深圳大学
Shenzhen University

操作系统

第五讲 存储器管理

谭舜泉
计算机与软件学院

第5章　存储器管理

第一节　基本概念

一、存储器管理的目的

- 为用户使用存储器提供方便

(1). 用户只需在自己的逻辑空间内编程，不需要了解自己将放在内存中的物理位置，也不需要了解其它用户程序在内存中的物理位置

(2). 为用户提供充分大的存储空间（*虚存管理*）

- 充分发挥内存的利用率，让尽量多的用户程序调入内存运行

第5章　存储器管理

第一节　基本概念

二、存储器管理的内容

1、内存分配

(1)、静态分配，程序装入内存后，不允许再申请新的内存空间，也不允许在内存中“移动”

(2)、动态分配，按需要，可以部分或全部将程序装入内存，并允许运行时“移动”位置，或继续申请内存

第5章　存储器管理

第一节　基本概念

二、存储器管理的内容

2、地址映射

解决如何将程序装入内存的问题

(1)、绝对映射（绝对装入方式）

程序中的逻辑地址与实际的内存地址完全相同

(2)、静态映射（可重定位装入方式）

在装入程序时，对逻辑地址进行修改

(3)、动态映射（动态运行时装入方式）

逻辑地址到内存地址的映射在真正执行时才进行

第5章　存储器管理

第一节　基本概念

二、存储器管理的内容

3、内存保护

(1)、保护内存不被非法访问

(2)、不非法访问其它用户（包括系统）内存

4、内存扩充

在逻辑上扩充内存的空间 *（虚拟内存）*

第5章　存储器管理

第二节　连续分配方式

- **连续分配**

  指为一个用户程序分配一个连续的内存空间

  主要包括:

  单一连续分配
  固定分区分配
  动态分区分区分配
  动态重定位分区分配

第5章　存储器管理

第二节　连续分配方式

一、单一连续分配

- 一个用户程序独占连续的内存用户区
- 只能用于单用户、单任务的OS中
- 系统分两个内存区：系统区和用户区
- 可以要求对系统区进行保护（基址R+界限R）
  也可以不对系统进行保护（出现差错时，
  损失小）

第5章　存储器管理

第二节　连续分配方式

二、固定分区分配

- 将内存划分为多个区域，每个内存区存放一个用户程序
- 可用于多用户、多任务环境

第5章　存储器管理

第二节　连续分配方式

二、固定分区分配

1、划分分区方法

- 分区大小相等

  将内存分成大小相等的多个分区

优点：简单

缺点：大程序装不下，小程序浪费严重

第5章　存储器管理

第二节　连续分配方式

二、固定分区分配

1、划分分区方法

- 分区大小不等

  将内存区分成多个较小的分区、适量的中等分区和少量的大分区

优点：适应性较强

缺点：特别大的程序可能仍装不下

第5章　存储器管理

第二节　连续分配方式

二、固定分区分配

2、内存分配

- 建立内存分区使用表

区号	大小	始址	状态
1	8K	16K	可用
2	16K	24K	可用
3	32K	40K	可用
4	64K	72K	可用
5	120K	136K	可用

- 算法

(1)、首次适应算法(FF)：按序选择第一个满足要求的内存区

(2)、最佳适应算法(BF)：仅当与程序大小相当的分区空闲时才予分配

第5章　存储器管理

第二节　连续分配方式

区号	大小	始址	状态
1	8K	16K	A占用
2	16K	24K	可用
3	32K	40K	B占用
4	64K	72K	可用
5	120K	136K	可用

二、固定分区分配

3、固定分区分配举例

- 有A、B、C、D、E五个作业，大小分别为7K，24K，20K，40K，90K。A、B两作业先分别装入内存区1、3运行，C、D、E处于后备队列

FF算法分配情况

区号	大小	始址	状态
1	8K	16K	A占用
2	16K	24K	可用
3	32K	40K	B占用
4	64K	72K	C占用
5	120K	136K	D占用

空闲区(2、4、5)200K只分配使用60K（30%）

SF算法分配情况

区号	大小	始址	状态
1	8K	16K	A占用
2	16K	24K	可用
3	32K	40K	B占用
4	64K	72K	D占用
5	120K	136K	E占用

空闲区(2、4、5)200K分配使用130K（65%）

第5章　存储器管理

第二节　连续分配方式

三、动态分区分配

- 根据用户程序的大小，动态地为之分配连续的内存空间
- 没有用户程序运行时，内存只有一个连续分区（所有的用户区）
- 当用户程序要求分配内存时，依次分配大小与用户程序大小相等的内存，余下为空闲区
- 用户程序执行完成，回收内存，留待后面的用户程序分配使用

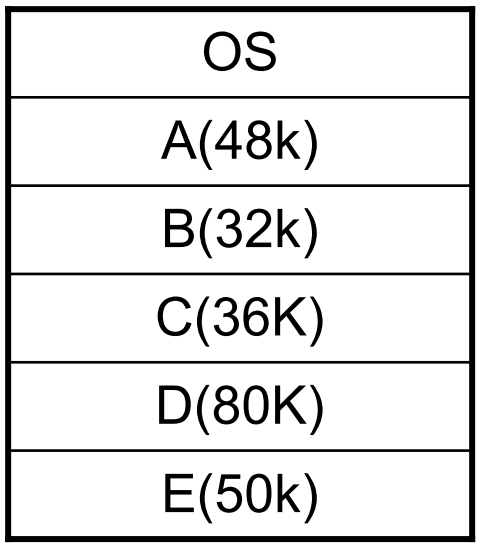
第5章　存储器管理

第二节　连续分配方式

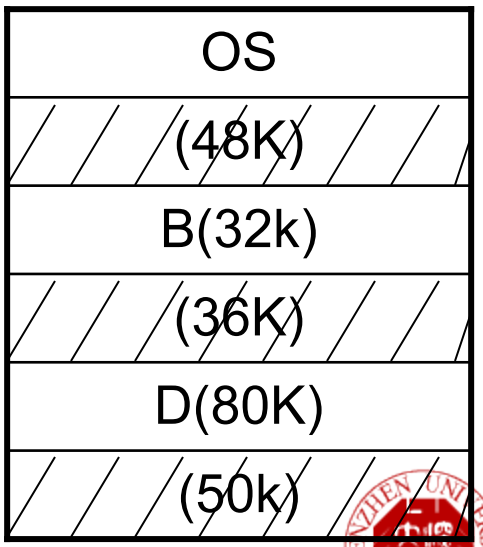
三、动态分区分配

- **动态分区的初始分配举例：**

有A(48K)、B(32K)、C(36K)、D(80K)、E(50K)五个作业



初始分配



A、C、E作业完成后

第5章　存储器管理

第二节　连续分配方式

三、动态分区分配

为了实现动态分区分配，必须解决三个问题：

- **分配所用的数据结构**(分区序号、始址和大小)
- **回收操作**
- **分配算法**

第5章　存储器管理

第二节　连续分配方式

三、动态分区分配

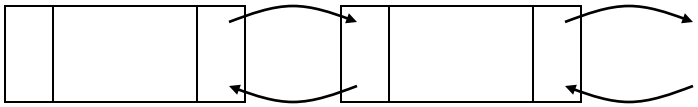
1、分配所用的数据结构

- 记录内存使用情况，为内存分配提供依据
- 主要采用两种方式：

(1)、空闲分区表

区号	大小	始址	状态
1	64K	44K	可用
2	24K	132K	可用
3	40K	210K	可用
4	30K	270K	可用
5	…	…	…

(2)、空闲分区（双向）链表



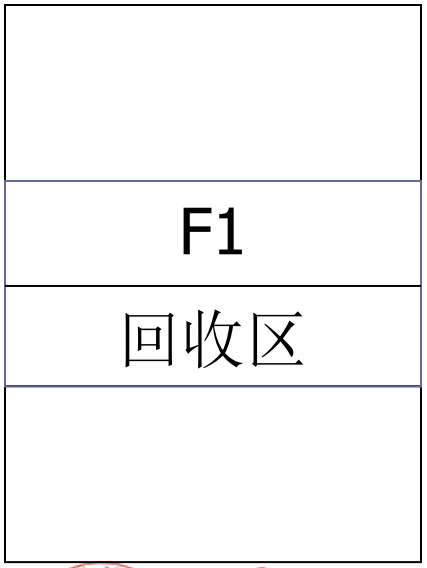
第5章　存储器管理

第二节　连续分配方式

三、动态分区分配

2、回收操作

- 回收区与插入点的前一个空闲分区F1相邻接
- 将回收区与前一分区F1合并，并修改F1大小（大小为两空闲区之和）



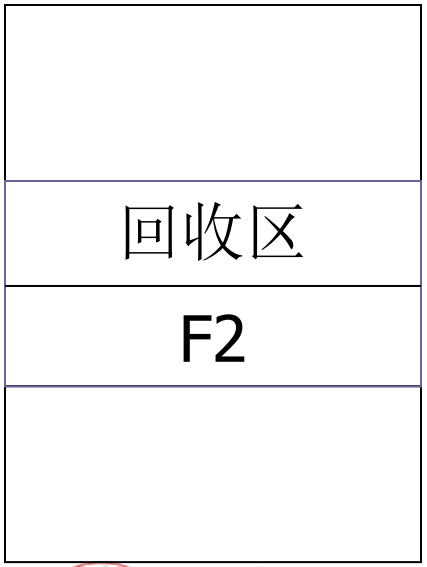
第5章　存储器管理

第二节　连续分配方式

三、动态分区分配

2、回收操作

- 回收区与插入点的后一个空闲分区F2相邻接
- 将回收区与后一空闲分区F2合并，用回收区的首址作为新空闲区的首址，大小为两空闲区之和



第5章　存储器管理

第二节　连续分配方式

三、动态分区分配

2、回收操作

- 回收区与插入点的前、后两个空闲分区相邻接

- 将回收区与前、后空闲分区F1、F2合并，用F1首址作为新空闲区的首址，大小为三空闲区之和，并取消F2表项

第5章　存储器管理

第二节　连续分配方式

三、动态分区分配

2、回收操作

- 回收区不与任何空闲区邻接
- 为回收区单独建立一个新表项，填写回收区的首址和大小，并插入空闲链表中适当的位置

第5章　存储器管理

第二节　连续分配方式

三、动态分区分配

3、分配算法

- **首次适应算法（FF）**

  按序选择第一个满足要求的内存区

优点：保留高地址部分的大空闲区

缺点：低地址存在很多小的、无法利用的空闲分区，且查找时间较长

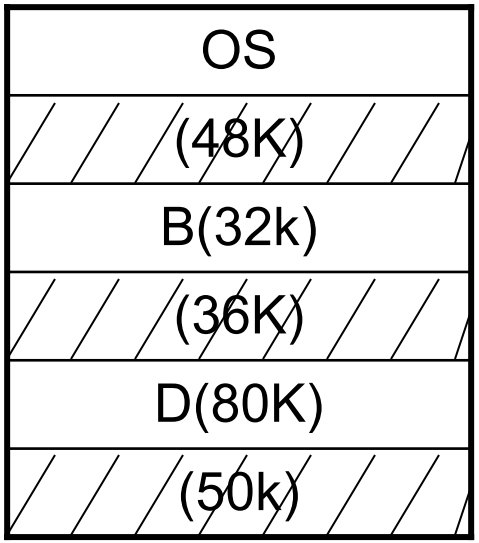
第5章　存储器管理

第二节　连续分配方式

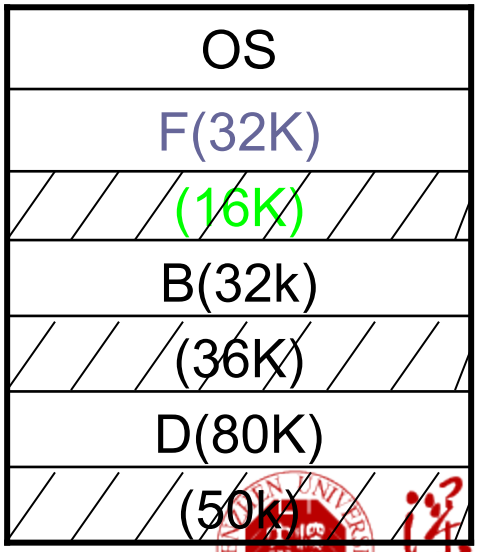
三、动态分区分配

3、分配算法

- 首次适应算法（FF）举例：现有F(32K)加入



F作业加入前



F作业加入后

第5章　存储器管理

第二节　连续分配方式

三、动态分区分配

3、分配算法

- **循环首次适应算法（CF）**

  从上次找到空闲分区的下一个分区开始，按序选择第一个满足要求的内存区

优点：空闲分区在内存中均匀分布，查找时间少

缺点：缺乏大的空闲分区

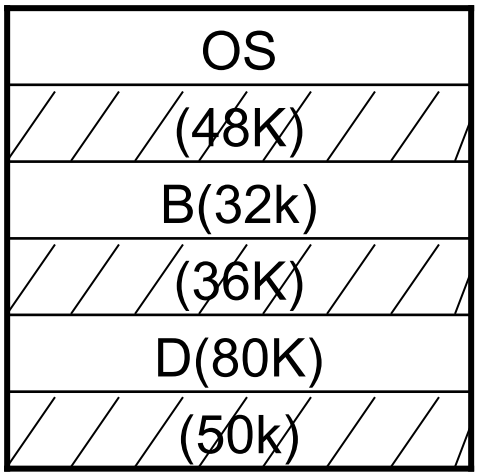
第5章　存储器管理

第二节　连续分配方式

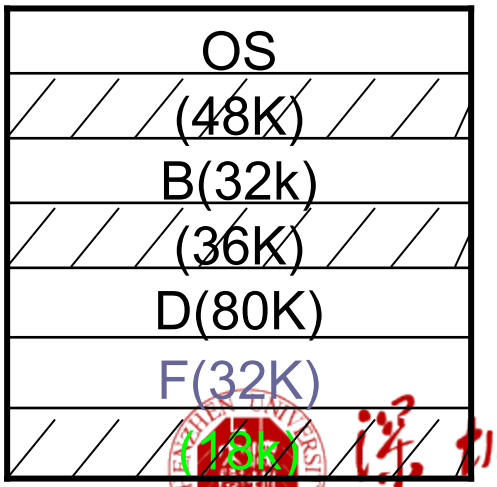
三、动态分区分配

3、分配算法

- 首次循环适应算法(FF)举例：现有F(32K)加入（假设上一次加入的作业为D）



F作业加入前



F作业加入后

第5章　存储器管理

第二节　连续分配方式

三、动态分区分配

3、分配算法

- **最佳适应算法（BF）（宏观上不一定最好）**

  在整个空闲空闲分区中查找大小与用户程序大小最相近的空闲分区

  为了加快查找速度，将空闲分区链表从小到大排列

优点：提高了内存使用效率，保留大的空闲区

缺点：存在许多很小的、无法利用的空闲分区

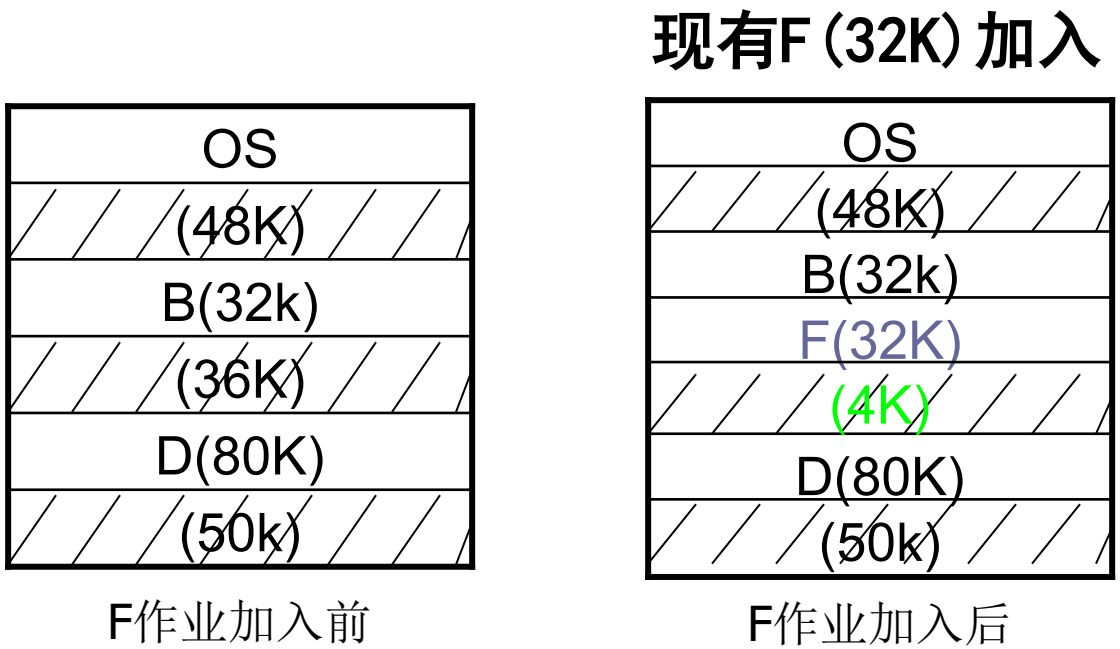
第5章　存储器管理

第二节　连续分配方式

三、动态分区分配

3、分配算法

- **最佳适应算法（BF）举例：**



F作业加入前　　F作业加入后

第5章　存储器管理

第二节　连续分配方式

三、动态分区分配

4、动态分区分配特性

- **优点：**
  可以按照用户进程实际大小，动态地分配内存空间，提高内存的使用效率

- **缺点：**
  不管采用何种算法，都必将产生小的、不可利用的空闲分区（碎片）

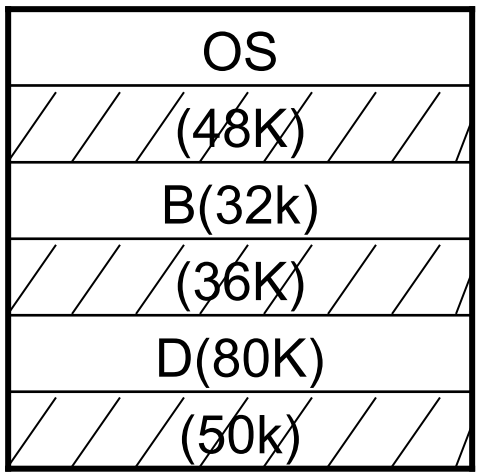
第5章　存储器管理

第二节　连续分配方式

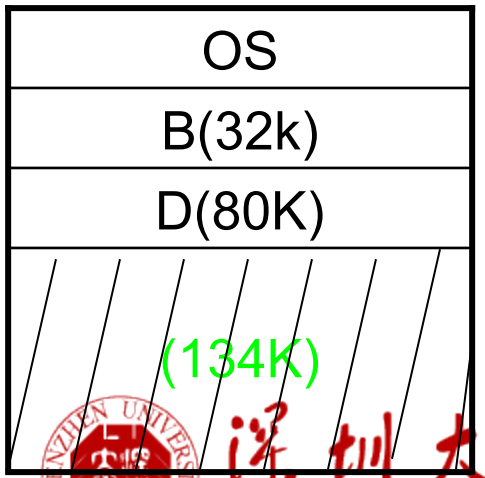
四、动态重定位分区分配

1、紧凑

- 对内存中正在使用的分区进行搬迁，使多个小的空闲分区（碎片）合并为一个大的空闲分区
- 紧凑也称拼接



拼接前



拼接后

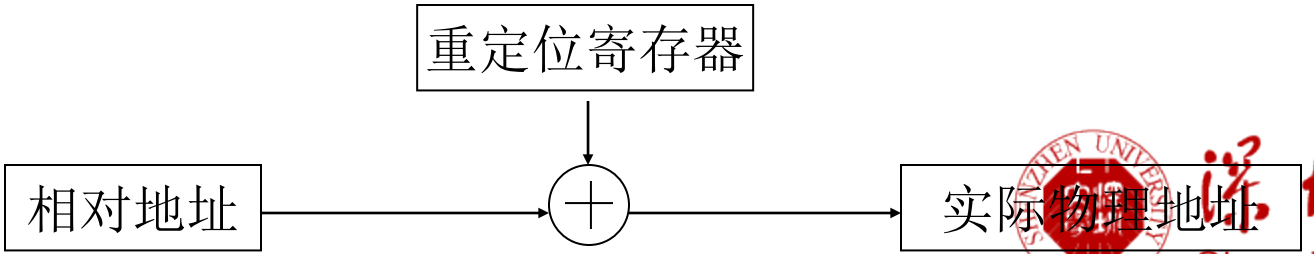
第5章　存储器管理

第二节　连续分配方式

四、动态重定位分区分配

2、动态重定位

- **实现拼接必须有*动态重定位*机制支持**
- **动态重定位功能由重定位寄存器（存放进程首址）实现**



第5章　存储器管理

第三节　对换

一、对换

- 对换是指把内存中暂不能运行的进程，或暂时不用的数据，换出到外存上，以腾出内存空间
- 整体对换：以进程为单位*(挂起操作)*
- 部分对换：以页或段为单位*(虚拟存储器)*

第5章　存储器管理

第三节　对换

二、对换空间的管理

- 将外存分为文件区和对换区
- 对换区是连续的外存存储区
- 整体对换可以采用与连续内存分配类似的方法，实现对换空间的管理（分配和回收）

第5章　存储器管理

第三节　对换

三、进程的换入和换出

1、进程的换出

- 选出被换出的进程：
  首先选择阻塞进程，然后选择低优先级的就绪进程

- 换出过程：
  换出非共享的程序段及数据段

第5章　存储器管理

第三节　对换

三、进程的换入和换出

2、进程的换入

- **对就绪且换出进程，先申请内存**
- **成功则换入**
- **失败，则先换出其它进程，再继续申请换入**

第5章　存储器管理

第四节　基本分页存储管理方式

- 内存离散分配方式：将一个进程分散地分配到许多不相邻接的分区中的内存分配方式
- 离散分配方式主要包括：

1、分页存储管理

2、分段存储管理

3、段页式存储管理

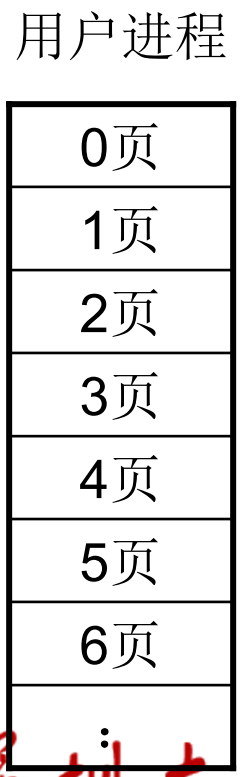
第5章　存储器管理

第四节　基本分页存储管理方式

一、页面和页表

1、页面

- 将*进程的逻辑空间*分成若干大小相等的片，称为页面、或页
- 同一进程的页面按序从小到大进行编号，称页号



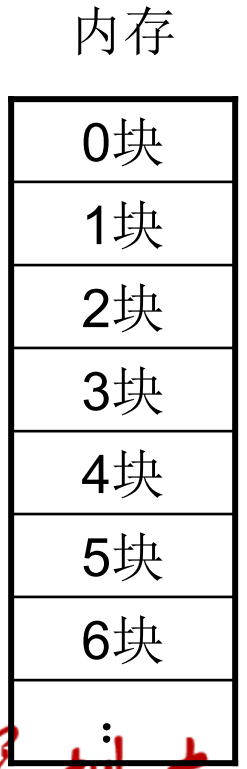
第5章　存储器管理

第四节　基本分页存储管理方式

一、页面和页表

2、内存物理块

- 将*内存空间*分成若干大小相等的存储块，称为**物理块**
- 内存块号按序从小到大进行编号，称**块号**



第5章　存储器管理

第四节　基本分页存储管理方式

一、页面和页表

3、进程页面与内存物理块关系

- 进程页面大小与内存物理块大小完全相等
- 进程的任何页面可以存放在内存任何物理块中
- 除进程最后一个页面外，其他页面都刚好装进一个内存物理块中

第5章　存储器管理

第四节　基本分页存储管理方式

一、页面和页表

4、页面大小（L）

- 为了提高内存使用效率，页面不能太大（尽量减少进程最后一个页面的空闲空间）
- 为了提高内存分配效率及换进/换出效率，页面不能太小
- 页面大小L一般选择2的次幂，如2^9(512Bytes)—2^{12}(4KBytes)之间

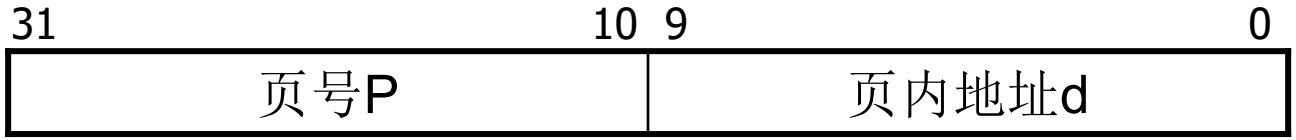
第5章　存储器管理

第四节　基本分页存储管理方式

一、页面和页表

5、进程地址结构

- 进程地址A分为两部分：页号P与页内地址d



- 页号P：　　P = INT(A/L)
- 页内地址d：　d = A MOD L
- 如A=2170B，L=1KB，则P=2，d=122

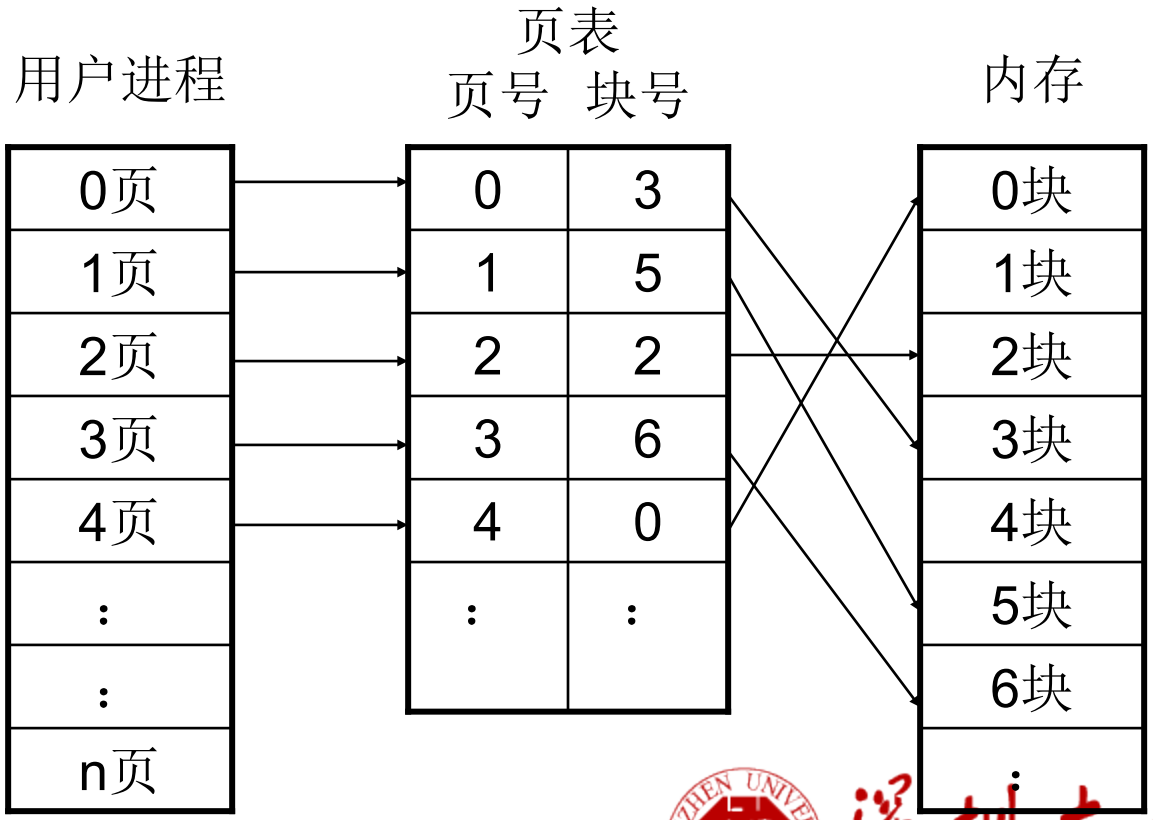
第5章　存储器管理

第四节　基本分页存储管理方式

一、页面和页表

6、页表

- 页表记录进程每一页在内存中存放的块号，即记录进程页面与内存物理块号之间映射关系



第5章　存储器管理

第四节　基本分页存储管理方式

二、地址变换机构

1、基本地址变换机构

- 页表一般存放在内存中
- 页表的始址和页表的长度存放在进程PCB中
- 在进程执行时，页表的始址和页表的长度送入页表寄存器PTR

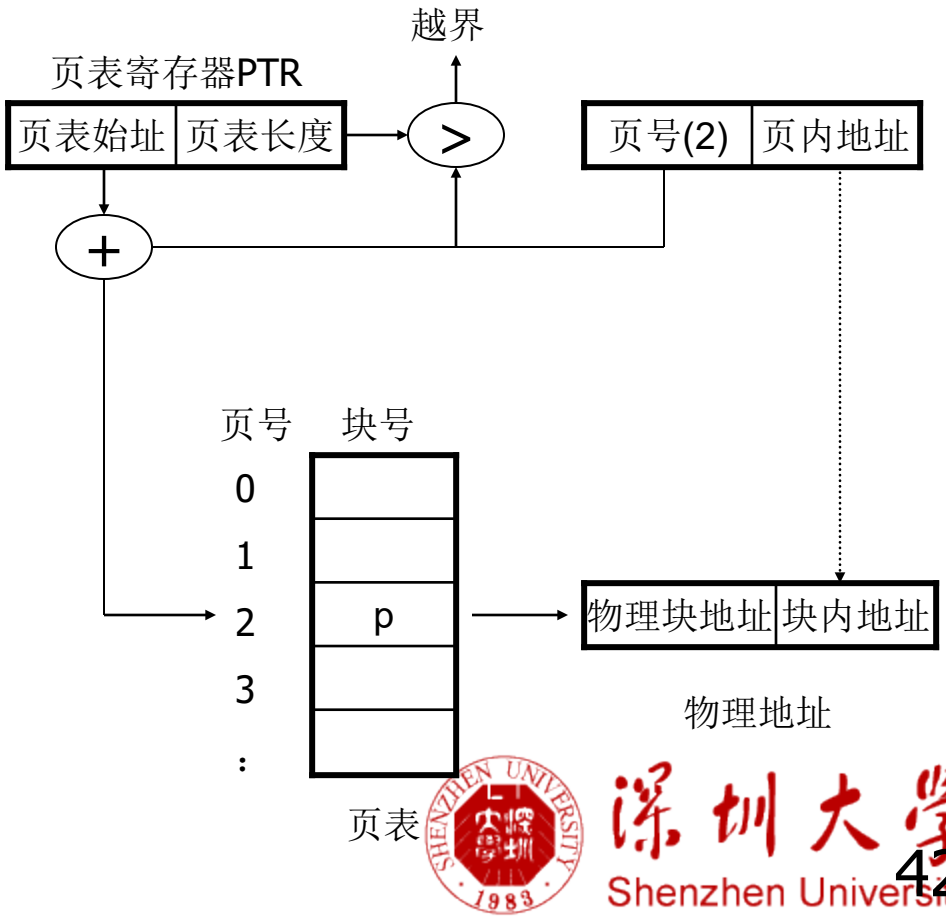
第5章　存储器管理

第四节　基本分页存储管理方式

二、地址变换机构

1、基本地址变换机构

- 当进程要访问某个逻辑地址时，分页地址机构自动将逻辑地址分为页号与页内地址
- 如果页号大于页表长度，越界错误
- 将页表始址与页号和页表项长度的乘积相加，得页表（内存中）中位置
- 从页表中相应位置，得到物理块号，送入物理地址寄存器的内存物理块字段中
- 将逻辑地址的页内地址部分送入物理地址寄存器的块内地址字段中



第5章　存储器管理

第四节　基本分页存储管理方式

二、地址变换机构

1、基本地址变换机构

- 缺点：
  访问一次逻辑地址，必须两次访问内存
- 第一次访问内存为取得逻辑地址对应的内存物理地址
- 第二次访问内存为取得逻辑地址中的数据

第5章　存储器管理

第四节　基本分页存储管理方式

二、地址变换机构

2、具有快表的地址变换机构

- 快表：具有并行查寻能力的特殊高速存储器，也称为“联想存储器”
- 快表的查找速度一般为内存访问的数倍，如快表访问一次费时20ns，而内存可能需要100ns
- 将页表放在快表中，将可以节省从逻辑地址到物理地址的映射时间

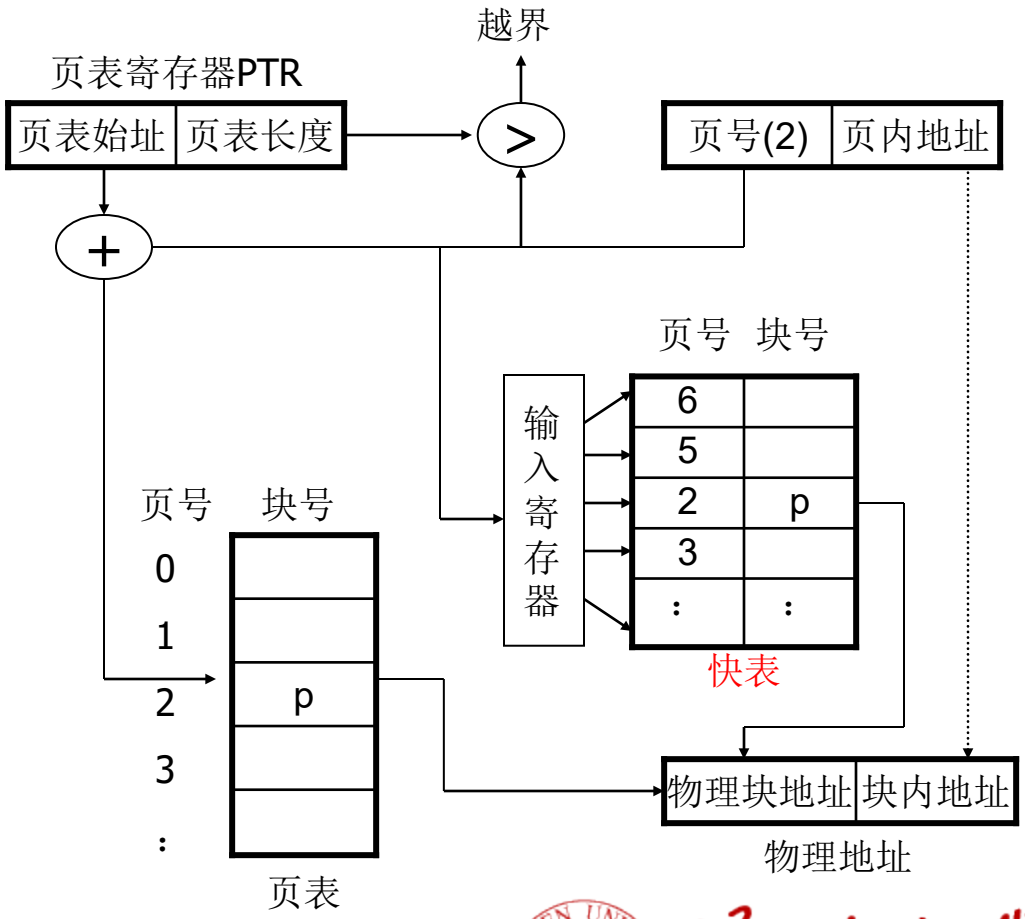
第5章　存储器管理

第四节　基本分页存储管理方式

二、地址变换机构

2、具有快表的地址变换机构

- **分页地址机构自动将逻辑地址分为页号与页内地址**
- **越界检查**
- **将页号送入快表中进行并行查找，如果从快表中找到所需页号，则将快表中对应的物理块号送物理地址寄存器**
- **否则将页表始址与页号和页表项长度的乘积相加，得页表中位置，进而得到物理块号，送入物理地址寄存器的内存物理块字段中**
- **页内地址送入物理地址寄存器**



第5章　存储器管理

第四节　基本分页存储管理方式

二、地址变换机构

2、具有快表的地址变换机构举例

- **快表访问时间为20ns，内存访问时间为100ns**
- **如果由快表访问成功，需时120ns，否则为220ns**

命中率	有效访问时间T
0%	T=0.0*120 + 1.0*220 = 220ns
50%	T=0.5*120 + 0.5*220 = 170ns
80%	T=0.8*120 + 0.2*220 = 140ns
90%	T=0.9*120 + 0.1*220 = 130ns
98%	T=0.98*120+0.02*220= 122ns

第5章　存储器管理

第四节　基本分页存储管理方式

二、地址变换机构

2、具有快表的地址变换机构举例

- **如果对快表的访问与对内存的访问同时进行**
- **如果由快表访问成功，需时120ns，否则为200ns**

命中率	有效访问时间T
0%	T=0.0*120 + 1.0*200 = 200ns
50%	T=0.5*120 + 0.5*200 = 160ns
80%	T=0.8*120 + 0.2*200 = 136ns
90%	T=0.9*120 + 0.1*200 = 128ns
98%	T=0.98*120+0.02*200= 121.6ns

第5章　存储器管理

第四节　基本分页存储管理方式

三、两级页表

1、一级页表存在的问题

- 当逻辑地址空间非常大（2^{32}–2^{64}）时，**页表将变得非常大**
- 如每个页面大小为4KB，逻辑地址空间为2^{32}时，需要2^{20}（即1M）页表项，每个页表项占用4个字节，每个进程共需要4MB页表(连续内存空间)

第5章　存储器管理

第四节　基本分页存储管理方式

三、两级页表

2、两级页表操作方式

- **采用两级页表**
- **将页表以离散方式放入内存**

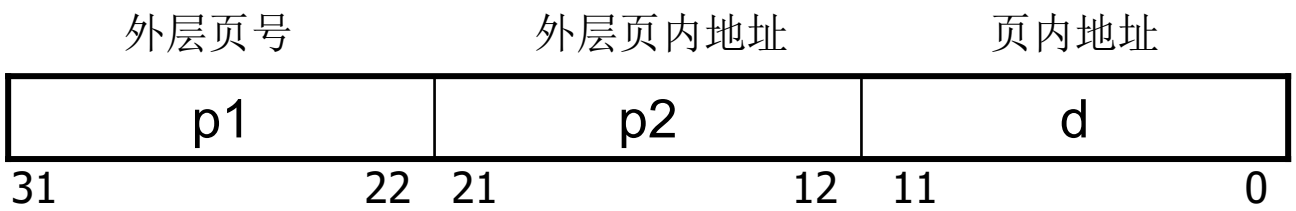
第5章　存储器管理

第四节　基本分页存储管理方式

三、两级页表

3、两级页表地址结构

- 采用外层页表、页表和页内地址三层结构



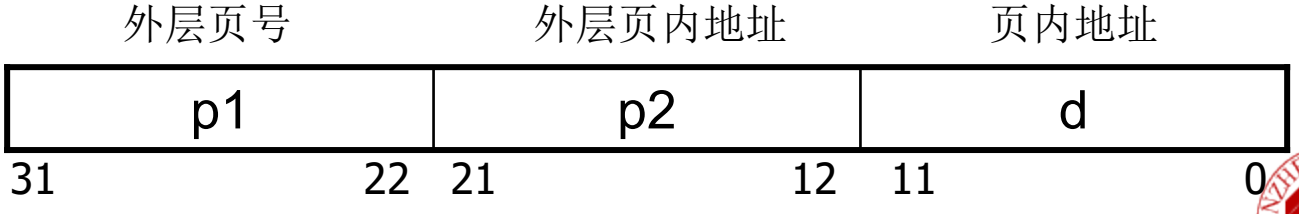
- 外层页表存放页表存放的位置

第5章　存储器管理

第四节　基本分页存储管理方式

三、两级页表

- 有一基本分页存储管理系统，内存块大小为512Bytes，（页表存放于一块中），每个块号占用4个Bytes，如果页表也以离散方式放在内存中，求采用一级页表和两级页表，系统能支持文件的逻辑空间最多为多少？假如逻辑地址为12345H，求页内地址和外层页号。



第5章　存储器管理

第四节　基本分页存储管理方式

三、两级页表

- 每个内存块号占4Bytes，即每个页表项占4B
- 每个内存块大小为512Bytes，
- 每个内存块可放下最多512/4=128个页表项
- 采用一级页表能支持文件的逻辑空间最多为：128x512=64KBytes
- 采用二级页表能支持文件的逻辑空间最多为：128x128x512=8MBytes

第5章　存储器管理

第四节　基本分页存储管理方式

三、两级页表

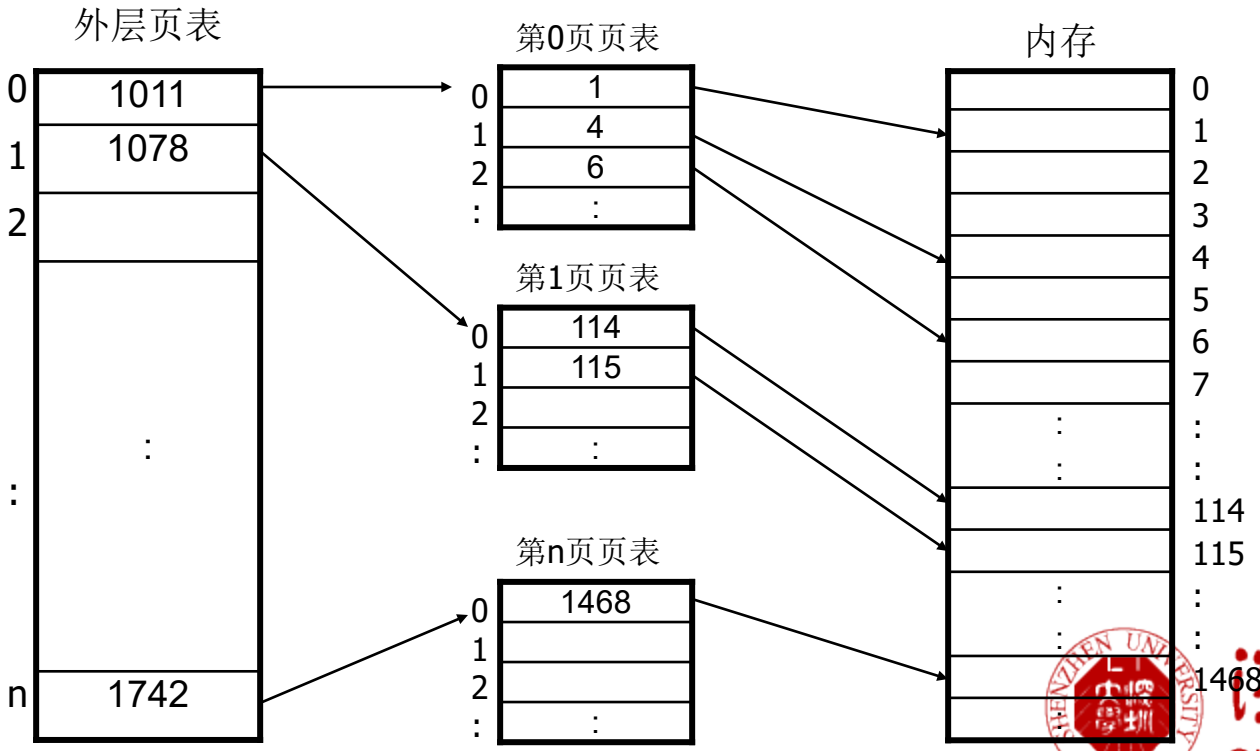
- 512占用9bits，即页内地址占9bits
- 128占7bits，即外层页内地址占7bits
- 12345H=0001，0010，0011，0100，0101b
- 页内地址=后9bits=1，0100，0101b=145H
- 外层页号=(9+7)bits以上地址=0001b=1H

第5章　存储器管理

第四节　基本分页存储管理方式

三、两级页表

3、两级页表地址结构

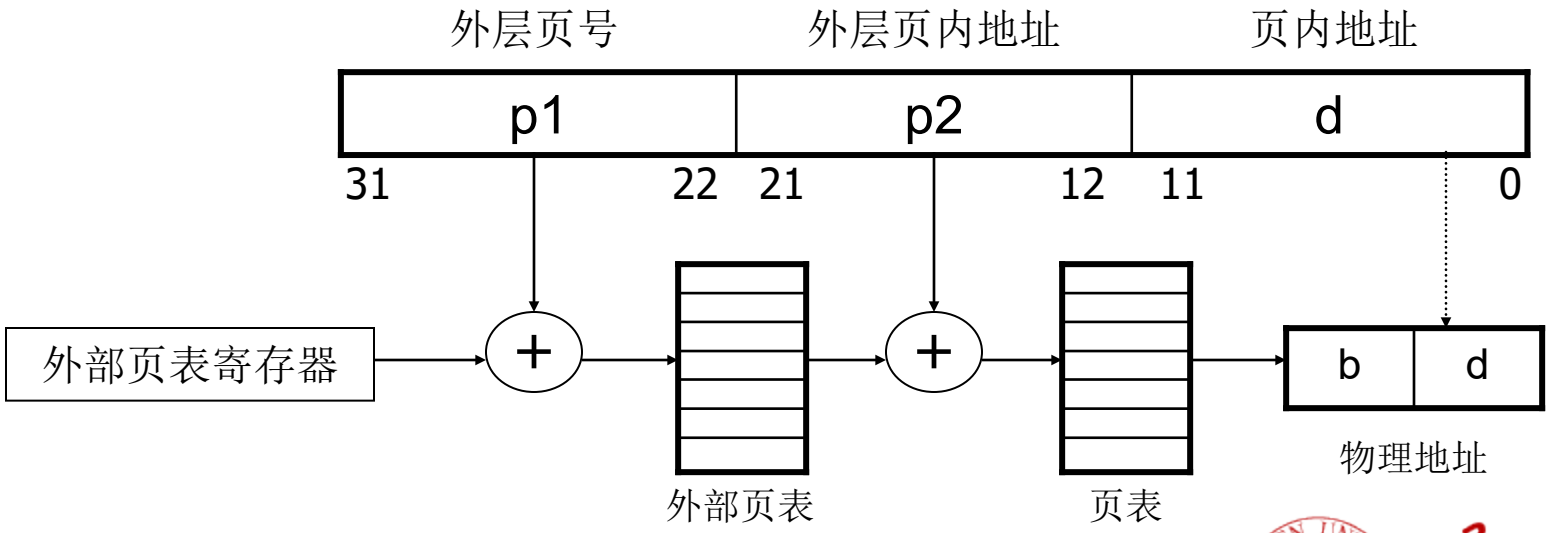


第5章　存储器管理

第四节　基本分页存储管理方式

三、两级页表

4、两级页表地址变换机构



第5章　存储器管理

第四节　基本分页存储管理方式

四、多级页表

- 当逻辑地址空间非常巨大（如2^{64}）时，页表将变得非常巨大
- 如每个页面大小为4KB，逻辑地址空间为2^{64}时，需要2^{52}（即4096G）页表项，每个页表项占用8个字节，每个进程共需要32768GB页表
- 必须分成更多级，如四级等

第5章　存储器管理

第五节　分段存储管理方式

一、分段

- 分段是指将一个作业（进程）分成多个具有独立逻辑意义的自然段
- 如主程序段、数据段、堆栈段、子程序段等
- 每个段都是从0开始编址的一段连续空间

第5章　存储器管理

第五节　分段存储管理方式

一、分段存储管理的目的

- **从内存管理的角度看，分段可以提高内存分配效率**

第5章　存储器管理

第五节　分段存储管理方式

二、分段存储管理的目的

从用户角度看，分段可以达到以下目的：

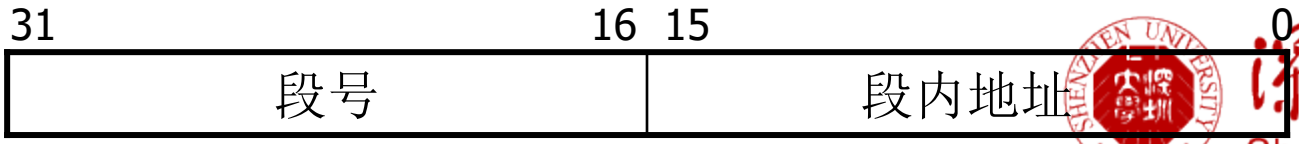
- **方便编程**
- **分段共享**
- **分段保护**
- **动态链接**
- **动态增长**

第5章　存储器管理

第五节　分段存储管理方式

三、分段地址结构

- 作业（进程）由多个段组成
- 每个段（从0开始）独立编址
- 整个作业（进程）的地址是二维结构
- 每个段有一个唯一的名称；为实现方便，一般用段号来代表
- 分段地址结构为：

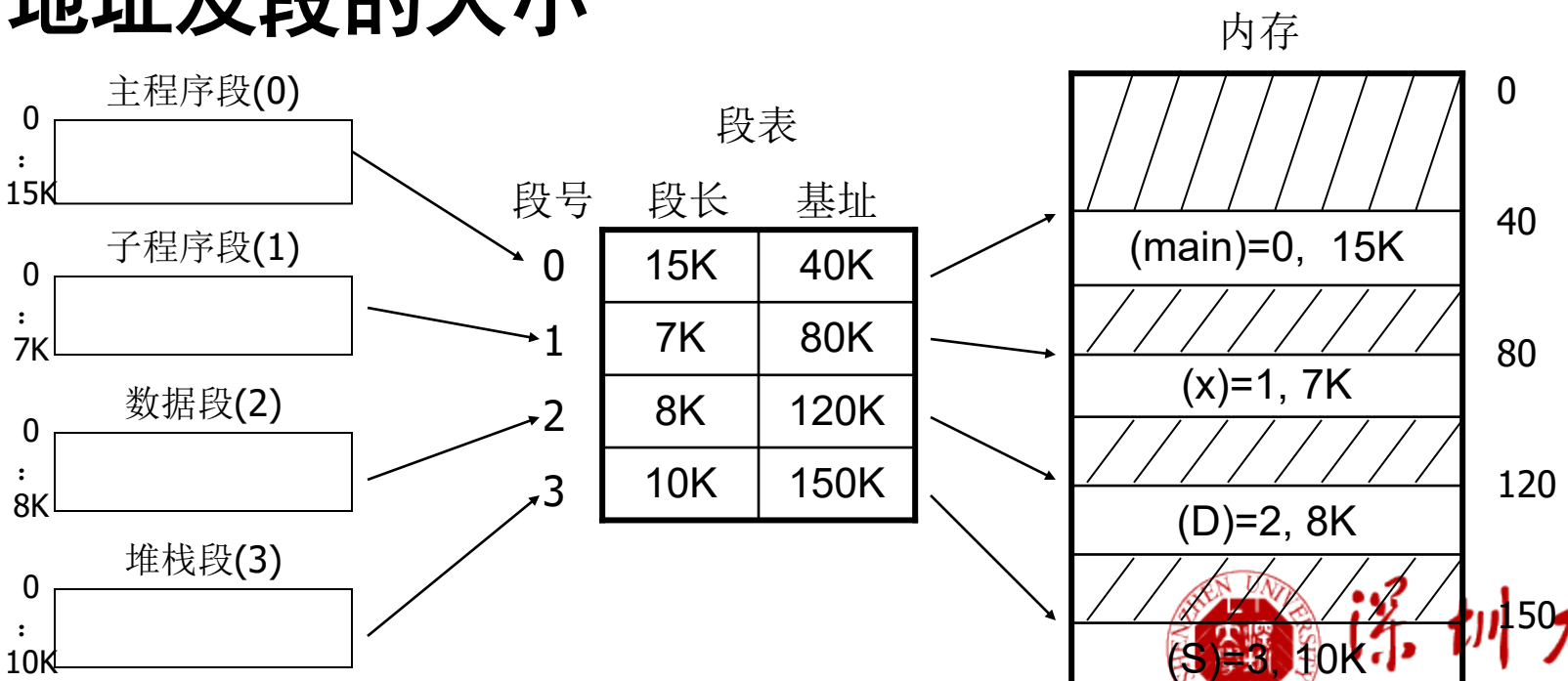


第5章　存储器管理

第五节　分段存储管理方式

四、段表

- **段表**记录进程每一段在内存中存放的(开始)地址及段的大小

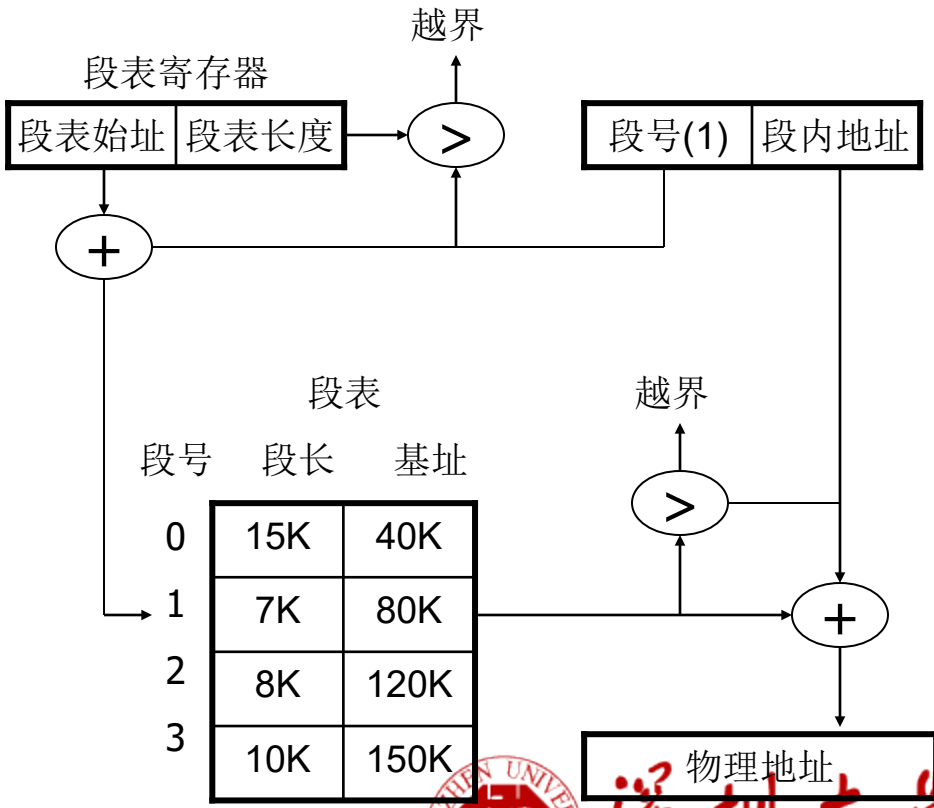


第5章　存储器管理

第五节　分段存储管理方式

五、地址变换机构

- 当进程要访问某个逻辑地址时，先取得段号
- 如果段号大于段表长度，越界错误
- 将段表始址与段号和段表项长度的乘积相加，得段表中位置
- 从段表中相应位置，得到段的首址及段的长度
- 如段内地址大于段长度，越界错误
- 将段的内存首址与段内地址相加，得内存物理地址



第5章　存储器管理

第五节　分段存储管理方式

六、具有快表的地址变换机构

- 为了访问段内数据，必须两次访问内存
- 同样可以采用快表
- 将段表放入快表中
- 由于进程中，段的数目远比页的数目少，因此，所需要的段也比页面少，因此，访问快表的命中率也比较高，因此，采用快表的分段地址变换，一般仅比常规内存访问慢10%~15%

第5章　存储器管理

第五节　分段存储管理方式

七、信息共享

- 由于段是信息的逻辑单位，含有一组有意义的相对完整的信息
- 因此，多个作业（进程）共享同一个有独立意义的段是非常方便的，只需在各自的段表中，标上相同的段基址和段长度即可
- 如Windows中的动态链接库中的纯代码段，可同时被多个进程共享

第5章　存储器管理

第五节　分段存储管理方式

八、分页和分段的主要区别

- 页是信息的物理单位（满足系统管理的需要）
  段是信息的逻辑单位（满足用户的需要）
- 页的大小固定（由系统决定）
  段的大小不固定（由用户决定）
- 分页的作业（进程）地址是一维的
  分段的作业（进程）地址是二维的

第5章　存储器管理

第六节　段页式存储管理方式

一、段页式管理的引入

- 由于段是由用户按编程需要设定的，因此其大小不固定
- 段需要连续分配内存空间，因此，同样存在连续分配存储管理方式中存在的缺点
- 结合段与页式管理的优点，既节省内存空间，提高内存分配的效率，又兼顾用户编程的需要

第5章　存储器管理

第六节　段页式存储管理方式

二、基本原理

- 段页式系统是分段和分页原理的组合
- 先将用户作业（进程）分成若干具有一定逻辑意义的段，再将段分成大小固定的若干个页
- 段页式系统需要同时设置段表和页表
- 段表中存放页表始址及页表大小

第5章　存储器管理

第六节　段页式存储管理方式

三、段页式地址结构

- **段页式地址结构为：**

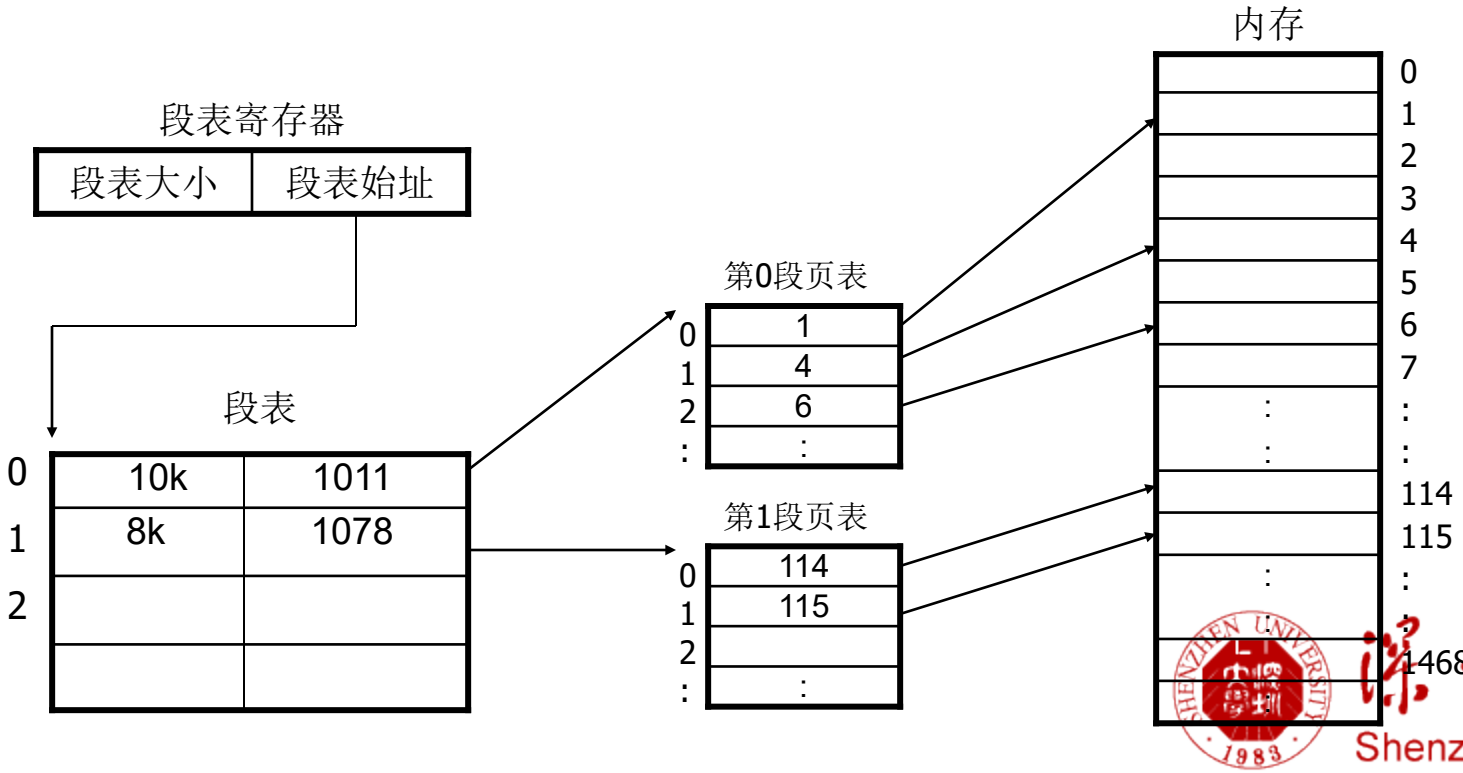
段号(S)	段内页号(p)	页内地址(d)

第5章　存储器管理

第六节　段页式存储管理方式

三、段页式地址结构

- **地址映射：**



第5章　存储器管理

第六节　段页式存储管理方式

四、段页式地址变换机构

- 当进程要访问某个逻辑地址时，先取得段号
- 如果段号大于段表长度，越界错误
- 将段表始址与段号相加，得段表中位置
- 从段表中相应位置，得到该段页表的首址
- 将段内页号与页表首址相加，得页表中位置
- 从页表中相应位置，得到物理块号p
- 将块号p与页内地址d组合成物理地址