# 虚拟内存综述

## 现状

## 架构和操作系统

# x86的虚拟内存管理

## 32-bit Paging模式

CR0.PG=1 & CR4.PAE=0

如果一个页的大小定义为4MB，则可用的内存达到1TB。如果页大小为4KB，则内存大小为4GB。

无论如何，线性地址的范围只有32位。

（似乎）MMU所使用的地址为40位。

一个线性地址（32-bit）转换成逻辑地址过程

4KB页帧：

页目录 PDE ： 39~32=0

31~12 从 CR3

11~2 从线性地址

1~0 =0 占有4个字节

页表 PTE (仅4KB模式下有PTE)： 39~32 = 0

31~12 从 PDE

11~2 从线性地址的21~12位

1~0 = 0 同样占用4字节

物理地址：39~32 = 0

31~12 从PTE

11~0 从线性地址

线性地址 ： [31~22] [21~12] [11~0]

CR3结构

PWT : CR3[3] 与cache相关，全称Page-level Write-Through 1=总是更新 0=只更新缓存

PCD ： CR3[4] Page-level cache disable是否禁用页级缓存。即是通过缓存还是直接访问内存。1=禁用 0=启用

CR3[31~12]:PDE基地址

允许的CR3首扇区：0 8 16 24…

PDE结构

31~12：PTE0高20位地址

11~8 ： ignored

7:0 (If CR4.PSE=1,this must be 0.Otherwise this entry maps a 4-Mbyte page; else is ignored)

6 : Ign (Ignored)

5:A (Accessed, set by processor, indicating that software has accessed this by this entry)

4:PCD

3:PWT

2:U/S (=1 all privileges can access, =0 only PL=0 can access this)

1:R/W (0=Read Only, could be used to protected code segment)

0:1 (Present)

典型值： [PTE0地址>>12] [0000] [0001] [1011]

PTE结构

31~12：高20位页地址

11~9: Ignored

8:G global, if set,indicate that when changing cr3 changed, prevents TLB from updating its address. Meaning that it will exists very longer

7:PAT if not knowing what it is, reserved as 0.

6 : D dirty,1=written indicate whether this page has been written. This bit is not updated by CPU.

5 : A

4 : PCD

3 : PWT

2 : U/S

1: R/W

0: 1 (Present)

典型值：[页帧0地址>>12][0001][0001][1011]

允许的PDE同样只能指向这些扇区，一个PDE指向1024个PTE，一个PTE指向4KB页面，

最大值：1个PDE=4MB，共8\*1024=8192个扇区。

x2的内核共：0xfffff（1MB,Code区域） + 0x1ffffe(2MB,Share区域) =3MB=6144个扇区

目前而言，内核使用1个PDE，其中的6144/8=768个PTE，需要恰好6个扇区映射逻辑地址

因此扩展x2的内核：扇区0作为PDE0（因此内核的栈缩小为1~3扇区）

扇区8~13作为PTE，映射为相同的物理地址。

PDE0： 1

PTEi：0 1 2 3 …767

## 临界状态：启用分页机制

为了使启用cr0.PG=1之后，EIP正常指向下一条代码，确定代码的物理地址范围为

code : 0x00000000 ~ 0x000fffff

data : 0x00000000 ~ 0x 002fffff

CR3[31~12]= Physical(PED0)/4096 (右移12位)

高10位为0，使用的是PDE的0号，则 PDE0[31~22]=Physical(PTE0)/4096

中10位为 0x00 ~ 0x2ff,

低12位 0x00 ~ 0xfff

编程实现：

//=================设置PDE0，PTE， 进入虚拟内存管理模式

\*(**int**\*)PMLoader::*PDE0\_START*= ((0x1u<<12)|0b11011);

**int** \*pte=(**int**\*)PMLoader::*PTE0\_START*;

size\_t n=(PMLoader::*DATA\_LIMIT*+1)/4096;

//0: Global,not dirty,

//1~3:not global,dirty

//4~6(IDT,GDT):global,dirty

//set all just global,not dirty

**for**(size\_t i=0;i<n;i++)

{

pte[i]= (i<<12)|0b100011011;

}

**\_\_asm\_\_** **\_\_volatile\_\_**(

"mov $0x18,%%eax \n\t"

"mov %%eax,%%cr3\n\t" //--> physical 0

"mov %%cr0,%%eax \n\t"

"or $0x80000000,%%eax \n\t"

"mov %%eax,%%cr0 \n\t"

:

:

:"eax"

);

## x2的系统编程接口

首先，系统必须有扩展自己的能力，为了达到这一点，PDE和PTE（或者至少PTE）必须位于虚拟内存的映射范围内。

设虚拟内存从线性地址到物理地址的映射函数为

vm ( l ) = p

新建一个进程的时候，进程本身拥有4GB独立寻址空间。考虑中断的情况：

原来的模型是：

改变寄存器ds的值以达到操作内核的目的

在虚拟内存之后：

必须依赖当前进程的CR3映射，由于内核的线性地址和物理地址相同，所以所有进程的CR3的PDE0都必须和内核的PDE0相同。(由于内核只有3MB，所以1个PDE会导致1MB的空间不能被进程使用，换另一种情况，这也是很常见的，如果内核不能用完4MB的整数倍，就会存在空间浪费。解决方法，在共享的PTE中，补全4MB，将其中后面的项设为global, dirty, user accessible)

在虚拟内存之下考虑进程模型，每个进程的内核空间都是相同的，它可自由使用的是非内核部分。相当于每个进程独占的4GB空间内，前面的部分总是内核代码，中断会用到前面部分的代码，其他时候不会，因此中断进入后仍然必须改变ds的值。

如何新建进程？

在物理内存中分配初始的进程大小，包括新的10个页目录（最多9\*4Mbyte=36Mbyte），

（+new）初始化CR3的值指向页目录的上一个4KB对齐

计算PDE0与4KB对齐的差值，将差值/4写入ds/cs/ss的高10位中。设这个差值为DIFF

PDE0的后1MB部分，以及PDE1~PDE9用于内存分配，新申请的小内存在PDE1中定义，所以其高10位是1+DIFF，中间的10位值为

大的内存块在数字较大的页目录中存放，所以数字较大的页目录通常也具有更多的页表，因此具有更多的物理内存。

进程初始化分页结构：

进程的CR3指向一个4KB对齐的地址（在进程空间，不与进程本身相邻），此处引入的一个问题就是如何分配一个对齐的地址？其实大可不必这样做，而是通过调整一系列的行为和值。

原来的来自GDT的ds不能再用了，因为原来的ds其CR3的index就是0，而每个进程的CR3的PDE0的index是不同的。所以，中断改变ds的行为中，ds来自于LDT的固定项。（cs,ss也需要同样的改变）

分配空间之后，前1个扇区的前10\*4个字节用于PDE保留，其中PDE0必须指向内核PDE0所指的PTE；后续的从10\*4开始之后的字节，依次存放其他信息

如果进程申请新的空间（为了避免频繁分配空间，内核一次分配4KB的空间，此后这个空间用于该进程的分配。如果进程申请4KB\*n的内存，内核则会分配4KB\*2n的分配器。每当进程的分配器不能完成分配请求时，内核会新建一个分配器，但是如果内核也不能完成请求，就发出内存不足的信号），内核将申请到的物理地址映射为有效的虚拟地址，然后返回。映射为有效的具体过程是：确定4KB的个数n，查找连续的n个未用的PTE（=0，或者使用associated memory），写入PTE的值。分配空间的同时还需要分配页表，一个4KB通常就需要1个页表）

一个问题的辨析：内核中有大量使用绝对地址的程序，最主要的是SimpleMemoryManager，它所依赖的地址必须是可访问的，而这个地址已经被分配给MemoryManager，基本上它们是不可以再改变的。考虑到内核只有 0~3MB，也就是说地址范围为0x0 ~ 0x2fffff 。 假设内核将使用的ds基地址是 [i][j][k=0]将其中的任意一个值加上,譬如 [0x4][0x24][0],

基地址是 : 0x1024000

PDE0允许的最大地址是 0x13fffff。

因此进程中内核使用的ds值就会依据PDE0来设置。

第二个问题：内核的内核部分已经通过虚拟内存可读写，内核的进程部分呢？如何向进程部分读写数据？

不要求内核存储其所有的页映射。假设现在需要访问进程空间的 （物理地址）A地址，共N字节；因为进程空间的ds基地址是确定的，在进程的上下文中，先新建一个PDE项，计算与PDE0的字差距，如果差距为0，则检查N，如果N<=1MB，则直接通过线性地址访问访问。

如果N>1MB,则剩余的M=N-1Mbyte，则在访问1MB之后的空间时，使用M作为参数重新访问。如果差距大于0，如果进程的PDEi存在空闲的PTE，则计算N1=N/4KB，计算需要使用的PTE个数，将物理地址写入这些PTE，然后返回线性虚拟基地址（包括高10位，中10位，低12位为0）。

问题3：给定PDE0的首地址，如何确定其CR3的高20位和线性地址高10位？

CR3[31~12] = A\_PDE0[31~12]

L[31~22] = A\_PDE0[11~0]

问题4：给定物理地址P，已知CR3, 在物理地址P0处设置第一个PDE，P1处设置第一个PTE，应当怎样设置P0,P1， 使得P拥有可访问的线性地址L？

PDE index = (P0 – (CR3.base <<12) )>>2

PTE index = (P1 – (PDE0.base<<12))>> 2

且令 P0 处 .base = P1>>12 & 0xfffff

P1处 .base = P >> 12 & 0xfffff

L = (PDE index << 22) | (PTE index << 12) | 0

函数接口 (size\_t P , size\_t P0, size\_t P1, CR3 cr3,size\_t &L)

{

}

辅助接口： getIndex(size\_t phyaddr, size\_t reg)

{

return (phyaddr – (reg & 0xfffff000))>>2;

}

getHighBase(size\_t phyaddr)

{

return phyaddr & 0xfffff000;

}

问题5：在虚拟内存模式下，给定物理地址P和一个小于4KB的范围R，如何访问P？给出一个有效的线性地址。

返回一个ds值即可。

如果存在可用的PTE，则在当前进程中，申请一个可用的PTE,物理地址为P1, P1.base = P >> 12 & 0xfffff;

然后申请一个ldt项，填写基线性地址为 : [PDE index] [PTE index] [0], 返回ldt选择符的index。

如果没有可用的PTE，先申请一个新的PDE，地址分别为P0,P1 (P0,P1可直接访问)，设置P0.base = P1,

关于PDE manager的设置：

PDE manager的的每一项都对应一个PTE manager。

class PDEManager:public AssociatedMemoryManager<PDE,1>{

//增加一个对应的PTEManager数组

PTEManager\* ptemans;

}

typedef AssociatedMemoryManager<PTE,1> PTEManager;

关于内存分配：

其实虚拟内存之后不要求实际的内存连续，所以虚拟内存的分配器会使用离散分配来完成工作。

# 附录

CR0结构

#### CR0

|  |  |  |
| --- | --- | --- |
| **bit** | **label** | **description** |
| 0 | pe | protected mode enable |
| 1 | mp | monitor co-processor |
| 2 | em | emulation |
| 3 | ts | task switched |
| 4 | et | extension type |
| 5 | ne | numeric error |
| 16 | wp | write protect |
| 18 | am | alignment mask |
| 29 | nw | not-write through |
| 30 | cd | cache disable |
| 31 | pg | paging |