**2021/04/30_Windows32位内核_第8课_内存管理(分页)以及PAE**

**笔记本：** Windows32位内核
**创建时间：** 2021/5/1 星期六 8:19
**作者：** ileemi


- 分页
- 页目录表、页表
- WinDbg 命令
- 跨进程读写内存
- PAE
  - 32位CPU开启PAE
  - 64位CPU开启PAE


# 分页

> 在分页基址中，分页涉及到页**目录表（PDE）、页表（PTE）**两张表。
> 分页机制开启标志为：CR0寄存器的最高位 PG = 1 ，分页大小以4k最为常见。

# 页目录表、页表

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Address of page directory[1] | | | | | | | | | | | | | | | | | | | | Ignored | | | | | | | PCD | PWT | Ignored | | | CR3 |
| Bits 31:22 of address of 4MB page frame | | | | | | | | | | Reserved (must be 0) | | | | | | | | | Bits 39:32 of address[2] | PAT | Ignored | | G | 1 | D | A | PCD | PWT | U/S | R/W | 1 | PDE: 4MB page |
| Address of page table | | | | | | | | | | | | | | | | | | | | Ignored | | | | 0 | Ign | A | PCD | PWT | U/S | R/W | 1 | PDE: page table |
| Ignored | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | PDE: not present |
| Address of 4KB page frame | | | | | | | | | | | | | | | | | | | | Ignored | | | G | PAT | D | A | PCD | PWT | U/S | R/W | 1 | PTE: 4KB page |
| Ignored | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | PTE: not present |

Figure 4-4. Formats of CR3 and Paging-Structure Entries with 32-Bit Paging

格式（主要记低3位）：

- PS（第7位）：为1，页目录表有4MB；为0，页目录表有4KB。
- P（第0位）：存在位。为1，页目录表存在；为0，页目录表不存在。
- A（第5位）：是否访问过。
- R/W（第1位）：读写位，用来表示内存属性。为0，可读可执行（RE）；为1，可读可写可执行（RWE）。OD中的内存执行断点就是修改P标志位（改为无效的内存），执行代码时，使其产生异常。

- U/S（第2位）：用来区分特权级（Ring0获取Ring3）。为0，系统级页（S）；为1，用户级页（U）。
- PAT（第12位）：如果支持PAT，则间接确定用于访问此条目引用的4-MB页面的内存类型（请参见第4.9.2节）；否则，则保留（必须为0）

页目录表为4MB时，表的寻址方式如下：低22位视为偏移，高10位进行页面目录查表

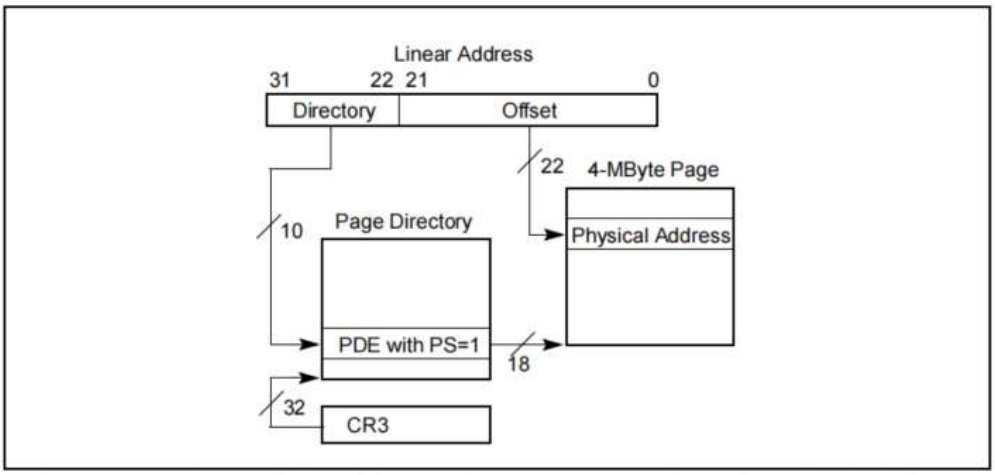
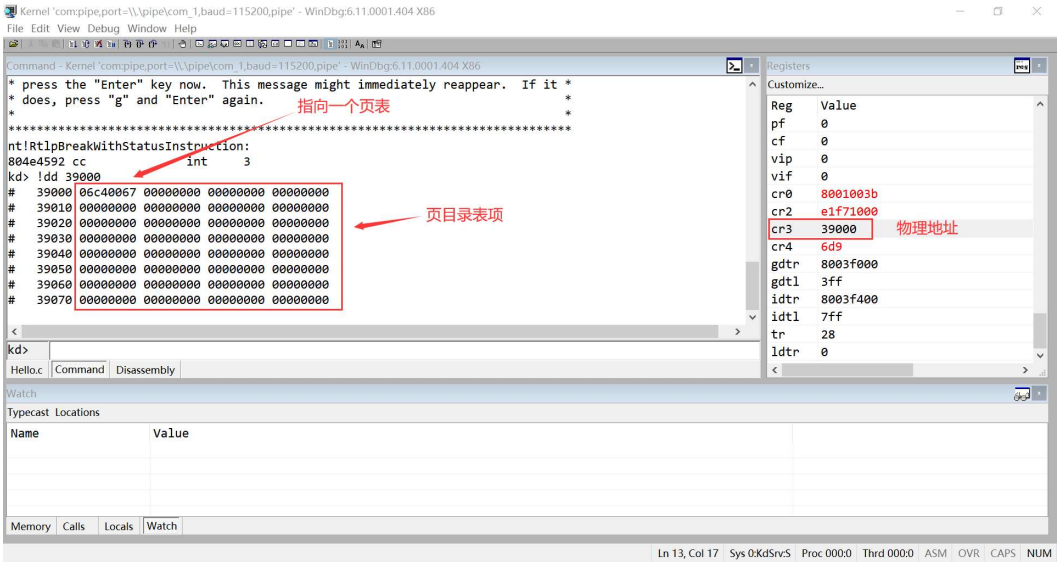



Figure 4-3. Linear-Address Translation to a 4-MByte Page using 32-Bit Paging

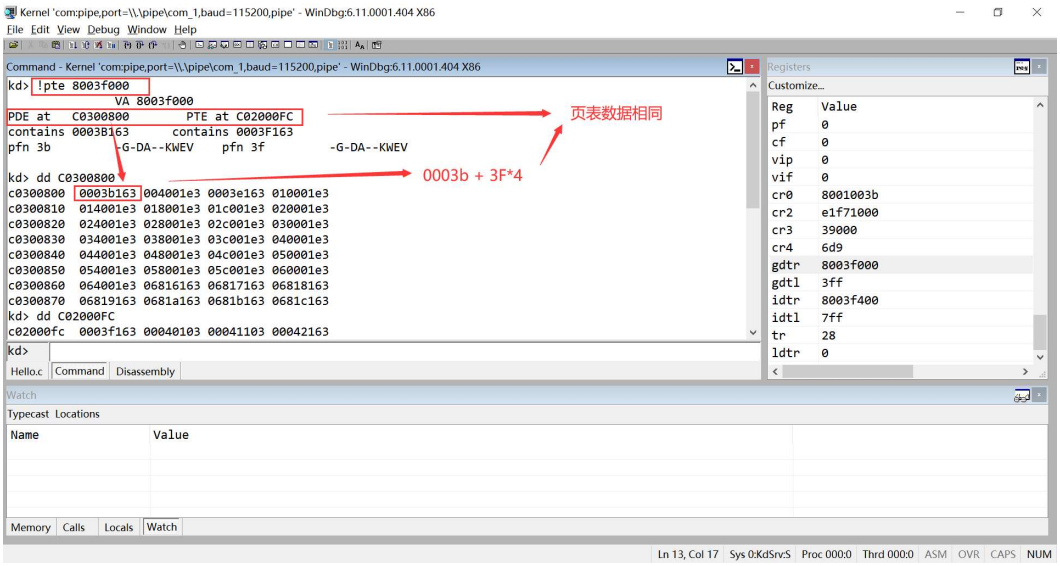
高18位（扩展为32位） + 低22位

页目录表、页表的识别：

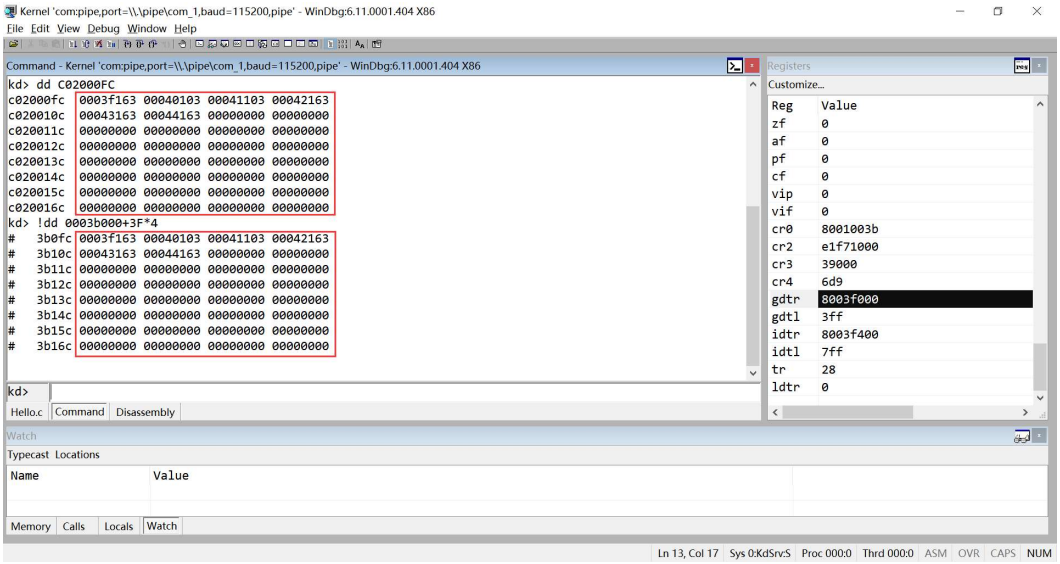



> 输入 "!dd addr L400" 命令，可显示当前进程中的所有页目录表项，从上图中可以看出当前进程第一个页表项有值，为"06c40067"。

输入 "!pte addr" 可以由windbg自动计算出其对应的页目录表以及页表的地址，如下图所示：



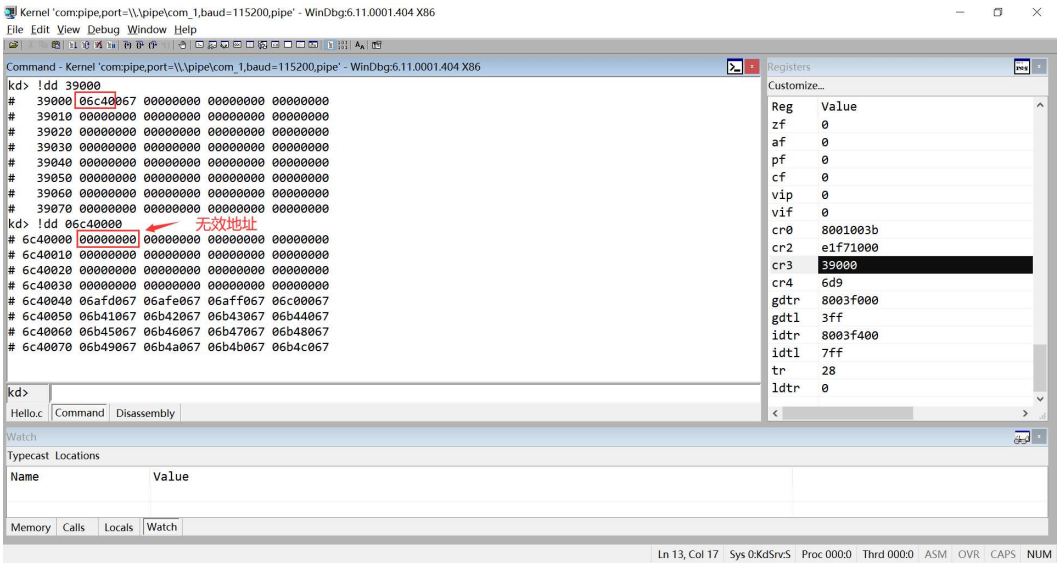

物理地址：0x0003F000



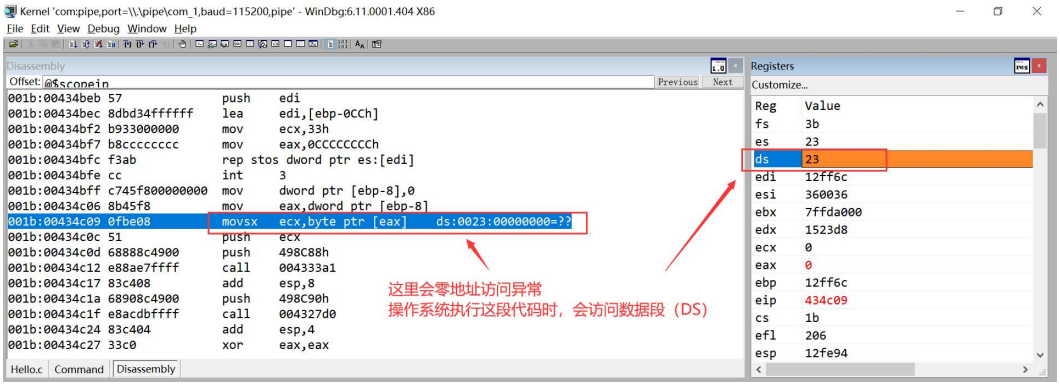

通过物理地址获取当前页表的下标，通过页表的下标可以得到页表的首地址。通过页表首地址去查询其在页目录表的项数，由此可以得到页表的高20位，低12位为gdtr的低12位。

0地址无法访问的原因：

CPU通过0地址查询第0下标（0地址的页目录表下标为0），然后访问对应的页表，查询对应页表中项是否有效，如下图所示：



CPU执行0xC05异常程序的过程：



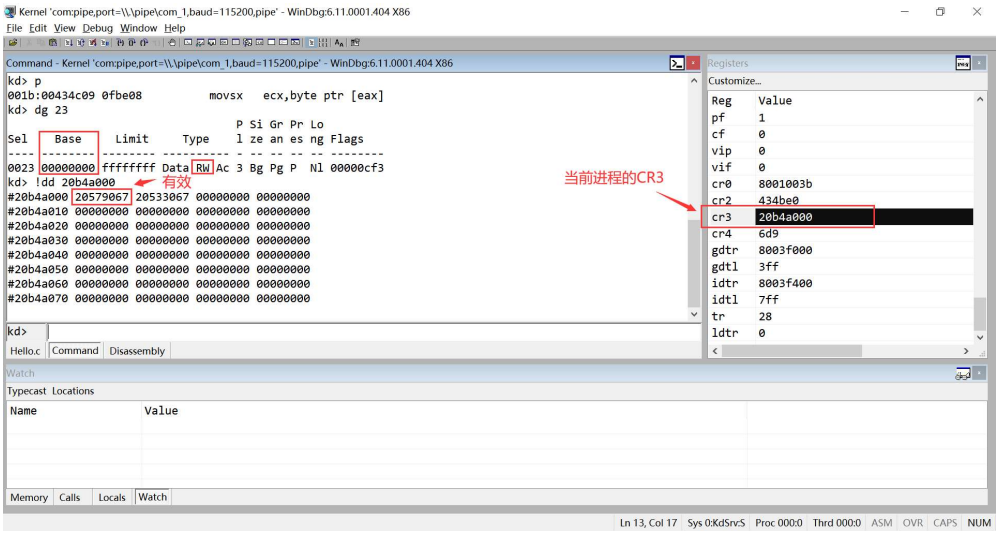

00000000 + 0（偏移）

线性地址 = 00000 000

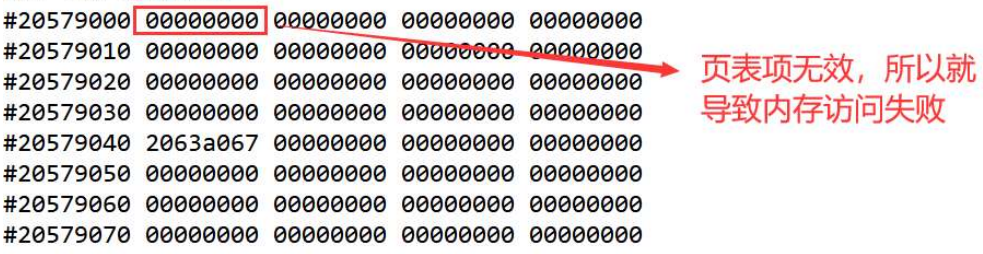
PDT index = 0

PTE index = 0

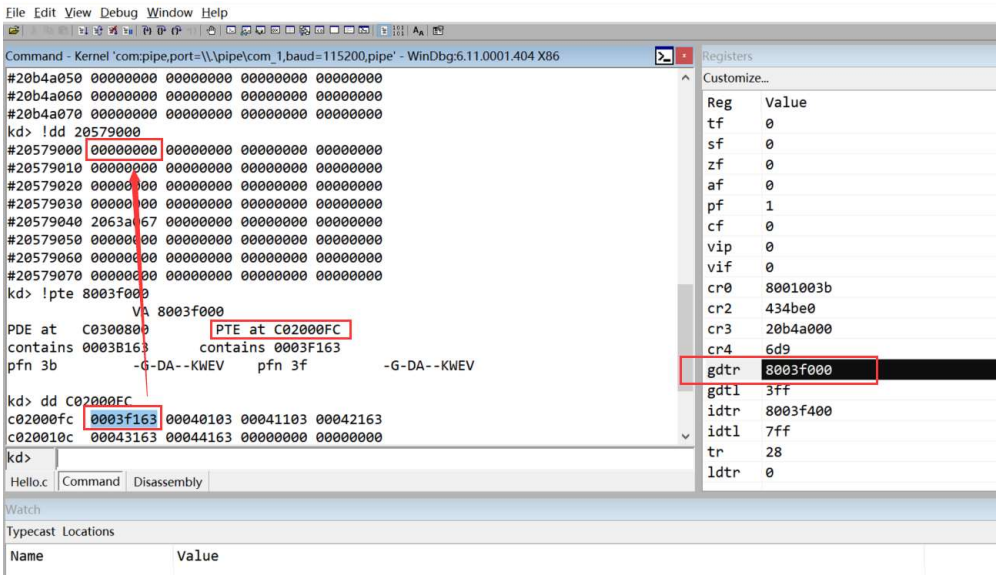
- 从CR3（每个进程都有一个CR3，切换进程就等价与切换CR3）地址处开始查表（CR3是当前程序）：



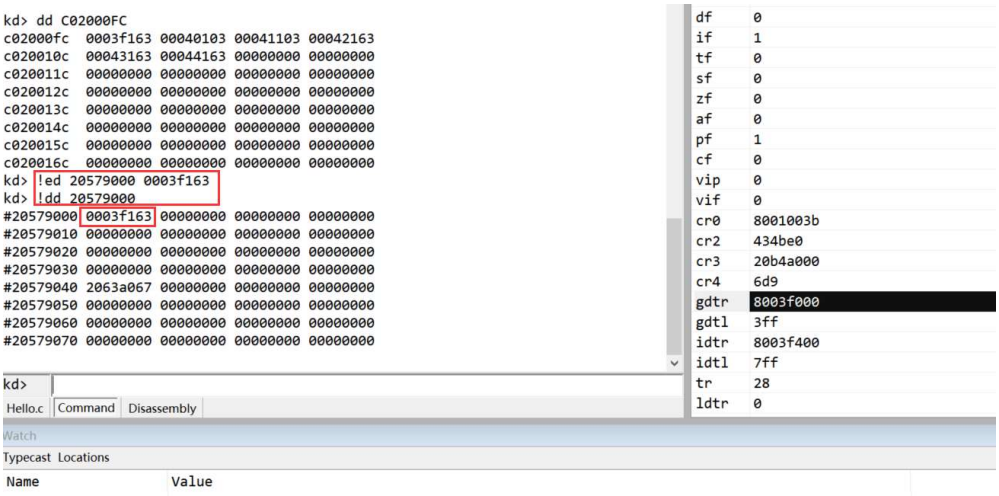

- "!dd 20b4a000" 对应的表项有效，"20579067"，最低位为1，取出高20位 "20579" 再补12位 "000"，"20579000"。
- Windbg 执行 "!dd 20579000" 命令，查看对应的表项是否有效，如下图所示




- 零地址可以使用，可将 "gdtr" 寄存器中值对应的PTE表的物理地址填充到这个零地址，操作如下：




- 修改目标地址 "!ed 20579000 0003f163"，CPU访问目标页表项时，为有效，就不会出现0xC05错误（0003f163，3：Ring3程序就有权限访问）：




- 修改内存后，在Windbg中输入 "g" 运行操作系统中的程序，程序可以运行，没有出现内存访问异常。

> 示例2：

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
  __asm int 3;
  char sz[] = "hello";
  printf("%p\n", sz);
  char* p = sz;

  printf("%02X",  *p);
  system("pause");

  return 0;
}
```

00000000 + 0（偏移）

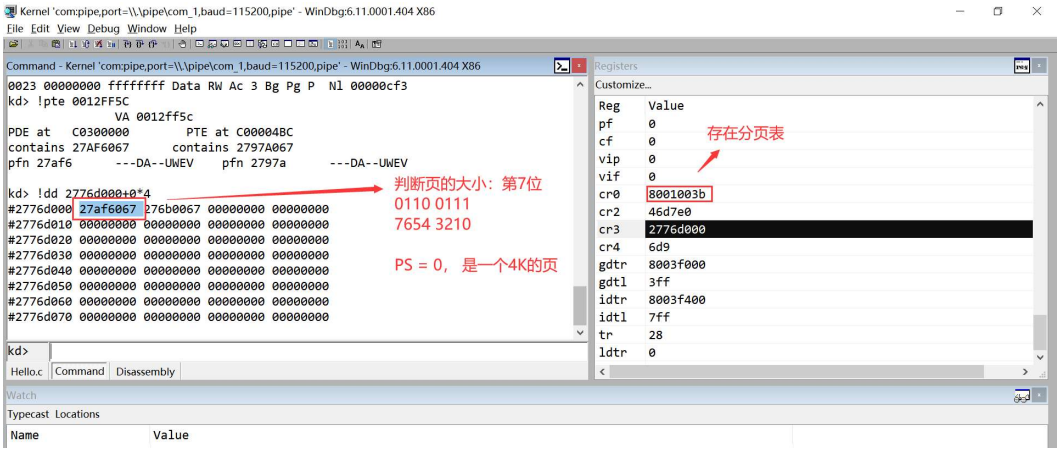
线性地址 = **0012F（物理地址） F5C**（偏移）

0012F -- 0000 0000 0010 0010 FFFF

PDT index = 0

PTE index = 12F

查询CR3寄存器值对应的表项：




确定分页的大小，当位4MB时，就需要取高18位（将这18位扩展位32位，后补零）+低22位的偏移。

根据目录表的物理地址 + 页表偏移去获取对应页表的物理地址：

```
kd> !dd 27af6000+12F*4
#27af64bc 2797a067 120ff025 12480025 00000000
#27af64cc 00000000 00000000 00000000 00000000
#27af64dc 00000000 00000000 00000000 00000000
#27af64ec 00000000 00000000 00000000 00000000
#27af64fc 00000000 27996025 00000000 00000000
#27af650c 00000000 00000000 00000000 00000000
#27af651c 00000000 00000000 00000000 00000000
#27af652c 00000000 00000000 00000000 00000000
```

得到表的物理地址

PA = 2797A000 + F5C（偏移）

CPU最后会访问物理地址：2997AF5C

访问物理地址上的数据：

```
kd> !db 2797af5c
#2797af5c 68 65 6c 6c 6f 00 cc cc-cc cc cc cc 0b 71 d6 10 hello........q..
#2797af6c b8 ff 12 00 ea 55 43 00-01 00 00 00 70 23 15 00 .....UC.....p#..
#2797af7c d8 23 15 00 df 71 d6 10-34 00 33 00 36 00 36 00 .#...q..4.3.6.6.
#2797af8c 00 70 fd 7f 00 00 00 00-00 00 00 00 00 00 00 00 .p..............
#2797af9c 00 00 00 00 80 ff 12 00-5c 9d 5a df e0 ff 12 00 ........\.Z.....
#2797afac f5 25 43 00 0f 44 8e 10-00 00 00 00 c0 ff 12 00 .%C..D..........
#2797afbc cd 57 43 00 f0 ff 12 00-67 70 81 7c 34 00 33 00 .WC.....gp.|4.3.
#2797afcc 36 00 36 00 00 70 fd 7f-b8 c6 54 80 c8 ff 12 00 6.6..p....T.....
kd> db 0012FF5C
0012ff5c 68 65 6c 6c 6f 00 cc cc-cc cc cc cc 0b 71 d6 10  hello........q..
0012ff6c b8 ff 12 00 ea 55 43 00-01 00 00 00 70 23 15 00  .....UC.....p#..
0012ff7c d8 23 15 00 df 71 d6 10-34 00 33 00 36 00 36 00  .#...q..4.3.6.6.
0012ff8c 00 70 fd 7f 00 00 00 00-00 00 00 00 00 00 00 00  .p..............
0012ff9c 00 00 00 00 80 ff 12 00-5c 9d 5a df e0 ff 12 00  ........\.Z.....
0012ffac f5 25 43 00 0f 44 8e 10-00 00 00 00 c0 ff 12 00  .%C..D..........
0012ffbc cd 57 43 00 f0 ff 12 00-67 70 81 7c 34 00 33 00  .WC.....gp.|4.3.
0012ffcc 36 00 36 00 00 70 fd 7f-b8 c6 54 80 c8 ff 12 00  6.6..p....T.....
```

CPU最后会访问的物理地址

缓冲区对应的虚拟地址

# WinDbg 命令

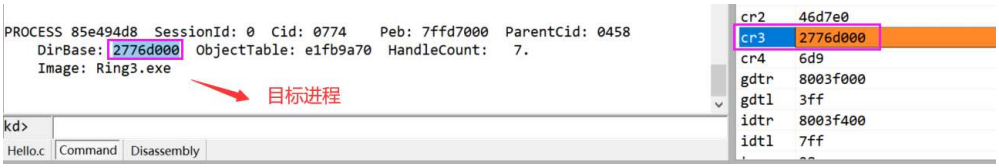
- 在 Windbg 中输入 "**!process 0 0**" 可以输出所有正在运行进程的CR3（页目录表）：

```
kd> !process 0 0
**** NT ACTIVE PROCESS DUMP ****
PROCESS 863b7830  SessionId: none  Cid: 0004    Peb: 00000000  ParentCid: 0000
    DirBase: 00039000  ObjectTable: e1000cf0  HandleCount: 164.
    Image: System

PROCESS 8619b458  SessionId: none  Cid: 0150    Peb: 7ffdd000  ParentCid: 0004
    DirBase: 0db5f000  ObjectTable: e100a490  HandleCount:  19.
    Image: smss.exe

PROCESS 86180020  SessionId: 0  Cid: 01ac    Peb: 7ffd5000  ParentCid: 0150
    DirBase: 0e7ef000  ObjectTable: e15423d0  HandleCount: 342.

...
```




- 在 Windbg 中输入 "**!process 0 7**" 可以输出所有正在运行进程的所有线程、所有线程对应的寄存器环境等信息：

```
PROCESS 85e494d8  SessionId: 0  Cid: 0774    Peb: 7ffd7000  ParentCid: 0458
    DirBase: 2776d000  ObjectTable: e1fb9a70  HandleCount:   7.
    Image: Ring3.exe
    VadRoot 86208058 Vads 21 Clone 0 Private 35. Modified 0. Locked 0.
    DeviceMap e15a2908
    Token                             e1ea0880
    ElapsedTime                       00:00:00.010
    UserTime                          00:00:00.010
    KernelTime                        00:00:00.000
    QuotaPoolUsage[PagedPool]         8864
    QuotaPoolUsage[NonPagedPool]      840
    Working Set Sizes (now,min,max)  (205, 50, 345) (820KB, 200KB, 1380KB)
    PeakWorkingSetSize                205
    VirtualSize                       6 Mb
    PeakVirtualSize                   6 Mb
    PageFaultCount                    201
    MemoryPriority                    BACKGROUND
    BasePriority                      8
    CommitCharge                      93

        THREAD 85e5bda8  Cid 0774.0754  Teb: 7ffdf000 Win32Thread: 00000000 RUNNING on processor 0
        Not impersonating
        DeviceMap                 e15a2908
        Owning Process            0       Image:         <Unknown>
        Attached Process          85e494d8       Image:         Ring3.exe
        Wait Start TickCount      224234         Ticks: 0
        Context Switch Count      37
        UserTime                  00:00:00.000
        KernelTime                00:00:00.000
        Win32 Start Address 0x00432d1b
        Start Address 0x7c8106f5
        Stack Init eedca000 Current eedc9c70 Base eedca000 Limit eedc7000 Call 0
        Priority 10 BasePriority 8 PriorityDecrement 2 DecrementCount 16
        ChildEBP RetAddr  Args to Child
WARNING: Frame IP not in any known module. Following frames may be wrong.
        0012ff6c 004355ea 00000001 00152370 001523d8 0x436572
        0012ffd0 8054c6b8 0012ffc8 85e5bda8 ffffffff 0x4355ea
        00130008 00000000 000000c4 00000000 00000020 nt!ExFreePoolWithTag+0x676 (FPO: [2,10,4])
```

# 跨进程读写内存

获取目标进程的CR3，可以做到欺骗CPU，进行跨进程读写内存。

```
// 访问系统进程 PID = 4 的 00401000 地址
// 在Windbg中使用 "!process 0 0"查看该进程的CR3（32位）：00039000
// GDTR(48bit)，CR3不需要查询GDTR表，因为其是物理地址
//
//强制读写进程内存
__asm {
    mov ebx, cr3
    mov eax, 00039000
```

```
    mov cr3, eax
    mov eax, [00401000h]
    mov cr3, ebx
}
```

```
PROCESS 85e257f0  SessionId: 0  Cid: 07e4    Peb: 7ffde000  ParentCid: 0458
    DirBase: 289c8000  ObjectTable: e1e88150  HandleCount:  15.
    Image: Ring3.exe

PROCESS 8621e4a0  SessionId: 0  Cid: 07cc    Peb: 7ffdd000  ParentCid: 07e4
    DirBase: 286ef000  ObjectTable: e1efd1e0  HandleCount:  30.
    Image: cmd.exe

PROCESS 86202b88  SessionId: 0  Cid: 043c    Peb: 7ffd3000  ParentCid: 0458
    DirBase: 29f2c000  ObjectTable: e1055350  HandleCount:  46.
    Image: InstDrv.exe

kd> g
[51asm] DriverEntry ch:68
[51asm] DriverUnload
```

读取成功

# PAE

> 物理地址扩展。
> 在32位CPU的基础上开启PAE，可将所有的物理地址升级到36位（此时32位CPU的地址总线已经扩展为36根，寄存器保留为32位），在64位上开启PAE，原物理地址由32位升级为52位 （64位太大也没必要）。

Win7开始、关闭PAE：

```
// {65223304-9093-11eb-888a-f6f1eb124313} -- 启动项对应的标识符
bcdedit /set {65223304-9093-11eb-888a-f6f1eb124313} pae ForceEnable
bcdedit /set {65223304-9093-11eb-888a-f6f1eb124313} pae forcedisable
```

一个进程不需要支持可以访问8G以上的内存，软件依然保留32位，支持访问4G的内存。只需要将32位的地址映射位36位的地址，软件兼容，不需要修改代码。对此Inter的做法：在原来的页目录表中，将32的物理地址升级为36位的物理地址。也就支持访问2^36 = 64G的内存。

使用标志位来告诉CPU，其开启了物理内存扩展。**在CR4寄存器中的第5位PAE Flag控制PAE是否开启。** 该标志不能强制进行修改，操作系统系统启动时就需要使用对应的表，强制修改会导致操作系统崩溃。
32位CPU：

64位CPU：

Table 4-1. Properties of Different Paging Modes

| Paging Mode | PG in CR0 | PAE in CR4 | LME in IA32_EFER | LA57 in CR4 | Lin.-Addr. Width | Phys.-Addr. Width[1] | Page Sizes | Supports Execute-Disable? | Supports PCIDs and protection keys? |
|---|---|---|---|---|---|---|---|---|---|
| None | 0 | N/A | N/A | N/A | 32 | 32 | N/A | No | No |
| 32-bit | 1 | 0 | 0[2] | N/A | 32 | Up to 40[3] | 4 KB<br>4 MB[4] | No | No |
| PAE | 1 | 1 | 0 | N/A | 32 | Up to 52 | 4 KB<br>2 MB | Yes[5] | No |
| 4-level | 1 | 1 | 1 | 0 | 48 | Up to 52 | 4 KB<br>2 MB<br>1 GB[6] | Yes[5] | Yes[7] |
| 5-level | 1 | 1 | 1 | 1 | 57 | Up to 52 | 4 KB<br>2 MB<br>1 GB[6] | Yes[5] | Yes[7] |

## 32位CPU开启PAE

原目录表、分页表可缩小位原来的一倍，整体缩小4倍。保证可以访问原内存大小，就增加了四个页目录指针表（PDPT）。CR3指向页目录指针表（保存四个页目录表的地址）。512 * 512 * 4096 * 4 = 4GB

在开启了PAE之后，页大小为4KB时，地址转换如下图（每个表项均为8字节）：

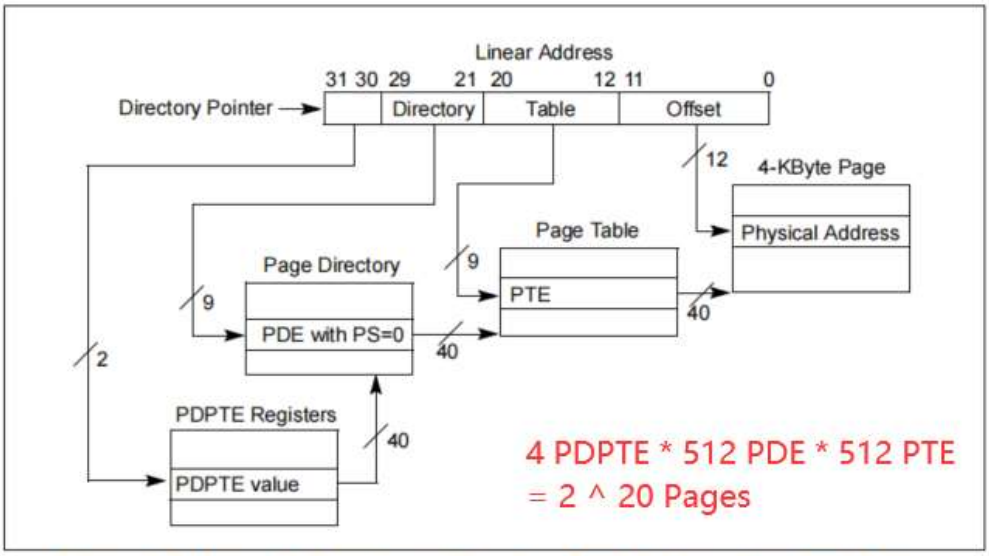



Figure 4-5. Linear-Address Translation to a 4-KByte Page using PAE Paging

页大小为2MB时，地址转换如下图：低21位为偏移

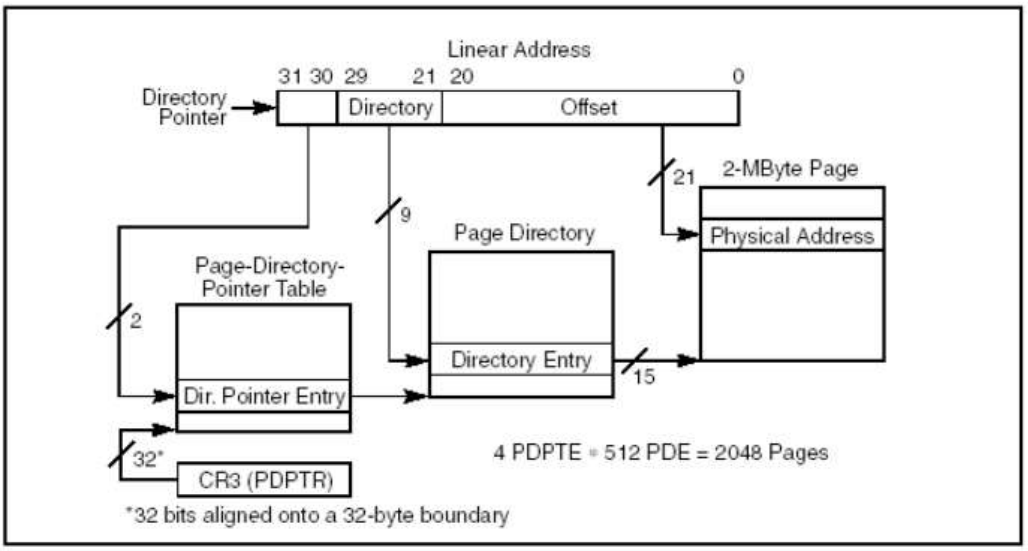



**Figure 3-19. Linear Address Translation With PAE Enabled (2-MByte Pages)**

当使用PAE分页时，CR3会引用一个32字节的页目录指针表的基础。表4-7说明了CR3如何与PAE分页中使用。

**Table 4-7. Use of CR3 with PAE Paging**

| Bit Position(s) | Contents |
|---|---|
| 4:0 | Ignored |
| 31:5 | Physical address of the 32-Byte aligned page-directory-pointer table used for linear-address translation |
| 63:32 | Ignored (these bits exist only on processors supporting the Intel-64 architecture) |

- CR3指向页目录指针表（PDPTE）
- 线性地址的高30 - 31位表示了PDPTE最多有4项
- 高21 - 29位的PDE索引和12 - 20位的PTE索引分别表示了PDE和PTE最多有512项
- 最后0 - 11位依然是偏移
- 在各个表中的项目中，拿低12位后的高24位是基址

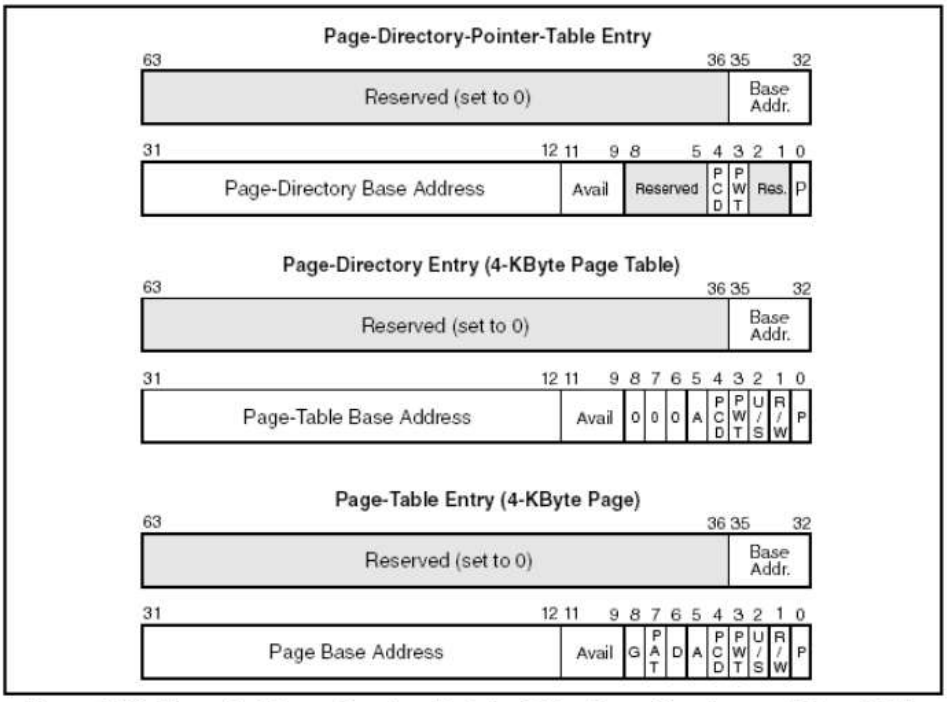



**Figure 3-20. Format of Page-Directory-Pointer-Table, Page-Directory, and Page-Table Entries for 4-KByte Pages with PAE Enabled**

> 页目录表项中的第7位判断页表的大小，为0，页表大小为4KB；为1，页表大小为2MB；

在PAE中查表：

- 操作系统需要打开PAE选项，开启PAE后，页的大小只能是4KB（PAE flag = 0）、2MB（PAE flag = 1），物理地址升级为36位。未开启PAE，物理地址为32位。

Table 3-3. Page Sizes and Physical Address Sizes

| PG Flag, CR0 | PAE Flag, CR4 | PSE Flag, CR4 | PS Flag, PDE | Page Size | Physical Address Size |
|---|---|---|---|---|---|
| 0 | X | X | X | — | Paging Disabled |
| 1 | 0 | 0 | X | 4 KBytes | 32 Bits |
| 1 | 0 | 1 | 0 | 4 KBytes | 32 Bits |
| 1 | 0 | 1 | 1 | 4 MBytes | 32 Bits |
| 1 | 1 | X | 0 | 4 KBytes | 36 Bits |
| 1 | 1 | X | 1 | 2 MBytes | 36 Bits |

- 以GDTR寄存器值 "8003f000" 为例，对其进行拆分：
  8003f 000 -- 共拆解成4份
  10 000000000 000111111 000000000000（偏移）
  -- PDPT index = 2
  -- PDE index = 0
  -- PTE index = 3F（**页大小为2MB时，所占位数当作偏移使用**）
- 获取页目录指针表地址： !dq b46000+2 * 8 -- b46000（CR3的值）

```
kd> !dq b46000 + 2 * 8
#  b46010 00000000`00b49001 00000000`00b4a001
#  b46020 00000000`00000000 00000000`00000000
...
00b49001 -- 最低为为1，表示有效
取高24位（物理地址）：000b49
```

- 获取页目录表地址：

```
kd>  !dq 000b49000 +0 * 8
// 第7位为0，页大小为4KB
#  b49000 00000000`00b51163 00000000`00b52163
// 第7位为0，页大小为2MB
#  b49010 00000000`004001e3 00000000`006001e3
#  b49020 00000000`00b55163 00000000`00b56163
#  b49030 00000000`010001e3 00000000`012001e3
#  b49040 00000000`014001e3 00000000`016001e3
#  b49050 00000000`018001e3 00000000`01a001e3
#  b49060 00000000`01c001e3 00000000`01e001e3
#  b49070 00000000`020001e3 00000000`022001e3
// 页大小为2MB时，虚拟地址一般在高2G中，系统进程一般才有
...
00b49001 -- 最低为为1，表示有效
```

```
取高24位：000b51
物理地址（36位）：000b51000
```

- 获取页表地址：

```
!dq 000b51000 + 3F * 8
```

- 获取物理地址：

```
kd> !dq 000b51000 + 3F * 8
#  b511F8 00000000`0003F163 00000000`00040103
#  b51208 00000000`00041103 00000000`00042163
#  b51218 00000000`00043163 00000000`00044163
#  b51228 00000000`00000000 00000000`00000000
...
取高24位：00003F
物理地址（36位）：00003F000
```

> **!db 00003F000 + 0**

```
kd> !db 00003F000 + 0          对应的物理地址
#   3f000 00 00 00 00 00 00 00 00-ff ff 00 00 00 9b cf 00 ................
#   3f010 ff ff 00 00 00 93 cf 00-ff ff 00 00 00 fb cf 00 ................
#   3f020 ff ff 00 00 00 f3 cf 00-ab 20 00 20 04 8b 00 80 ......... . ....
#   3f030 01 00 00 f0 df 93 c0 ff-ff 0f 00 00 00 f3 40 00 ..............@.
#   3f040 ff ff 00 04 00 f2 00 00-00 00 00 00 00 00 00 00 ................
#   3f050 68 00 00 af 54 89 00 80-68 00 68 af 54 89 00 80 h...T...h.h.T...
#   3f060 ff ff 40 2f 02 93 00 00-ff 3f 00 80 0b 92 00 00 ..@/.....?......
#   3f070 ff 03 00 70 ff 92 00 ff-ff ff 00 00 40 9a 00 80 ...p........@...
kd> db 8003f000          gdtr 对应的虚拟地址
8003f000  00 00 00 00 00 00 00 00-ff ff 00 00 00 9b cf 00  ................
8003f010  ff ff 00 00 00 93 cf 00-ff ff 00 00 00 fb cf 00  ................
8003f020  ff ff 00 00 00 f3 cf 00-ab 20 00 20 04 8b 00 80  ......... . ....
8003f030  01 00 00 f0 df 93 c0 ff-ff 0f 00 00 00 f3 40 00  ..............@.
8003f040  ff ff 00 04 00 f2 00 00-00 00 00 00 00 00 00 00  ................
8003f050  68 00 00 af 54 89 00 80-68 00 68 af 54 89 00 80  h...T...h.h.T...
8003f060  ff ff 40 2f 02 93 00 00-ff 3f 00 80 0b 92 00 00  ..@/.....?......
8003f070  ff 03 00 70 ff 92 00 ff-ff ff 00 00 40 9a 00 80  ...p........@...
```

WinXP开启PAE后，内存访问不能超过4GB，操作系统启动时，会判断系统的版本，个人版本会按照36位建表。服务器版本会支持4GB以上的内存。WinXP打补丁就可以支持4GB以上的内存。

## 64位CPU开启PAE

> 64位CPU开启PAE后，32位的程序支持将地址升级到52位的物理地址；不开启，32位的程序支持将地址升级到40位的物理地址：

Table 4-1. Properties of Different Paging Modes

| Paging Mode | PG in CR0 | PAE in CR4 | LME in IA32_EFER | LA57 in CR4 | Lin.-Addr. Width | Phys.-Addr. Width[1] | Page Sizes | Supports Execute-Disable? | Supports PCIDs and protection keys? |
|---|---|---|---|---|---|---|---|---|---|
| None | 0 | N/A | N/A | N/A | 32 | 32 | N/A | No | No |
| 32-bit | 1 | 0 | 0[2] | N/A | 32 | Up to 40[3] | 4 KB<br>4 MB[4] | No | No |
| PAE | 1 | 1 | 0 | N/A | 32 | Up to 52 | 4 KB<br>2 MB | Yes[5] | No |
| 4-level | 1 | 1 | 1 | 0 | 48 | Up to 52 | 4 KB<br>2 MB<br>1 GB[6] | Yes[5] | Yes[7] |
| 5-level | 1 | 1 | 1 | 1 | 57 | Up to 52 | 4 KB<br>2 MB<br>1 GB[6] | Yes[5] | Yes[7] |

线性地址只用了48位。

> MmGetPhysicalAddress：将虚拟地址转换位对应的物理地址。
> MmMapIoSpace：将物理地址映射一块虚拟地址到当前进程中（在页表中找到一块空闲位置，填写获取到的物理地址，同时通过物理就可以得到一个对应的虚拟地址）。