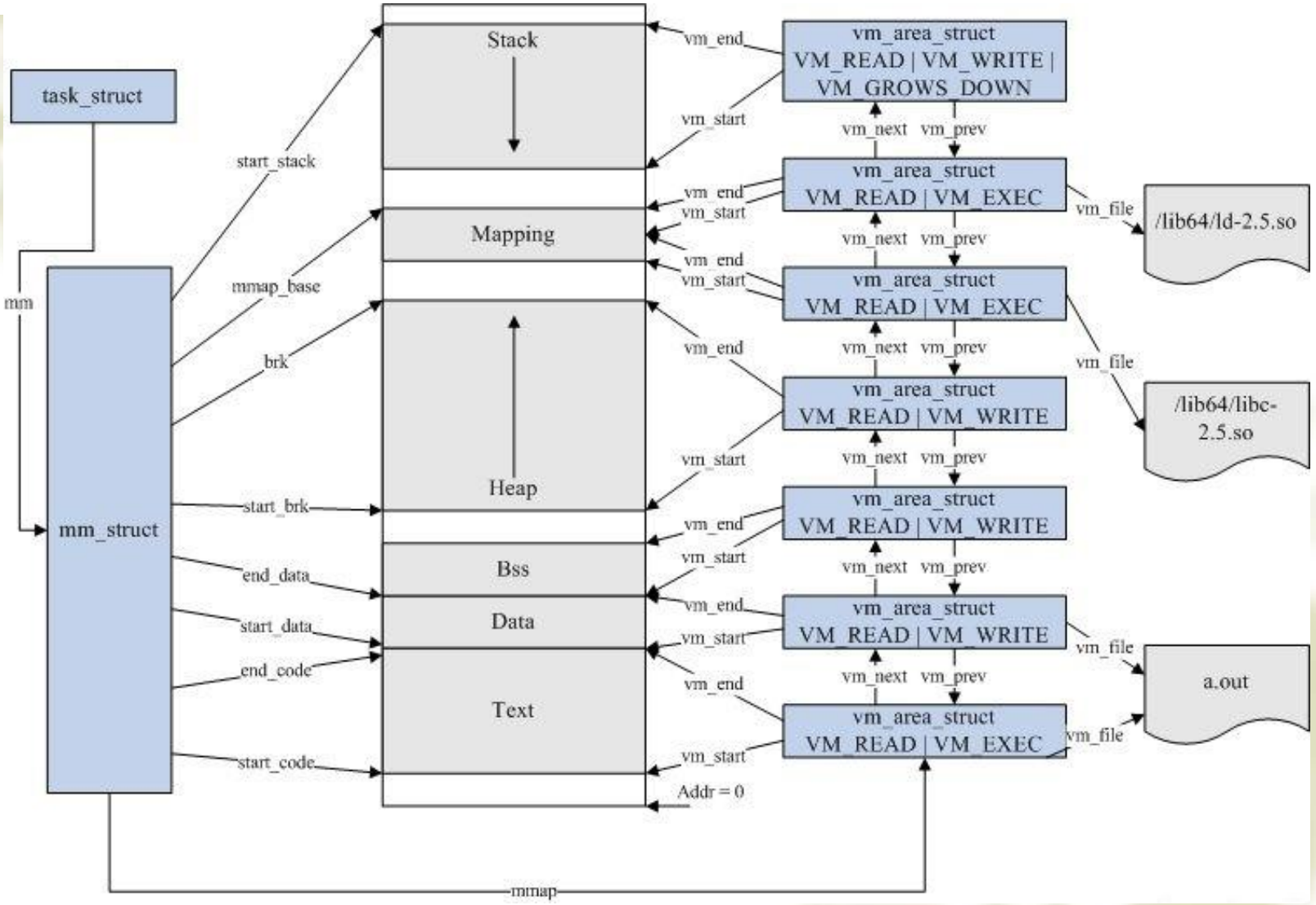
如何用数据结构描述进程的用户空间

Linux把进程的用户空间划分为若干个区间，便于管理，这些区间称为虚拟内存区域（简称vma）。一个进程的用户地址空间主要由mm_struct结构和vm_area_structs结构来描述。

mm_struct结构对进程整个用户空间进行描述。

vm_area_structs结构对用户空间中各个内存区进行描述。

如何用数据结构描述进程的用户空间



mm_struct在源代码中的部分字段

mm_struct定义在include/linux/mm_types.h

```
struct mm_struct {
    struct vm_area_struct *mmap;        /* list of VMAs */
    struct rb_root mm_rb;
    u32 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 */
   pgd_t * pgd;
   atomic_t mm_users;            /* How many users with user space? */
   atomic_t mm_count;            /* How many references to "struct mm_struct" (us
   atomic_long_t nr_ptes;            /* Page table pages */
   int map_count;                /* number of VMAs */
   unsigned long start_code, end_code, start_data, end_data;
   unsigned long start_brk, brk, start_stack;
   unsigned long arg_start, arg_end, env_start, env_end;
```

mm_struct基本字段的含义

域名	说明
mmap	指向线性区对象的链表头
mm_rb	指向线性区对象的红-黑树的根
mmap_cahce	最近一次用到的虚存区很可能下一次还要用到，因此，把最近用到的虚存区结构放入高速缓存，这个虚存区就由mmap_cache指向
pgd	进程的页目录基地址，当调度程序调度一个进程运行时，就将这个地址转成物理地址，并写入控制寄存器（CR3）
mm_user	表示共享地址空间的进程数目
mm_count	对mm_struct结构的引用进行计数。为了在Linux中实现线程，内核调用clone派生一个线程，线程和调用进程共享用户空间，即mm_struct结构，派生后系统会累加mm_struct中的引用计数
map_count	在进程的整个用户空间中虚存区的个数
mmap_sem	线性区的读写信号量
page_table_lock	线性区的自旋锁和页表的自旋锁
mmlist	所有mm_struct通过mmlist域链接成双向链表，链表的第一个元素是idle进程的mm_struct结构
start_code，end_code start_data，end_data	进程的代码段和数据段的起始地址和终止地址
start_brk，brk start_stack	每个进程都有一个特殊的地址区间，这个区间就是所谓的堆，也就是图4.5中的空洞。前两个域分别描述堆的起始地址和终止的地址，最后一个域描述堆栈段的起始地址
arg_start，arg_end env_start，env_end	命令行参数所在的堆栈部分的起始地址和终止地址； 环境串所在的堆栈部分的起始地址和终止地址
Rss，total_vm locked_vm	进程贮留在物理内存中的页面数，进程所需的总页数，被锁定在物理内存中的页数
def_flags	线性区默认的访问标志

mm_struct基本字段的含义

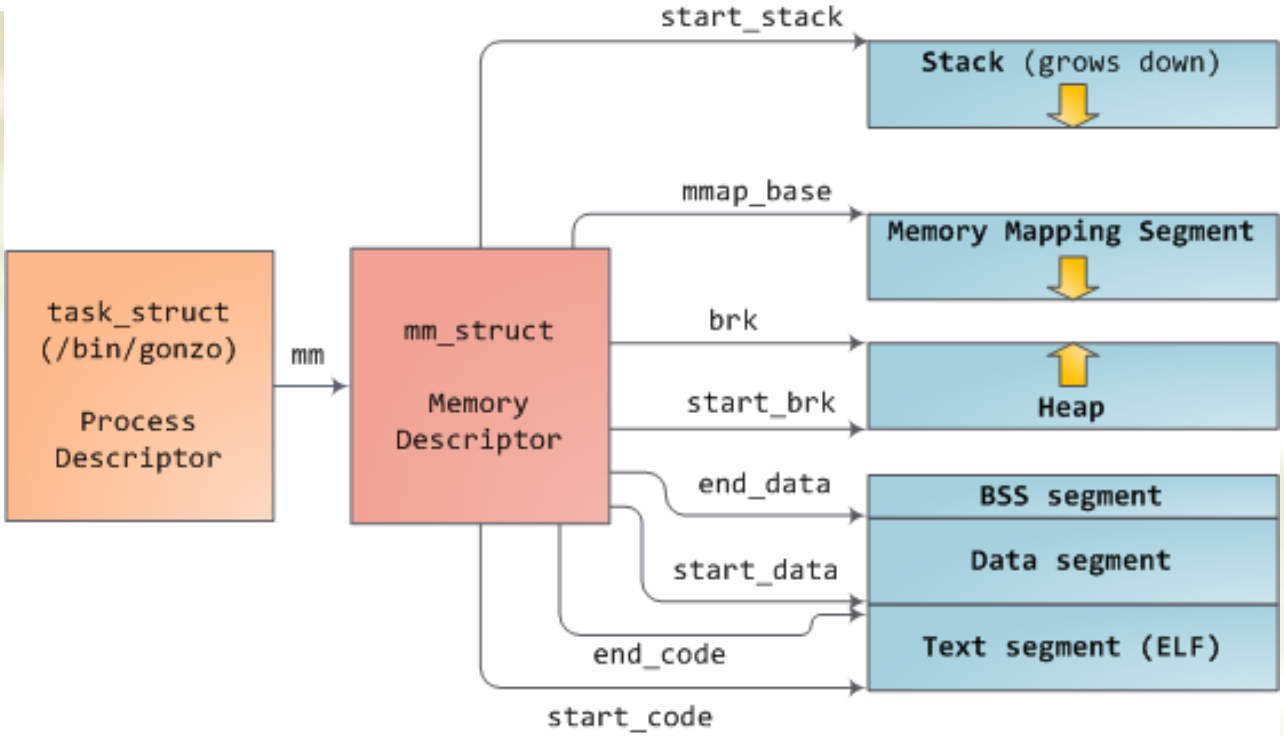
这个表中对mm_struct结构中的一些字段含义给出简要描述，在这里只提及几个重要的字段：

一个进程的虚拟空间中可能有多个虚拟区间，对这些虚拟区间的组织方式有两种，当虚拟区较少时采用单链表，由mmap指针指向这个链表，当虚拟区间多时采用红黑树结构，由mm_rb指向。

指针pgd指向该进程的页目录，当调度程序调度一个程序运行时，就将这个虚地址地址转成物理地址，并写入控制寄存器（CR3）。

其他字段就不一一介绍。

mm_struct如何描述地址空间



在进程的task_struct结构中，有一个字段mm指向mm_struct结构，mm_struct结构中各个区域的起始和结束字段描述了进程地址空间的各个虚存区（VMA）。

虚存区（VM_AREA_STRUCT） 结构

vm_area_struct描述进程用户空间的一个虚拟内存区间(Virtual Memory Area,简称VMA)，其定义如下：

```
struct vm_area_struct {
        struct mm_struct * vm_mm;
        unsigned long vm_start;
        unsigned long vm_end;
        struct vm_area_struct * vm_next;
        pgprot_t vm_page_prot;
        unsigned long vm_flags;
        struct rb_node_t vm_rb;
        struct vm_operations_struct * vm_ops;
        unsigned long vm_pgoff;
        struct file * vm_file;
        void * vm_private_data;
  ……
};
```

< >

虚存区（VMA）主要字段

虚存区的主要字段含义如下：

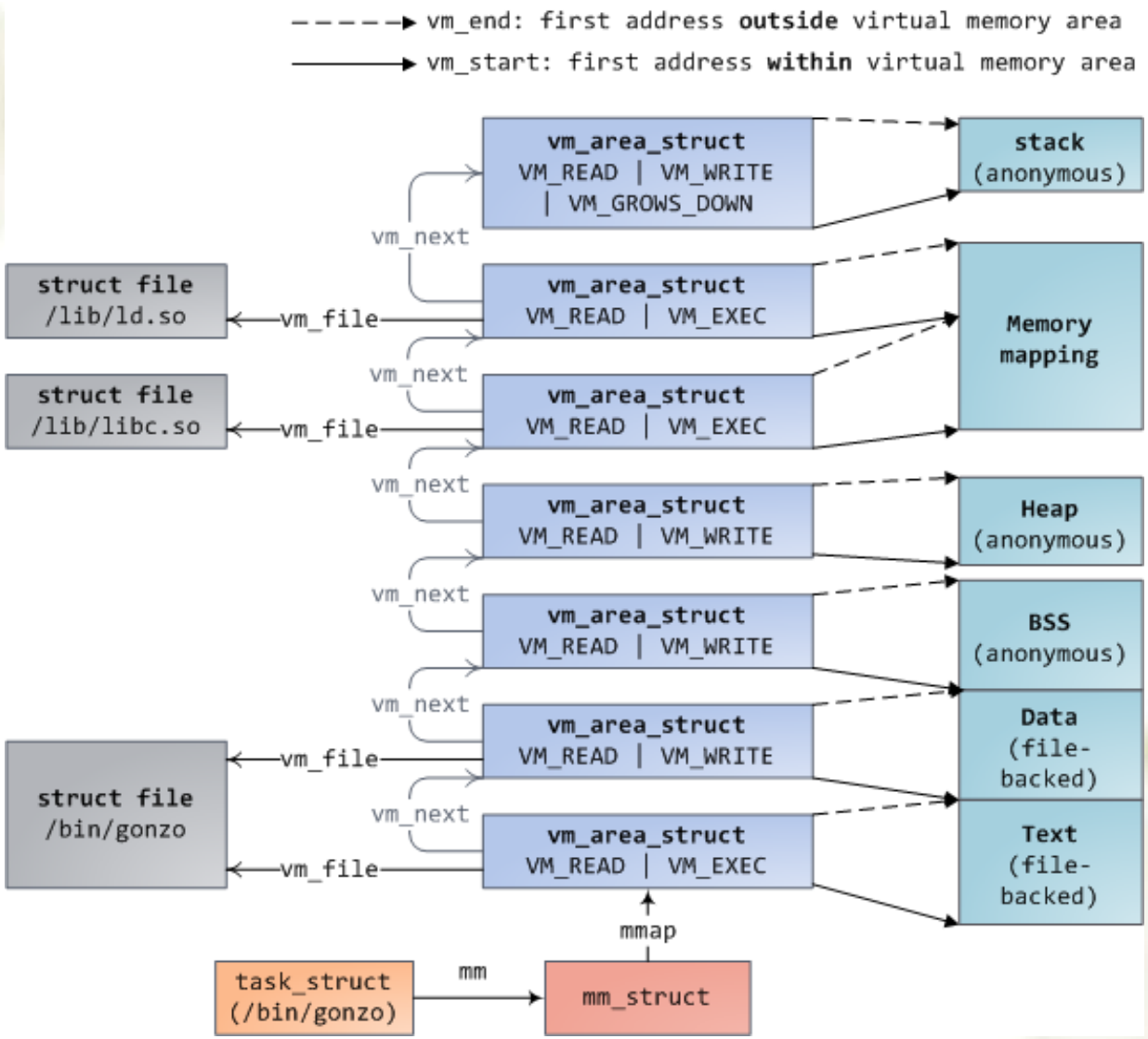
域名	说明
vm_mm	指向虚存区所在的mm_struct结构的指针。
vm_start，vm_end	虚存区的起始地址和终止地址。
vm_page_prot	虚存区的保护权限。
vm_flags	虚存区的标志。
vm_next	构成线性链表的指针，按虚存区基址从小到大排列。
vm_rb	用于红-黑树结构
vm_ops	对虚存区进行操作的函数。这些给出了可以对虚存区中的页所进行的操作。
vm_pgoff	映射文件中的偏移量。对匿名页，它等于0、vm_start或PAGE_SIZE
vm_file	指向映射文件的文件对象
vm_private_data	指向内存区的私有数据

vm_area_struct基本字段的含义

为什么把进程的用户空间要划分为一个个区间？这是因为每个虚存区可能来源不同，有的可能来自可执行映像，有的可能来自共享库，而有的则可能是动态分配的内存区，对不同的区间可能具有不同的访问权限，也可能有不同的操作。因此Linux 把进程的用户空间分割管理，并利用了虚存区处理结构（vm_ops）来抽象对不同来源虚存区的处理方法,其定义如下:

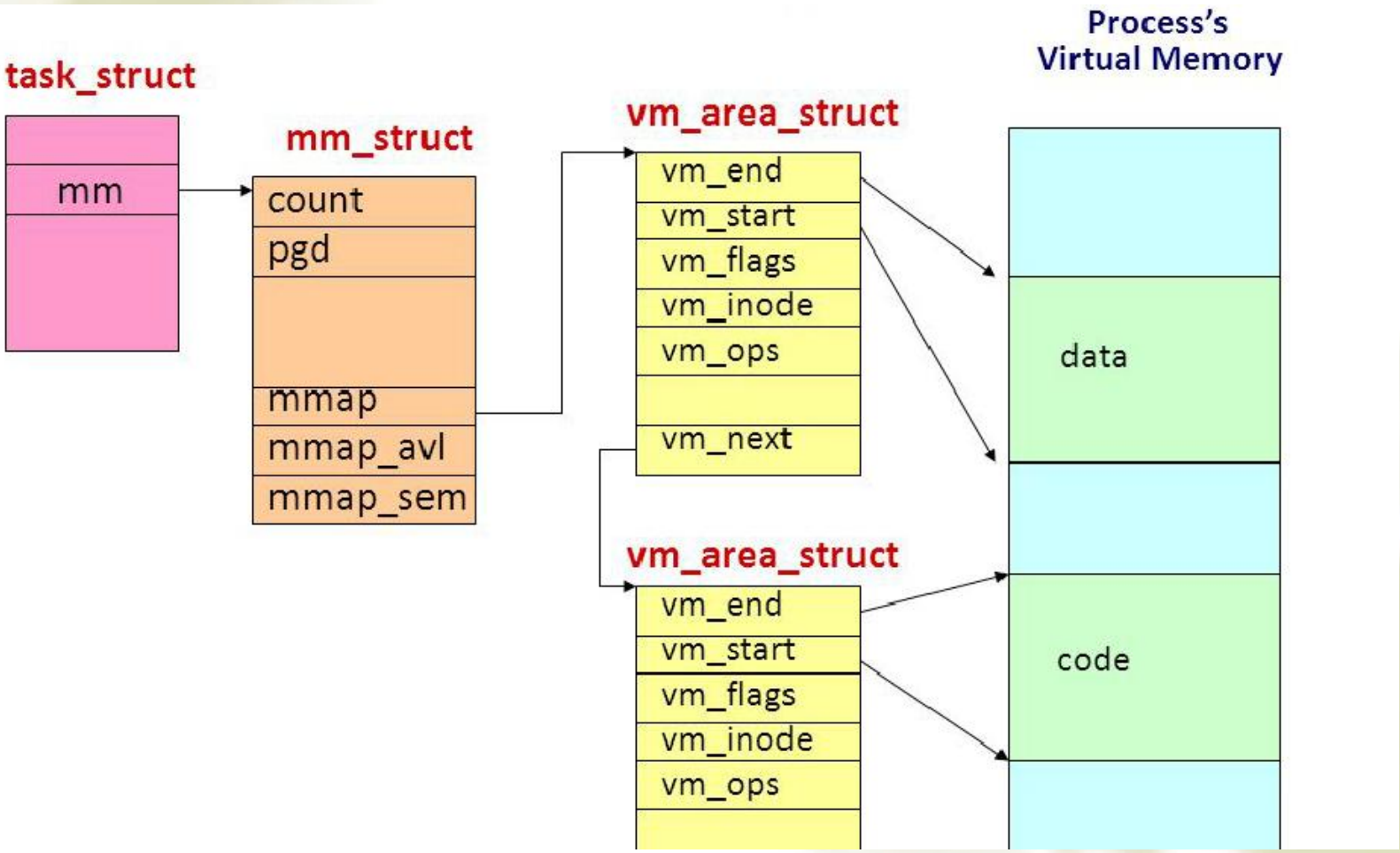
```
struct vm_operations_struct {
        void (*open)(struct vm_area_struct * area);
        void (*close)(struct vm_area_struct * area);
       struct page * (*nopage)(struct vm_area_struct * area,
       unsigned long address, int unused);
…;
};
```

内存区（VMA）如何映射到地址空间？



从这个图看出，mm_struct结构由一个个的VMA组成，进程的代码段和数据段映射到Text段和Data段，共享库（.so）映射到内存映射区。

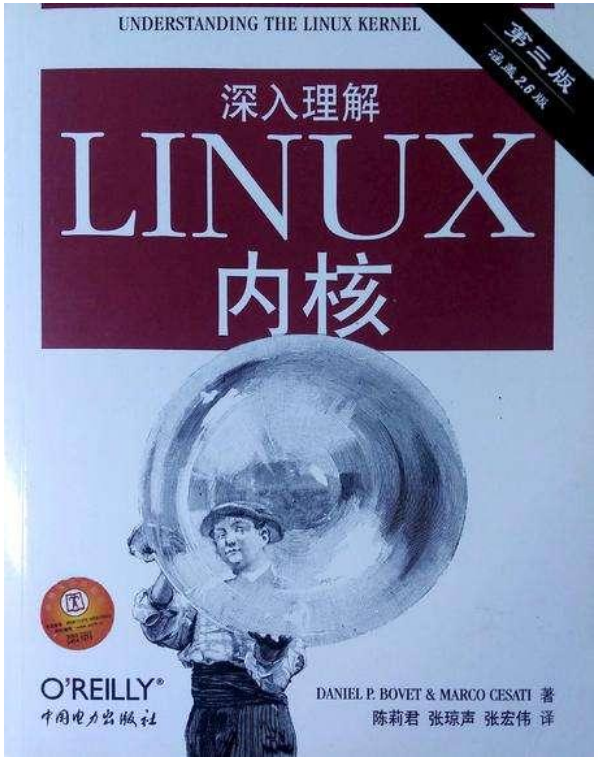
相关数据结构的关系图


task_struct
mm
mm_struct
count
pgd
mmap
mmap_avl
mmap_sem
vm_area_struct
vm_end
vm_start
vm_flags
vm_inode
vm_ops
vm_next
vm_area_struct
vm_end
vm_start
vm_flags
vm_inode
vm_ops
Process's
Virtual Memory
data
code

相关数据结构之间的关系

进程控制块是内核中的核心数据结构。在进程的task_struct 结构中包含一个mm域，它是指向mm_struct 结构的指针。而进程的 mm_struct结构则包含进程的可执行映像信息以及进程的页目录指针pgd等。该结构还包含有指向 虚存区（vm_area_struct ）结构的几个指针，每个 VMA代表进程的一个虚拟地址区间。这几个结构之间的关系如图所示。

参考资料



深入理解Linux内核 第三版第八、九章

带着思考离开

进程的地址空间（mm_struct结构和vm_area_strcut结构）到底是什么时候被映射的？

谢谢大家！

THANK YOU