

《操作系统》课程

第三章 内存管理

授课教师：孙海龙

82339063, sunhl@buaa.edu.cn

2024年春季，北航计算机学院

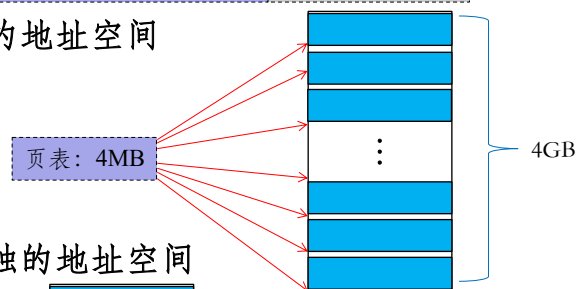
1

思考：页表的虚拟地址如何布局？

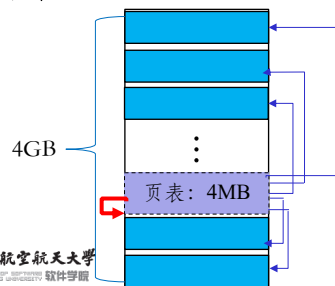
- 以一级页表为例：

页号：20位	偏移：12位
--------	--------

- Case 1: 页表有单独的地址空间



- Case 2: 页表没有单独的地址空间

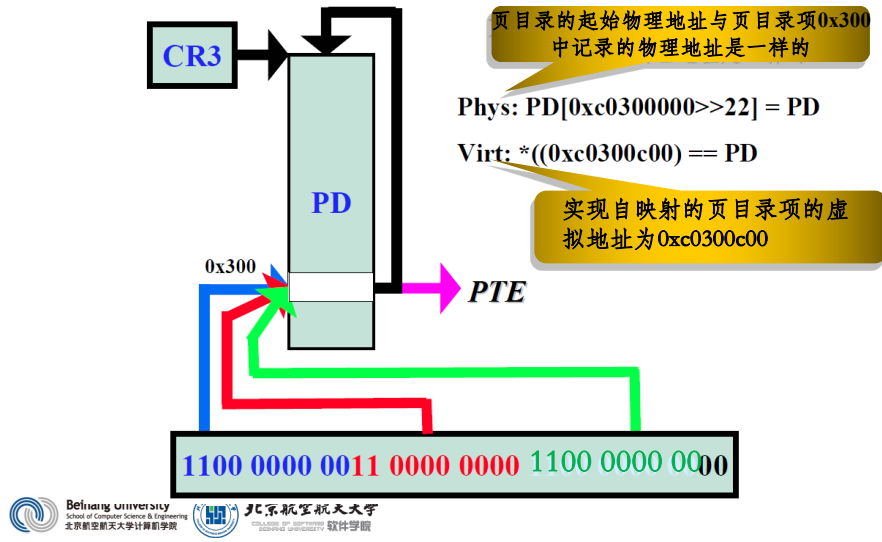


2

2

专题：页目录自映射

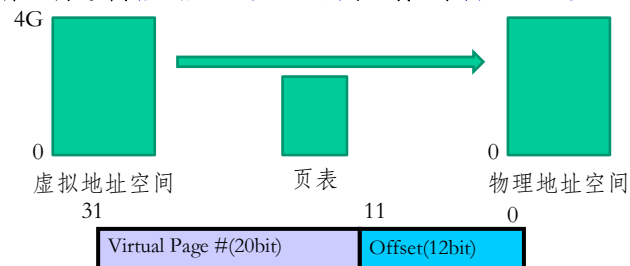
Virtual Access to PageDirectory[0x300]



3

回顾：页式内存管理

- 页表的作用是将虚拟地址空间映射到物理地址空间



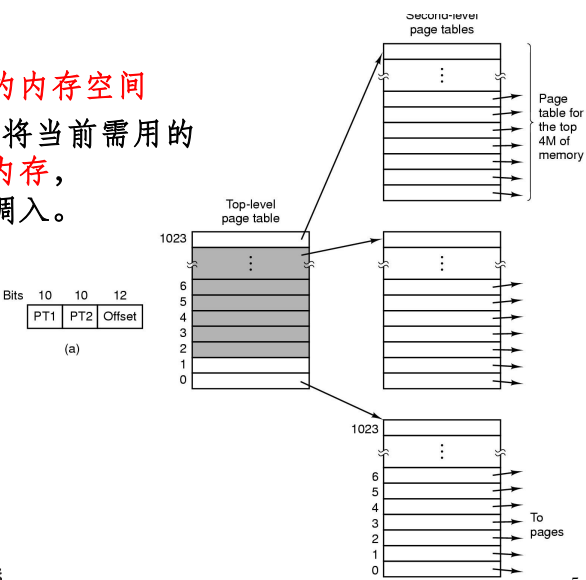
- 32位地址：可寻址空间为4GB
- 采用12位页内偏移，表明内存页大小为4KB
- 每个页表项负责记录1页（4KB）的地址映射关系
- 整个4GB地址空间被划分为 $4GB/4KB=1M$ 页，所以需要1M个页表项来记录逻辑-物理映射关系

4

4

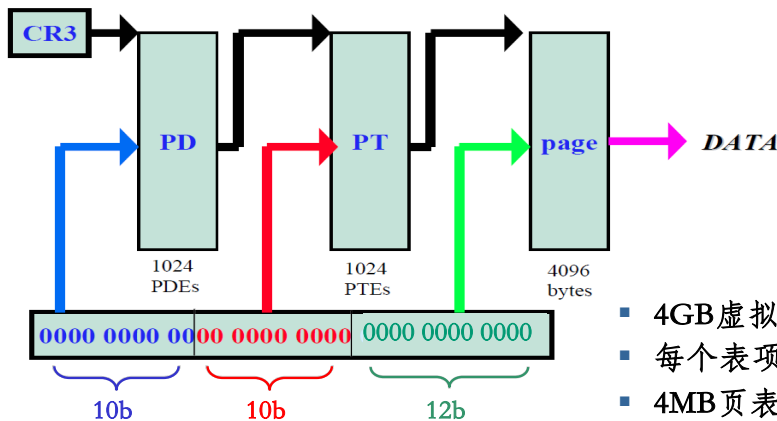
多级页表

- 多级页表的动机
 - 节省页表所占用的内存空间
 - 动态调入页表: 只将当前需用的部分页表项调入内存, 其余的需用时再调入。
- 例: 二级页表
 - 32位地址空间
 - 4K页面大小
 - 1M+1K页表项
 - 地址结构
 - 10 10 12



多级页表

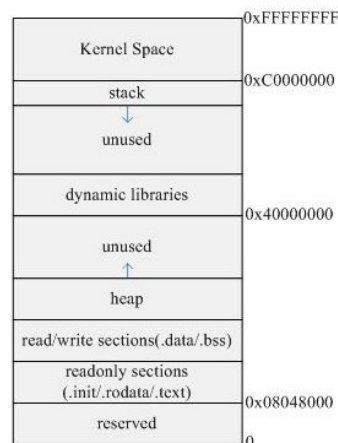
Virtual Address Translation



- 4GB虚拟地址空间
- 每个表项4个字节
- 4MB页表项(PTE)
- 4KB页目录表项(PDE)

页表管理

- 谁来管理（填写）页表？
 - 当然是OS
 - 填写页表目的？
 - 反映内存布局
 - 如何填写、修改页表？
 - 读写页表所在内存
 - 用虚拟地址还是物理地址？ 虚拟地址。



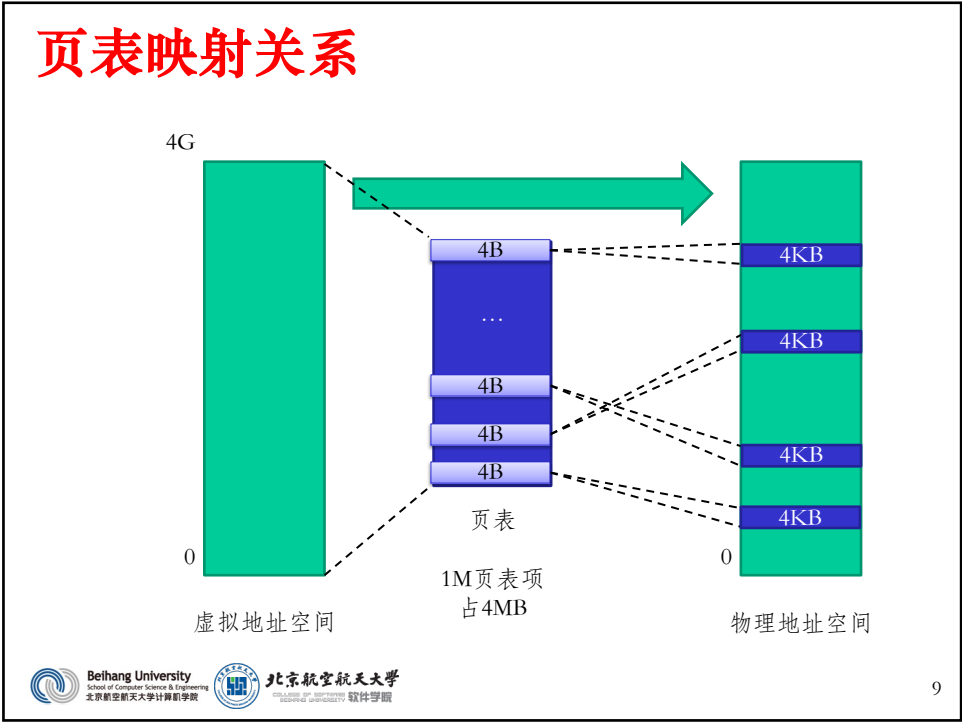
7

9	o	/ Invalid memory	/ /\
10	o	+-----+-----+	-Physics Memory Max
11	o	/ ... / kseg0	
12	o	VPT,KSTACKTOP--> +-----+-----+	-0x8040 0000----end
13	o	/ Kernel Stack / KSTKSIZE	/\
14	o	+-----+-----+	/
15	o	/ Kernel Text /	PDMAP
16	o	KERNBASE --> +-----+-----+	-0x8001 0000 /
17	o	/ Interrupts & Exception / \/\	\/\
18	o	ULIM --> +-----+-----+	-0x8000 0000-
19	o	/ User VPT / PDMAP	/\
20	o	UVPT --> +-----+-----+	-0x7fc0 0000 /
21	o	/ PAGES / PDMAP	/
22	o	UPAGES --> +-----+-----+	-0x7f80 0000
23	o	/ ENVS / PDMAP	
24	o	UTOP,UENVS --> +-----+-----+	-0x7f40 0000
25	o	UXSTACKTOP -/ user exception stack / BY2PG	
26	o	+-----+-----+	-0x7f3f f000
27	o	/ Invalid memory / BY2PG	
28	o	USTACKTOP --> +-----+-----+	-0x7f3f e000
29	o	/ normal user stack / BY2PG	
30	o	+-----+-----+	-0x7f3f d000 /
31	a	/ /	/
32	a	.	/
33	a	.	kuseg
34	a	.	/
35	a	.	/
36	a	/-----/	/
37	a	/ /	/
38	o	UTEXT --> +-----+	
39	o	/ /	2 * PDMAP \/\
40	a	0 ----->	
41	o		
42	*/		

只有4MB，一级页表（即页目录）的4KB去哪里了？

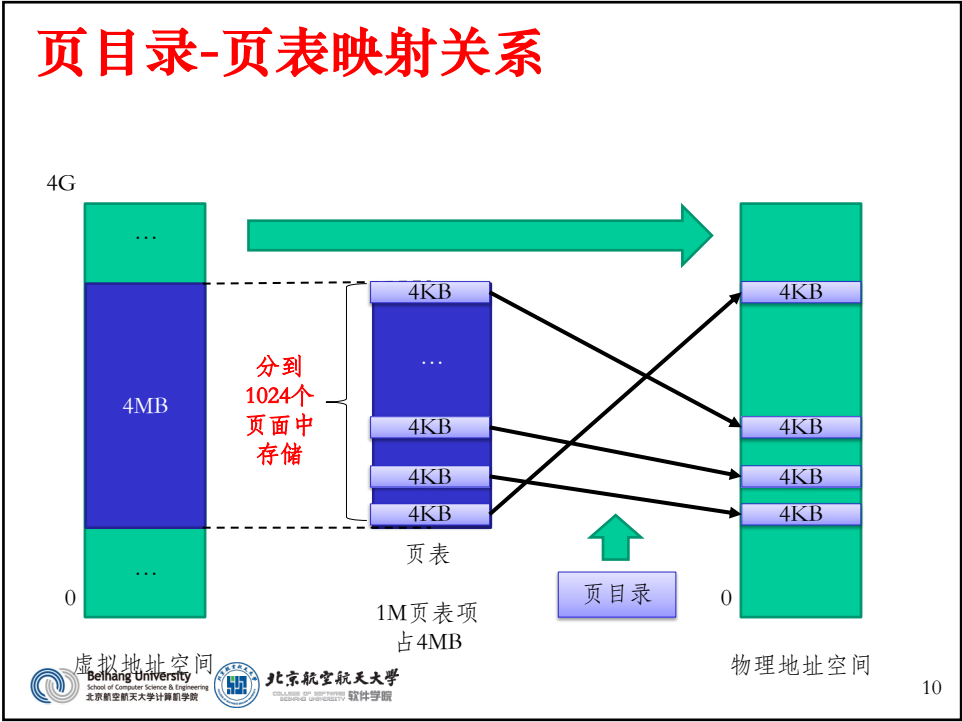
8

页表映射关系



9

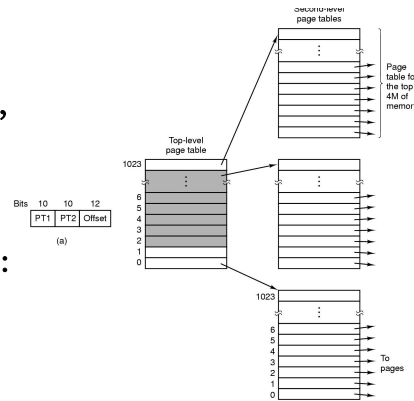
页目录-页表映射关系



10

页目录

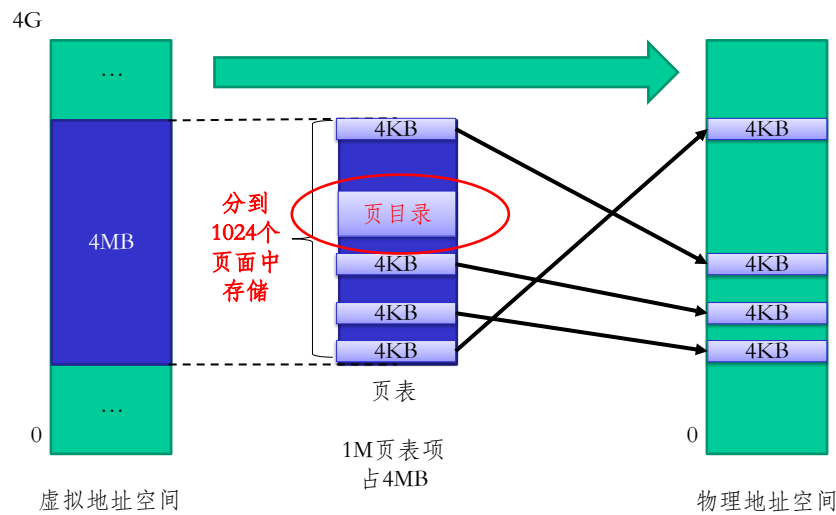
- 页目录定义：页表页的地址映射
 - 1024个页表页逻辑上连续，物理上可以分散，其对应逻辑-物理映射关系记录在页目录中
 - 页目录占1页（4KB）空间，有1024项（页目录项），每一项指向一个页表页
 - 每一页目录项对应4MB内存，1024个页目录项正好对应4GB内存（整个地址空间）
 - 页目录和页表一共应该占据：
4KB + 4MB地址空间



页目录自映射

- 关键点
 - 存储页表的4MB地址空间中是整个4GB虚拟地址空间中的一部分，OS设计者可规定其所在位置（4MB对齐）
 - 一方面根据页目录的定义：记录这4MB（连续）地址空间到物理地址空间映射关系的，是一个4KB的页目录
 - 另一方面根据页表页的定义：记录这4MB（连续）地址空间到物理地址空间映射关系的，是一个4KB的页表页（当然，它属于整个4MB页表的一部分）
 - 所以，页目录和上述页表页内容相同，页目录无需额外分配单独的存储空间
- **4MB + 4KB → 4MB?**

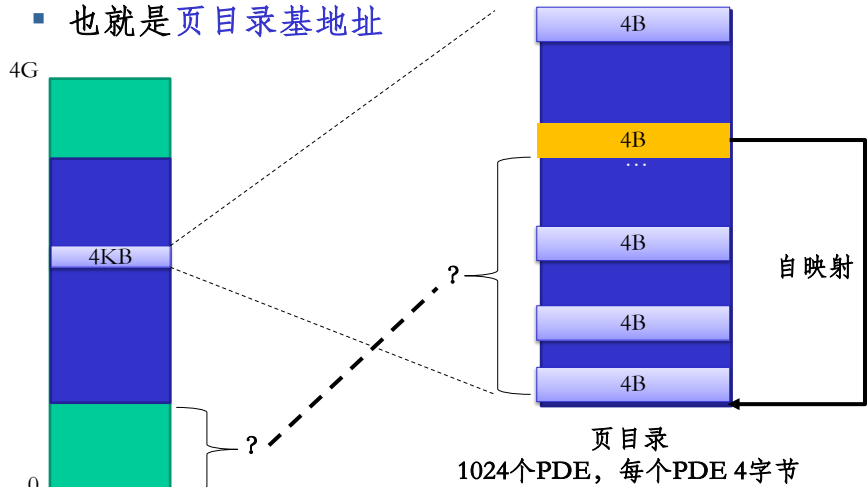
页目录自映射



13

页目录自映射

- 自映射：页目录中有一条PDE指向自身物理地址
- 也就是页目录基地址



14

构建方法

1. 给定一个页表基址 PT_{base} ，该基址需4M对齐，即：

$$\mathbf{PT}_{\text{base}} = ((\mathbf{PT}_{\text{base}}) \gg 22) \ll 22;$$

不难看出， PT_{base} 的低22位全为0。

- ## 2. 页目录表基址PD_{base}在哪里?

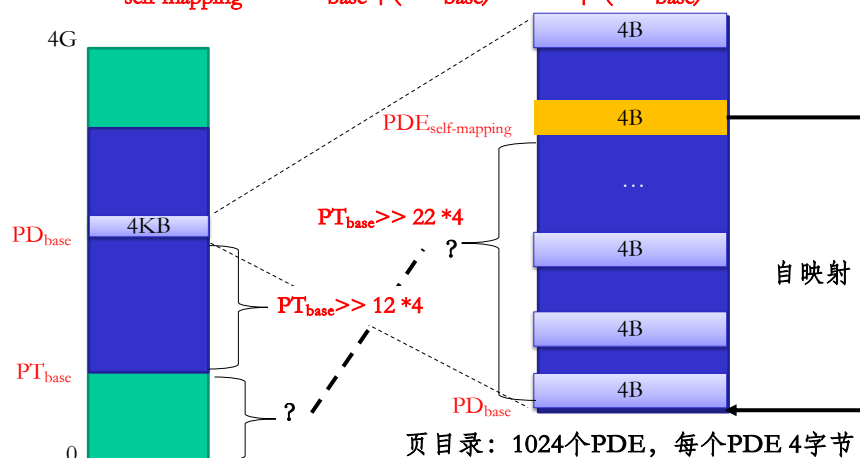
$$PD_{base} = PT_{base} \mid (PT_{base}) \gg 10$$

- ### 3. 自映射目录表项PDE_{self-mapping}在哪里?

$$\text{PDE}_{\text{self-mapping}} = \text{PT}_{\text{base}} \mid (\text{PT}_{\text{base}}) \gg 10 \mid (\text{PT}_{\text{base}}) \gg 20$$

页目录自映射

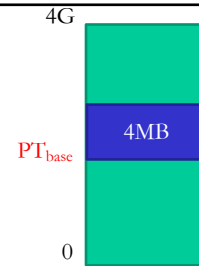
- $PD_{\text{base}} = PT_{\text{base}} \mid (PT_{\text{base}}) \gg 10$
- $PDE_{\text{self-mapping}} = PT_{\text{base}} \mid (PT_{\text{base}}) \gg 10 \mid (PT_{\text{base}}) \gg 20$



思考

- 是不是一定要4M对齐？
- 如果仅考虑映射关系，不是必须的。
- 采用4M对齐，可使页目录表单独地存在于一个页面（页表）中，从使用方便性的角度，是必须的。
- 采用4M对齐，还可以简化计算，各部分地址可以采取“拼接”的方式。
- 思考：4MB对齐的地址有什么特点？以下哪个地址是4MB对齐的

• 0x7fc0 0000, 0x7fd0 0000, 0x8020 0000, 0x1000 0000



特别强调

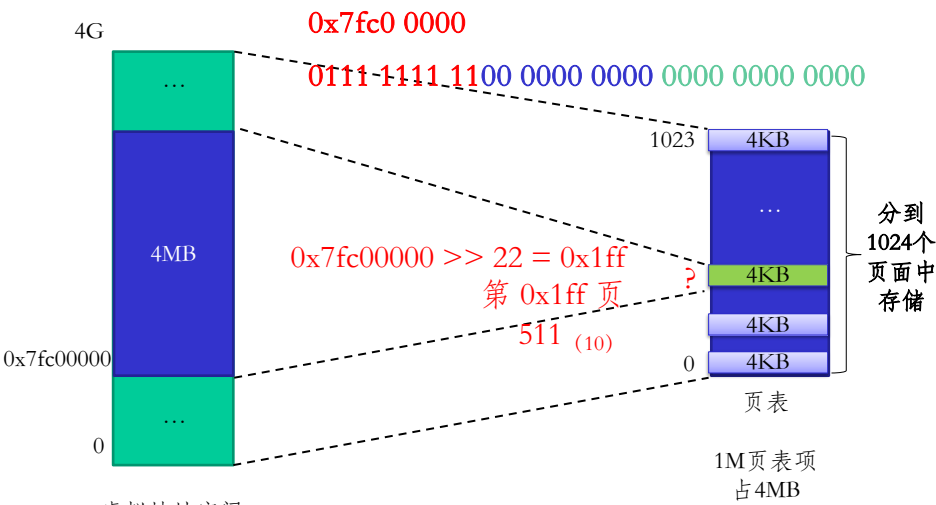
- 只要给定4M对齐的页表基址（虚拟地址），就可以得到所有页表项对应的地址，也就包括页目录表基址和自映射页目录项在页目录表中的位置。因此页目录表基址和自映射页目录项在虚空间中是**计算**出来的。
- 页表主要供OS使用的，因此页表和页目录表通常放置在OS空间中（如Win的高2G空间）；
- “页目录自映射”的含义是页目录包含在页表当中，是我们采用的映射（或组织）方法的一个特征，**是虚拟地址空间内的映射，与虚拟地址到物理地址的映射无关！**
- 支持“页目录自映射”可节省4K（虚拟地址）空间

页目录自映射

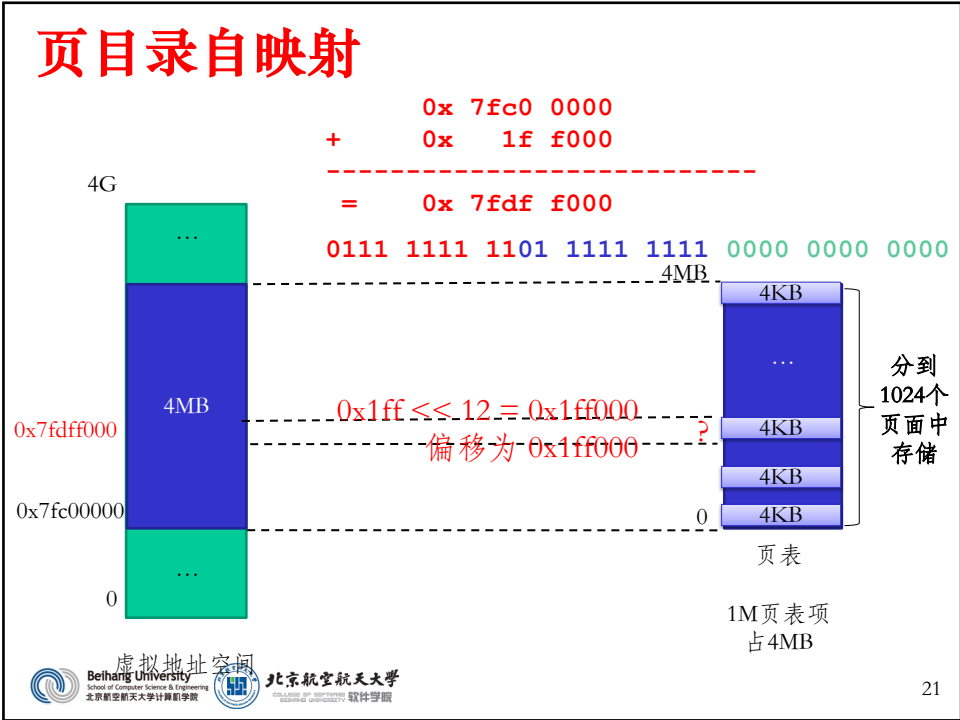
- 举例：页目录在哪？
 - 给定页表虚拟地址起始位置，例如0x7fc00000
 - 可知，从这个地址开始的4MB是存储页表的空间
 - 这4MB地址空间是整个4GB地址空间中第（ $0x7fc00000 \gg 22$ ）个4MB地址空间，因此其逻辑-物理映射关系应该记录在第（ $0x7fc00000 \gg 22$ ）个页表页中

19

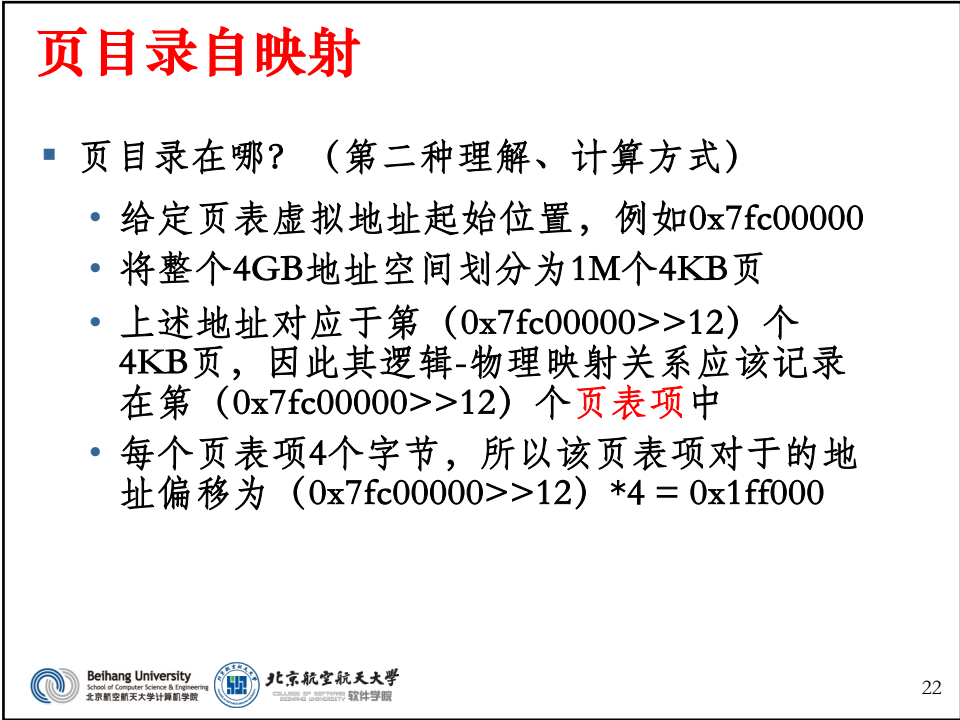
页目录自映射



20

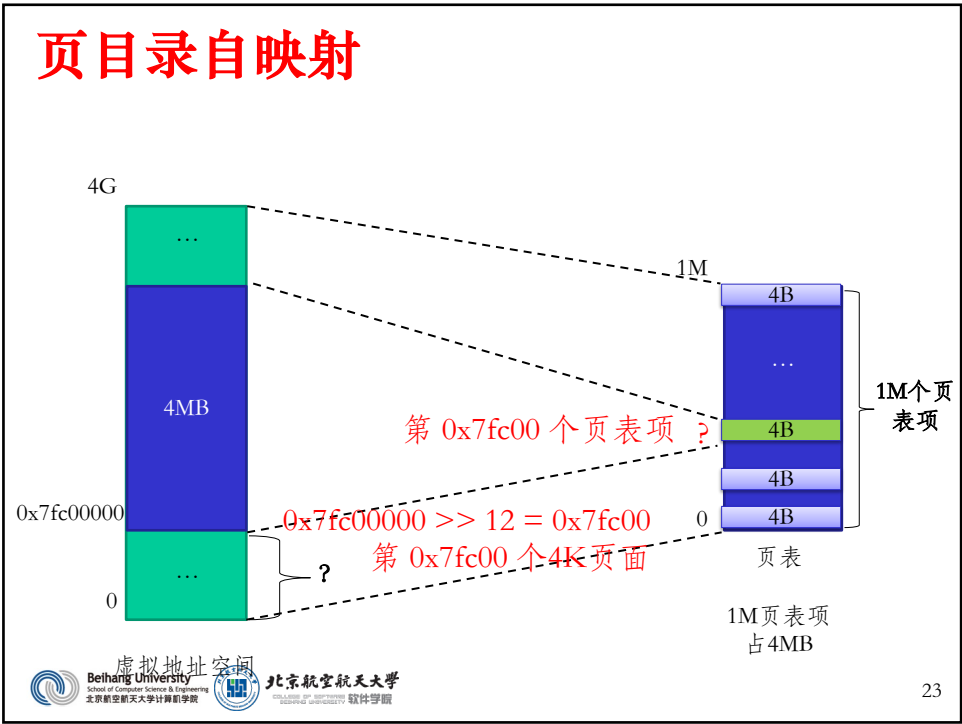


21



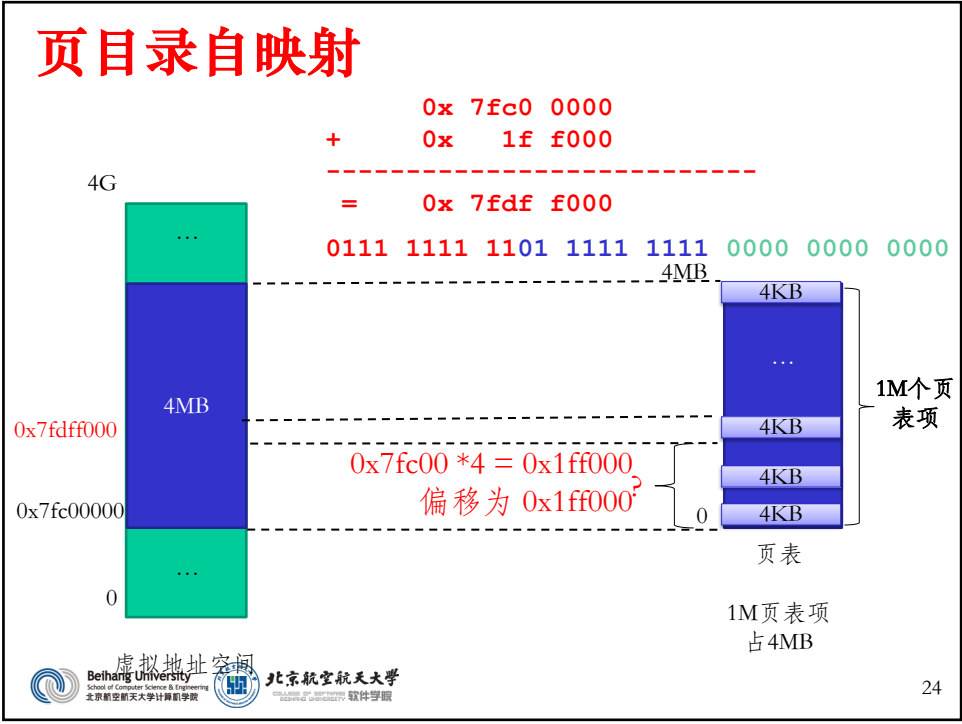
22

页目录自映射



23

页目录自映射



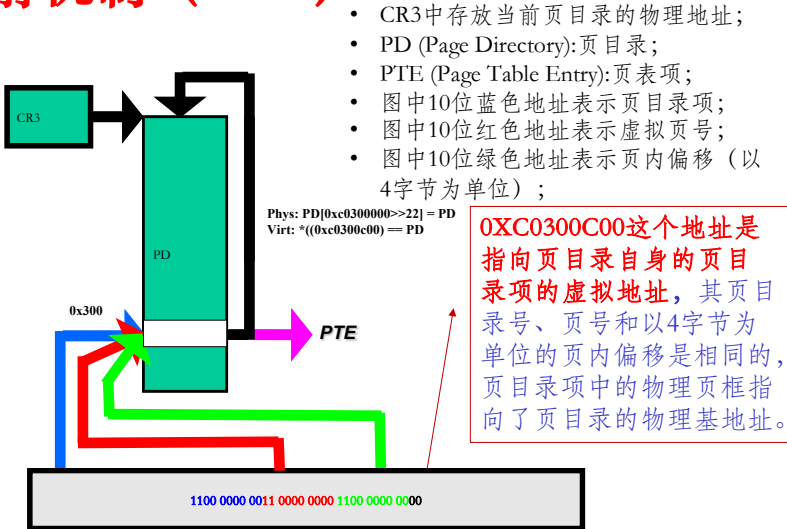
24

页目录自映射

- 简化计算
 - 对于32位地址字长，2级页表，4KB页面大小
 - 给定页表起始地址（虚拟地址，4MB对齐）b
 - 页目录起始地址 = $b + (b >> 10) = b + b/1024$
- 练习：
 - 页表起始地址0x80000000，页目录起始地址=?
 - $0x80000000 + 0x200000 = 0x80200000$
- 反过来：如果给定页目录起始地址，求页表起始地址？
 - E.g. 页目录起始地址0xC0300000，页表起始？

25

自映射机制（X86）



请参考：<http://i-web.i.u-tokyo.ac.jp/edu/training/ss/msprojects/data/07-ProcessesThreadsVM.pdf>

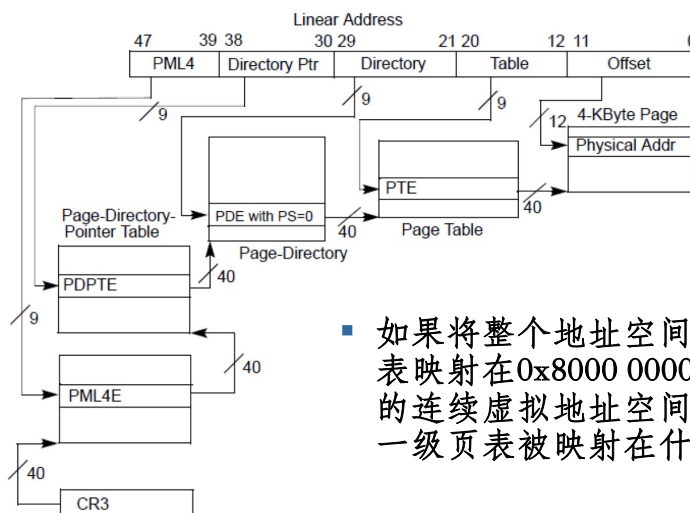
26

Windows下地址转换举例：

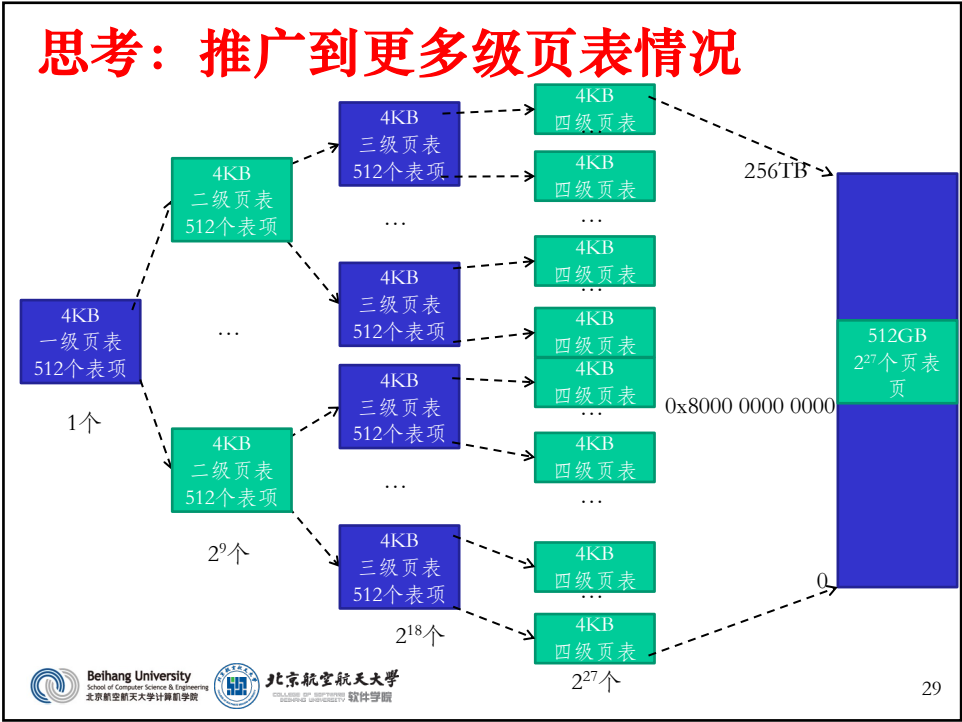
- 给定一个虚拟地址 $va = 0x0012F980$ ，如何访问对应物理内存中的内容。
 1. 转换成二进制： $va = 0b\ 00000000\ 00\ 010010\ 1111\ 1001\ 10000000$
 2. 页目录索引 = $0x0$, 页表索引 = $0x12F$, 偏移 = $0x980$
 3. 页表项基地址 $PTE_BASE = 0xC0000000$ (虚地址)
 4. va 对应的页表项虚地址 = $PTE_BASE + \text{页目录索引} * 4KB + \text{页表索引} * 4 = 0xC00004BC$ (相当于 $PTE_BASE + va \gg 12 \ll 2$, 注意: $va \gg 12 \ll 2$ 与 $va \gg 10$ 不同)
 5. 假定从 $0xC00004BC$ 中得到的内容为 $0x09DE9067$, 其中低12位为状态标志, 高20位为页框号 (即 $0x09DE9$)
 6. va 对应的物理地址为 $(0x09DE9 \ll 12) + 0x980 = 0x09DE9980$

思考：推广到更多级页表情况

- 一个48位页式存储系统，采用4级页表：



- 如果将整个地址空间对应的页表映射在 $0x8000\ 0000\ 0000$ 起始的连续虚拟地址空间上，则第一级页表被映射在什么地址？



29

思考：推广到更多级页表情况

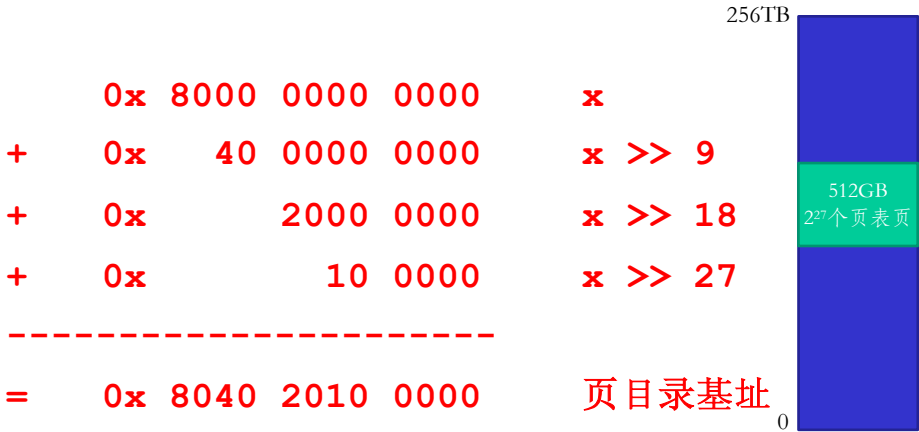
- 整个地址空间大小： $256\text{ TB} = 2^{48}\text{ B}$
- 页大小4KB，每个页表项占8字节
- 页表数量：共有1个一级页表， 2^9 个二级页表， 2^{18} 个三级页表， 2^{27} 个四级页表
- 2^{27} 个四级页表映射在整个地址空间中一段连续的 2^9 GB 地址空间上，起始地址是 $0\text{x}8000\ 0000\ 0000$
- 实际上， 2^{18} 个三级页表也是连续的，是四级页表的一个子集； 2^9 个二级页表也是连续的，是三级页表的一个子集（当然也是整个四级页表的子集）；1个一级页表是上述二级页表的子集（当然最终也是整个四级页表的子集）。也就是说，那个一级页表是整个 2^{27} 个四级页表中的一个。**问题是：是哪个？**

30

30

思考：推广到更多级页表情况

- 因为：页表映射起始地址是0x8000 0000 0000



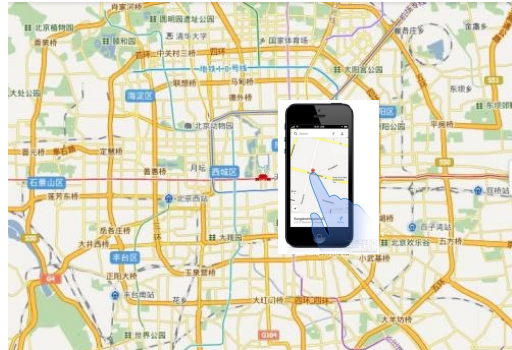
小结

- 自映射是Windows操作系统对内存管理的一种实现机制
- 对于32位地址空间来说：
 - 前10位用于指定页目录号，因此一共有1个页目录，其中包括1024个页目录项，即对应1024个页表；页目录一共占1024*4 = 4KB空间。
 - 中间10位用于指定页号，因此每个页表有1024个页表项，一共有1024*1024个页表项；页表一共占1024*1024*4 = 4MB空间。
 - 在自映射机制下，从4GB地址空间中拿出4MB空间存放1024个页表；同时，从1024个页表中拿出一个页表用于存放页目录；这样页目录和页表一共占4MB，而这4MB空间又是4GB空间的一部分，不必单独分配4MB + 4KB来存放页目录和页表。
- 核心问题：1024个页表中哪个页表用来存放页目录？页目录中的哪个页目录项指向页目录自身？

要求：保证虚拟地址是线性、连续的

自映射的数学意义

- 地图的比喻
 - 手持北京地图在北京
 - 必有地图上一点与其表示的地理位置与该点的实际地理位置重合
- 不动点： $f(x) = x$
 - 压缩映像



Q&A

微信群/课程中心论坛
sunhl@buaa.edu.cn

基本功练习1

- 1 KB = 1024 B = 2^{10} B
- 4 KB = 4096 B = 2^{12} B
- 1MB = 2^{20} B
- 4MB = 2^{22} B
- 2^{16} B = 64KB
- 2^{32} B = 4GB
- 2^{64} B = 16EB
- 2^{10} B = 1KB
- 2^{20} B = 1MB
- 2^{30} B = 1GB
- 2^{40} B = 1TB
- 2^{50} B = 1PB
- 2^{60} B = 1EB
- 2^{70} B = 1ZB

以下用最大字节单位

基本功练习2

- 十六进制数0x12345678转成二进制有29位
- $0x12345678 \gg 12 = \underline{0x12345}$
- $0x80000000 \gg 22 = \underline{0x200}$
- $0x1000 = \underline{4096}_{(10)} = \underline{4} \text{ K}$
- $16\text{K} = 0x \underline{4000}$; $64\text{K} = 0x \underline{10000}$
- $1\text{M} = 0x \underline{100000}$; $4\text{M} = 0x \underline{400000}$
- $1\text{G} = 0x \underline{40000000}$; $2\text{G} = 0x \underline{80000000}$
- $3\text{G} = 0x \underline{C0000000}$; $4\text{G} = 0x \underline{100000000}$
0

页目录自映射

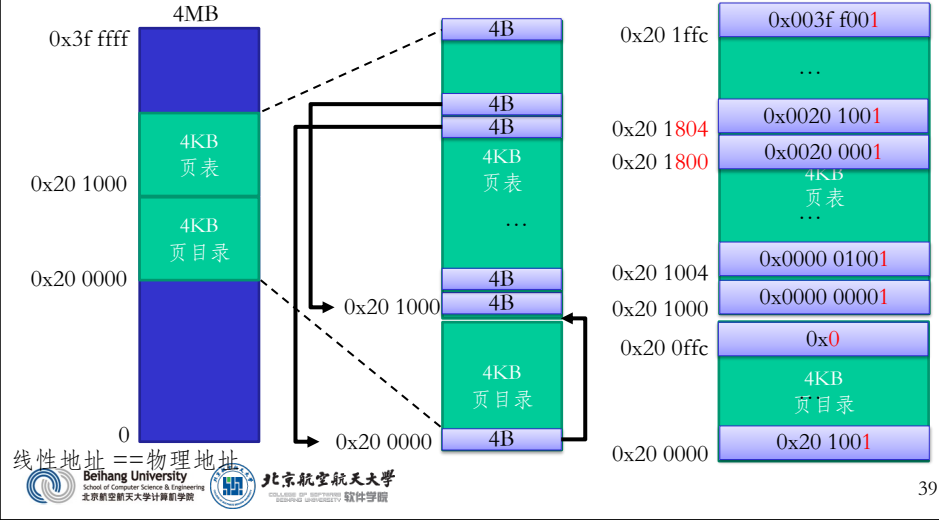
- 页表在虚拟地址空间中映射
 - 每个页表项需要4字节，所以1M个页表项需要4MB，所以整个页表占用的地址空间大小就是4MB
 - 4MB页表也要分页存储，共需要 $4\text{MB}/4\text{KB}=1024$ 个页面存储（页表页）
 - 每一页中存储 $4\text{KB}/4\text{B}=1024$ 项页表项
 - 由于1个页表项对应4KB内存，所以每1个页表页对应 $1024*4\text{KB}=4\text{MB}$ 内存

X86初始系统页表建立

- X86 CPU引导时处于实模式。
- 开启分页（Enable Paging）前，CPU使用物理地址寻址
- 进入保护模式后，可以开启分页，此时CPU将开始使用线性地址寻址。线性地址需要由MMU根据页表翻译成物理地址。
- 问题：怎么实现切换？
- 一个思路：事先构造页表，初始化一部分线性地址空间，使得该空间内的虚拟地址等于物理地址。
 - 例如：虚拟地址 $0\text{x}0000\ 0000 \sim 0\text{x}0040\ 0000$ （4MB）直接映射到物理地址 $0\text{x}0000\ 0000 \sim 0\text{x}0040\ 0000$ （4MB）
 - 页表怎么构造？

X86初始系统页表建立

- 在0x20 0000地址处分配2个4K页面，存储页目录和页表。启用分页后，如下页表保证线性地址 == 物理地址（4MB以内）



39

内容回顾

- 1. 什么是belady异常？为什么会产生belady异常？
- 2. 虚拟内存管理中的“抖动”是一种什么现象？产生的原因是什么？
- 3. 虚拟内存管理中的“工作集”是指什么？

40

40