北京航空航天大学
Beijing University of Aeronautics and Astronautics

《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

内容提要

- 存储管理基础
- 页式内存管理
  - 基本原理
  - 基本概念：页表、地址变换、多级页表、快表
  - 页表类型：哈希页表、反置页表
  - 页共享与保护
- 段式内存管理
- 虚拟内存管理
- 内存管理实例

2

补充知识：程序、进程和作业

- 程序是静止的，是存放在磁盘上的可执行文件
- 进程是动态的。进程包括程序和程序处理对象（数据集），是一个程序的执行过程，是分配资源的基本单位。通常把进程分为系统进程和用户进程：
  - 系统进程：完成操作系统功能的进程；
  - 用户进程：完成用户功能的进程。
- 作业是用户需要计算机完成的某项任务，是要求计算机所做工作的集合。

3

作业、进程和程序之间的联系

- 作业通常包括程序、数据和操作说明书3部分。
- 每一个进程由进程控制块PCB、程序和数据集合组成。这说明程序是进程的一部分，是进程的实体。
- 因此，一个作业可划分为若干个进程来完成，而每一个进程有其实体——程序和数据集合。

4

回顾：分区式内存管理

- 内存的碎片问题：无论固定分区，还是可变分区都普遍存在的问题
  - 内存紧缩
- 程序需要装入连续的物理内存空间
  - 小内存运行大程序：覆盖、交换
- 很难满足程序对内存空间的动态需求
- 很难找到一个绝对有效的内存分配/回收策略
  - First Fit/Next Fit/Best Fit/Worst Fit
  - Buddy System
- 内存利用率低

5

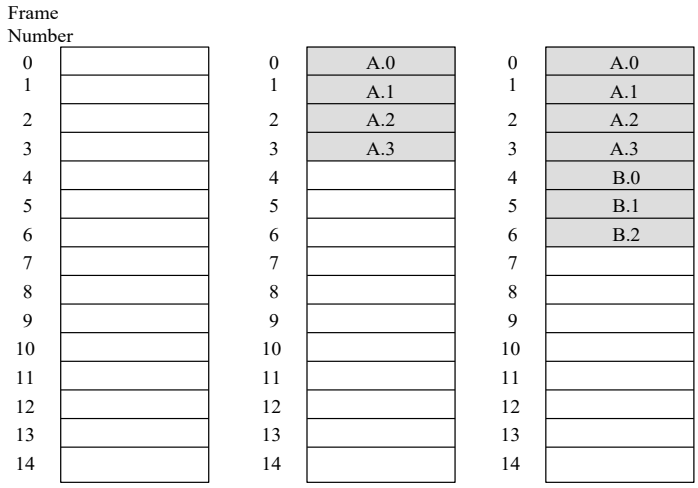
分页式存储管理的基本思想

- 基本思想：把一个逻辑地址连续的程序分散存放到若干不连续的内存区域内，并保证程序的正确执行。
- 带来的好处：既可充分利用内存空间，又可减少移动带来的开销。
- 页式管理首先由英国Manchester大学提出并使用。
  - 1962年建成，世界上最强的超级计算机之一
  - 首次提出了基于页式管理的Virtual memory技术

https://en.wikipedia.org/wiki/Atlas_(computer)

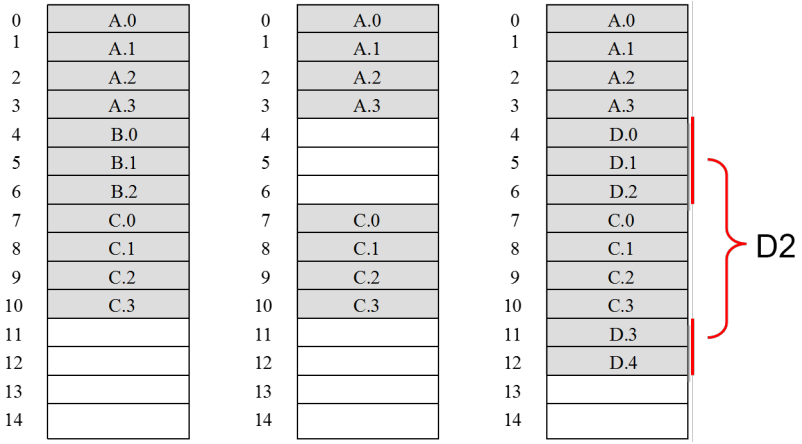
6

页式管理示例



7

页式管理示例（续）



8

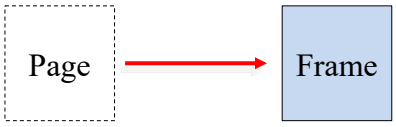
纯分页系统

- 不具备页面对换功能的分页存储管理方式，不支持虚拟存储器功能，这种存储管理方式称为纯分页或基本分页存储管理方式。(v. s. 请求分页)
- 在调度一个作业时，必须把它的所有页一次装到主存的页框内；如果当时页框数不足，则该作业必须等待，系统再调度另外作业。
- 优点：
  - 没有外碎片，每个内碎片不超过页大小
  - 一个程序不必连续存放
  - 便于改变程序占用空间的大小
- 缺点：
  - 程序全部装入内存；有内碎片

9

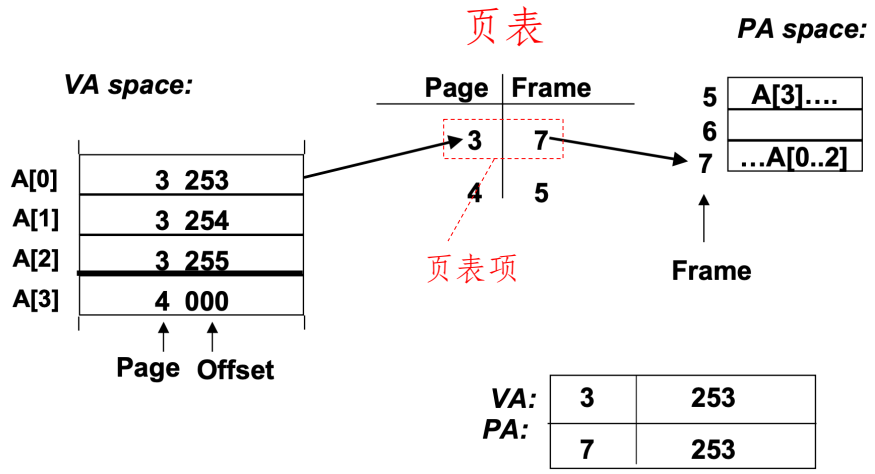
基本概念

- 页：把每个作业的地址空间分成一些大小相等的片，称之为页面（Page）或页，各页从0开始编号。
- 存储块：把物理内存的存储空间也分成与页面相同大小的片，这些片称为存储块，或称为页框（Frame），同样从0开始编号。
- 页表：为便于在内存找到进程的每个页面所对应的页框，分页系统中为每个进程配置一张页表。进程逻辑地址空间的每一页，在页表中都对应有一个页表项。



10

举例：页表、页表项



11

页面的大小

- 页大小（与块大小一样）是由硬件来决定的，通常为2的幂
- 选择页的大小为2的幂可以方便地将逻辑地址转换为页框号和页偏移。
  - 如果逻辑地址空间为2^m，且页大小为2^n单元，那么逻辑地址的高m-n位表示页号（页表的索引），而低n位表示页偏移。
  - 每页大小从512B到16MB不等。
- 现代操作系统中，最常用的页面大小为4KB。
- 练习：32位地址空间，页大小为4KB，用多少位表示页内偏移？多少位表示页号？

页号：20位	偏移：12位

12

页面的大小

若页面较小

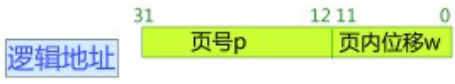
- 减少页内碎片和总的内存碎片，有利于提高内存利用率。
- 每个进程页面数增多，使页表长度增加，占用内存较大。
- 页面换进换出速度将降低。

若页面较大

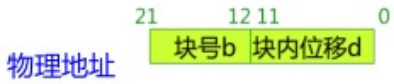
- 增加页内碎片，不利于提高内存利用率。
- 每个进程页面数减少，页表长度减少，占用内存较小。
- 页面换进换出速度将提高。

13

地址结构



例：地址长为 32 位，其中 0-11 位为页内地址，即每页的大小为 2^{12}=4KB；
12-31 位为页号，地址空间最多允许有 2^{20} =1M 页。



例：地址长为 22 位，其中 0-11 位为块内地址，即每块的大小为 2^{12}=4KB，与页相等；
12-21 位为块号，内存地址空间最多允许有 2^{10} =1K 块。

14

地址结构

已知逻辑地址求页号和页内地址

- 给定一个逻辑地址空间中的地址为 A，页面的大小为 L，则页号 P 和页内地址 d（从 0 开始编号）可按下式求得：

$$P = INT\left[\frac{A}{L}\right], d = [A] mod L$$

其中，INT 是整除函数，mod 是取余函数。

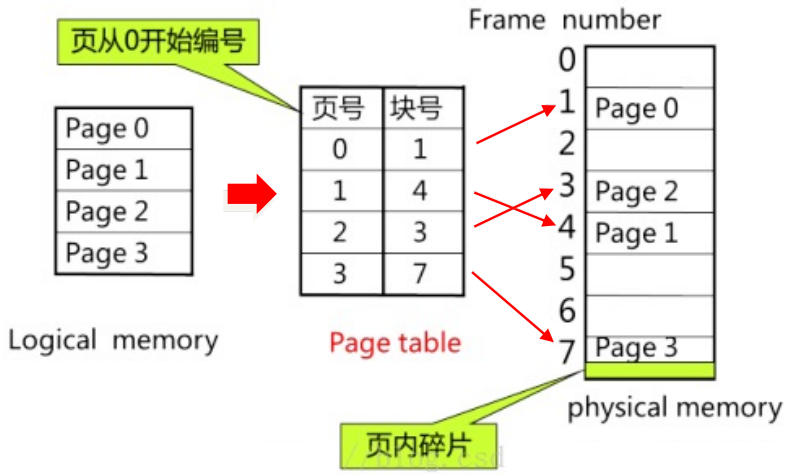
例如：
如果页面大小是4KB，逻辑地址0x a032。对应的页号应该是p=10（十进制数），页内偏移是0x 32 = 0d 50

关于页表

- 页表存放在内存中，属于进程的现场信息。
- 用途：
  1. 记录进程的内存分配情况
  2. 实现进程运行时的动态重定位。
- 访问一个数据需访问内存 2 次（页表一次，内存一次）。
- 页表的基址及长度由页表寄存器给出。

页表始址	页表长度

地址变换——页表查找



17

地址变换机构

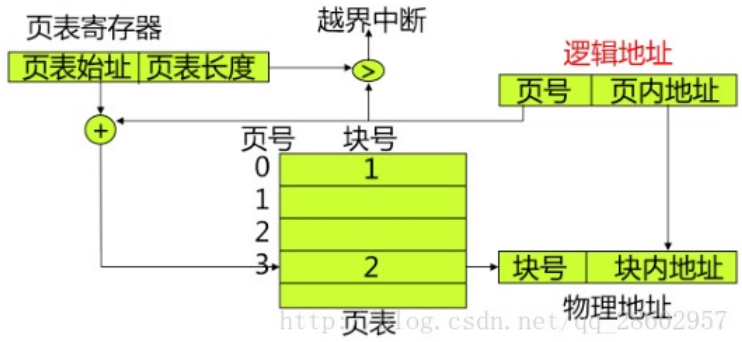
- 当进程要访问某个逻辑地址中的数据时，分页地址变换机构会自动地将有效地址（相对地址）分为“页号”和“页内地址”两部分。
- 将“页号”与“页表长度”进行比较，如果页号大于或等于页表长度，则表示本次所访问的地址已超越进程的地址空间，产生地址越界中断。
- 将“页表始址”与“页号和页表项长度的乘积”相加，得到该页表项在页表中的位置，于是可从页表项中得到该页的物理块号，将之装入物理地址寄存器中。

Frame Number	Present/Absent	Protection	Reference	Caching	Dirty

- 将有效地址寄存器中的“页内地址”送入物理地址寄存器的“块内地址”字段中。

18

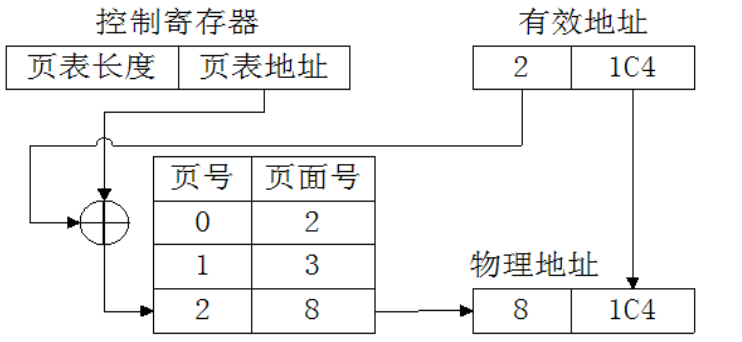
地址转换机构



逻辑地址：把相对地址分为页号和页内地址两部分。
页表定位：页表始址 + 页号 × 页表项长度。
查询页表：读出块号。
物理地址：块号 + 块内地址。 （块内地址 = 页内地址）

19

地址转换例 （21C4→81C4)



① 逻辑地址21C4分为页号2和页内位移1C4
② 根据寄存器所指页表地址与页号2找到对应的 **页框号8**
③ 将8与页内位移1C4合并成物理地址81C4

20

课堂练习

- 假设虚拟地址占8位，物理地址占10位，每个页面的大小是64 Bytes。问：
  - 一共有多少虚拟页面？
  - 一共有多少物理页面（页框）？
  - 每个页表中有多少个页表项？
  - 假定页表是一个数组，[2，5，1，8]，请问对应虚拟地址0d 241的物理地址是多少？

- 解答
  - 页面大小64 = 2^6→8位地址中6位是页内偏移，2位表示页号→4个页面

  - 物理地址10位，页大小是2^6→2^10/2^6=16个物理页框
  - 4个页面→4个页表项
  - 0d 241=0b 11110001→ 页号是3，页内偏移是 0b 110001 → 物理页框8→ 物理地址：0b 1000 110001 = 0d 561 或 0X 231

21

内容回顾

1. 覆盖解决了在有限内存中执行具有大内存需求的程序的问题，举例说明什么是覆盖？
2. 交换可以解决在有限的内存中运行多个程序的问题，交换会引起较大的计算开销，为什么？
3. 页式内存管理属于固定分区分配，还是可变分区分配？
4. 页式管理中虚拟地址是如何构成的？页内偏移的地址位数由什么决定？
5. 页表项中是否需要存放页内偏移？
6. 如果页面很大，会带来什么问题？页面很小呢？

22

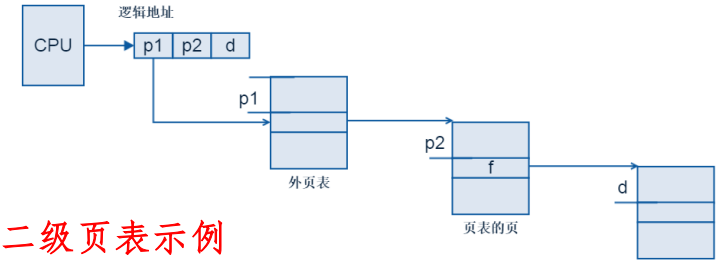
一级页表的问题

- 若逻辑地址空间很大（2^32 ~2^64 ），则划分的页比较多，页表就很大，占用的存储空间大（要求连续），实现较困难。
- 例如，对于 32 位逻辑地址空间的分页系统，如果规定页面大小为 4 KB 即 2^12 B，则在每个进程页表就由高达2^20 页表项组成。设每个页表项占用4个字节，每个进程仅仅页表就要占用 4 MB 的内存空间。
- 解决问题的方法：多级页表
  - 页表分级管理
  - 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。

23

二级页表

- 将页表再进行分页，离散地将各个页表页面存放在不同的物理块中，同时也再建立一个外层页表用以记录页表页面对应的物理块号。
- 正在运行的进程先把外层页表（页表的页表）调入内存，而后动态调入内层页表。只将当前所需的一些内层页表装入内存，其余部分根据需要再陆续调入。



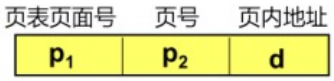
二级页表示例

24

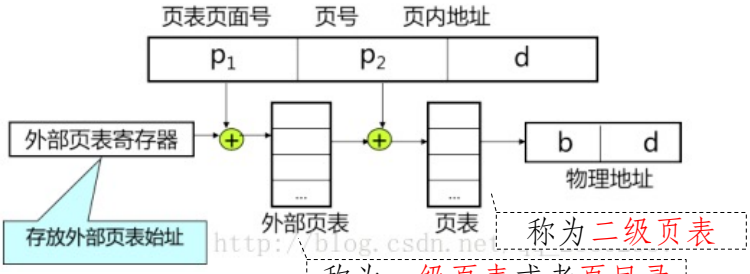
二级页表

对于32位地址空间，页面大小4K，如每个页表项占4个字节，则页表需占4MB空间。
P_1=10，P_2=10，d=12。

- 逻辑地址

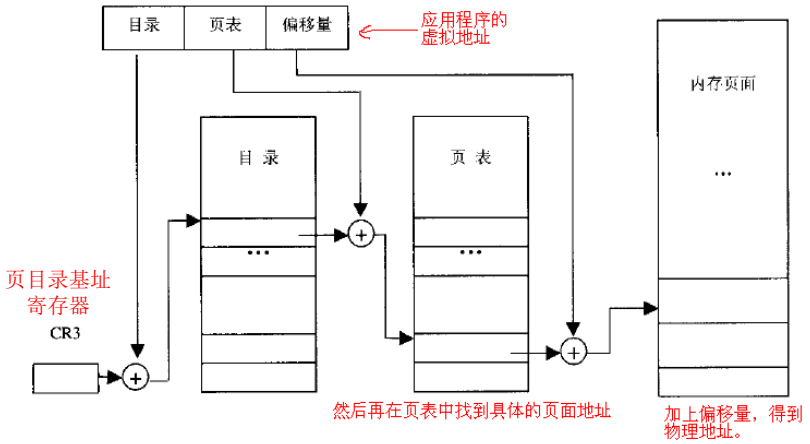


- 地址变换



25

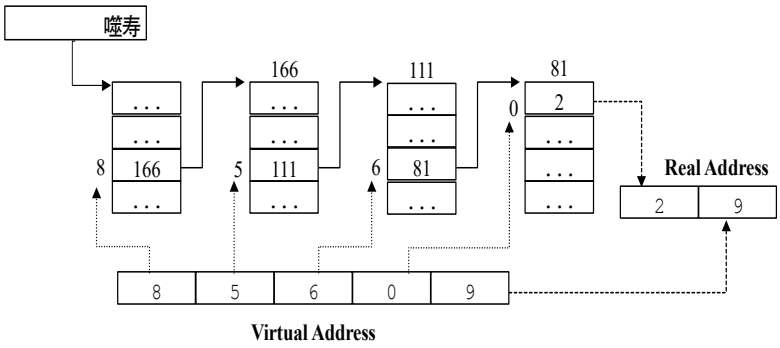
X86 二级页表结构



26

多级页表

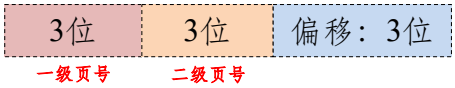
- 多级页表结构中，指令所给出的地址除偏移地址之外的各部分全是各级页表的页表号或页号，而各级页表中记录的全是物理页号，指向下级页表或真正的被访问页。



课堂练习

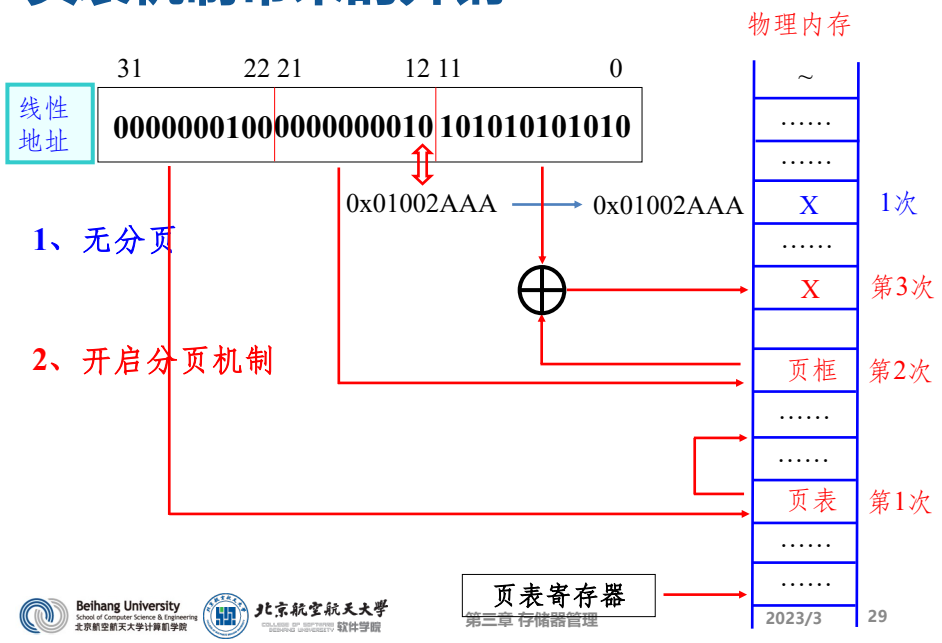
- 页面的大小是8 bytes，物理内存为128 bytes，虚拟地址空间为512 bytes。假设每个页表项占1个字节，设计一个多级页表，并回答如何查询对应虚拟地址0x3A的物理地址。
- 解答：
  - 页面大小是8字节→页内偏移3位

  - 页表项1字节→一个页框存储8个页表项→页表最多管理8个页面→需要3位表示页号



  - 0X3A=00111010，其中00表示查找一级页表中的第0页表项，从中找到对应的二级页表的物理地址，读出相应的物理页框；
  - 111表示查找对应二级页表的第7个页表项，从中找到对应的物理页框号；
  - 上一步得到的物理页框的初始地址+偏移量010即为物理地址。

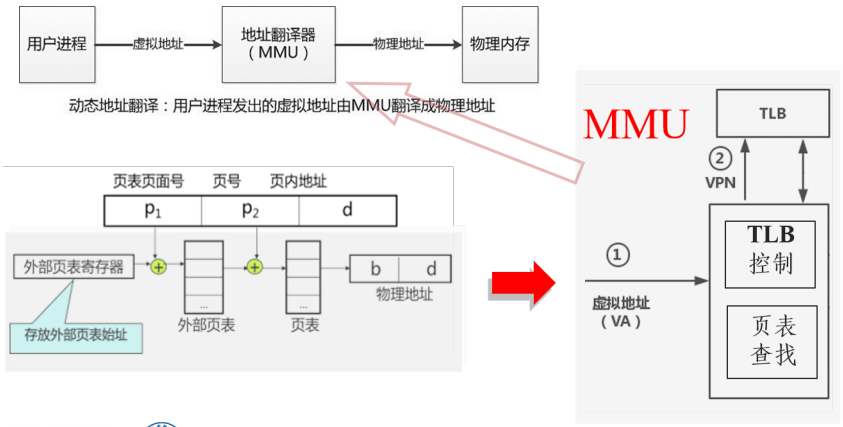
页表机制带来的开销



29

页表机制带来的问题

- 页表机制带来的严重问题就是内存访问效率的严重下降，以二级分页地址机制为例，由不分页时的 1 次，上升到了 3 次，这个问题必须解决。



30

具有快表的地址变换机构

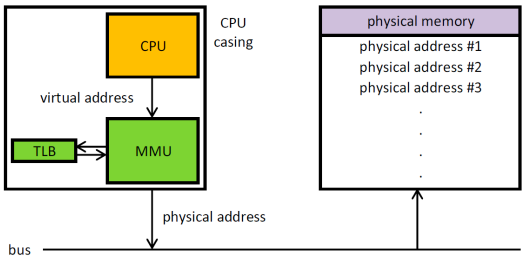
- 快表又称联想存储器（Associative Memory）、TLB（Translation Lookaside Buffer）转换表查找缓冲区，IBM最早采用TLB。
- 快表是一种特殊的高速缓冲存储器（Cache），内容是页表中的一部分或全部内容。
- CPU 产生逻辑地址的页号，首先在快表中寻找，若命中就找出其对应的物理块；若未命中，再到页表中找其对应的物理块，并将相应的页表项复制到快表。若快表中内容满，则按某种算法淘汰某些页。
- 通常，TLB中的条目数并不多，在64~1024之间。

31

具有快表的地址变换机构(续)

- 支持页式管理的硬件部件通常称为MMU（Memory Management Unit）

动态地址翻译：用户进程发出的虚拟地址由MMU翻译成物理地址

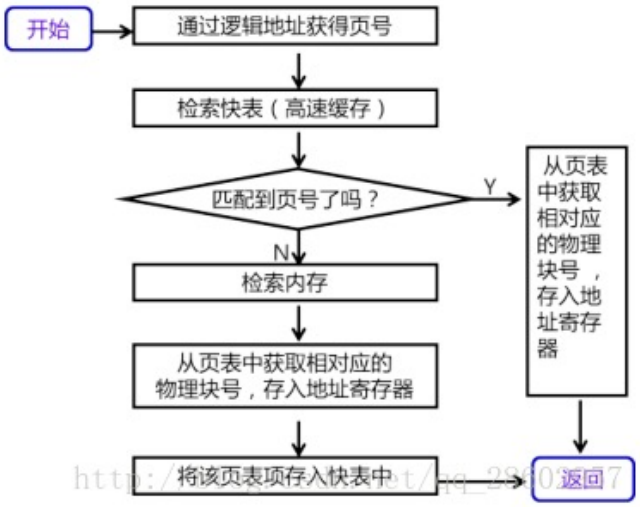


CPU: Central Processing Unit
MMU: Memory Management Unit
TLB: Translation lookaside buffer

By Mdjango, Andrew S. Tanenbaum - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=20478405

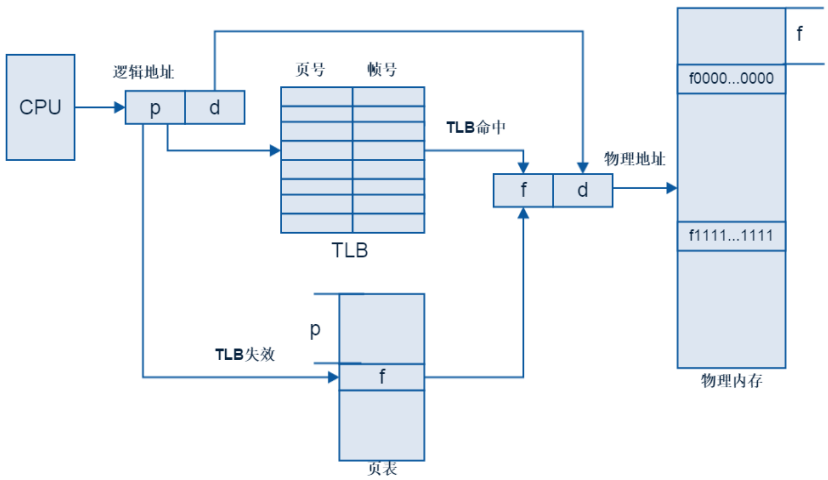
32

快表（TLB）



33

快表（TLB）



34

快表（TLB）的结构

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

与页表项中的valid位不同，这里指的是所缓存的条目是否有效

35

快表（TLB）

TLB的性质和使用方法与Cache相同：

- TLB只包括页表中的一小部分条目。当CPU产生逻辑地址后，其页号提交给TLB。如果不在TLB中（称为TLB失效），那么就需要访问页表,并将页号和帧号增加到TLB中。
- 如果TLB中的条目已满，那么操作系统会选择一个来替换。替换策略有很多，从最近最少使用替换（LRU）到随机替换等。
- 另外，有的TLB允许有些条目固定下来。通常内核代码的条目是固定下来的。

36

快表（TLB）

TLB的其它特性

- 有的TLB在每个TLB条目中还保存地址空间标识码（address-space identifier，ASID）。ASID可用来唯一标识进程，并为进程提供地址空间保护。当TLB试图解析虚拟页号时，它确保当前运行进程的ASID与虚拟页相关的ASID相匹配。如果不匹配，那么就作为TLB失效。
- 除了提供地址空间保护外，ASID允许TLB同时包含多个进程的条目。如果TLB不支持独立的ASID，每次选择一个页表时（例如，上下文切换时），TLB就必须被冲刷（flushed）或删除，以确保下一个进程不会使用错误的地址转换。

37

举例

- 假设虚拟地址16位，页内偏移是8位，页号为8位

页表

VPN (Index)	Valid	PFN/ Disk Block
0	1	1
1	0	5
2	1	7
3	1	8
4	1	2
5	1	6

TLB

Valid	Tag (VPN)	PFN
0		
1	4	2
1	2	7
1	5	6

- 如何翻译下列虚拟地址？
  - 0x0506 TLB命中→PFN: 6→0x 0606
  - 0x02F0 TLB命中→PFN: 7→0x 07F0
  - 0x00F0 TLB没命中→查页表，有效→PFN: 1→0x 01F0→更新TLB
  - 0x0104 TLB没命中→查页表，无效→?

38

有效内存访问时间（一级页表）

- EAT-Effective Access Time
- TLB查询时间= ε 时间单位
- 单次内存访问时间= τ 时间单位
- TLB的命中率是α
- EAT的计算：
  - EAT = (τ+ε)*α + (2τ+ε)*(1-α) = 2τ + ε - τα

39

有效内存访问时间：举例

- 假如查找TLB需要20ns，访问内存需要100ns，TLB命中率为80%
  - 有效内存访问时间 = 2*100 + 20 - 80%* 100 = 140 ns
- 过程分析：如果访问位于TLB中的页号，那么需要20+100=120ns。如果不能在TLB中找到（20ns），那么必须先访问位于内存中的页表得到帧号（100ns），并进而访问内存中所需字节（100ns），这总共需要220ns。。
  - 有效内存访问时间= 0.80 * 120 + 0.2 * 220 = 140（ns）
  - 比不用页表的内存访问速度要慢40%（100ns v.s. 140ns）
- 如果TLB命中率为98%，那么有效内存访问时间 = 2*100 + 20 -0.98 * 100 = 122(ns)
  - 由于提高了命中率，内存访问时间只慢了22%

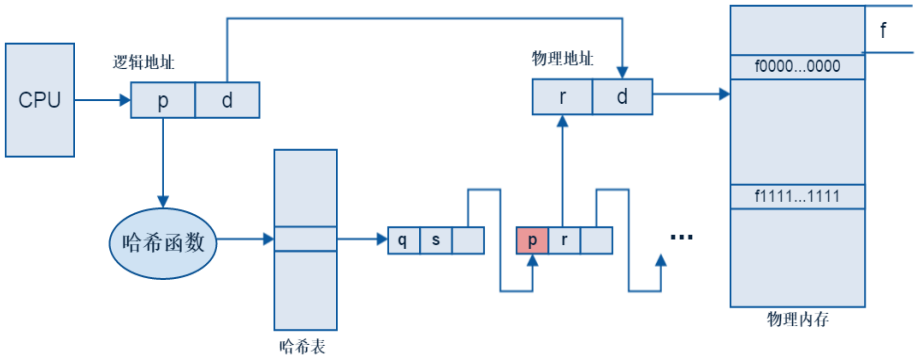
40

哈希页表（hashed page table）

- 处理超过32位地址空间的一种常用方法是使用哈希页表（hashed page table），根据虚拟页号的哈希值来访问页表。哈希页表的每一表项都包括一个链表的元素，这些元素哈希成同一位置（要处理碰撞）。每个元素有3个域：
  1. 虚拟页号
  2. 所映射的页框号
  3. 指向链表中下一个元素的指针。
- 具体过程：将虚拟页号转换为哈希值，据此访问哈希表的表项（链表）。用虚拟页号与链表中的元素的第一个域相比较。如果匹配，那么相应的页框号（第二个域）就用来形成物理地址；如果不匹配，那么就进一步比较链表的下一个节点，以找到匹配的页号。

41

哈希页表（Hashed Page Table）



42

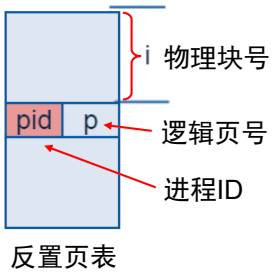
反置页表(Inverted page table)

- 每个进程都有一个相关页表，该进程所使用的每个页都在页表中有一项。这种页的表示方式比较自然，这是因为进程是通过页的虚拟地址来引用页的。操作系统必须将这种引用转换成物理内存地址。
- 这种方法的缺点之一是每个页表可能有很多项。这些表可能消耗大量物理内存，却仅用来跟踪物理内存是如何使用的。对于每个使用32位逻辑地址的进程，其页表长度均为4MB。
- 为了解决这个问题，可以使用反向页表（inverted pagetable）。

43

反置页表(Inverted page table)

- 反置页表不是依据进程的逻辑页号来组织，而是依据该进程在内存中的页框号来组织（即：按页框号排列），其表项的内容是逻辑页号 P 及隶属进程标志符 pid 。
- 反置页表的大小只与物理内存的大小相关，与逻辑空间大小和进程数无关。如：64M主存,若页面大小为4K,则反向页表只需 64KB。
- 如64位的PowerPC，UltraSparc等处理器。



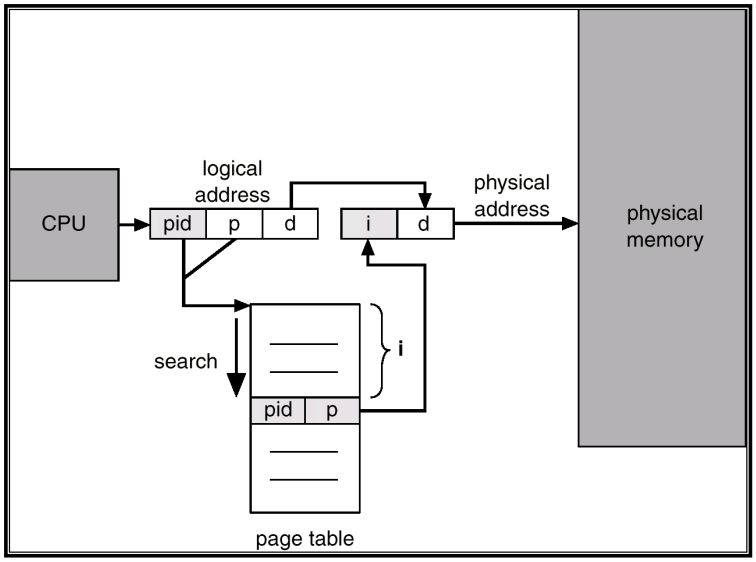
44

反置页表

利用反置页表进行地址变换：

- 用进程标志符和页号去检索反置页表。
- 如果检索完整个页表未找到与之匹配的页表项，表明此页此时尚未调入内存，对于具有请求调页功能的存储器系统产生请求调页中断，若无此功能则表示地址出错。
- 如果检索到与之匹配的表项，则表项的序号 i 便是该页的物理块号，将该块号与页内地址一起构成物理地址。

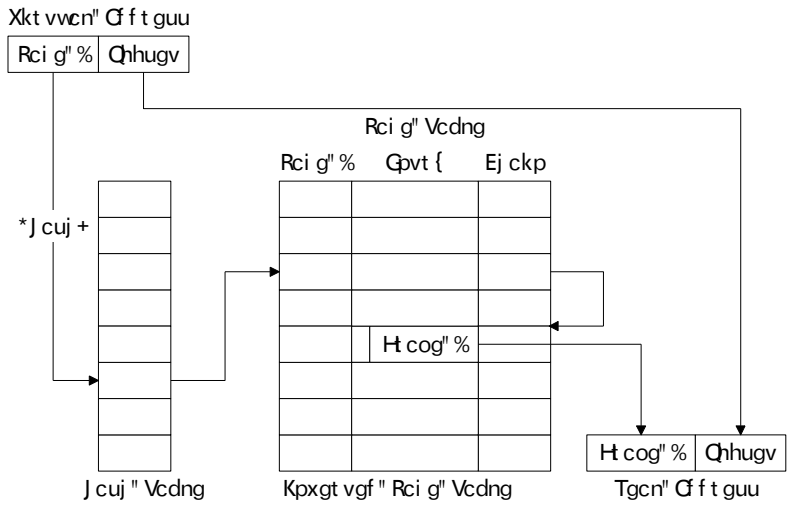
反置页表(Inverted page table)



反置页表(Inverted page table)

- 反置页表按照物理地址排序，而查找依据虚拟地址，所以可能需要查找整个表来寻找匹配。
- 可以使用哈希页表限制页表条目或加入 TLB 来改善。
- 通过哈希表(hash table)查找可由逻辑页号得到物理页面号。虚拟地址中的逻辑页号通过哈希表指向反置页表中的表项链头（因为哈希表可能指向多个表项），得到物理页面号。
- 采用反置页表的系统很难共享内存，因为每个物理帧只对应一个虚拟页条目。

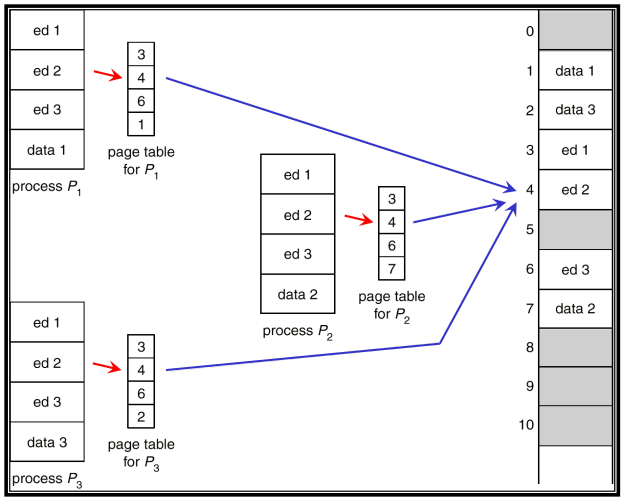
47



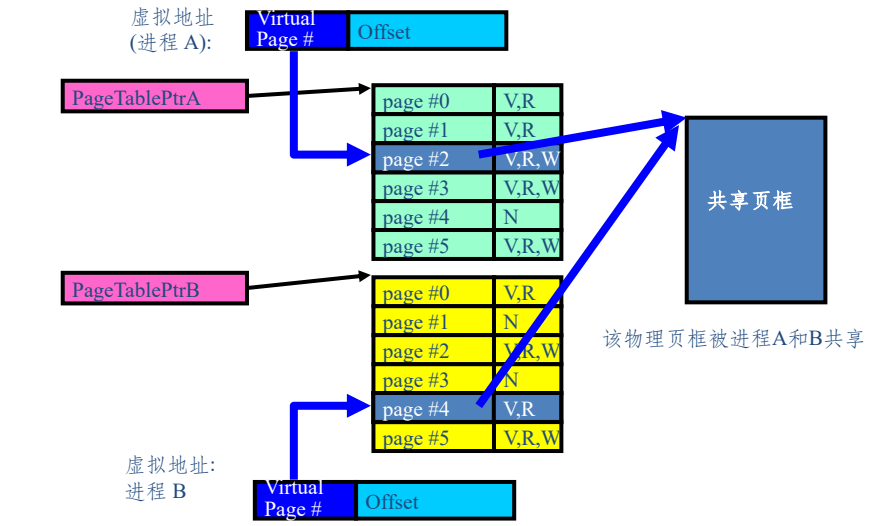
48

页共享与保护

- 各进程把需共享的数据/代码的相应页面指向相同页框



页共享与保护



页共享与保护

页的保护

- 页式存储管理系统提供了两种方式：
  - 地址越界保护
  - 在页表中设置保护位（定义操作权限：只读，读写，执行等）

共享带来的问题

- 若共享数据与不共享数据划在同一页框中，则：
  - 有些不共享的数据也被共享，不易保密。
- 实现数据共享的最好方法：分段存储管理。

51

内容小结

- 基本原理
  - 非连续的内存分配
  - 固定的分配单位
  - 虚拟页面与物理页框的分离、映射
- 基本概念：页表、地址变换、多级页表、快表
- 其他页表类型：哈希页表、反置页表
- 页共享与保护

- 思考
  - OS在修改页表项时使用的是物理地址，还是虚拟地址？页表项的虚拟地址是如何转换成物理地址的？
  - 页式管理有什么不足？

52

Q&A

微信群/课程中心论坛

sunhl@buaa.edu.cn

53

- 1. 基于页式的内存分配，属于固定分区分配，还是动态分区分配？是否会产生内存碎片？
- 2. 页面的大小对内存管理产生怎样的影响？
- 3. 页表项主要记录哪些信息？
- 4. 为什么需要多级页表？
- 5. 在只有一级页表的内存管理中，页表内容是否是连续存储的？

54

55