Virtual Memory: System

Chapter 9.7 - 9.8

Intel Core i7/Linux 内存系统

- 支持48位（256TB）虚拟地址空间
- 支持52位（4PB）物理地址空间

Intel Core i7/Linux 内存系统

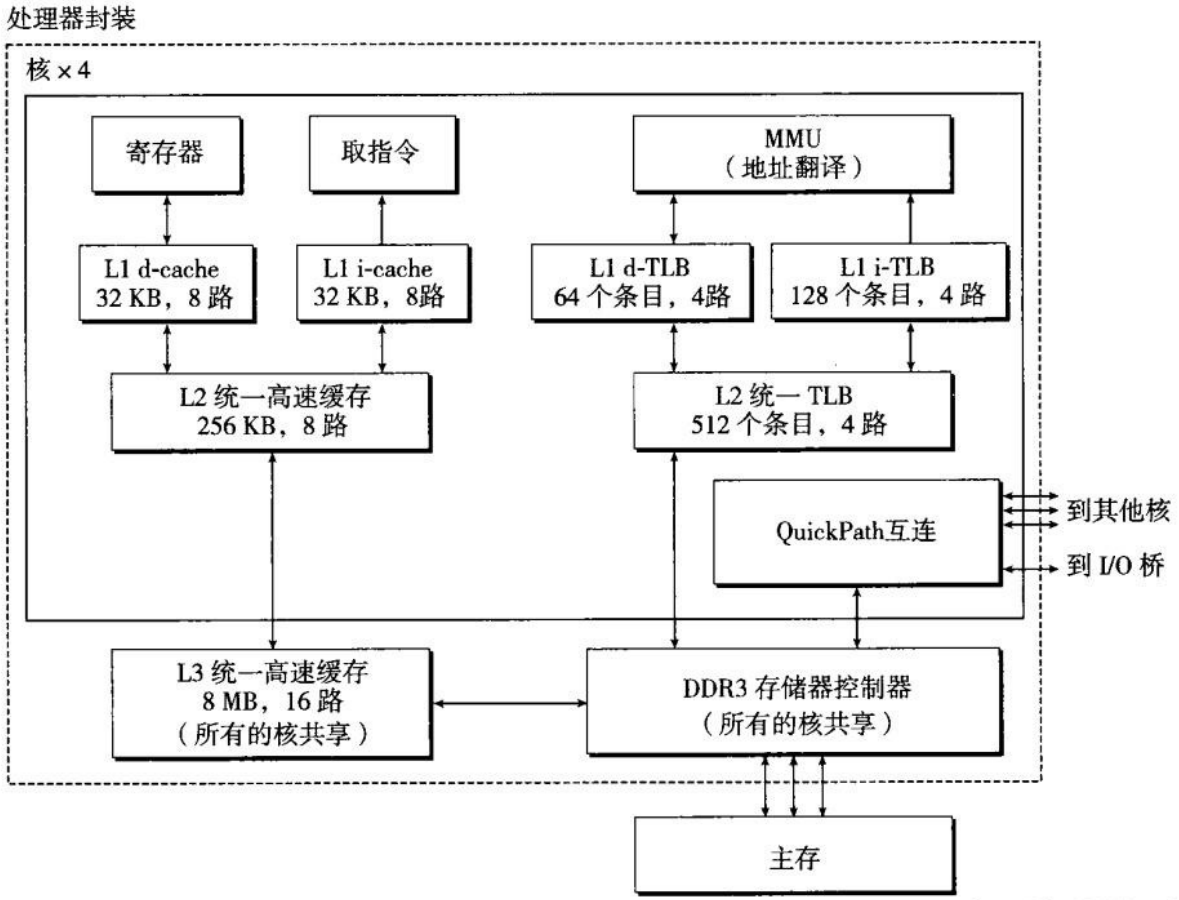


图 9-21　Core i7 的内存系统

- 四个核心
- L1,L2,L3 高速缓存
- L1,L2 TLB
- DDR3 存储器控制器

Core i7 地址翻译

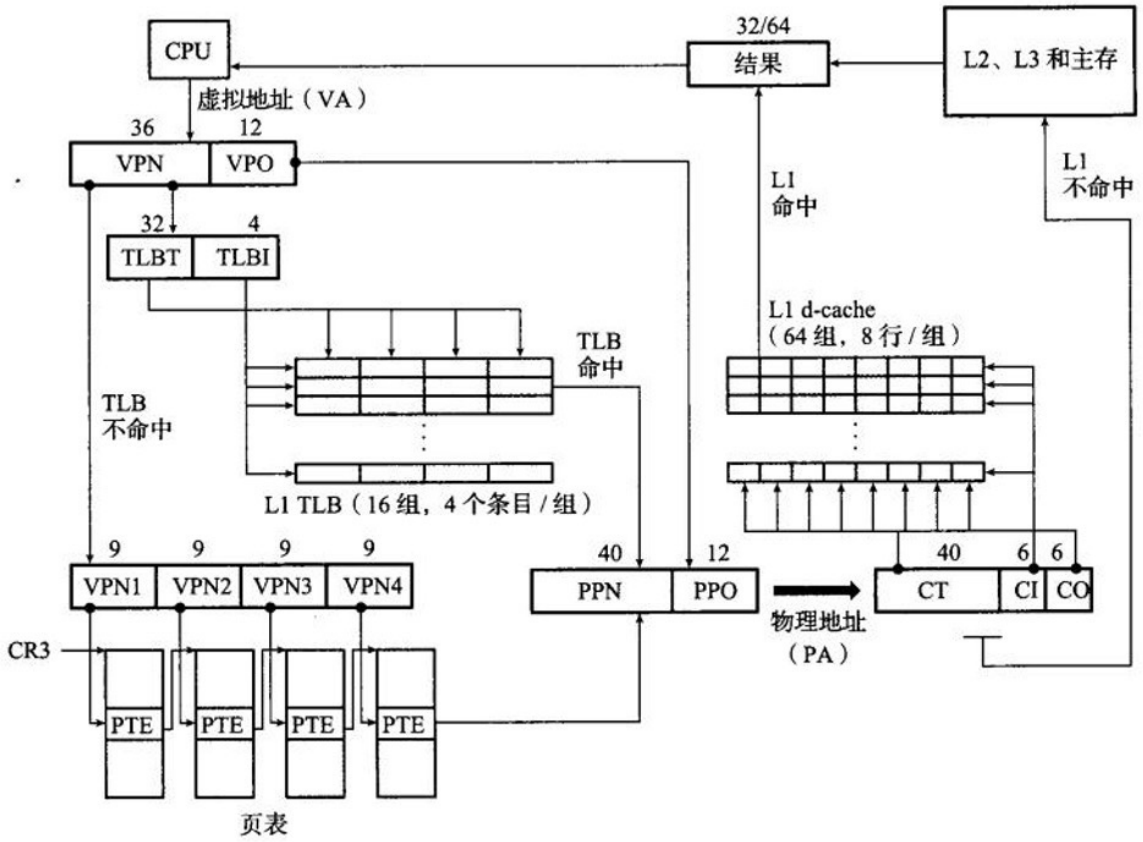
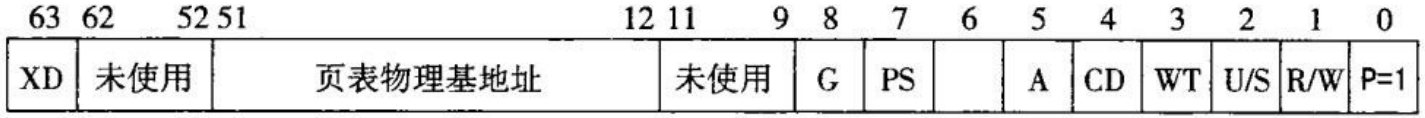


图 9-22　Core i7 地址翻译的概况。为了简化，没有显示 i-cache、i-TLB 和 L2 统一 TLB

- 四级页表
- 每个进程的页表层次均是私有的
- 已经分配了页的页表都是驻留在内存中的
- CR3（控制寄存器）指向一级页表的起始位置，其值是进程上下文的一部分

前三级页表条目格式



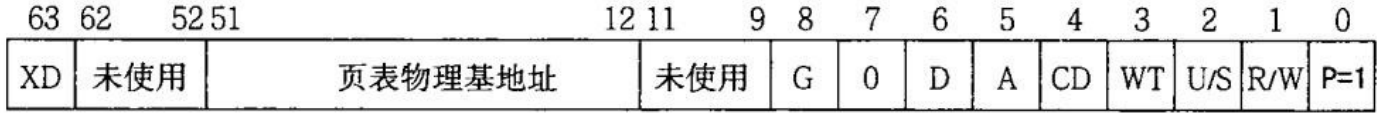
63	62–52	51–12	11–9	8	7	6	5	4	3	2	1	0
XD	未使用	页表物理基地址	未使用	G	PS		A	CD	WT	U/S	R/W	P=1
OS 可用（磁盘上的页表位置）												P=0

字段	描述
P	子页表在物理内存中（1），不在（0）
R/W	对于所有可访问页，只读或者读写访问权限
U/S	对于所有可访问页，用户或超级用户（内核）模式访问权限
WT	子页表的直写或写回缓存策略
CD	能 / 不能缓存子页表
A	引用位（由 MMU 在读和写时设置，由软件清除）
PS	页大小为 4 KB 或 4 MB（只对第一层 PTE 定义）
Base addr	子页表的物理基地址的最高 40 位
XD	能 / 不能从这个 PTE 可访问的所有页中取指令

图 9-23　第一级、第二级和第三级页表条目格式。每个条目引用一个4KB子页面

- 要求物理页表4KB对齐
- 40bit中存放下一级页表的开始处
- P 代表是否缓存到内存
- R/W 限制读写权限
- U/S 限制模式权限

最后一级页表条目格式



63	62 52	51 12	11 9	8	7	6	5	4	3	2	1	0
XD	未使用	页表物理基地址	未使用	G	0	D	A	CD	WT	U/S	R/W	P=1

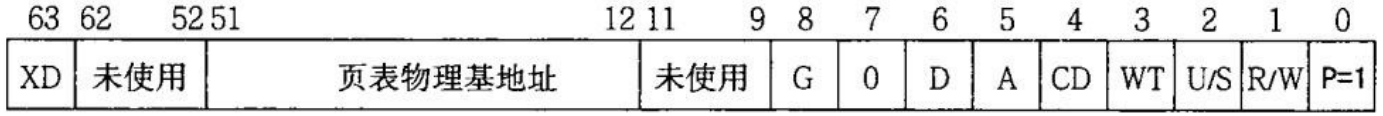
OS 可用（磁盘上的页表位置）	P=0

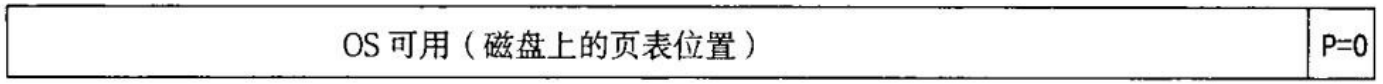
字段	描述
P	子页表在物理内存中（1），不在（0）
R/W	对于子页，只读或者读写访问权限
U/S	对于子页，用户或超级用户（内核）模式访问权限
WT	子页的直写或写回缓存策略
CD	能 / 不能缓存
A	引用位（由 MMU 在读和写时设置，由软件清除）
D	修改位（由MMU在读和写时设置，由软件清除）
G	全局页（在任务切换时，不从 TLB 中驱逐出去）
Base addr	子页物理基地址的最高 40 位
XD	能 / 不能从这个子页中取指令

图 9-24　第四级页表条目的格式。每个条目引用一个 4KB 子页

- 要求物理页4KB对齐
- 40bit中存放PPN

缺页处理程序会用到的位





字段	描述
P	子页表在物理内存中（1），不在（0）
R/W	对于子页，只读或者读写访问权限
U/S	对于子页，用户或超级用户（内核）模式访问权限
WT	子页的直写或写回缓存策略
CD	能 / 不能缓存
A	引用位（由 MMU 在读和写时设置，由软件清除）
D	修改位（由MMU在读和写时设置，由软件清除）
G	全局页（在任务切换时，不从 TLB 中驱逐出去）
Base addr	子页物理基地址的最高 40 位
XD	能 / 不能从这个子页中取指令

图 9-24　第四级页表条目的格式。每个条目引用一个 4KB 子页

- A 引用位(reference bit)，实现页替换算法
- D 修改位(dirty bit)，每次对页进行写时都会修改，告诉内核替换前是否需要写回

四级页表的翻译过程

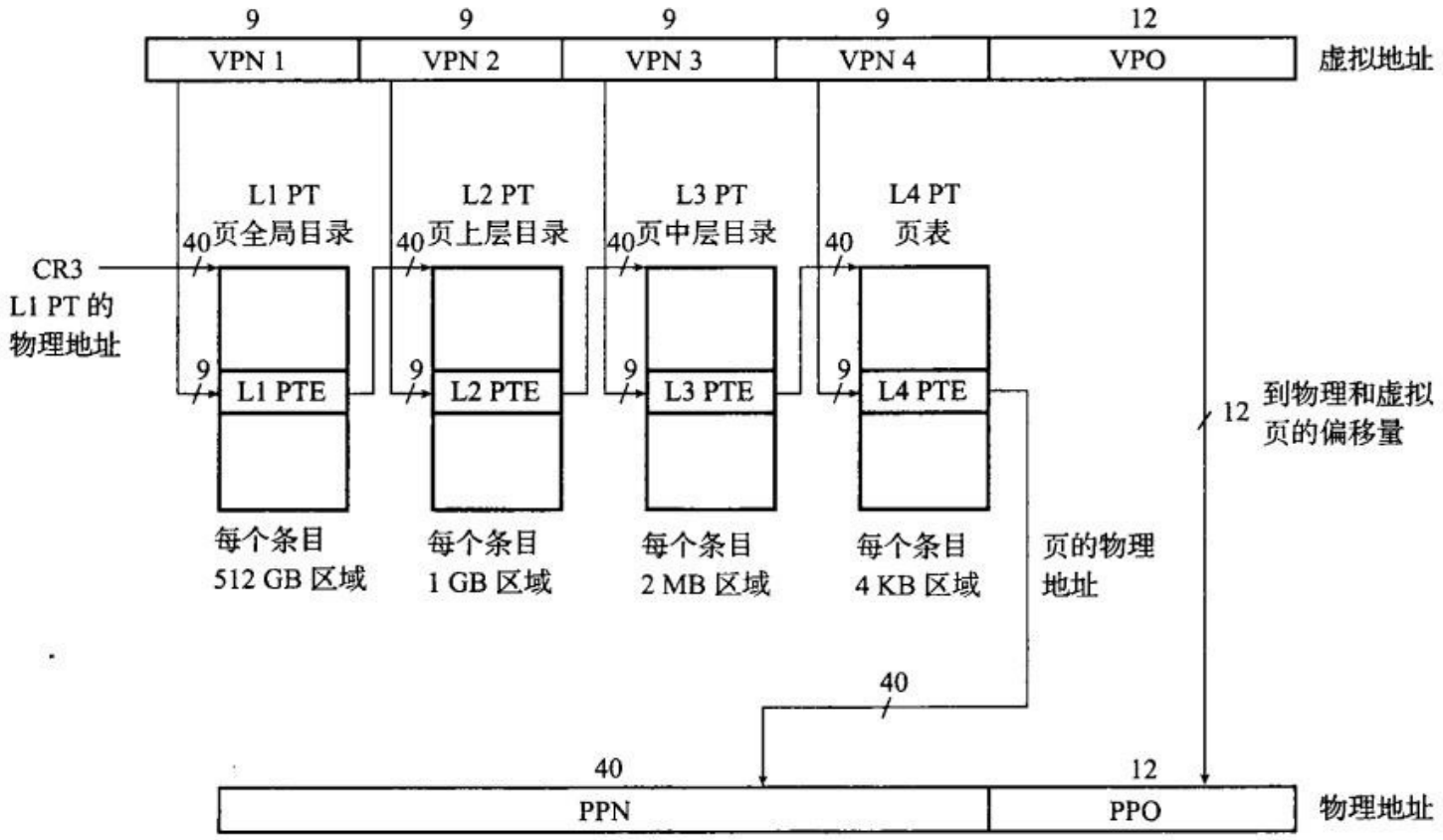


图 9-25 Core i7 页表翻译(PT：页表，PTE：页表条目，VPN：虚拟页号，VPO：虚拟页偏移，PPN：物理页号，PPO：物理页偏移量。图中还给出了这四级页表的 Linux 名字)

- 36位的VPN被划分为4个部分，分别对应一级页表，表示目标条目在页表内的偏移量
- PPO依然与VPO一样

优化地址翻译

- VPO与PPO相同（虚拟地址的低12位）
- L1高速缓存有64组，每组64个字节，6个偏移位，6个索引位，恰好是低12位
- 因此MMU在查询PPN时，L1也会利用VPO去提前查询相应的组，并读出这个组里的8个标记和相应偏移量的数据字了，当MMU得到PPN时，L1已经准备好把这个PPN与取出的8个标记位做匹配了

Linux 虚拟内存系统

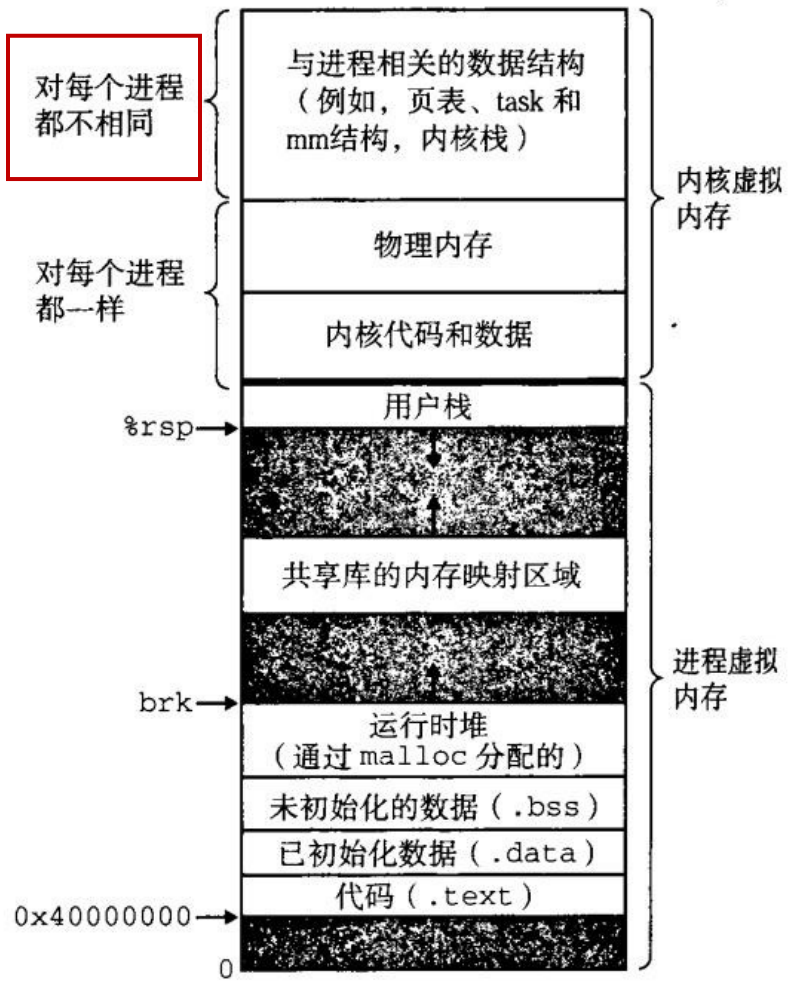


图 9-26　一个 Linux 进程的虚拟内存

勘误

- **Chapter 9: Virtual Memory**
  - p. 829, Figure 9.26. The kernel portion of the address space is identical for each process. There is no part of the kernel virtual memory that is different for each process. *Posted 05/08/2018. Godmar Bak*

地址空间的内核部分对每个进程都相同。

Task struct and mm struct

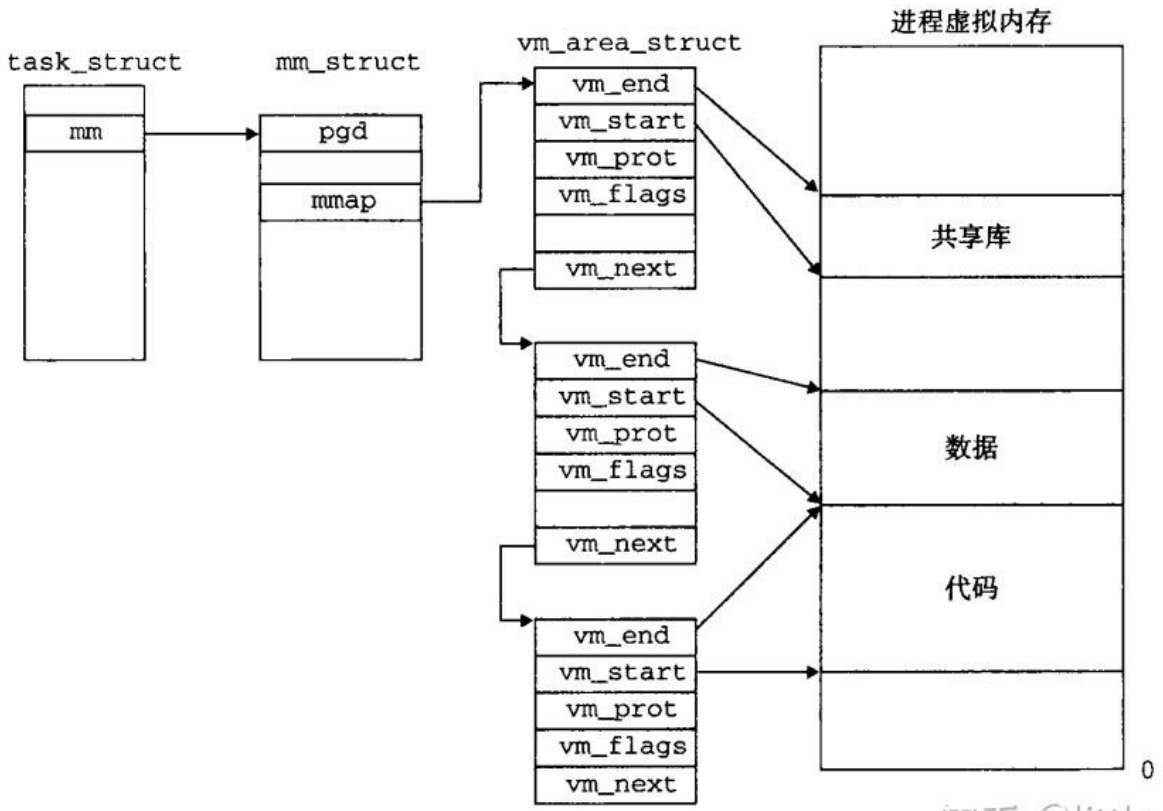


图 9-27　Linux 是如何组织虚拟内存的

内核会为系统中的每个进程维护一个单独的任务结构，在源代码中被命名为 **task_struct**

其中包含了这个进程相关的所有信息

Task struct and mm struct

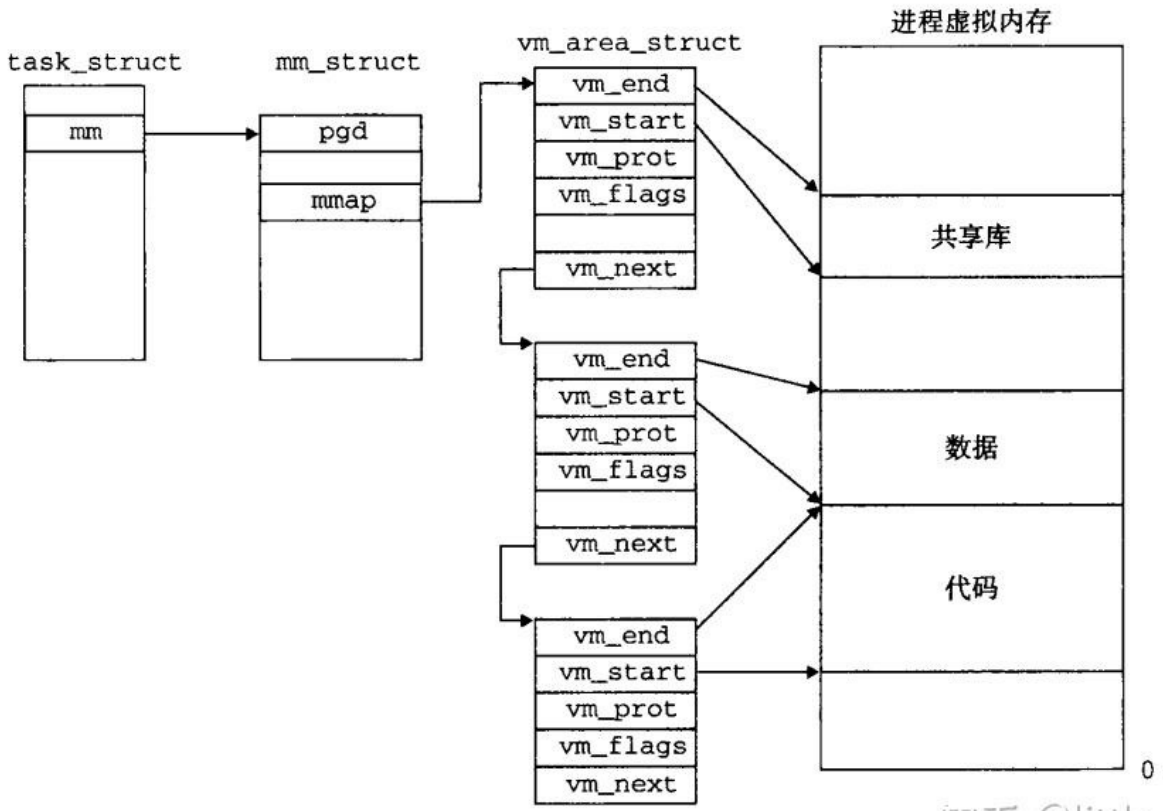


图 9-27　Linux 是如何组织虚拟内存的

`task_struct`其中的一个条目指向`mm_struct`，

它描述了虚拟内存的当前状态

`pgd`：指向第一级页表的基址

`mmap`：指向一个`vm_area_struct`的链表

Task struct and mm struct

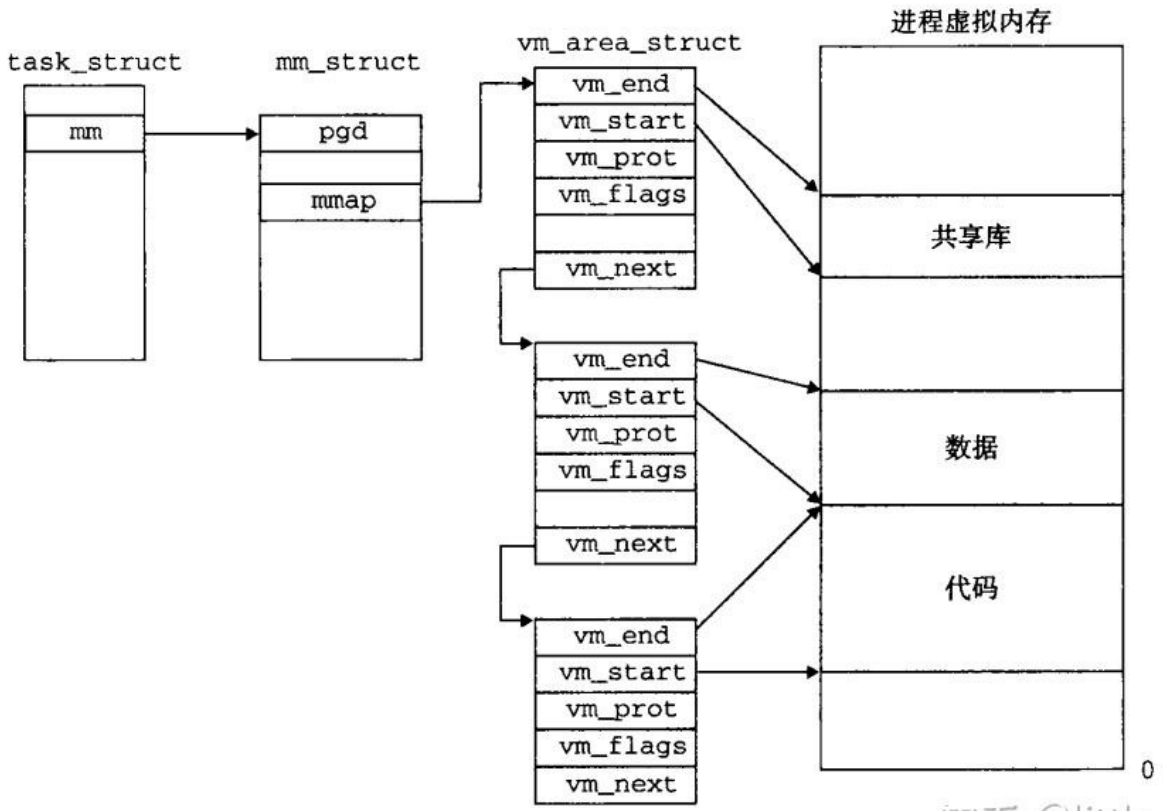


图 9-27　Linux 是如何组织虚拟内存的

每一个**vm_area_struct**都描述了当前虚拟地址空间的一个区域

- **vm_start**：起始
- **vm_end**：结束
- **vm_prot**：区域内所有页的读写权限
- **vm_flags**：页面是与共享还是私有
- **vm_next**：指向下一个区域结构

Linux 缺页异常处理

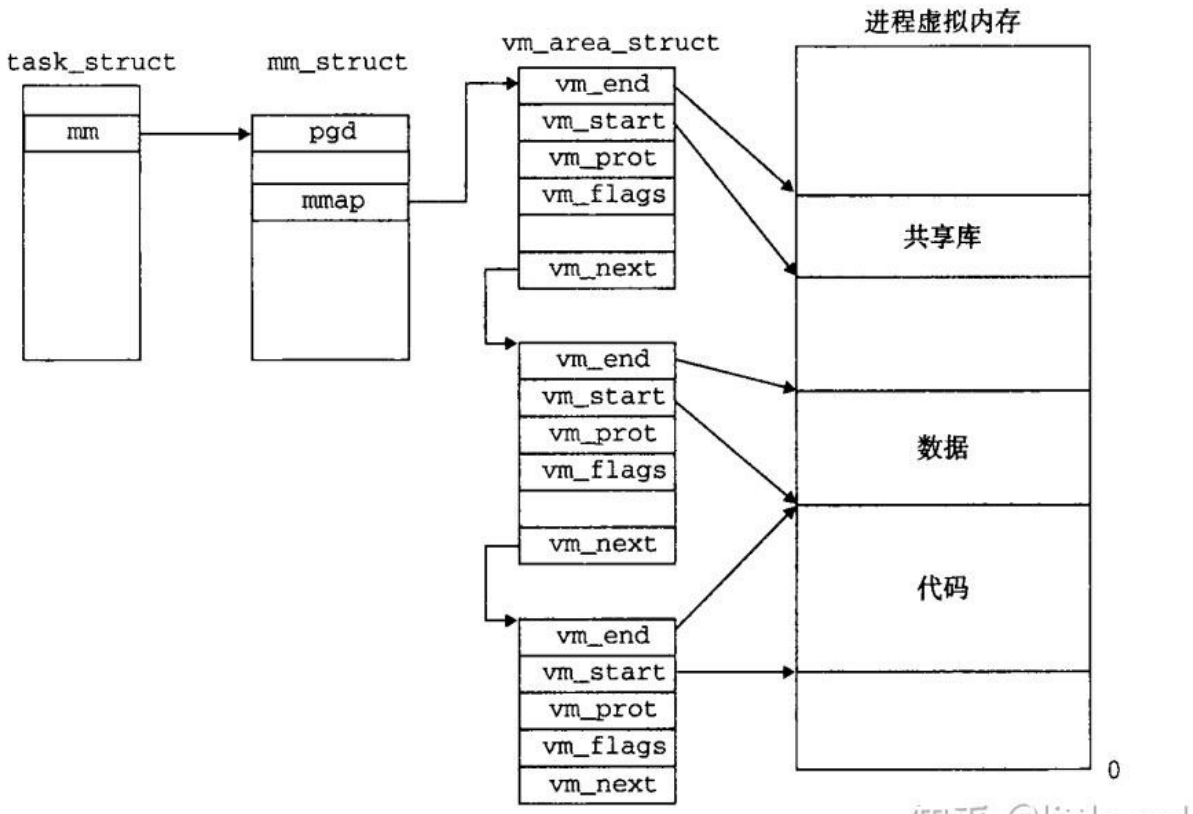


图 9-27　Linux 是如何组织虚拟内存的

步骤：

- 首先确定虚拟地址A是否合法
- 然后判断进程是否有读、写或者执行这个区域内页面的权限
- 如果前两个条件都满足，则选择一个牺牲页面，如果这个牺牲页面被修改过，那么就将它交换出去，换入新的页面并更新页表。

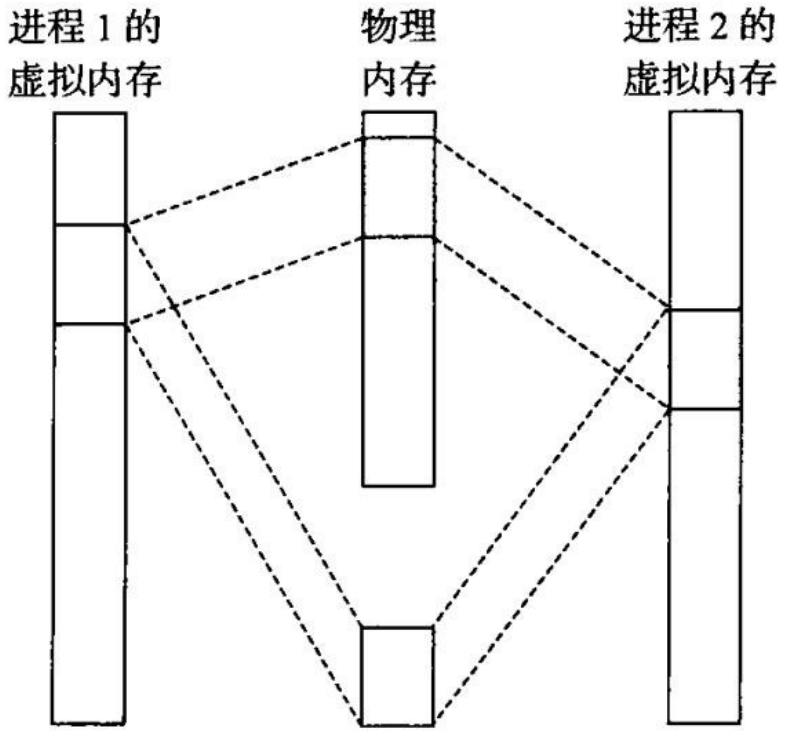
内存映射

Linux 通过将一个虚拟内存区域与一个磁盘上的对象关联起来，以初始化这个虚拟内存区域的内容

可以映射到的两种文件类型

- 普通文件
  - 文件区被分成页大小的片
  - lazy allocation
- 匿名文件
  - 由内核创建，内容只包含二进制0
  - 磁盘和内存之间并没有实际的数据传送

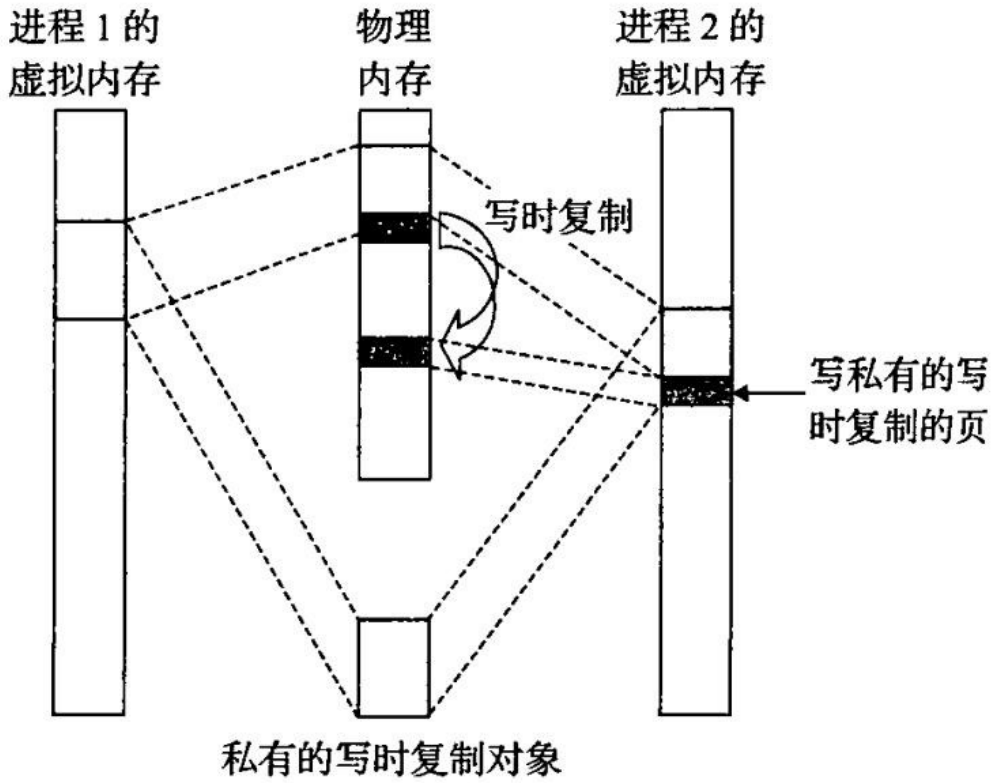
共享对象



a）两个进程都映射了私有的写时复制对象之后

- 多个进程将自己的一段虚拟内存区域映射到相同的物理内存区域，并且写操作引起的变化彼此可见
- 映射到共享对象的虚拟内存区域叫做共享区域
- 共享的判断和实现由内核完成

私有对象



b）进程 2 写了私有区域中的一个页之后

- 初始情况下也会共享同一个物理副本
- 读不会引发问题，但写彼此不可见
- 私有区域的页表条目被标为只读，并且区域结构被标记为**私有的写时复制**
- 典例：folk函数

execve 函数

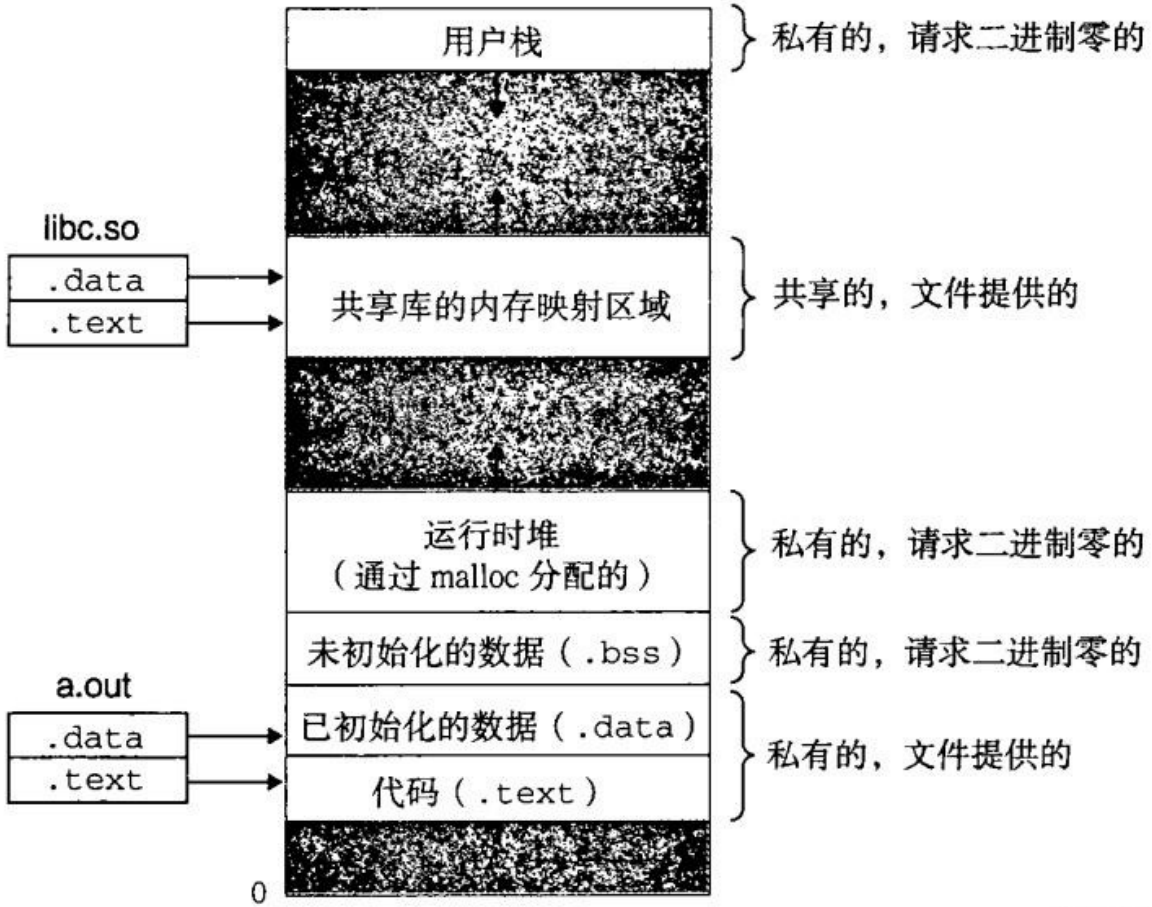


图 9-31　加载器是如何映射用户地址空间的区域的

- 删除已存在的用户区域
- 映射私有区域
- 映射共享区域
- 设置程序计数器(PC)

使用mmap函数的用户级内存映射

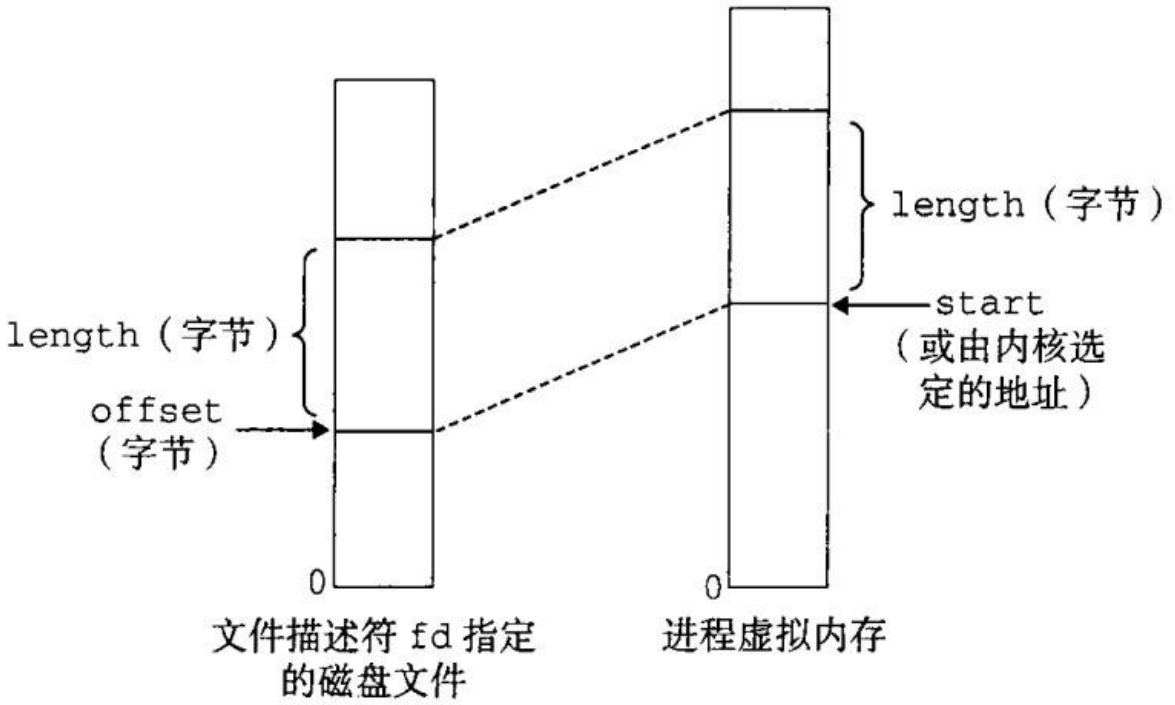


图 9-32　mmap 参数的可视化解释


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

void *mmap(void *start,
size_t length, int prot, int flags,
int fd, off_t offset)
```

prot：描述访问权限（对应vm_prot）　flags：描述被映射对象的类型（普通/匿名、共享/私有写时复制）

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

void *munmap(void *start, size_t length)
```

用来删除虚拟内存