2.3 Windows内存结构与管理

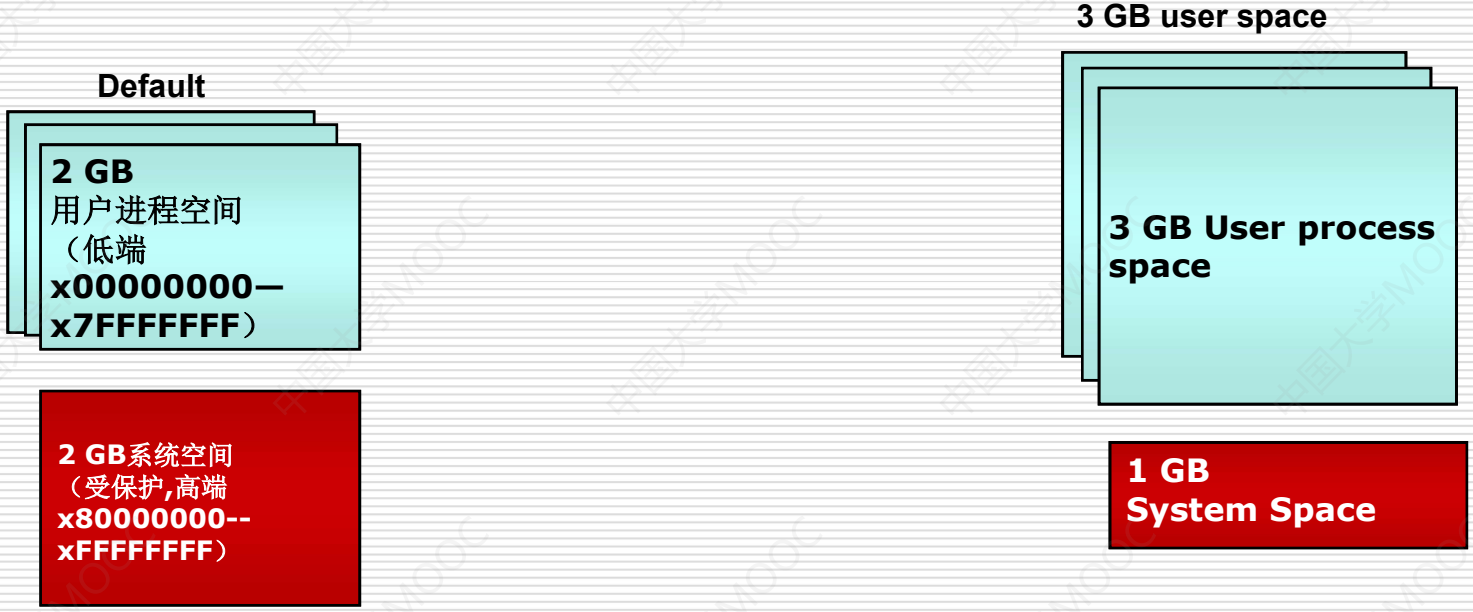
- 内存布局与页面地址转换
  - DOS实模式下的内存布局
  - Windows下的虚拟地址空间布局
  - 虚拟地址与物理地址的转换
- Windows内存页面权限管理

2.3.1 内存布局与地址转换

DOS实模式下的内存布局

2.3.1 内存布局与地址转换

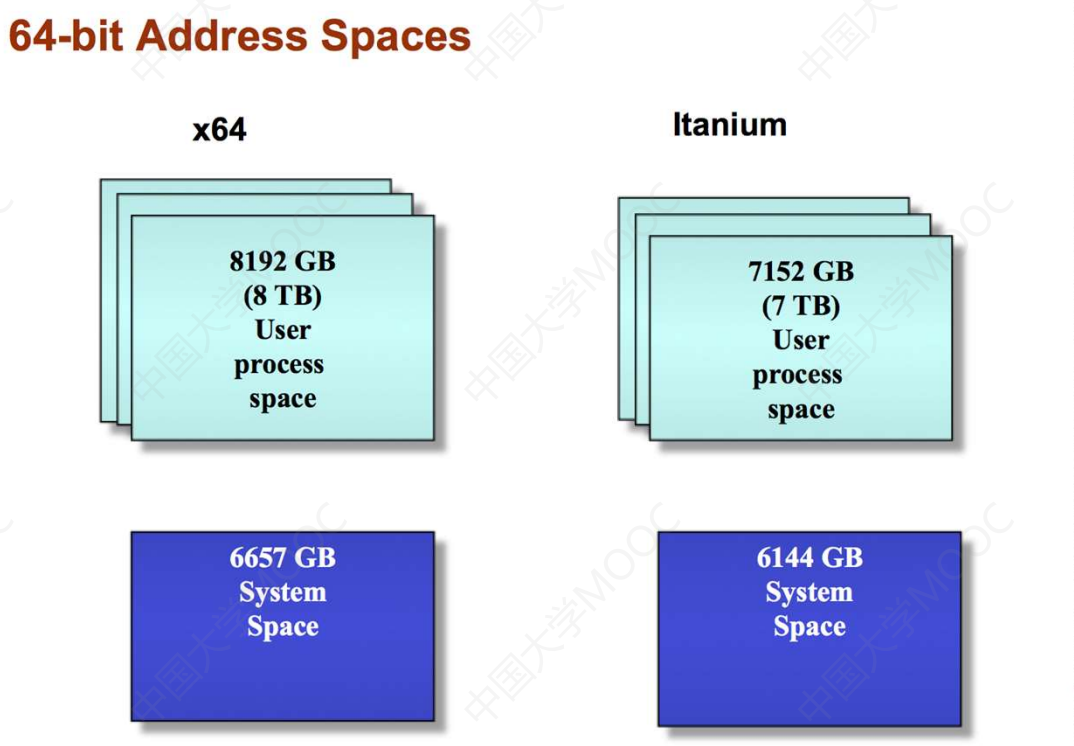
Windows虚拟地址空间（32位）



（32-bit x86 虚拟地址空间最大为4GB）

2.3.1 内存布局与地址转换

Windows虚拟地址空间（64位）



CPU特权级与内存访问

- 与进程虚拟内存中用户模式区和内核模式区相对应，Windows为了确保系统的稳定性，将处理器存取模式划分为用户模式（Ring 3）和内核模式（Ring 0）。
  - 用户应用程序一般运行在用户模式。
    - 其访问空间局限于用户区；
  - 操作系统内核代码（如系统服务和设备驱动程序等）运行在内核模式。
    - 可以访问所有的内存空间（包括用户模式分区）和硬件，可使用所有处理器指令。

用户区内存

- 用户区是每个进程真正独立的可用内存空间，进程中的绝大部分数据都保存在这一区域。
  - 主要包括：应用程序代码、全局变量、所有线程的线程栈以及加载的DLL代码等
- 每个进程的用户区的虚拟内存空间相互独立，一般不可以直接跨进程访问，这使得一个程序直接破坏另一个程序的可能性非常小。

OD查看Notepad进程内存区

地址	大小	属主	区段	包含	类型	访问
00010000	00001000				Priv 00021004	RW
00020000	00001000				Priv 00021004	RW
00030000	00005000				Priv 00021004	RW
0006E000	00001000				Priv 00021104	RW 保护
0006F000	00011000			堆栈 于 主线程	Priv 00021104	RW 保护
00080000	00003000				Map 00041002	R
00090000	00002000				Map 00041002	R
000A0000	00009000				Priv 00021004	RW
001A0000	00006000				Priv 00021004	RW
001B0000	00003000				Map 00041004	RW
001C0000	00016000				Map 00041002	R
001E0000	00041000				Map 00041002	R
00230000	00041000				Map 00041002	R
00280000	00006000				Map 00041002	R
00290000	00041000				Map 00041002	R
002E0000	0000E000				Map 00041020	R E
003A0000	00002000				Map 00041020	R E
003B0000	00008000				Priv 00021004	RW
003C0000	00001000				Priv 00021004	RW
003D0000	00001000				Priv 00021004	RW
003E0000	00002000				Map 00041002	R
003F0000	00002000				Map 00041002	R
00400000	00003000				Priv 00021004	RW
00410000	00008000				Priv 00021004	RW
00420000	00004000				Priv 00021004	RW
00430000	00003000				Map 00041002	R
00440000	00003000				Priv 00021004	RW
00480000	00103000				Map 00041002	R
00590000	00123000				Map 00041020	R E
00890000	00001000				Priv 00021004	RW
01000000	00001000	notepad		PE 文件头	Imag 01001002	R
01001000	00008000	notepad	.text	代码,输入表	Imag 01001002	R
01009000	00002000	notepad	.data	数据	Imag 01001002	R
0100B000	00008000	notepad	.rsrc	资源	Imag 01001002	R
58FB0000	00001000	AcGenral		PE 文件头	Imag 01001002	R
58FB1000	00032000	AcGenral	.text	代码,输入表,输出表	Imag 01001002	R
58FE3000	00009000	AcGenral	.data	数据	Imag 01001002	R
58FEC000	00188000	AcGenral	.rsrc	资源	Imag 01001002	R
59174000	00006000	AcGenral	.reloc	重定位	Imag 01001002	R
5ADC0000	00001000	UxTheme		PE 文件头	Imag 01001002	R
5ADC1000	00030000	UxTheme	.text	代码,输入表,输出表	Imag 01001002	R
5ADF1000	00001000	UxTheme	.data	数据	Imag 01001002	R
5ADF2000	00003000	UxTheme	.rsrc	资源	Imag 01001002	R
5ADF5000	00002000	UxTheme	.reloc	重定位	Imag 01001002	R
5CC30000	00001000	ShimEng		PE 文件头	Imag 01001002	R
5CC31000	0000E000	ShimEng	.text	代码,输入表,输出表	Imag 01001002	R
5CC3F000	00014000	ShimEng	.data	数据	Imag 01001002	R
5CC53000	00001000	ShimEng	.rsrc	资源	Imag 01001002	R
5CC54000	00002000	ShimEng	.reloc	重定位	Imag 01001002	R
62C20000	00001000	LPK		PE 文件头	Imag 01001002	R
62C21000	00005000	LPK	.text	代码,输入表,输出表	Imag 01001002	R
62C26000	00001000	LPK	.data	数据	Imag 01001002	R
62C27000	00001000	LPK	.rsrc	资源	Imag 01001002	R

地址	大小	属主	区段	包含	类型	访问
77C35000	00003000	msvcrt	.reloc	重定位	Imag 01001002	R
77D10000	00001000	USER32		PE 文件头	Imag 01001002	R
77D11000	00060000	USER32	.text	代码,输入表,输出表	Imag 01001002	R
77D71000	00002000	USER32	.data	数据	Imag 01001002	R
77D73000	0002A000	USER32	.rsrc	资源	Imag 01001002	R
77D9D000	00003000	USER32	.reloc	重定位	Imag 01001002	R
77DA0000	00001000	ADVAPI32		PE 文件头	Imag 01001002	R
77DA1000	00075000	ADVAPI32	.text	代码,输入表,输出表	Imag 01001002	R
77E16000	00005000	ADVAPI32	.data	数据	Imag 01001002	R
77E1B000	00029000	ADVAPI32	.rsrc	资源	Imag 01001002	R
77E44000	00005000	ADVAPI32	.reloc	重定位	Imag 01001002	R
77E50000	00001000	RPCRT4		PE 文件头	Imag 01001002	R
77E51000	00084000	RPCRT4	.text	代码,输入表,输出表	Imag 01001002	R
77ED5000	00007000	RPCRT4	.orpc	代码	Imag 01001002	R
77EDC000	00001000	RPCRT4	.data	数据	Imag 01001002	R
77EDD000	00001000	RPCRT4	.rsrc	资源	Imag 01001002	R
77EDE000	00005000	RPCRT4	.reloc	重定位	Imag 01001002	R
77EF0000	00001000	GDI32		PE 文件头	Imag 01001002	R
77EF1000	00043000	GDI32	.text	代码,输入表,输出表	Imag 01001002	R
77F34000	00002000	GDI32	.data	数据	Imag 01001002	R
77F36000	00001000	GDI32	.rsrc	资源	Imag 01001002	R
77F37000	00002000	GDI32	.reloc	重定位	Imag 01001002	R
77F40000	00001000	SHLWAPI		PE 文件头	Imag 01001002	R
77F41000	0006C000	SHLWAPI	.text	代码,输入表,输出表	Imag 01001002	R
77FAD000	00001000	SHLWAPI	.data	数据	Imag 01001002	R
77FAE000	00002000	SHLWAPI	.rsrc	资源	Imag 01001002	R
77FB0000	00006000	SHLWAPI	.reloc	重定位	Imag 01001002	R
77FC0000	00001000	Secur32		PE 文件头	Imag 01001002	R
77FC1000	0000D000	Secur32	.text	代码,输入表,输出表	Imag 01001002	R
77FCE000	00001000	Secur32	.data	数据	Imag 01001002	R
77FCF000	00001000	Secur32	.rsrc	资源	Imag 01001002	R
77FD0000	00001000	Secur32	.reloc	重定位	Imag 01001002	R
7C800000	00001000	kernel32		PE 文件头	Imag 01001002	R
7C801000	00084000	kernel32	.text	代码,输入表,输出表	Imag 01001002	R
7C885000	00005000	kernel32	.data	数据	Imag 01001002	R
7C88A000	0008E000	kernel32	.rsrc	资源	Imag 01001002	R
7C918000	00006000	kernel32	.reloc	重定位	Imag 01001002	R
7C920000	00001000	ntdll		PE 文件头	Imag 01001002	R
7C921000	0007D000	ntdll	.text	代码,输出表	Imag 01001002	R
7C99E000	00005000	ntdll	.data	数据	Imag 01001002	R
7C9A3000	00010000	ntdll	.rsrc	资源	Imag 01001002	R
7C9B3000	00003000	ntdll	.reloc	重定位	Imag 01001002	R
7D590000	00001000	SHELL32		PE 文件头	Imag 01001002	R
7D591000	001FF000	SHELL32	.text	代码,输入表,输出表	Imag 01001002	R
7D790000	0001D000	SHELL32	.data	数据	Imag 01001002	R
7D7AD000	005BD000	SHELL32	.rsrc	资源	Imag 01001002	R
7DD6A000	0001B000	SHELL32	.reloc	重定位	Imag 01001002	R
7F6F0000	00006000				Map 00041020	R E
7FFA0000	00033000				Map 00041002	R
7FFDA000	00001000				Priv 00021004	RW
7FFDF000	00001000			数据块 于 主线程	Priv 00021004	RW
7FFE0000	00001000				Priv 00021002	R

内核区内存

- 内存内核区中的所有数据是所用进程共享的，是操作系统代码的驻地。
  - 其中包括：操作系统内核代码，以及与线程调度、内存管理、文件系统支持、网络支持、设备驱动程序相关的代码。
- 该分区中所有代码和数据都被操作系统保护。
  - 用户模式代码无法直接访问和操作：如果应用程序直接对该内存空间内的地址访问，将会发生地址访问违规。

两个思考问题？

我的电脑内存才2G，为何每个进程可用4GB内存空间？

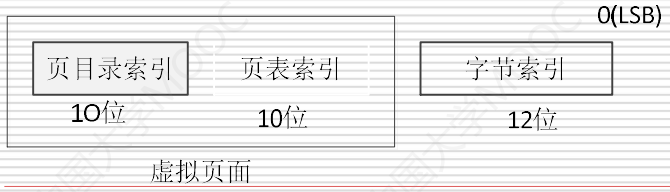
- ？

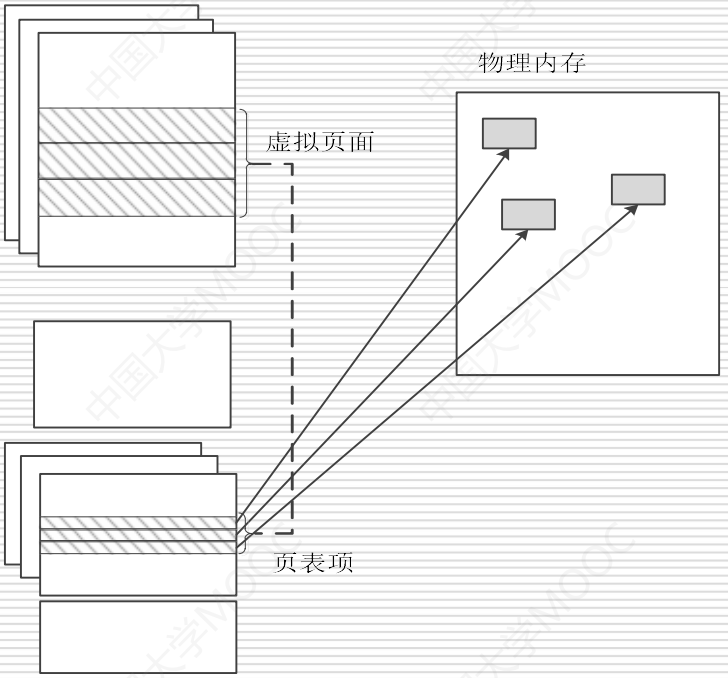
A、B两个进程的可执行程序映像加载地址都是00400000H，但同一地址对应的的代码却不一样，为什么？

- ？

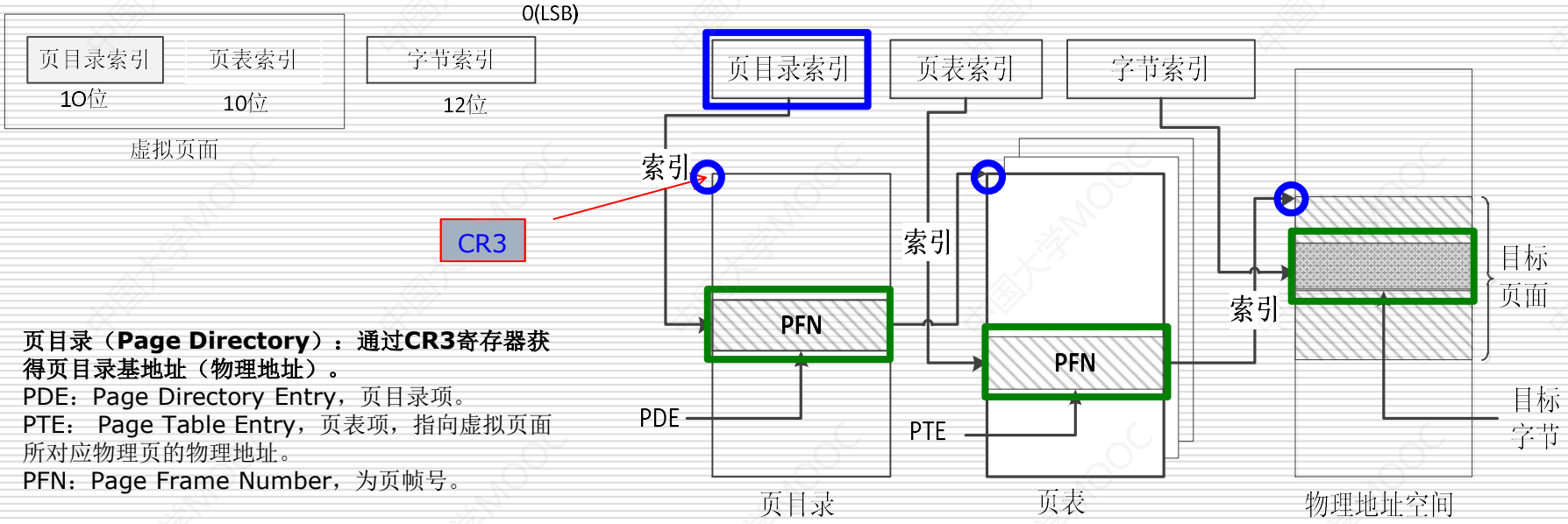
Windows分页机制

- X86 Windows默认使用二级页表来把虚拟地址转译为物理地址。
  - 一个32位地址被划分为三个单独部分：页目录索引、页表索引和字节索引。
- 在x86系统上默认页面大小为4K，故页内字节索引宽度为12位。





虚拟地址转译到物理地址的过程



页目录（Page Directory）：通过CR3寄存器获得页目录基地址（物理地址）。
PDE：Page Directory Entry，页目录项。
PTE： Page Table Entry，页表项，指向虚拟页面所对应物理页的物理地址。
PFN：Page Frame Number，为页帧号。

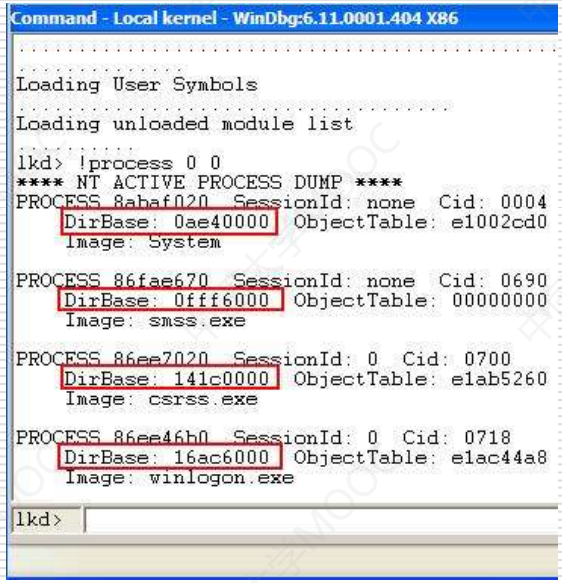
页目录表，页表，以及页均以4k对齐，低12位做什么用？

PFN（PTE与PDE，4K页大小）

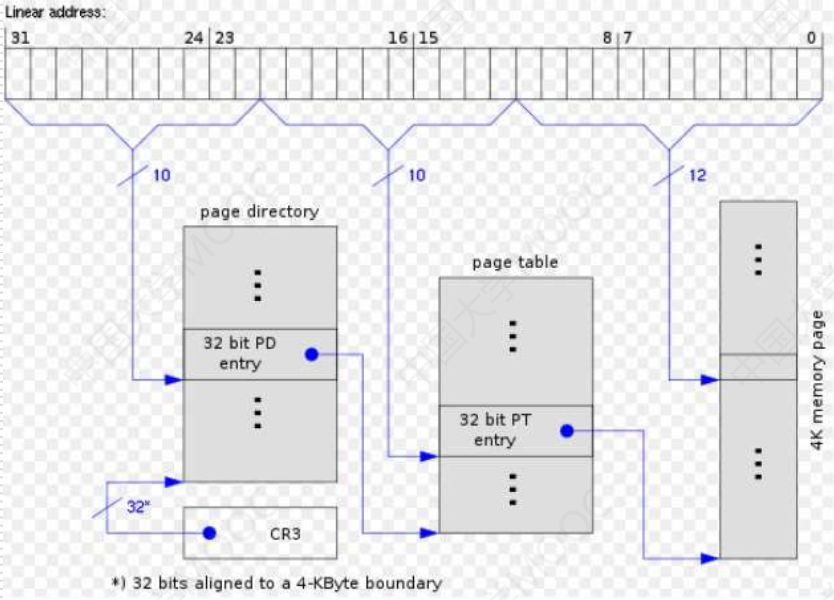
CPU对物理地址的寻址能力有多大？与PFN有什么关系？

CR3寄存器

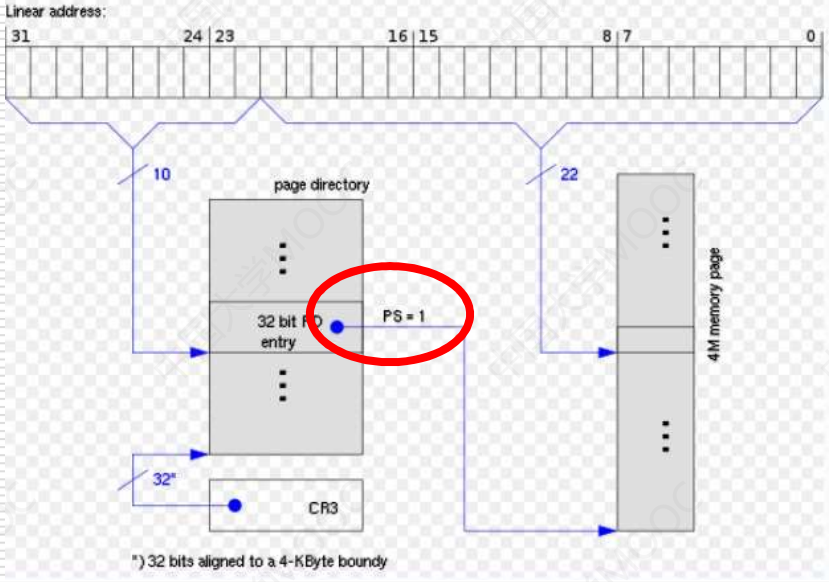
- CR3指向页目录基地址的物理地址。
  - 也被称为页目录基地址寄存器PDBR（Page-Directory Base address Register）
- 二级页表下，32位系统中的CR3寄存器末尾12位为0（以4k对齐）
- Windbg：!process 0 0
  - DirBase即CR3



分页大小只能为4K么？



4K分页的页表结构

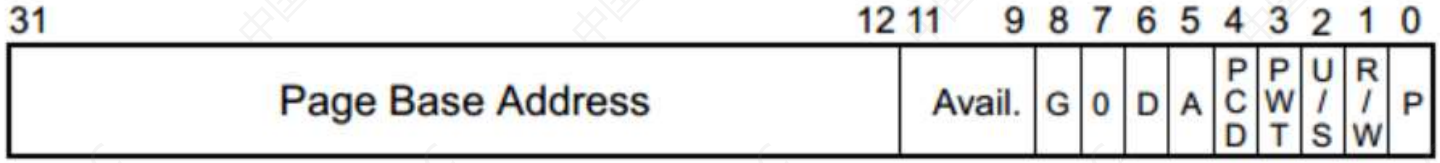


4M分页的页表结构

如果物理内存大于4G，如何办？

- 为了提高处理器的寻址能力，Intel在处理器上把管脚数从32增加到36，使其达到2^36=64GB
- 为此引入一种新的分页机制-PAE（ Physical Address Extension ）

32位页目录项（PDE）和页表项（PTE）



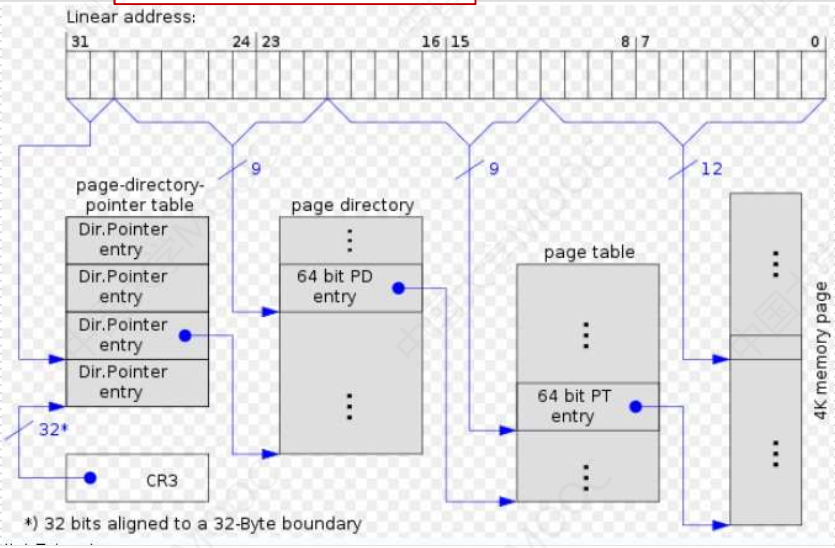
页面基地址：20b→24b　　**PDE/PTE**长度：32b→36b（64b）

4KB/32b=1024（2^{10}）项 → 4KB/64b=512（2^{9}）项

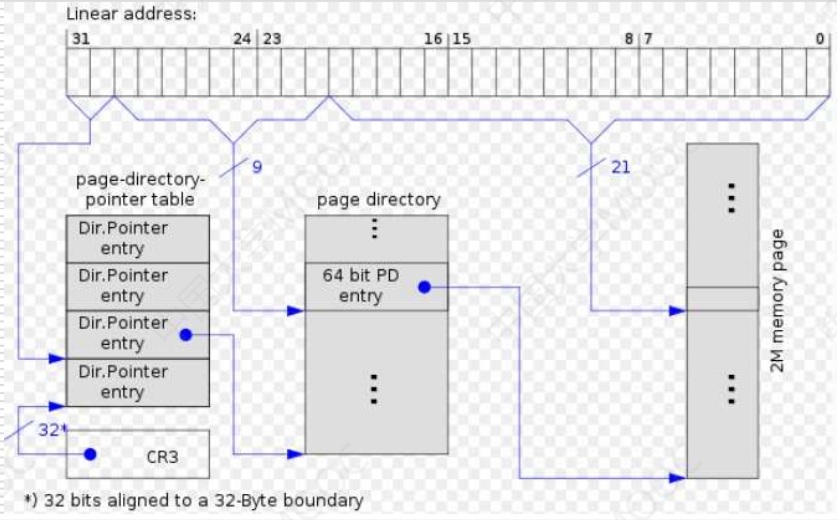
启用PAE

三级页表：2+9+9

2

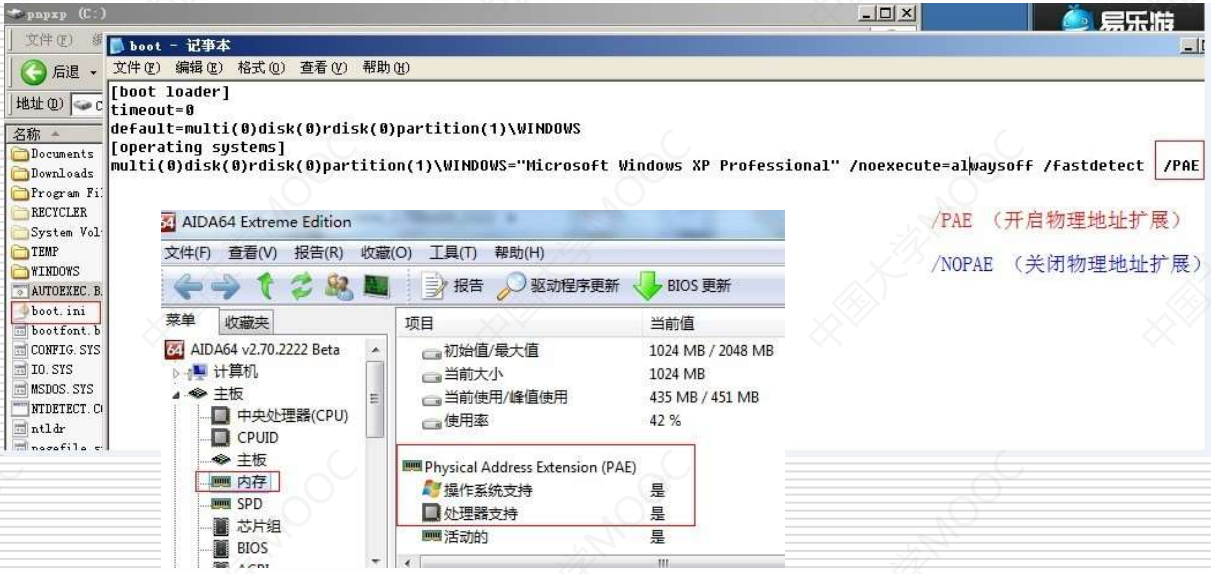


启用PAE下4K分页的页表结构



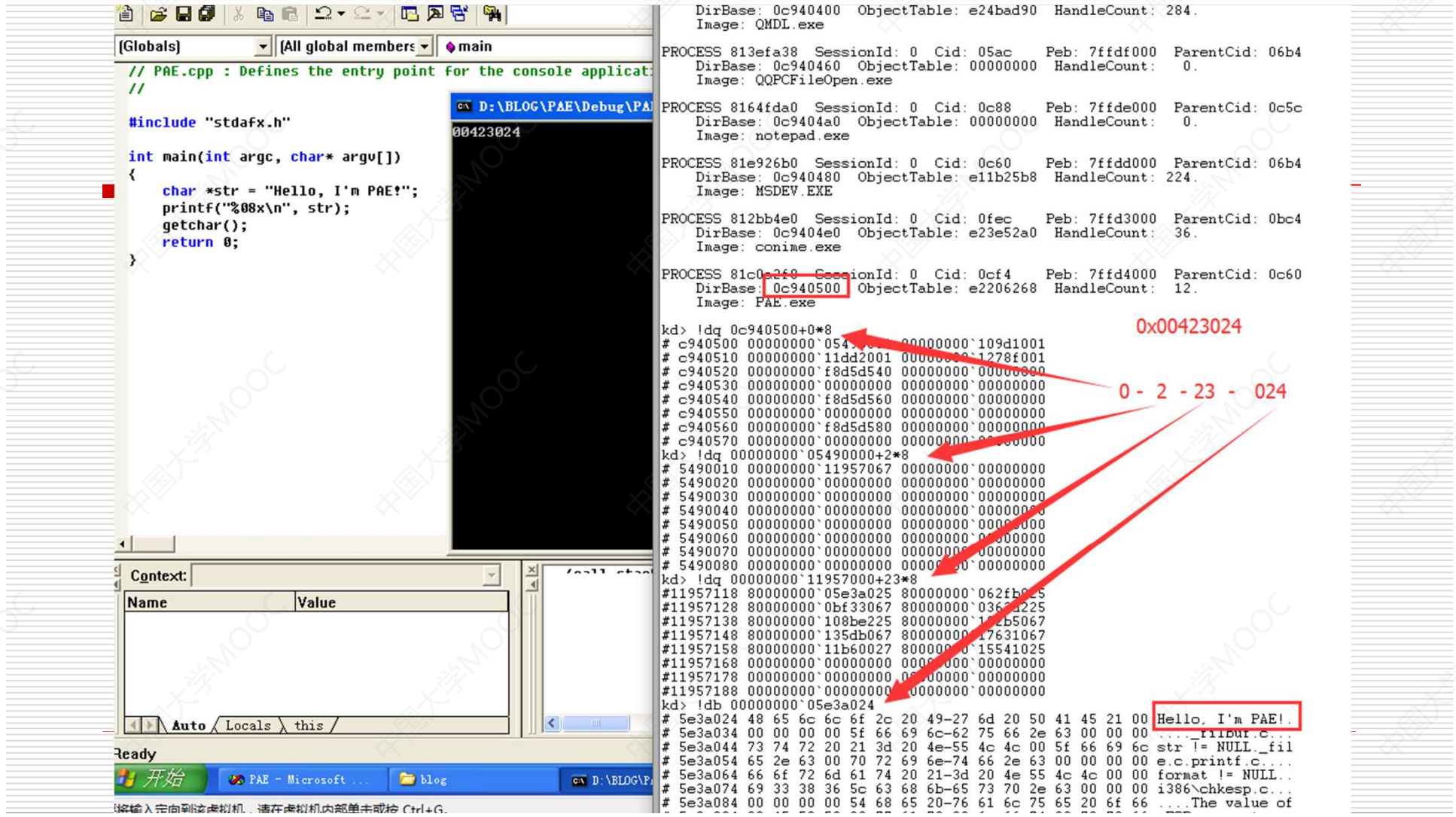
启用PAE下2M分页的页表结构

操作系统如何开启PAE

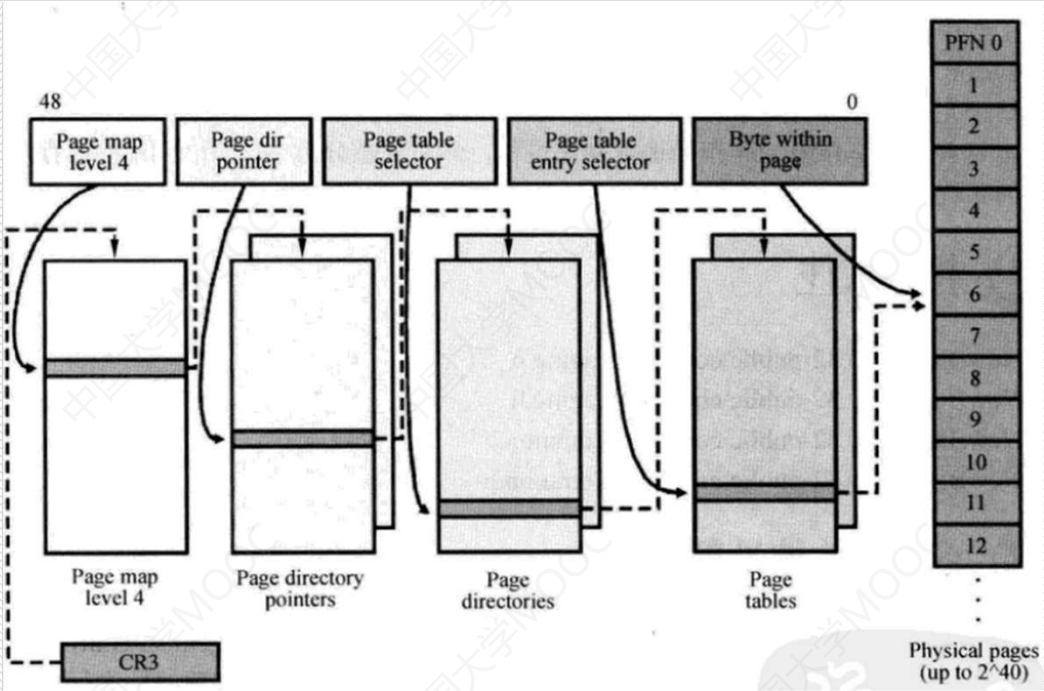


XP开启PAE的办法

WIN7开启PAE的办法


(Globals)
(All global members
main
// PAE.cpp : Defines the entry point for the console applicat
//
#include "stdafx.h"
int main(int argc, char* argv[])
{
char *str = "Hello, I'm PAE!";
printf("%08x\n", str);
getchar();
return 0;
}
D:\BLOG\PAE\Debug\PA
00423024
Context:
Name
Value
Auto
Locals
this
Ready
开始
PAE - Microsoft ...
blog
D:\BLOG\P
DirBase: 0c940400 ObjectTable: e24bad90 HandleCount: 284.
Image: QMDL.exe
PROCESS 813efa38 SessionId: 0 Cid: 05ac Peb: 7ffdf000 ParentCid: 06b4
DirBase: 0c940460 ObjectTable: 00000000 HandleCount: 0.
Image: QQPCFileOpen.exe
PROCESS 8164fda0 SessionId: 0 Cid: 0c88 Peb: 7ffde000 ParentCid: 0c5c
DirBase: 0c9404a0 ObjectTable: 00000000 HandleCount: 0.
Image: notepad.exe
PROCESS 81e926b0 SessionId: 0 Cid: 0c60 Peb: 7ffdd000 ParentCid: 06b4
DirBase: 0c940480 ObjectTable: e11b25b8 HandleCount: 224.
Image: MSDEV.EXE
PROCESS 812bb4e0 SessionId: 0 Cid: 0fec Peb: 7ffd3000 ParentCid: 0bc4
DirBase: 0c9404e0 ObjectTable: e23e52a0 HandleCount: 36.
Image: conime.exe
PROCESS 81c0c2f8 SessionId: 0 Cid: 0cf4 Peb: 7ffd4000 ParentCid: 0c60
DirBase: 0c940500 ObjectTable: e2206268 HandleCount: 12.
Image: PAE.exe
kd> !dq 0c940500+0*8
kd> !dq 00000000`05490000+2*8
kd> !dq 00000000`11957000+23*8
kd> !db 00000000`05e3a024
5e3a024 48 65 6c 6c 6f 2c 20 49-27 6d 20 50 41 45 21 00 Hello, I'm PAE!.
5e3a034 00 00 00 00 5f 66 69 6c-62 75 66 2e 63 00 00 00 ...._filbuf.c...
5e3a044 73 74 72 20 21 3d 20 4e-55 4c 4c 00 5f 66 69 6c str != NULL._fil
5e3a054 65 2e 63 00 70 72 69 6e-74 66 2e 63 00 00 00 00 e.c.printf.c....
5e3a064 66 6f 72 6d 61 74 20 21-3d 20 4e 55 4c 4c 00 00 format != NULL..
5e3a074 69 33 38 36 5c 63 68 6b-65 73 70 2e 63 00 00 00 i386\chkesp.c...
5e3a084 00 00 00 00 54 68 65 20-76 61 6c 75 65 20 6f 66 ....The value of
0x00423024
0 - 2 - 23 - 024

64位系统下的分页机制



- PML4T(Page Map Level4 Table)
- PDPT (Page Directory Pointer Table)
- PDT (Page Directory Table)
- PT(Page Table)及表内额PTE结构

每个table entry 的结构都是8个字节64位宽，而virtual address中每个索引值都是9位

64位转换

0x00401000

PML4E→0

PDPE→0

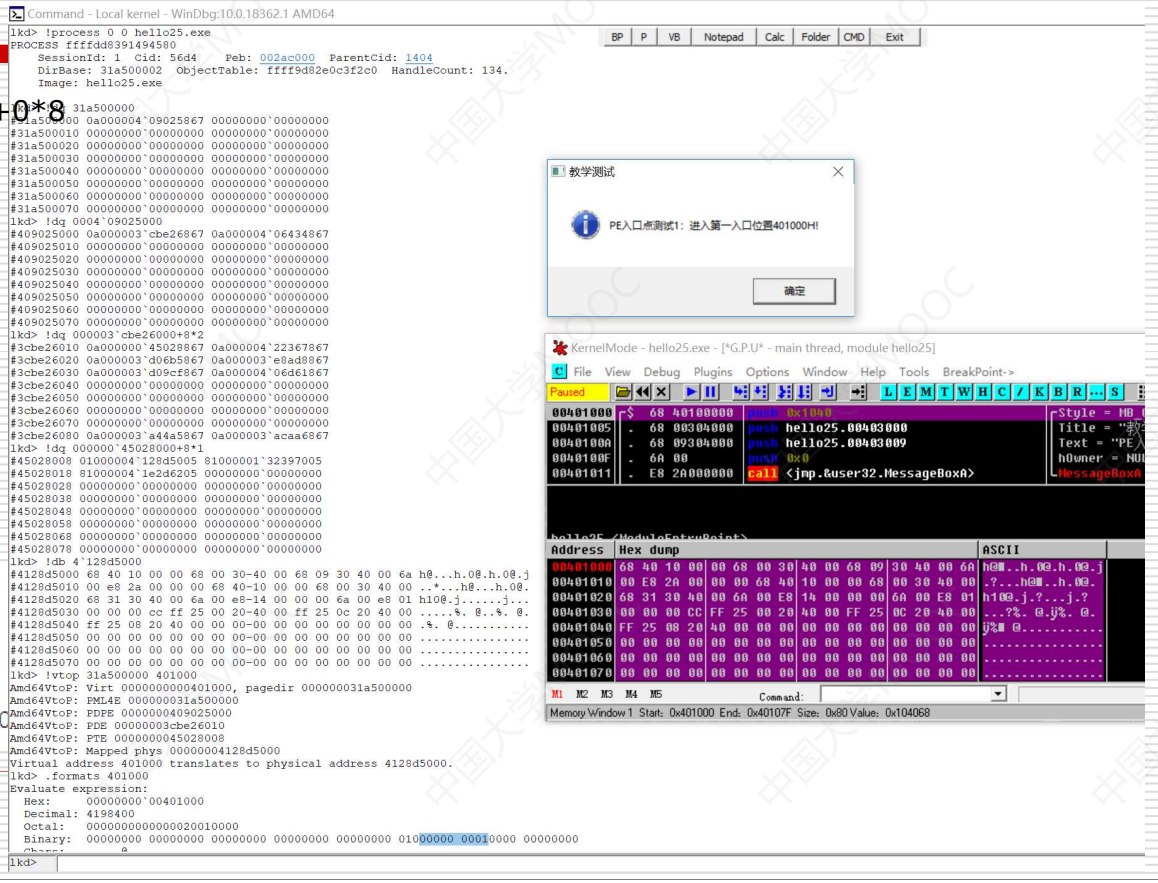
PDE→2

PTE→1

页内地址：0

```
lkd> .formats 401000
Evaluate expression:
  Hex:     00000000`00401000
  Decimal: 4198400
  Octal:   0000000000000020010000
  Binary:  00000000 00000000 00000000 000
  Chars:        @
```

+0*8



课后作业

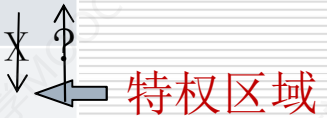
- 在自己电脑上查看hello-2.5.exe程序0x00401000h对应的物理地址，并验证（物理地址与虚拟地址对应的数据一致）。

2.3.2 Windows内存页面权限

- 内存空间的特权区域
- 内存页面权限管理

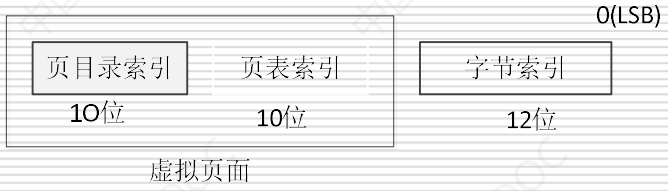
内存空间的特权区域

32位	64位	Itanium	模式	访问
2G	8192G	7152G	用户态（ring3）	区域内
2G	6657G	6144G	内核态（ring0）	区域内

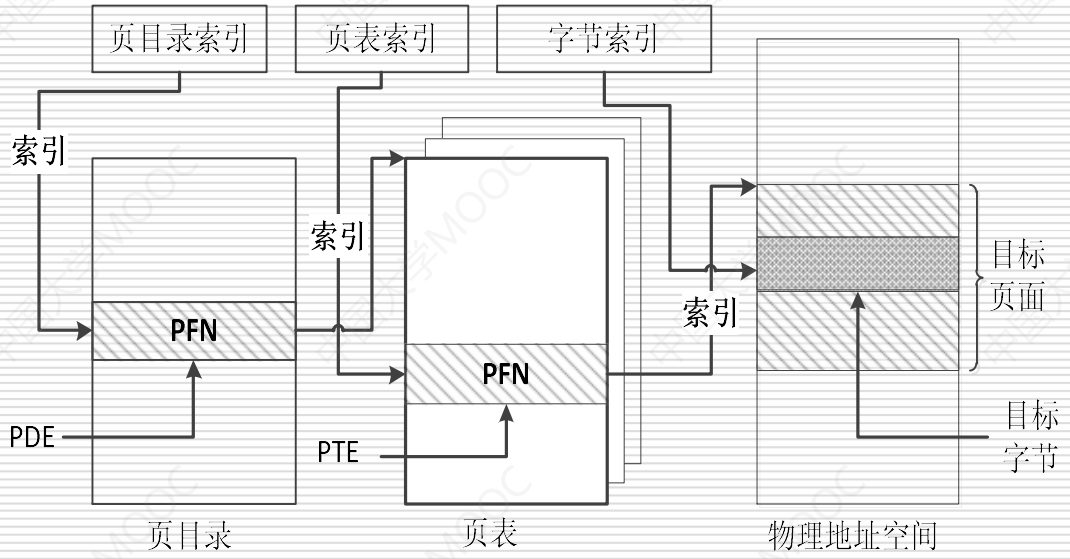


- 内核态的数据和代码是进程间共享的
  - 把权限从用户态提升到内核态的漏洞称为提权漏洞。

内存页面权限管理

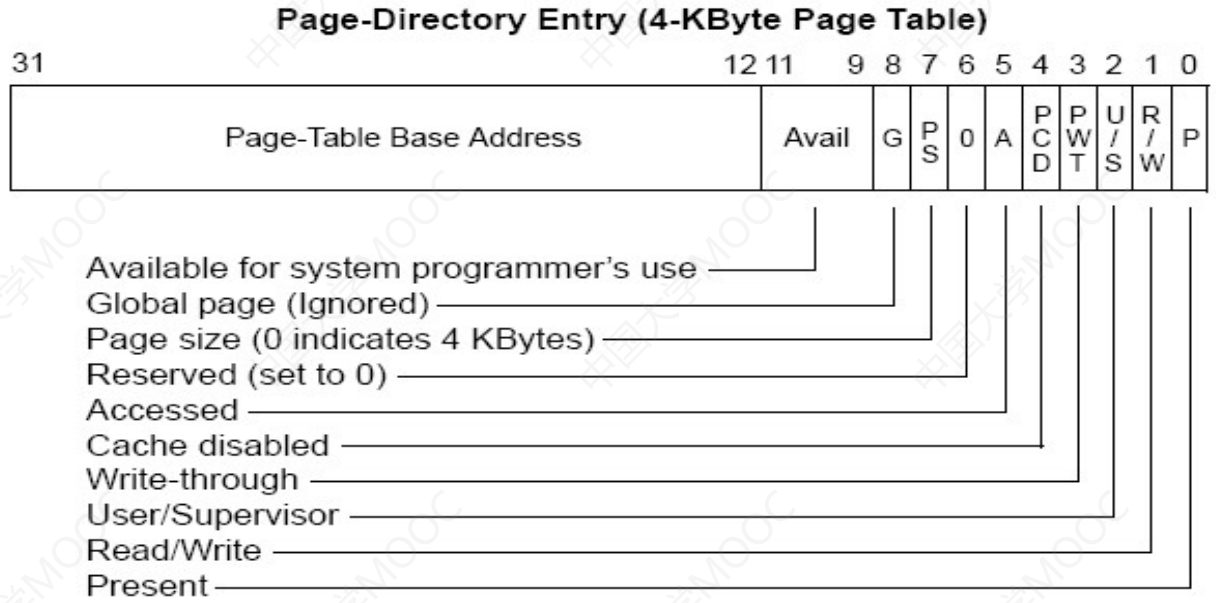


- 页面权限由PDE和PTE属性值设定
  - PDE有1024个
  - PTE有1024*1024个



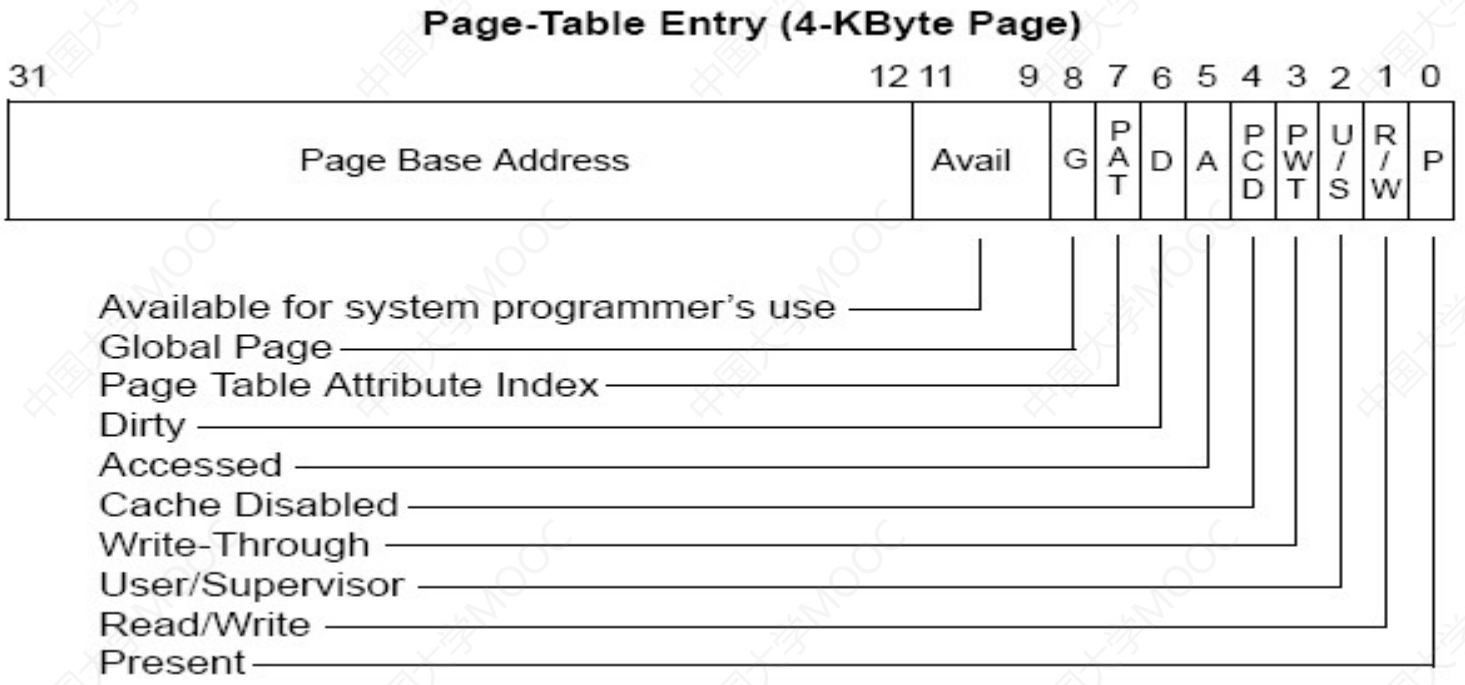
1024 PDE * 1024 PTE = 2^{20} Pages

内存页面权限管理



- R/W：读/写
- U/S：用户态/核心态

内存页面权限管理



- R/W：读/写
- U/S：用户态/核心态

内存页面权限管理

```
typedef struct _PTE
{
        ULONG Present          :1;
        ULONG Writable         :1;
        ULONG Owner            :1;
        ULONG WriteThrough     :1;
        ULONG CacheDisable     :1;
        ULONG Accessed         :1;
        ULONG Dirty            :1;
        ULONG LargePage        :1;
        ULONG Global           :1;
        ULONG ForUse1          :1;
        ULONG ForUse2          :1;
        ULONG ForUse3          :1;
        ULONG PageFrameNumber  :20;

} PTE, *PPTE;
```

```
typedef struct _HARDWARE_PTE
 {
    ULONG64 Valid:          1;
    ULONG64 Write:          1;
    ULONG64 Owner:          1;
    ULONG64 WriteThrough:1;
    ULONG64 CacheDisable:1;
    ULONG64 Accessed:       1;
    ULONG64 Dirty:          1;
    ULONG64 LargePage:      1;
    ULONG64 Global:         1;
    ULONG64 CopyOnWrite:1;
    ULONG64 Prototype:      1;
    ULONG64 reserved0:      1;
    ULONG64 PageFrameNumber:28;
    ULONG64 reserved1:      12;
    ULONG64 SoftwareWsIndex:11;
    ULONG64 NoExecute:      1;
} HARDWARE_PTE,
```

内存页面权限管理

	Read	Write	Execute	Read-on-write	Copy-on-write
PAGE_EXECUTE	-	-	Y	-	-
PAGE_EXECUTE_READ	Y	-	Y	-	-
PAGE_EXECUTE_READWRITE	Y	Y	Y	-	-
PAGE_EXECUTE_WRITECOPY	-	-	Y	Y	Y
PAGE_READONLY	Y	-	-	-	-
PAGE_NOACCESS	-	-	-	-	-
PAGE_READWRITE	Y	Y	-	-	-
PAGE_WRITECOPY	-	-	-	-	Y

BOOL VirtualProtect(LPVOID lpAddress, DWORD dwSize, DWORD flNewProtect, PDWORD lpflOldProtect);

内存页面权限管理

- 内存的管理函数
  - VirtualAlloc 和 VirtualFree
  - VirtualLock 和 VirtualUnlock
  - VirtualQuery 或 VirtualQueryEx
  - VirtualProtect 或 VirtualProtectEx
- 其他函数
  - GetSystemInfo(at kernel32.dll)
  - GlobalMemoryStatusEx (at kernel32.dll)
  - Module32First/Next (at kernel32.dll)
  - Heap32First/Next (at kernel32.dll)
  - Process32First/Next(…)
  - Thread32First/Next(…)
  - CreateToolhelp32Snapshot(…)

页面权限管理

- 获取进程的所有内存页的属性，并尝试修改它们

基 地 址	类 型	大 小	块	保护属性	描 述
77F80000	映像	483 328	5	ERWC	C：\WINNT\System32\ntdll.dll
77FF6000	空闲	40 960			
78000000	映像	290 816	6	ERWC	C：\WINNT\System32\MSVCRT.dll
78047000	空闲	124 424 192			
7F6F0000	映射	1 048 576	2	ER--	00400000 提交 openshellm.exe
7F7F0000	空闲	8 126 464			00400000 镜像 -R- 4096
7FFB0000	映射	147 456	1	-R--	00401000 镜像 ER- 8192
7FFD4000	空闲	40 960			00403000 镜像 WCOPY 4096
7FFDE000	私有	4096	1	ERW-	00404000 镜像 -R- 8192
7FFDF000	私有	4096	1	ERW-	00406000 镜像 -RW 8192
7FFE0000	私有	65 536	2	-R--	00408000 镜像 WCOPY 8192
					0040A000 镜像 -R- 20480