

## 4.5 页式存储管理

- 4.5.1 页式管理的实现原理
- 4.5.2 页式动态地址变换
- 4.5.3 快表和联想存储器
- 4.5.4 多级页表
- 4.5.5 页式管理的主存分配
- 4.5.6 页式管理的保护
- 4.5.7 页式管理的共享

58

### 4.5.1 页式管理的实现原理(1)

#### ➤ 页面与物理块

- 物理块（页框）

- ◆ 将物理存储空间划分成的大小相等的若干存储块
- ◆ 大小为2的整次幂，大小在4KB到1GB之间

- 页面

- ◆ 将进程的逻辑地址空间划分成的与物理块大小相同的若干片

- 为进程分配内存时，以块为单位将进程的若干页分别装入多个可以不相邻接的物理块中

59

## 4.5.1 页式管理的实现原理(2)

### ➤ 地址计算

- 逻辑地址空间的大小是  $2^m$
- 每一页的大小是  $2^n$  个逻辑地址单元
- 页号与页内位移计算公式：
  - ◆ 逻辑地址空间中的地址为 A
  - ◆ 每一页的大小为  $L = 2^n$ ，则

$$\text{页号: } P = A \div L$$

$$\text{页内地址: } d = A \mod L$$

60

## 4.5.1 页式管理的实现原理(3)

### ➤ 地址结构

- 页号(p)：用做页表的索引，包含了每一页在物理内存中的起始地址
- 页内位移(d)
- 页号与页内位移各占多少位，与页的大小和主存最大容量有关
- 地址分离工作由硬件实现

页号	页内位移
p	d
$m-n$	n

61

## 4.5.1 页式管理的实现原理(4)

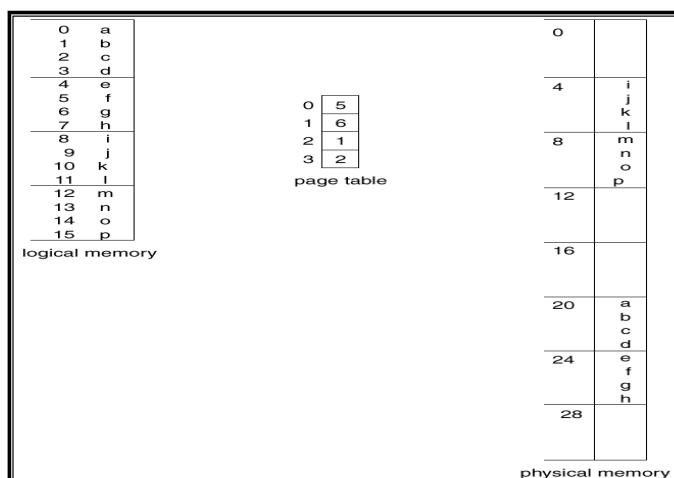
### ➤示例

- 若一个进程申请6150B的存储区，当页大小为1024B时，试问该进程需要多少个内存块？该进程第4000B处的目标代码在第几页？
  - 因为该进程有 $[6150/1024] = 7$ 个页，所以需要7个内存块
  - 方法1： $4000/1024 = 3$
  - 方法2： $(4000)_{10} = (11110100000)_2$

➤采用分页技术不会产生外部碎片，但可能产生内部碎片

62

## 4.5.1 页式管理的实现原理(5)



32字节内存，页面大小为4字节分页示例

63

## 4.5.1 页式管理的实现原理(6)

### ➤ 页表

- 记录进程的逻辑页与主存中物理块的对应关系，实现从页号到物理块号的地址映射
- 进程地址空间的每一页对应一个表目，指出该逻辑页在主存中的物理块号
- 页表存放于主存，页表的主存始址与页表长度保存在进程控制块中
- 控制寄存器
  - 页表基址寄存器(PTBR)：指向页表
  - 页表长度寄存器 (PTLR)：页表长度

64

## 4.5.2 页式动态地址变换(1)

### ➤ 变换过程

- 将进程的页表始址和页表长度送入控制寄存器中；
- 比较程序计数器内的页号 $P$ 与控制寄存器中的页表长度，若页号小于页表长度则继续，否则产生地址越界，终止程序运行；
- 将程序计数器中的页号 $P$ 与控制寄存器中的页表始址相加，得到该访问操作的页号在页表中的入口地址；

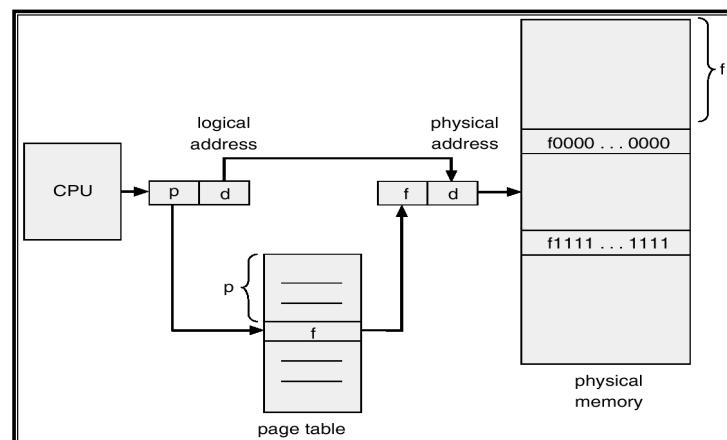
65

## 4.5.2 页式动态地址变换(2)

- 以该地址访问页表，获得该页所对应的主存块号f；
  - 将主存块号f与程序计数器中的页内偏移相拼接，得到该操作所在主存的物理地址： $f \times L + d$ ；
  - 根据这个物理地址，完成指定操作。
- 存在的问题
- 执行一次访问操作至少要两次访问主存
    - ◆ 访问页表
    - ◆ 实现指定操作

66

## 4.5.2 页式动态地址变换(3)



页式管理地址变换过程

67

## 4.5.2 页式动态地址变换(4)

➤ **示例：**进程地址空间共有7个页，每页的大小为1024B，其对应的主存块在页表中已列出。假定页表在主存始址为500，若该程序从第0页开始运行。某时刻，程序计数器内容为4000，则该地址对应的物理地址是多少？

➤ **解：**首先分解该逻辑地址，则

$$\text{页号 } p = 4000 / 1024 = 3$$

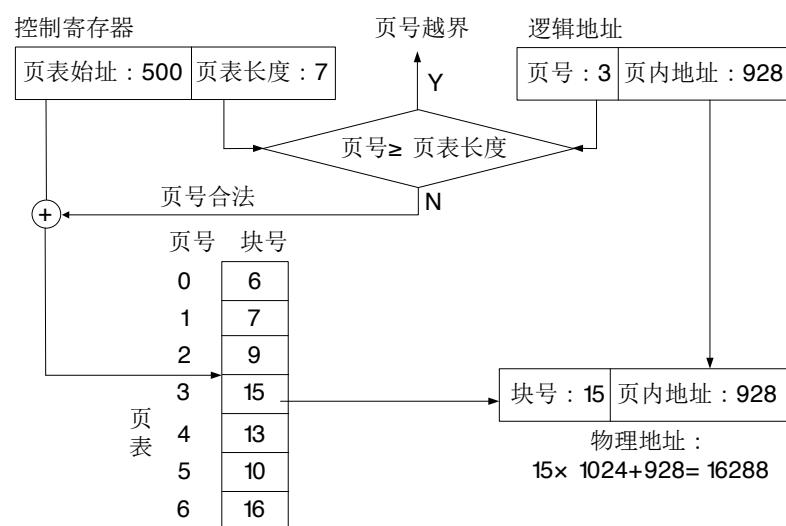
$$\text{页内地址 } d = 4000 \bmod 1024 = 928$$

程序计数器： 

3	928
---	-----

68

## 4.5.2 页式动态地址变换(5)



69

### 4.5.3 快表和联想存储器(1)

#### ➤ 快表

- 为了提高存取速度，在地址变换机构中设置的具有并行查找能力的专用高速缓冲寄存器组（32~1024个寄存器），用来存放页表的一部分

#### ➤ 快表结构

页号	块号	访问位	状态位
0	5	1	1
1	7	0	1

70

### 4.5.3 快表和联想存储器(2)

- 页号：程序当前访问的地址空间的页号
- 块号：该页所对应的主存块号
- 访问位：指示该页最近是否被访问过
  - 0：未被访问
  - 1：访问过
- 状态位：指示该寄存器是否被占用
  - 0：空闲
  - 1：占用

71

### 4.5.3 快表和联想存储器(3)

#### ➤ 地址变换过程

- 硬件地址转换机构在进行地址变换时，有两个变换过程：
  - ◆ 利用快表进行的快速变换过程
  - ◆ 利用主存页表进行的正常变换过程
- 一旦快表中与查找的页号相符合时，则将快表中的块号与CPU给出的页内位移相拼接，得到访问主存的绝对地址，结束快表查找工作

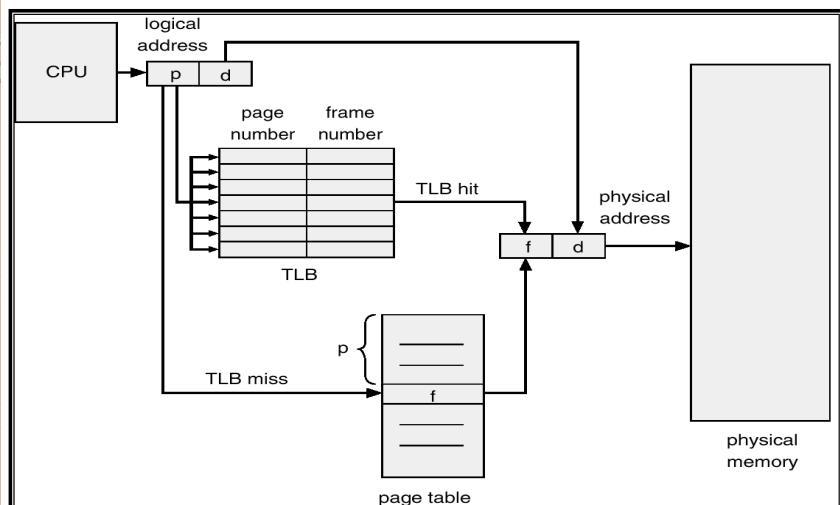
72

### 4.5.3 快表和联想存储器(4)

- ◆ 若利用快表进行变换时，没有找到要查询的页，则继续正常的转换过程，直到形成访问主存的绝对地址
- ◆ 将从主存页表中取出的块号和CPU给出的页号一起写入快表中状态位为0的一行中
- ◆ 若没有这样行存在，则写入访问位为0的某一行中，并同时置状态位和访问位为1（置换）

73

### 4.5.3 快表和联想存储器(5)



使用快表后的页式管理地址变换过程

74

### 4.5.3 快表和联想存储器(6)

- TLB(Translation Lookaside Buffer)命中率
  - 页号在TLB中被查找到的百分比
  - 示例：假设查找TLB需要20ns，访问内存需要100ns，则内存数据访问时间
    - ◆ 页号位于TLB： $20+100=120\text{ns}$
    - ◆ 页号不在TLB： $20+100+100=220\text{ns}$
    - ◆ TLB命中率为80%：  
 $0.8 \times 120 + 0.2 \times 220 = 140\text{ns}$
    - ◆ TLB命中率为98%：  
 $0.98 \times 120 + 0.02 \times 220 = 122\text{ns}$

75

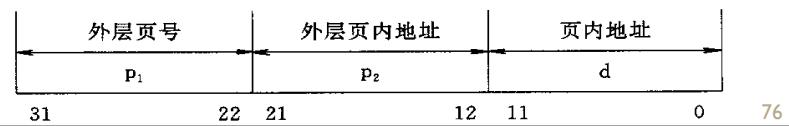
## 4.5.4 多级页表(1)

### ➤ 问题提出

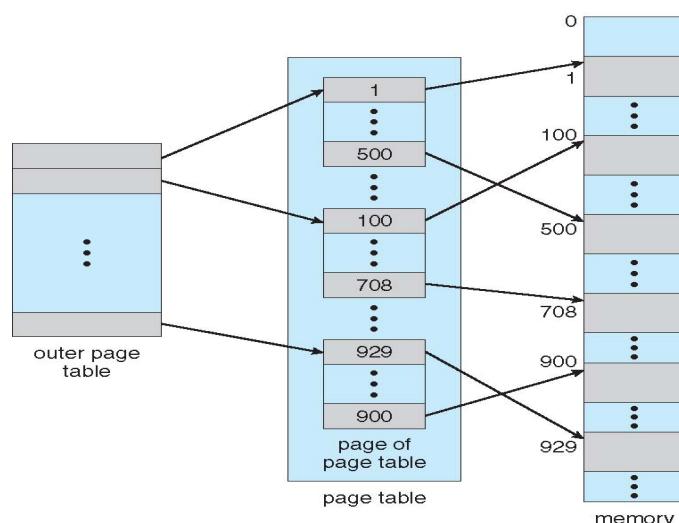
- 一个具有32位地址空间的计算机系统，如果系统的页大小为4KB( $2^{12}$ )，则一个页表可以包含 $2^{32}/2^{12}=2^{20}$ 个表项，若一个表项占用4B，则每个进程需要4MB连续物理地址空间存储页表

### ➤ 解决方法：页表不连续存储

- 将页表再分页



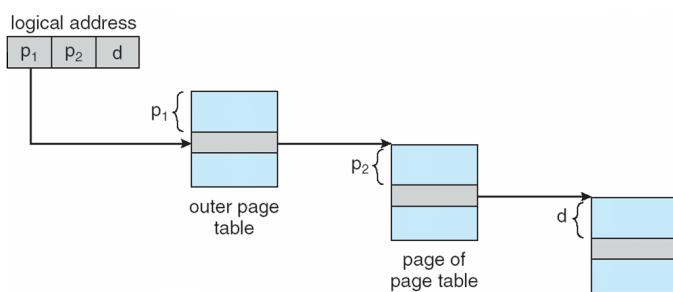
## 4.5.4 多级页表(2)



两级页表结构

77

#### 4.5.4 多级页表(3)



具有两级页表的地址变换机构

78

#### 4.5.4 多级页表(4)

➤ **示例：**某计算机采用二级页表的分页存储管理方式，按字节编址，页大小为 $2^{10}$ 字节，页表项大小为2字节，逻辑地址结构为

页目录号	页号	页内偏移量
------	----	-------

逻辑地址空间大小为 $2^{16}$ 页，则表示整个逻辑地址空间的页目录表中包含表项的个数至少是

- A.64    B.128    C.256    D.512

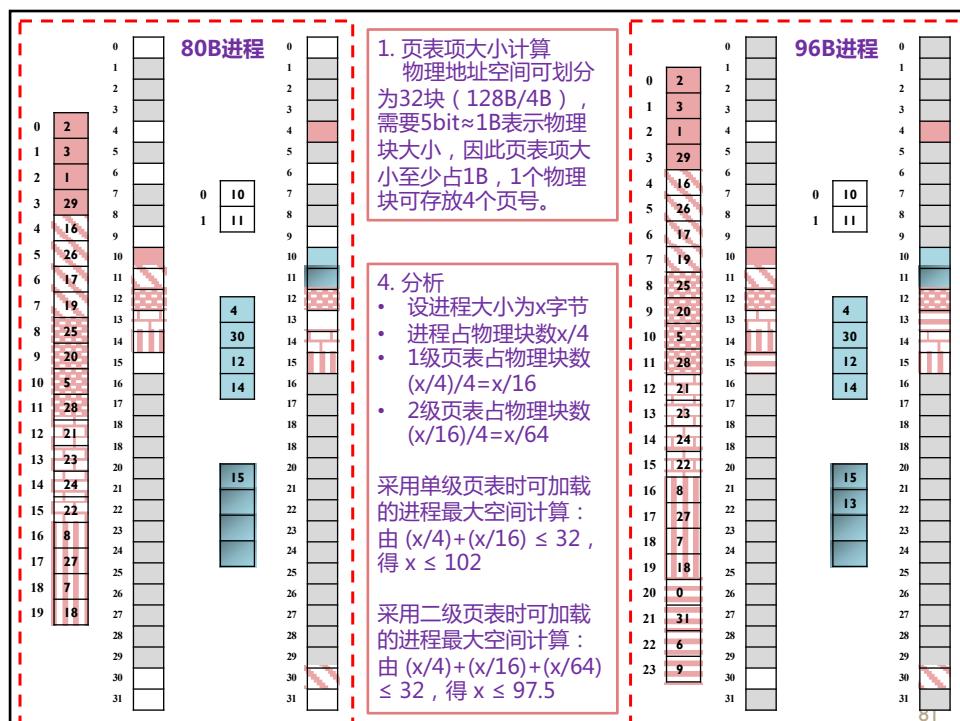
79

## 4.5.4 多级页表(5)

思考题：假设物理地址空间为128B，每页大小为4B，则：

- (1) 每个页表项占多少字节？
- (2) 如果1个进程的逻辑地址空间为80B，请分别给出采用单级页表和二级页表时存储空间的使用情况。
- (3) 如果1个进程的逻辑地址空间为96B，请分别给出采用单级页表和二级页表时存储空间的使用情况。
- (4) 试对比(2)和(3)两种情况，分析单级页表和多级页表对存储空间，以及进程大小的影响。

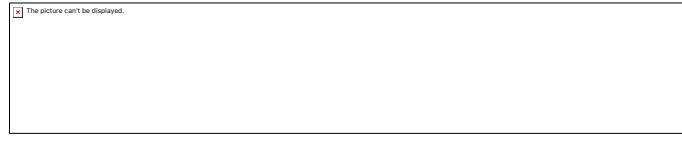
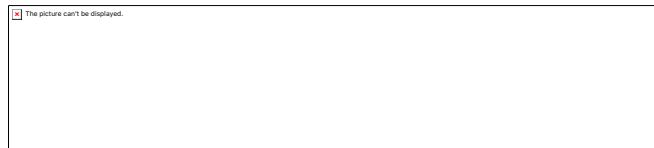
80



## 4.5.4 多级页表(6)

### ➤ 多级页表

- 64位逻辑地址的计算机系统



82

## 4.5.4 多级页表(7)

### ➤ 哈希页表

- 超过32位地址空间的常用方法
- 哈希表
  - ◆ 以逻辑页号作为哈希值
  - ◆ 哈希页表的每一个表项都包括一个链表，链表中的元素哈希为同一个位置
  - ◆ 每个元素包含3个域
    - 逻辑页号
    - 映射的物理块号
    - 指向链表下一元素的指针

83

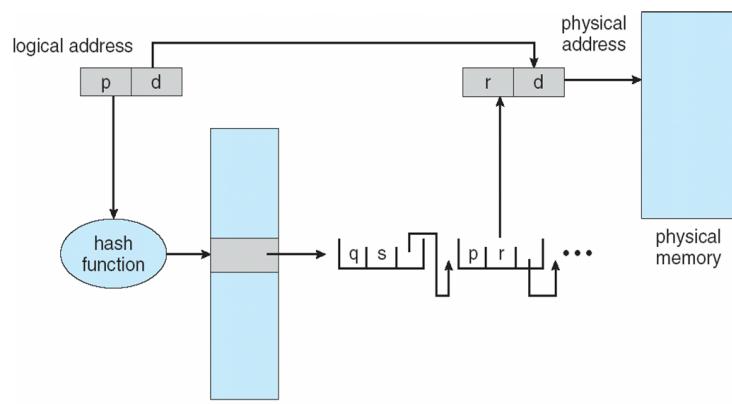
## 4.5.4 多级页表(8)

### ● 地址映射

- ◆ 逻辑地址中的逻辑页号转换到哈希表中，用逻辑页号与链表中的每一个元素的第一个域进行比较
  - 若匹配，则利用该元素的第2个域形成物理地址
  - 若不匹配，则与链表中的下一个节点进行比较，直至找到一个匹配的页号

84

## 4.5.4 多级页表(9)



哈希页表

85

## 4.5.4 多级页表(10)

- 群集页表 (clustered page table)
  - Variation for 64-bit addresses is clustered page tables
  - Similar to hashed but each entry refers to several pages (such as 16) rather than 1
  - Especially useful for **sparse** address spaces (where memory references are non-contiguous and scattered)
- 反向页表 (Inverted Page Table)
  - 前向映射页表 (Forward-mapped page table)
    - ◆ 每个进程拥有一个页表，每个逻辑页占据页表一项

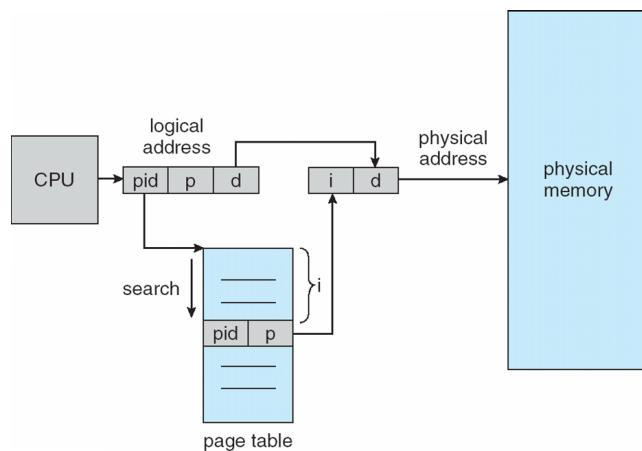
86

## 4.5.4 多级页表(11)

- Rather than each process having a page table and keeping track of all possible logical pages, **track all physical pages**
- Only **one page** table is in the system, and one entry for each real page of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries
  - ◆ TLB can accelerate access

87

#### 4.5.4 多级页表(12)



反向页表

88

#### 4.5.5 页式管理的主存分配(1)

##### ➤ 数据结构

- 页表

- ◆ 每个进程一张，用于将页的逻辑地址  
转换成内存的物理地址

- 进程控制块

- ◆ 进程的页表在主存的起始地址以及页  
表长度

- 存储空间使用情况表

- ◆ 记录存储空间的使用情况

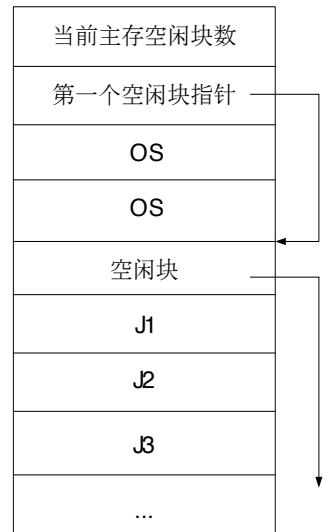
- ◆ 存储分块表/位示图

89

## 4.5.5 页式管理的主存分配(2)

### ➤ 存储分块表

- 记录存储器中各块的状态
- 第一项指出当前主存空闲块总数
- 第二项是指向第一个空闲块指针
- 分配
  - ◆ 检查存储分块表能否满足进程要求



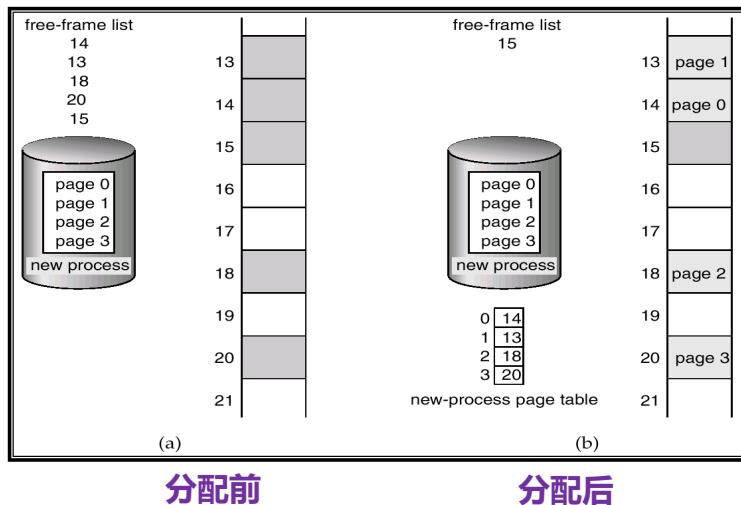
90

## 4.5.5 页式管理的主存分配(3)

- ◆ 若不能满足，则进程等待
- ◆ 若能满足，由存储分块表的第一项中减去本次分配块数，再由第二项空闲块指针找到所需各块，并为进程建立页表、修改存储分块表第二项的空闲块指针
- 回收
  - ◆ 将进程占用主存块归还系统，并修改存储分块表的有关各项

91

## 4.5.5 页式管理的主存分配(4)



92

## 4.5.5 页式管理的主存分配(5)

### ➤ 位示图

- 使用一个位向量，磁盘中的每个块占用其中的一位
- 表中为0的位相应于空闲块，为1的位相应于已被占用的块

1001101101101100
011011011110111
1010110110110110
0110110110111011
1110111011101111
1101101010001111
0000111011010111
1011101101101111
1100100011101111
~ ~
0111011101110111
1101111101110111

A bit map

93

## 4.5.5 页式管理的主存分配(6)

- 分配

- ◆ 分配主存时，查找位示图中位为0的个数能否满足进程的要求。若能满足，则将查到的字节、位转换成主存相应块号，并将相应的位置1

- 回收

- ◆ 释放主存时，系统应将主存块转换为位示图中的字节、位，并将相应的位置0

94

## 4.5.6 页式管理的保护(1)

- 越界保护

- 页号与页表长度寄存器存放的值进行比较

- 访问保护

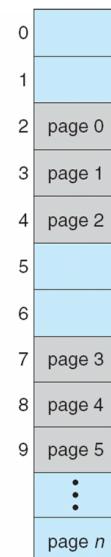
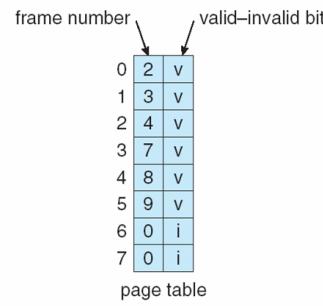
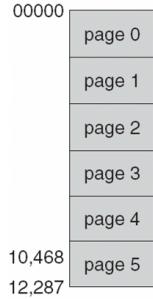
- 在页表中为每个物理块设置保护位

- ◆ 读写保护

- ◆ 有效-无效保护

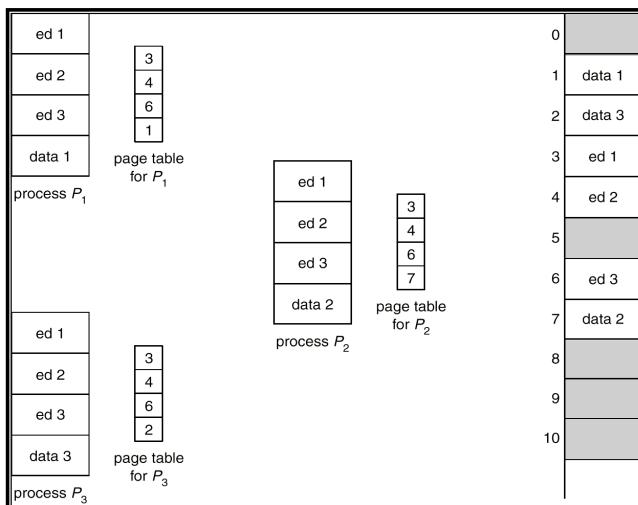
95

## 4.5.6 页式管理的保护(2)



96

## 4.5.7 页式管理的共享



页式管理的共享

97

## 4.6 段式存储管理

- 4.6.1 段式管理的实现原理
- 4.6.2 段式动态地址变换
- 4.6.3 段式管理的存储保护
- 4.6.4 段式管理的共享
- 4.6.5 段式管理的主存分配
- 4.6.6 段式管理与页式管理的比较

98

### 4.6.1 段式管理的实现原理(1)

- 分段管理方式的引入
  - 满足用户（程序员）在编程和使用上多方面的要求
    - ◆ 方便编程
      - 逻辑地址由段名和段内偏移决定
    - ◆ 信息共享
      - 程序和数据的共享以信息的逻辑单位为基础

99

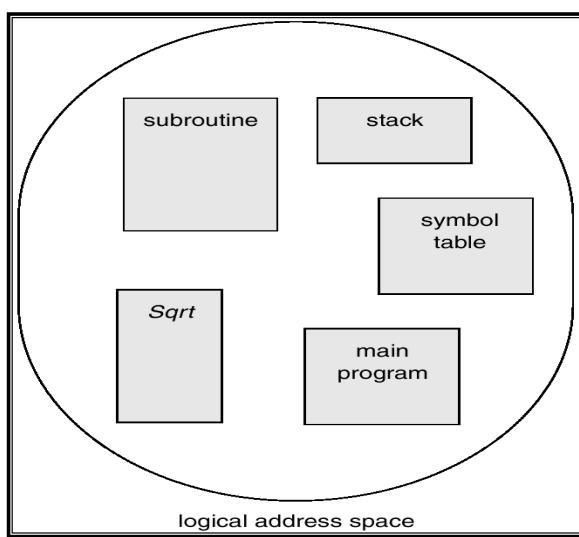
## 4.6.1 段式管理的实现原理(2)

### ➤ 分段

- 按照程序自身的逻辑关系将作业的空间划分成的若干部分
- 每个段都有自己的名字
  - ◆ 主程序段MAIN
  - ◆ 子程序段X
  - ◆ 数据段D
  - ◆ 栈段S
- 每个段都占用从0开始编址的连续地址空间

100

## 4.6.1 段式管理的实现原理(3)

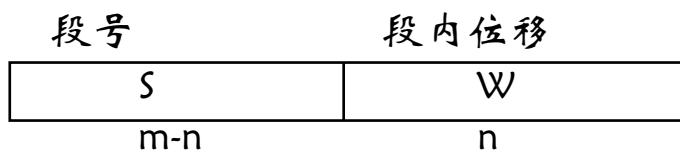


101

## 4.6.1 段式管理的实现原理(4)

➤ 二维地址空间结构

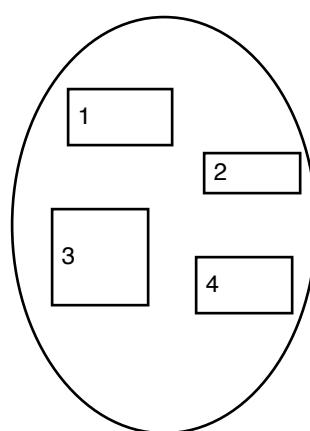
- 段号：内部段号
- 段内位移



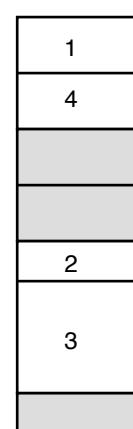
➤ 段式存储分配以段为单位进行，为作业的每一个分段分配一个连续的主存空间，各段之间可以不连续

102

## 4.6.1 段式管理的实现原理(5)



逻辑地址空间



物理存储空间

分段管理存储分配示意图

103

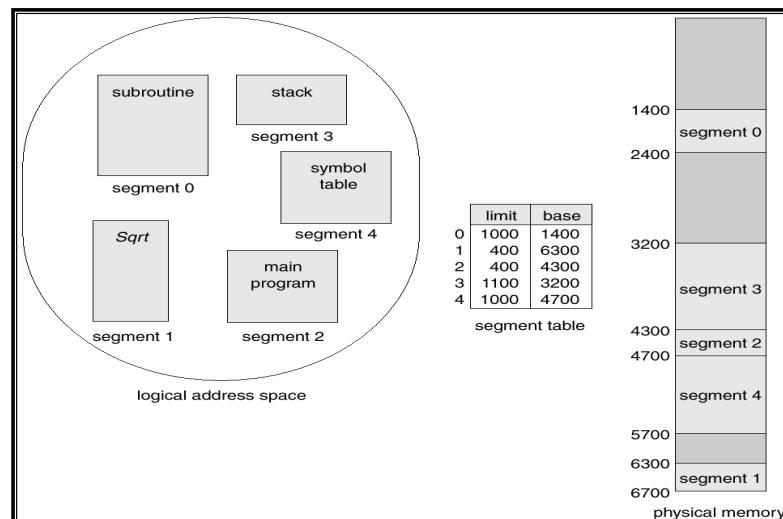
## 4.6.1 段式管理的实现原理(6)

### ➤ 段表

- 记录进程分段与物理存储空间的对应关系, 实现从逻辑分段到物理内存的地址映射
- 每个逻辑分段对应一个表目, 指出该逻辑分段在主存中的起始地址和段的长度
- 段表存放于主存
- 段表的主存始址与段表长度
  - 保存在进程控制块中
- 控制寄存器
  - 段表基址寄存器(STBR) : 指向段表
  - 段表长度寄存器 (STLR) : 段表长度

104

## 4.6.1 段式管理的实现原理(7)



段表

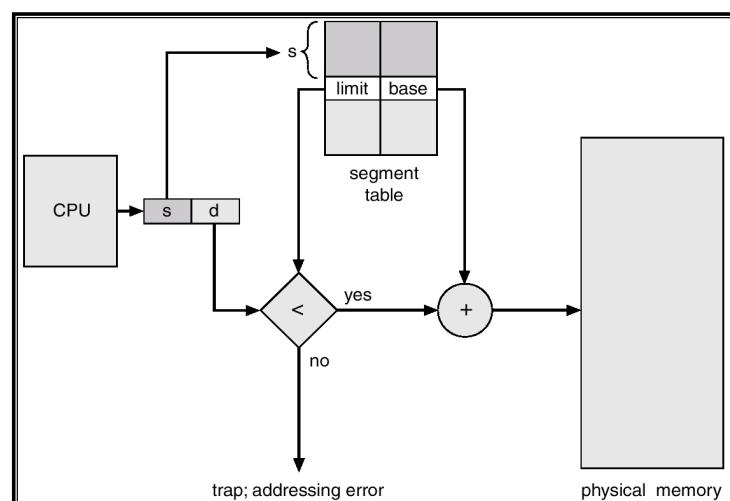
105

## 4.6.2 段式动态地址变换(1)

- 把进程的段表始址和段表长度送入控制寄存器中；
- 比较作业逻辑地址( $S, W$ )中的段号 $S$ 与控制寄存器中的段表长度，若 $S >$ 段表长度，则产生**越界中断**；否则将段号 $S$ 与控制寄存器的段表始址相加，形成访问段表相应表项的主存地址；
- 访问段表，比较段内位移 $W$ 与相应表项中的段长，若 $W >$ 段长，则产生**越界中断**；否则将该段的起始地址与段内位移相加，形成访问主存的物理地址；

106

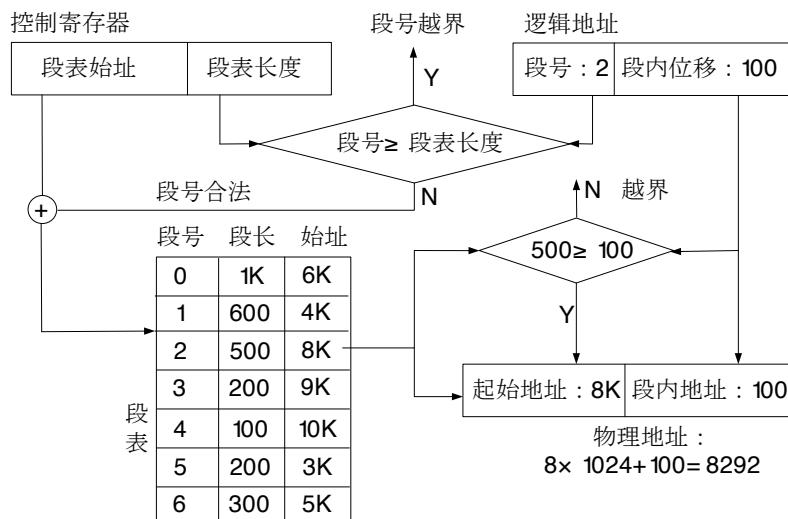
## 4.6.2 段式动态地址变换(2)



段式地址变换过程

107

## 4.6.2 段式动态地址变换(3)



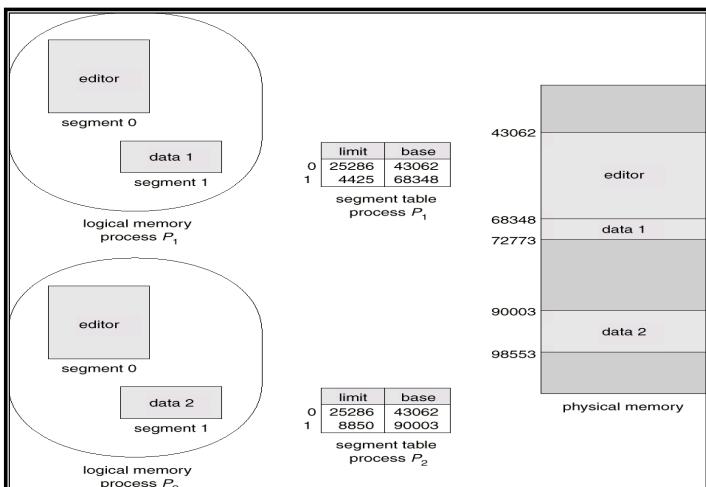
108

## 4.6.3 段式管理的存储保护

- 利用控制寄存器的段表长度对段号越界进行第一级保护；
- 利用段表中的段长对逻辑地址中的段内位移进行检查，实现第二级保护；
- 通过在段表中增加相应的操作方式字段，对相应的段规定读、写、执行是否许可的操作权限，实现第三级保护；

109

#### 4.6.4 段式管理的共享



段式管理的共享

110

#### 4.6.5 段式管理的主存分配

- 类似于可变式分区
  - 首次适应法
  - 最佳适应法
  - 最坏适应法
- 可变式分区管理以**进程**为单位分配一个**连续分区**；段式管理以**段**为单位分配分区，各段之间可以占有**不连续的分区**
- 存在碎片问题

111

#### 4.6.6 段式管理与页式管理的比较(1)

- 段是信息的逻辑单位，它根据用户需要进行划分，对用户是可见的；页是信息的物理单位，它为了方便管理主存而划分，对用户是透明的；
- 段式管理向用户提供的是二维地址空间，页式管理向用户提供的则是一维地址空间，确定其页号和页内偏移由机器硬件实现；
- 页的大小固定不变，由系统决定；段的大小不固定，由其完成的功能决定；

112

#### 4.6.6 段式管理与页式管理的比较(2)

- 分段作为信息的逻辑单位，便于进行存储保护和信息共享；分页的存储保护和信息共享则受到限制；
- 段式管理与分区管理一样可能产生主存碎片，而页式管理则很好地消除了碎片；
- 段式管理与页式管理都需要在作业运行前将全部信息装入主存，不能充分利用存储器；
- 为实现地址变换，段式管理与页式管理都要花费较大的开销；

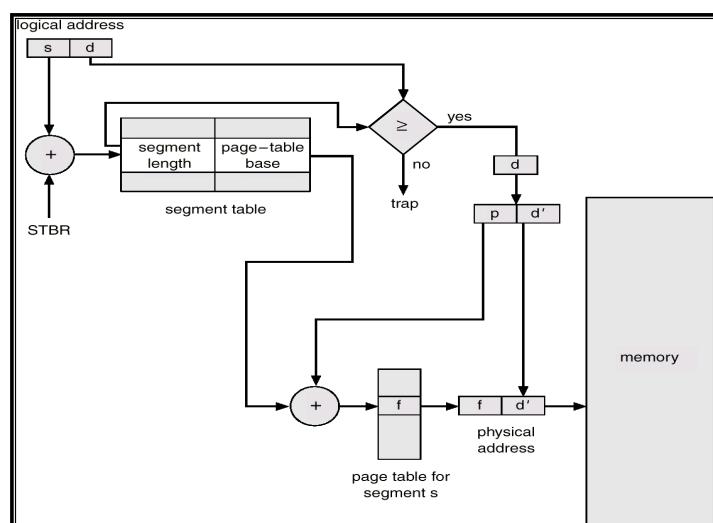
113

## 4.7 段页式管理(1)

- 结合分段原理与分页原理
- 先将用户程序分成若干个段，再把每个段分成若干个页，并为每个段赋予一个段名；
- 地址结构由三部分构成：段号、页号、页内位移
- 地址变换
  - 三次访问内存
  - 快表

114

## 4.7 段页式管理(2)



MULTICS的段页式管理地址变换过程

115

## 4.8 虚拟存储器

4.8.1 基本概念

4.8.2 页式虚拟存储器管理

4.8.3 页式管理设计中的问题

116

### 4.8.1 基本概念(1)

➤ 实存管理技术

- 特点

- ◆ 一次性

- ◆ 驻留性

- 缺点

- ◆ 进程请求的内存空间超过内存总容量，  
进程不能全部装入内存，无法运行

- ◆ 内存容量不足以容纳大进程，只能先  
运行小进程，大进程等待

117

## 4.8.1 基本概念(2)

### ➤ 虚存存储技术

#### ● 虚拟存储器

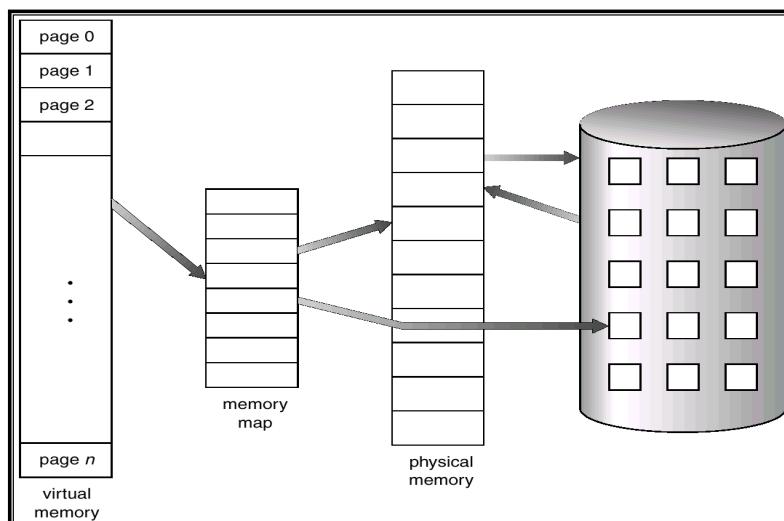
- ◆ 具有请求调入功能和置换功能，能从逻辑上对内存容量加以扩充的一种存储器系统
- ◆ 逻辑容量 = 内存容量 + 外存容量
- ◆ 运行速度接近于内存速度

#### ◆ 特征

- 多次性
- 对换性
- 虚拟性

118

## 4.8.1 基本概念(3)



虚拟存储器

119

## 4.8.1 基本概念(4)

- 虚拟存储技术的基础
  - ◆ 局部性原理
    - 1968, Denning P: 程序在执行时将呈现出局部性规律, 即在一较短时间里, 程序的执行仅局限于某个部分; 相应地, 它所访问的存储空间也局限于某个区域
    - 时间局限性: 循环
    - 空间局限性: 互斥执行

|20

## 4.8.2 页式虚拟存储器管理(1)

### ➤ 实现原理

- 页式管理 + 交换技术
- 将进程信息的副本存放在外存中, 并在页表中指出各页对应的外存地址。当进程被调度运行时, 先将进程中的较少页装入主存, 在执行过程中, 访问不在主存的页时, 再将其调入

### ➤ 页表

- 页号
- 物理块号

|21

## 4.8.2 页式虚拟存储器管理(2)

- 状态位：指示该页是否在主存
  - ◆ 0：不在                  缺页中断
  - ◆ 1：在
- 外存地址：指示该页在外存上的地址
- 访问位：指示该页最近是否被访问过
  - ◆ 0：未访问
  - ◆ 1：访问
- 修改位：指示该页调入主存后是否修改过
  - ◆ 0：未修改过
  - ◆ 1：修改过

|22

## 4.8.2 页式虚拟存储器管理(3)

### ➤ 缺页中断处理

- 操作系统接到此中断信号后，就调出缺页中断处理程序。如果内存中有空闲块，则分配一页，根据页表中给出的外存地址，将新调入页装入内存，并修改页表中相应页表项的状态位及相应的内存块号，使进程继续运行下去
- 若此时内存中没有空闲块，则要淘汰某页，若该页在内存期间被修改过，则要将其写回外存

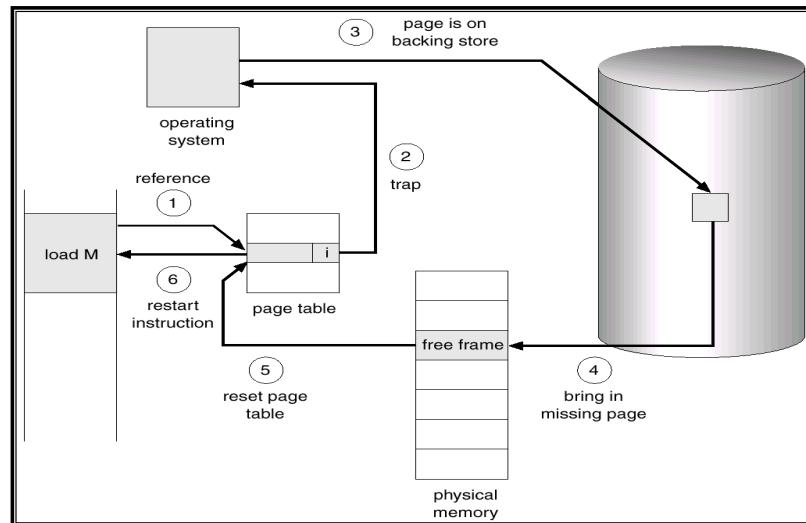
|23

## 4.8.2 页式虚拟存储器管理(4)

- 由于从外存向主存调入一页需要的时间较长，故在调页过程中应将请求调页的进程置为阻塞状态，但此时进程的页绝对不允许被换出，否则，无法装入所需的页
- 该页装入主存后，将等待进程唤醒
- 在产生缺页中断时，一条指令并没有执行完，在操作系统进行缺页中断处理后，应重新执行被中断的指令

|24

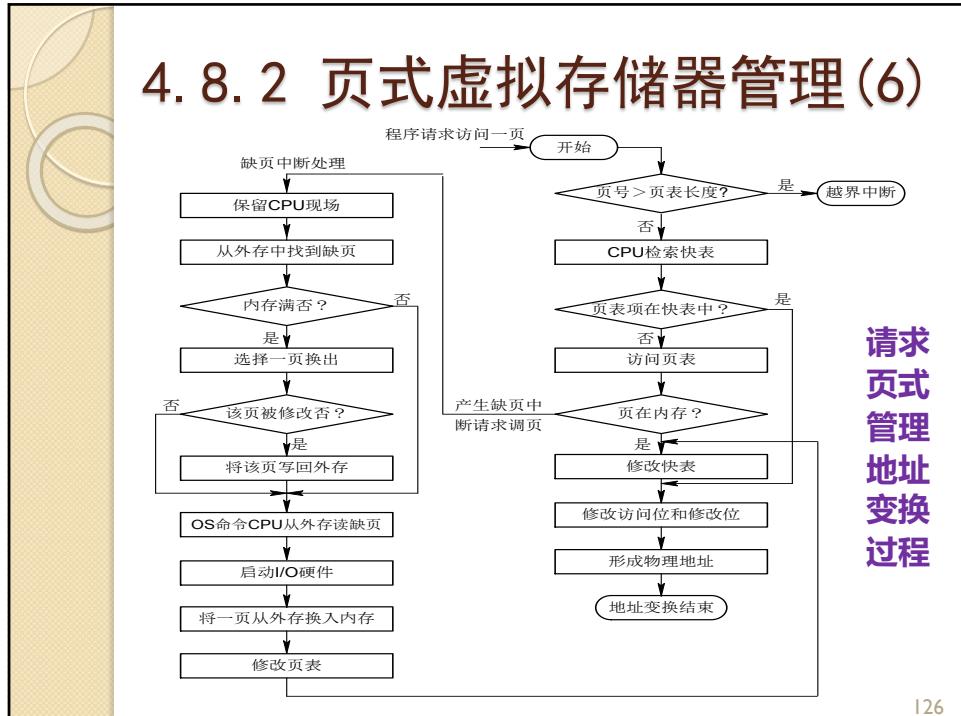
## 4.8.2 页式虚拟存储器管理(5)



缺页中断处理步骤

|25

## 4.8.2 页式虚拟存储器管理(6)



|26

请求  
页式  
管理  
地址  
变换  
过程

## 4.8.2 页式虚拟存储器管理(7)

### ➤ 页面置换算法

#### ● 抖动(Thrashing)

- ◆ 刚被淘汰的页面马上又要用，因而又要调入；调入不久再被淘汰，淘汰不久再次装入。如此反复，使整个系统处于频繁地调入调出状态，大大降低系统的处理效率；

#### ● 算法假设

- ◆ 一个进程分配的主存块数固定不变
- ◆ 采用局部淘汰(淘汰一页时，只考虑在本进程内部实施淘汰)

|27

## 4.8.2 页式虚拟存储器管理(8)

- 算法性能

- ◆ 进程  $P_i$  共有  $m$  页，系统分配给它的主存块为  $n$  块， $m > n$
- ◆ 开始时，主存没有装入任何一页信息
- ◆ 如果进程  $P_i$  在运行中成功访问的次数为  $S$ ，不成功的访问次数为  $F$ （产生缺页中断的次数），则进程执行过程中总的访问次数为  $A = S + F$
- ◆ 进程  $P_i$  执行过程中的缺页率  $f = F/A$

| 28

## 4.8.2 页式虚拟存储器管理(9)

- 理想调度算法

- ◆ 选择以后不再访问的页或经过很长时间之后才可能访问的页进行淘汰

- 置换算法

- ◆ 最佳算法 (OPT, Optimal)
- ◆ 先进先出算法 (FIFO)
- ◆ 最近最久未用算法 (LRU, Least Recently Used)

- ◆ 时钟页面置换算法

- 页面序列 ( $A=12$ )

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

| 29

## 4.8.2 页式虚拟存储器管理(10)

➤ 最佳算法

• 3个物理块—缺页7次

1	1	1	1	1	1	1	1	1	3	3	3
	2	2	2	2	2	2	2	2	2	4	4
		3	4	4	4	5	5	5	5	5	5

• 4个物理块—缺页6次

1	1	1	1	1	1	1	1	1	1	4	4
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3	3	3	3	3
			4	4	4	5	5	5	5	5	5

130

## 4.8.2 页式虚拟存储器管理(11)

➤ FIFO算法

• 3个物理块—缺页9次

1	1	1	4	4	4	5	5	5	5	5	5
	2	2	2	1	1	1	1	1	3	3	3
		3	3	3	2	2	2	2	2	4	4

• 4个物理块—缺页10次 (Belady异常)

1	1	1	1	1	1	5	5	5	5	4	4
	2	2	2	2	2	2	1	1	1	1	5
		3	3	3	3	3	3	3	2	2	2
			4	4	4	4	4	4	3	3	3

131

## 4.8.2 页式虚拟存储器管理(12)

➤ LRU算法

• 3个物理块—缺页10次

1	1	1	4	4	4	5	5	5	3	3	3
	2	2	2	1	1	1	1	1	1	4	4
		3	3	3	2	2	2	2	2	2	5

• 4个物理块—缺页8次

1	1	1	1	1	1	1	1	1	1	1	5
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	5	5	5	5	4	4
			4	4	4	4	4	4	3	3	3

132

## 4.8.2 页式虚拟存储器管理(13)

• 硬件实现

- 系统设置一个64位的硬件计数器C，每当一条指令引用后都自动计数
- 页表的每一项必须含有足够大的字段存放该计数器的值。每次访问主存后，将计数器C的值存入刚被访问过的页的相应字段
- 产生缺页中断时，操作系统查看所有页表，找出计数值最小的页作为最近最少使用的页

133

## 4.8.2 页式虚拟存储器管理(14)

◆ LFU(Least Frequently Used) 与 LRU

- 页面序列 2,1,2,1,2,3,4
- $T=10\text{min}$ , 每分钟发生一页调页

LRU	2	2	2	2	2	2	2
	1	1	1	1	1	1	4
						3	3
LFU	2	2	2	2	2	2	2
	1	1	1	1	1	1	1
						3	4

| 34

## 4.8.2 页式虚拟存储器管理(15)

● 软件实现

◆ 栈式页面置换算法

- 主存维护一张运行进程所需页链表
- 利用栈记录页的使用。正在引用的页放在栈顶，最近最久未用的页放在栈底
- 为了便于栈中元素移动，可采用双向链

| 35

## 4.8.2 页式虚拟存储器管理(16)

➤ LRU算法

• 3个物理块—缺页10次

		3	4	1	2	5	1	2	3	4	5
		2	2	3	4	1	2	5	1	2	3
1	1	1	2	3	4	1	2	5	1	2	3

• 4个物理块—缺页8次

			4	1	2	5	1	2	3	4	5
			3	3	4	1	2	5	1	2	3
			2	2	2	3	4	1	2	5	1
1	1	1	1	2	3	4	4	4	5	1	2

136

## 4.8.2 页式虚拟存储器管理(17)

• 具有LRU算法特性的一类算法，都具有这样的性质：

$$\diamond \quad M(m,r) \leq M(m+1,r)$$

• 这里m随进程分配的主存块而变，r是引用串的索引。这个公式表明，主存块为m+1时，在任何时刻t，存于主存中的一串页面中必然包含有主存块为m时存于主存中的各页。

• 这种关系对所有的m和r都成立。它绝不会出现FIFO算法中的Belady异常。

137

## 4.8.2 页式虚拟存储器管理(18)

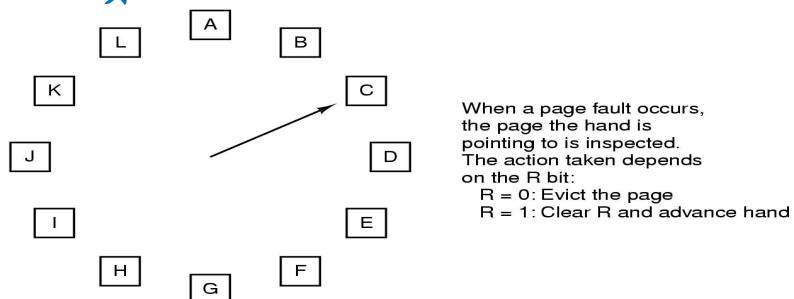
### ➤ 时钟 (CLOCK) 页面置换算法

- 将进程所访问的页像时钟一样存放于一个循环链中
- 设置一个指针指向最早进入主存的页
- 产生缺页中断时，按照下列原则选择某一页进行淘汰
- 检查其访问位，如果为0，则淘汰该页，新装入的页插入到该位置，指针向前移动一个位置

| 38

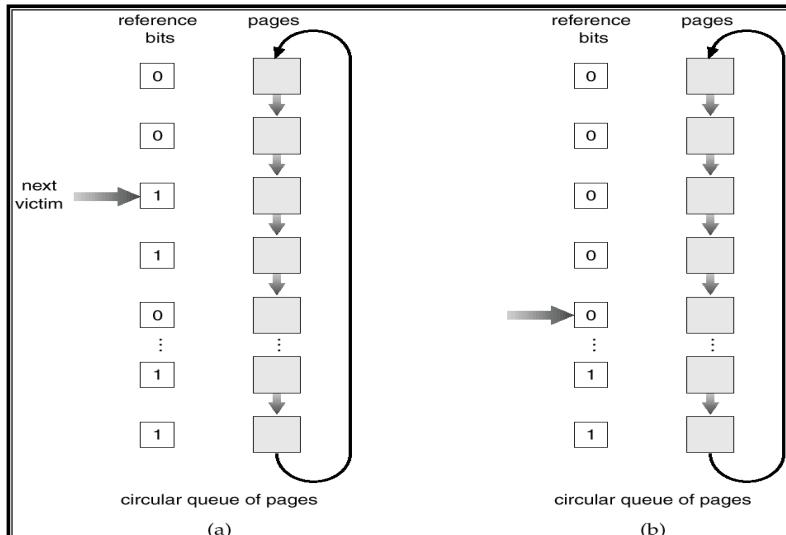
## 4.8.2 页式虚拟存储器管理(19)

- 如果为1，则将访问位置0，并将指针前进一个，继续检查页的访问位（第二次机会）
- 重复该过程，直到找到访问位为0的页



9

## 4.8.2 页式虚拟存储器管理(20)



140

## 4.8.2 页式虚拟存储器管理(21)

➤ CLOCK算法

• 3个物理块- 缺页10次

1	1	1	1	1	1	4	1	4	1	4	1	5	1	5	1	5	1	3	1	3	1	3	1
		2	1	2	1	2	0	1	1	1	1	1	0	1	1	1	1	1	0	4	1	4	1
			3	1	3	0	3	0	2	1	2	0	2	0	2	1	2	0	2	0	5	1	

• 4个物理块- 缺页8次

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		2	1	2	1	2	1	2	1	2	1	2	1	2	0	2	0	2	1	2	1	2	0
			3	1	3	1	3	1	3	1	3	1	5	1	5	1	5	1	5	0	4	1	4
				4	1	4	1	4	1	4	0	4	0	4	0	3	1	3	1	3	0		

141

### 4.8.3 页式管理设计中的问题(1)

#### ➤ 确定页面大小

- 页面越大，无用程序装入主存就越多，从而使主存浪费严重
- 页面越小，程序需要的页越多，页表越大
- 最佳页尺寸
  - 设进程平均大小为 $s$ 字节，页大小为 $p$ 字节，页表项占 $e$ 字节，则
  - 进程所需页数近似为 $s/p$ 页
  - 页表占用空间为 $se/p$ 字节
  - 进程内部碎片浪费空间平均为 $p/2$

142

### 4.8.3 页式管理设计中的问题(2)

- 因此页表和内部碎片引起的系统的总开销为 $se/p + p/2$
- 页表开销中，页越小，页表越大
- 内部碎片开销中，页越大，内部碎片越大
- $d(se/p + p/2)/dp = -se/p^2 + 1/2$
- 令上式为0，可得最佳页面大小为

$$p = \sqrt{2se}$$

#### ➤ 确定最小物理块数

- 保证进程正常运行所需的最小物理块数

143

### 4.8.3 页式管理设计中的问题(3)

#### ➤ 物理块分配算法

- 可变分配
- 固定分配
  - ◆ 平均分配
  - ◆ 按比例分配
    - 系统中有n个进程，每个进程的页面数为 $S_i$ ，系统中各进程页面数总和为S
    - 系统中可用物理块总数为m，每个进程能分配到的物理块数为 $b_i$ ，则
    - $S = \sum S_i \quad b_i = (S_i / S) * m$
  - ◆ 按优先权分配

| 44

### 4.8.3 页式管理设计中的问题(4)

#### ➤ 页面置换策略

- 局部置换
- 全局置换

	Age
A0	10
A1	7
A2	5
A3	4
A4	6
A5	3
B0	9
B1	4
B2	6
B3	2
B4	5
B5	6
B6	12
C1	3
C2	5
C3	6

(a) 存储器当前情况

	Age
A0	
A1	
A2	
A3	
A4	
A5	
B0	
B1	
B2	
B3	
B4	
B5	
B6	
C1	
C2	
C3	

(b) 局部置换

	Age
A0	
A1	
A2	
A3	
A4	
A5	
B0	
B1	
B2	
B3	
B4	
B5	
B6	
C1	
C2	
C3	

(c) 全局置换

| 45

### 4.8.3 页式管理设计中的问题(5)

#### ➤ 物理页框分配+页面置换策略

- 固定分配局部置换
  - ◆ 为每个进程分配多少物理块难以确定
    - 太少，会频繁产生缺页中断，降低系统吞吐量
    - 太多，内存驻留进程减少，可能造成资源空闲，且进程交换时会耗费更多时间
  - 可变分配局部置换
  - 可变分配全局置换
    - ◆ 有更好的系统吞吐量，易于实现，较为常用

146

### 4.8.3 页式管理设计中的问题(6)

#### ➤ 调页时机

- 预调页
  - ◆ 根据空间局部性，将不久之后会被访问的页面预先调入内存
    - 成功率仅有50%，主要用于进程的首次调入
- 请求调页
  - ◆ 进程运行时，发现其所访问的程序页面不在内存，请求系统将所需页面调入内存
    - 系统开销大，易于实现
    - 虚拟存储器大多采用此策略

147

### 4.8.3 页式管理设计中的问题(7)

#### ➤ 交换区的管理

- 交换区是OS利用磁盘扩充主存的主要方法
- 系统使用交换区的方法
  - ◆ 用交换区保存进程运行的整个映像
  - ◆ 存储分页系统可能被淘汰的页
- 交换空间的设置
  - ◆ 从文件系统中分割一部分空间作为一个大文件使用
  - ◆ 占用一个独立的磁盘或磁盘分区

| 48

### 4.8.3 页式管理设计中的问题(8)

#### ➤ 抖动与工作集

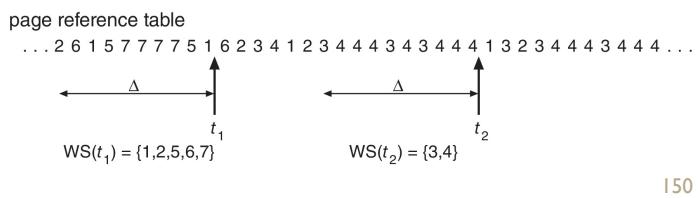
- 产生抖动的原因
  - ◆ 同时在系统中运行的进程太多，因此分配给每一个进程的物理块太少，不能满足进程正常运行的基本要求，使得进程运行时频繁出现缺页，从而产生抖动
- 工作集
  - ◆ 理论基础：1968年由Denning提出，基于程序运行时的局部性原理
  - ◆ 概念：在某段时间间隔里，进程实际所要访问页面的集合

| 49

### 4.8.3 页式管理设计中的问题(9)

#### ◆ 工作集的定义

- 进程在时间  $t$  的工作集为  $WS(t, \Delta)$
- 进程在时间间隔  $(t-\Delta, t)$  中引用页面的集合
- 变量  $\Delta$  为工作集的“窗口尺寸”
- 示例：  $\Delta=10$



150

### 4.8.3 页式管理设计中的问题(10)

引用页序列	窗口大小		
	3	4	5
24	24	24	24
15	15 24	15 24	15 24
18	18 15 24	18 15 24	18 15 24
23	23 18 15	23 18 15 24	23 18 15 24
24	24 23 18	—	—
17	17 24 23	17 24 23 18	17 24 23 18 15
18	18 17 24	—	—
24	—	—	—
18	—	—	—
17	—	—	—
17	—	—	—
15	15 17 18	15 17 18 24	—
24	24 15 17	—	—
17	—	—	—
24	—	—	—
18	18 24 17	—	—

151

### 4.8.3 页式管理设计中的问题(11)

- 抖动的预防

- ◆ 采用局部置换策略

- 限制影响范围

- 效果不佳，会延长其他进程缺页中断的处理时间

- ◆ 把工作集算法融入到处理器调度中

- 当造成处理器利用低的原因是“内存中分配的物理块数不够”

- 工作集页面置换算法

152

### 4.8.3 页式管理设计中的问题(12)

- ◆ 利用 “ $L=S$ ” 准则调节缺页率

- $L$ 为缺页之间的平均时间， $S$ 为平均缺页服务时间

- $L>>S$ , 表示很少发生缺页

- $L < S$ , 表示频繁发生缺页

- $L \approx S$ , 表示磁盘与处理器均能达到其最大利用率

- ◆ 选择暂停的进程

153

### 4.8.3 页式管理设计中的问题(13)

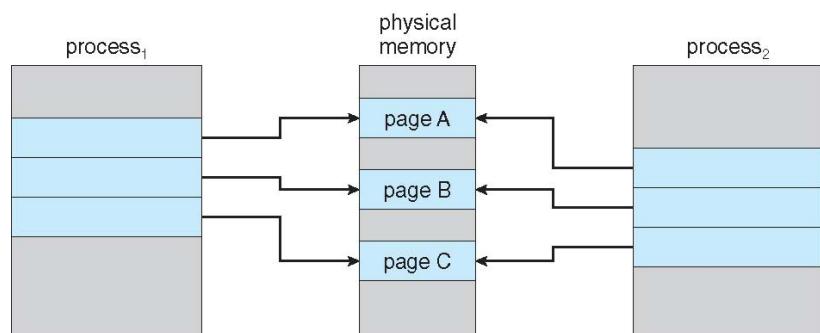
#### ➤ 页面共享

- 写时复制

- ◆ 内存共享技术
- ◆ 当两个进程要读写相同内容的内存时，允许两个进程共享同一物理块，这些物理块标记为写时复制页
- ◆ 若两个进程对该块只读，则二者共享
- ◆ 若某一进程对该块写，则系统将该物理块复制到主存的另一物理块中，并更新该进程的页表指向该物理块

| 54

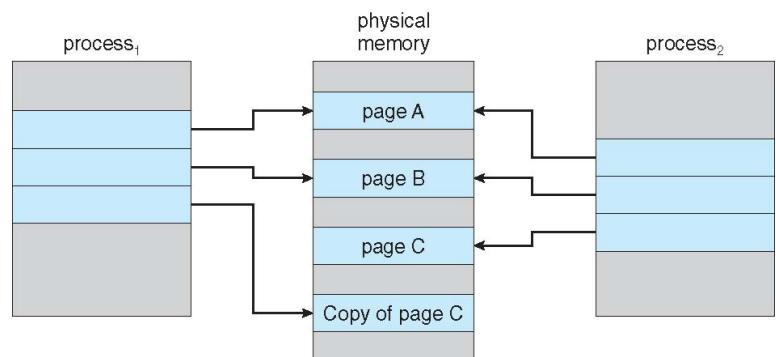
### 4.8.3 页式管理设计中的问题(14)



进程1修改页C之前

| 55

### 4.8.3 页式管理设计中的问题 (15)



进程1修改页C之后

156

### 存储分配方式

作业占用  
连续存储空间

单一连续区

单道程序

作业占用  
不连续存储空间

固定式分区  
可变式分区

多道程序

分页管理方式  
分段管理方式  
段页式管理方式

作业一次  
全部装入

请求分页管理方式  
请求分段管理方式  
请求段页式管理方式

作业多次  
分批装入

157