内存管理概念

常考点

与覆盖交换技术相关	1. 使用交换技术时，一个进程正在I/O操作，则不能交换出主存；开辟一个缓冲I/O区后，可以交换出主存 2. 覆盖和交换的提出是为了解决主存空间不足的问题，不是从物理上解决，而是将暂时不用的部分换出主存以节省空间，从而在逻辑上扩充主存 3. 单一连续存储管理可采用覆盖技术
程序运行过程相关	1. 形成逻辑地址的阶段：链接 2. 形成物理地址的阶段：装入或程序运行时
与重定位相关	1. 动态重定位是在作业的执行过程中进行的 2. 静态重定位是在转入的时候进行的 3. 固定分区可以采用静态重定位（装入后位置不再改变） 4. 在整个系统设置一个重定位寄存器，用来存放程序在内存中的始址
其他	内存保护需要由OS和硬件机构合作完成，以保证进程空间不被非法访问

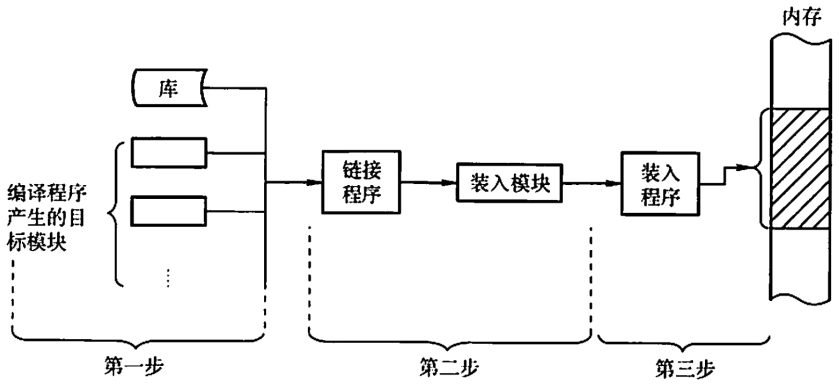
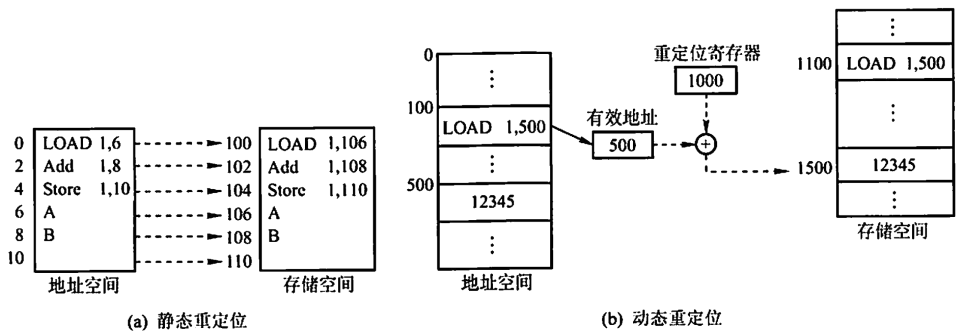
基本概述

内存管理是什么?	内存管理是操作系统对内存的划分和动态分配	
内存管理的目的	1. 为了更好地支持多道程序并发执行 2. 方便用户 3. 提高内存利用率	
内存管理的功能	**1. 内存空间的分配与回收**	由OS完成主存储器空间的分配和管理
	2. 地址转换	存储管理将逻辑地址转换为物理地址
	3. 内存空间的扩充	利用虚拟存储技术/自动覆盖技术，从逻辑上扩充内存
	4. 内存共享	允许多个进程访问内存的同一部分
	5. 存储保护	保证多道作业在各自的存储空间运行，互不干扰
内存管理的分配	**1. 连续分配**	单一连续分配→固定分区分配→动态分区分配
	2. 不连续分配	分段存储管理→分页存储管理→段页存储管理

逻辑地址与物理地址

	逻辑地址	**物理地址**
定义	每个目标模块都从0号单元开始编址的地址	物理地址空间是指内存中物理单元的集合
特点	• 不同进程可以有系统的逻辑地址，这些逻辑地址可以映射到主存的不同位置 • 进程运行时，看到和使用的地址都是逻辑地址 • 将逻辑地址转换为物理地址的过程叫做地址重定位	• 物理地址是地址转换的最终地址

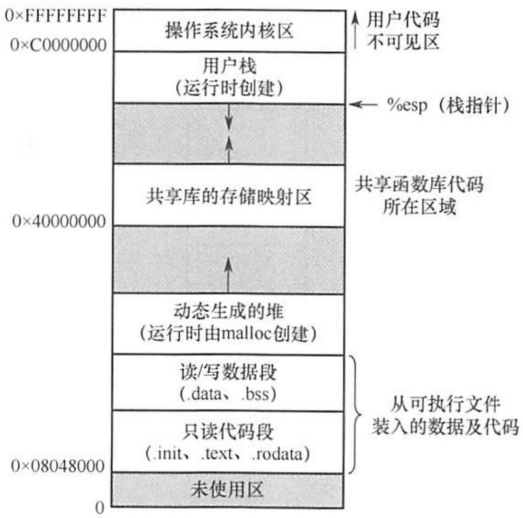
将程序变成可在内存中执行的程序过程【也就是程序的链接与接入过程】

过程图	 图 3.1　将用户程序变为可在内存中执行的程序的步骤
step1：编译	编译：由编译程序将用户源代码编译成若干目标模块
step2：链接	链接：由链接程序将目标模块和库函数链接，形成完整的装入模块 链接类别 1. 静态链接 2. 动态链接 3. 运行时动态链接
step3：装入	装入：是由装入程序将装入模块装入内存运行 装入类别 1. 静态装入：在编程时把物理地址计算好 2. 可重定位装入：装入时把逻辑地址转换为物理地址，但装入后不能改变 3. 动态重定位装入：执行时再决定装入的地址并装入，装入后有可能会换出
例图	 图 3.2　重定位类型

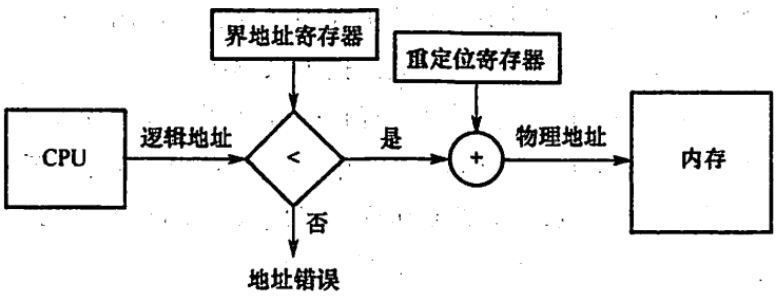
内存共享

概念	• 只有只读区域的进程空间可用共享 • 纯代码/可重入代码 = 不能修改的代码，不属于临界资源 • 可重入程序通过减少交换数量来改善系统性能
实现方式	1. 段的共享 2. 基于共享内存的进程通信（第二章的同步互斥） 3. 内存映射文件

进程的内存映像

定义	当一个进程调入内存运行时，就构成了进程的内存映像	
组成要素	1. 代码段	代码段是只读的，可以被多个进程共享
	2. 数据段	程序运行时加工处理的对象，包括全局变量和静态变量
	3. 进程控制块PCB	存放在系统区，OS通过PCB控制和管理进程
	4. 堆	用来存放动态分配的变量
	5. 栈	用来实现函数调用的
	 图 3.3　内存中的一个进程	

内存共享

目的	确保每个进程都有一个单独的内存空间
方法	方法1：在CPU中设置一对上下限存储器，判断CPU访问的地址是否越界 方法2：使用重定位寄存器和界地址寄存器（只有OS内存才可以使用这两个寄存器） • 重定位寄存器/基地址寄存器含最小的物理地址值【用于“加”】 • 界地址寄存器含逻辑机制的最大值【用于“比”】 • 逻辑地址+重定位寄存器的值=物理地址
例图	 图 3.4　重定位寄存器和界地址寄存器的硬件支持

内存管理方法

分页和分段的比较

	分段	分页	段页式
地址映射表	每个进程由多个不等的段组成	每个进程一张页表，且进程的页表驻留在内存中	每个进程一张段表，每个段一张页表
地址结构	段号+段内偏移	页号+页内偏移	段号+段内页号+页内偏移量
地址结构维度	二维	一维	二维
供什么感知	供用户感知	供操作系统感知	无
以什么单位划分	• 以段为单位分配 • 每段是一个连续存储区 • 每段不一定等长 • 段与段之间可连续，也可不连续	• 逻辑地址分配按页分配 • 物理地址分配按内存块分配	• 分段方法来分配管理用户地址空间 • 分页方法来管理物理存储空间
长度是否固定	段长不固定	页长固定	段长不固定，页长固定
访问主存次数	2次	2次	3次
碎片情况	只产生外部碎片	产生内部碎片	产生内部碎片

常考点

共享相关	1. 段的共享是通过两个作业的段表中相应表项指向被共享的段的统一物理副本实现的
存储器相关	1. 对主存储器的访问以字节或字为单位 2. 对主存储器的分配以块或段为单位
其他	1. 确定一个地址需要几个参数，作业地址空间就是几维的

方法一：连续分配

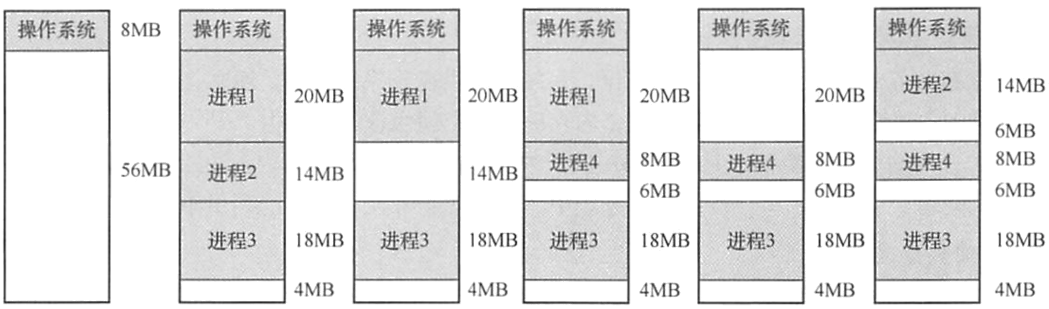
<table>
<tr><td>定义</td><td colspan="3">连续分配管理是为一个用户程序分配一个连续的内存空间</td></tr>
<tr><td>特点</td><td colspan="3">• 用户程序在主存中都是连续存放的
• 非连续分配的方式的存储密度 < 连续分配方式</td></tr>
<tr><td>碎片</td><td colspan="3">• 内部碎片：当程序小于固定分区大小时，也要占用一个完整的内存分区，导致分区内部存在空间浪费
• 外部碎片：内存中产生的小内存块</td></tr>
<tr><td rowspan="6">分类</td><td></td><td>1. 单一连续分配</td><td>2. 固定分区分配</td></tr>
<tr><td>定义</td><td>• 在此方法下，内存分为两个区
• 系统区：供OS用，在地址区
• 用户区：内存用户空间由一道程序独占</td><td>• 用户内存空间划分为固定大小(分区大小相等或不等)的区域
• 每个区装一道作业</td></tr>
<tr><td>优点</td><td>• 简单，无外部碎片
• 无需进行内存保护（内存中永远只有一道程序）</td><td>• 简单</td></tr>
<tr><td>缺点</td><td>• 只能用于单用户单任务的OS
• 有内部碎片，存储器利用率极低</td><td>• 程序太大可能放不下任何一个分区，有内部碎片
• 不能实现多进程共享一个主存区，存储空间利用率低</td></tr>
<tr><td colspan="3">3. 动态分区分配：进程转入内存时，根据进程的实际需要，动态地分配内存；动态分区是在作业装入时动态建立的</td></tr>
<tr><td colspan="3"><table>
<tr><th>动态分配算法</th><th>按什么次序链接的</th><th>特点</th></tr>
<tr><td>a. 首次适应算法</td><td>按地址递增的次序</td><td>最简单，效果最好，速度最快</td></tr>
<tr><td>b. 邻近适应算法</td><td>从上次分配出去的地址访问</td><td>比首次适应算法差</td></tr>
<tr><td>c. 最佳适应算法</td><td>按容量递增的次序</td><td>性能很差，会产生最多的外部碎片</td></tr>
<tr><td>d. 最坏适应算法</td><td>按容量递减的次序</td><td>可能导致没有可用的大内存块，性能差</td></tr>
</table></td></tr>
<tr><td colspan="3"> 

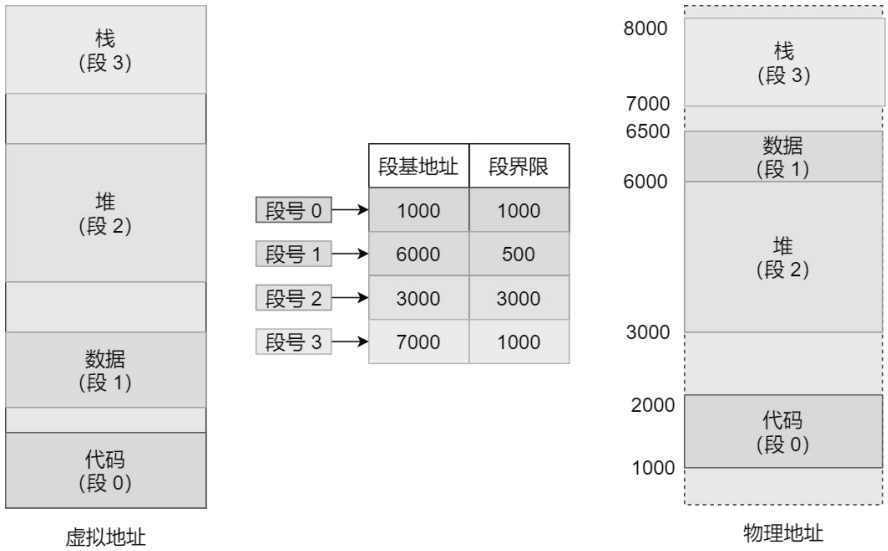
图 3.6　动态分区分配</td></tr>
</table>

方法二：分段

概念

分段 = 程序由若干个逻辑分段组成，不同的段有不同的属性，用分段的方式把程序进行分离
段表 = 一张逻辑空间与内存空间映射的表

概念图



段的共享与保护

- 段的共享 = 通过两个作业的段表中相应表项指向被共享的段的同一物理副本
- 段的保护有两种
  - 存储控制保护
  - 地址越界保护

优缺特点

优点	缺点	特点
• 能产生连续的内存空间 • 分段存储管理能反映程序的逻辑结构并有利于段的共享和保护 • 程序的动态链接与逻辑结构有关，分段存储管理有利于程序的动态链接	• 会产生外部碎片 • 内存交换的效率低	• 满足程序员/用户 • 方便编程，分段共享 • 分段保护，动态链接，增长 • 分段管理地址空间是二维的 • 每段的长短不同

地址变换

物理地址 = 段基址 + 偏移量

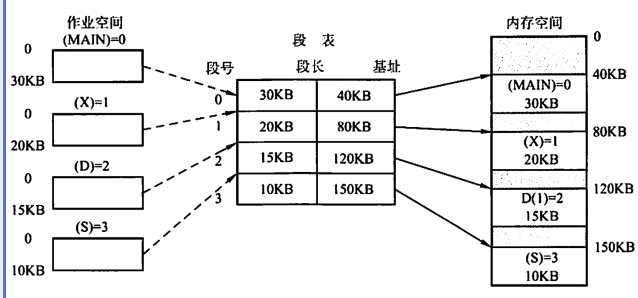


图 3.15 利用段表实现物理内存区映射

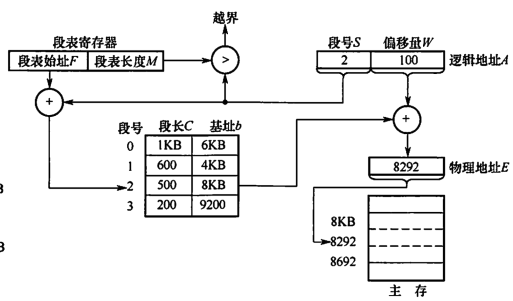


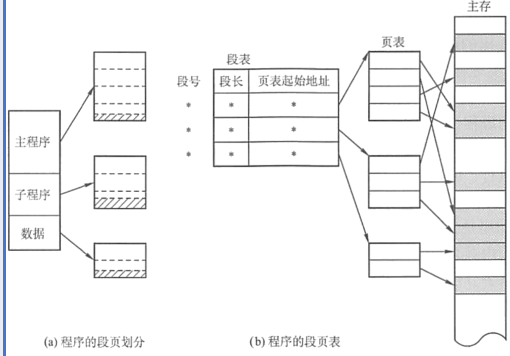
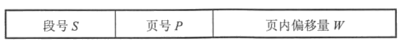
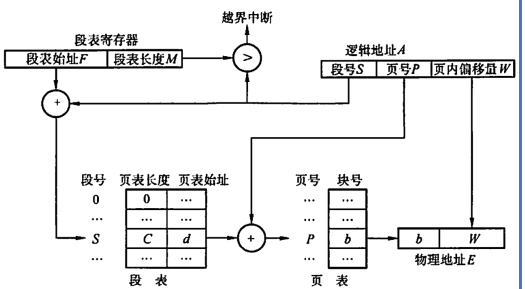
图 3.16 分段系统的地址变换过程

方法三：分页

概念	分页	• 把整个虚拟和物理内存空间切成一段段固定尺寸的大小，在linux中，每一页的大小为4kB
	页/页面	• 页就是进程中的块 • 页面大→页内碎片增多，降低内存的利用率 • 页面小→进程的页面数大，页表过长，占用大量内存，增加硬件地址转换的开销，降低页面换入/出的效率
	地址结构	• 地址结构 = 页号P+页内偏移量W • 地址结构决定了虚拟内存的寻址空间有多大 • 完成地址转换工作的是有硬件的地址转换机构，而不是地址转换程序
	页表	• 页表就是记录页面在内存中对应的物理块号 • 页表的起始地址放在页表基址寄存器PTBR中 • 页表是存储在内存里的，内存管理单元 （MMU）就做将虚拟内存地址转换成物理地址的工作

优缺特点	优点	缺点	特点
	• 能有效地提高内存利用率	• 会产生内部碎片	• 逻辑地址分配按页分配 • 物理地址分配按内存块分配 • 划分的页面大小都相同 • 所有进程都有一张页表 • 页表存在内存中 • 分页是面向计算机的 • 整个系统设置一个页表寄存器用于存放页表在内存中起始地址和长度

方法四：段页

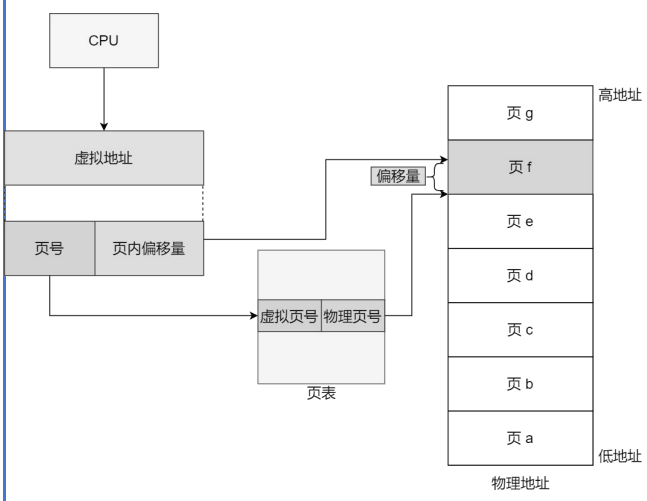
实现方法	step1：将程序划分为多个有逻辑意义的段【分段】 step2：对分段划分出来的连续空间，再划分固定大小的页【分页】
地址变换	• 作业的逻辑地址划分为：段号，段内页号，页内偏移量 • 对内存的管理以存储块为单位，地址空间是二维的  图 3.17 段页式管理方式  图 3.18 段页式系统的逻辑地址结构  图 3.19 段页式系统的地址变换机构
得到物理地址的3次内存访问	1. 访问段表，得到页表起始地址 2. 访问页表，得到物理页号 3. 将物理页号与页内偏移组合，得到物理地址

逻辑地址与物理地址

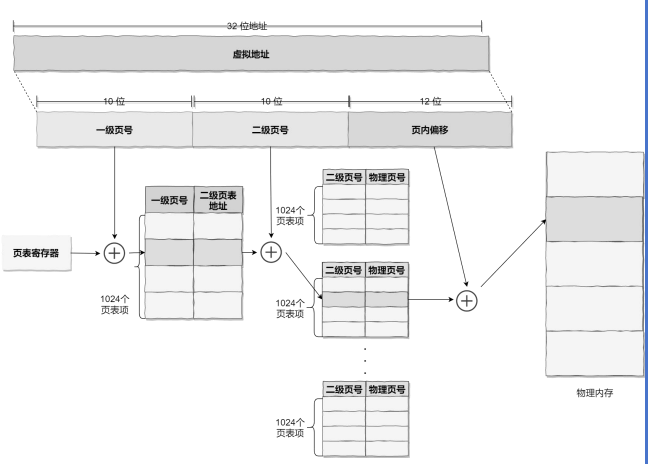
	知识点
基本地址变换	1. 把虚拟内存地址，切分成页号和偏移量 2. 根据页号，从页表里面，查询对应的物理页号 3. 直接拿物理页号，加上前面的偏移量，就得到了物理内存地址 分页产生的页表过大，使用多级页表，解决空间上的问题
两级页表地址变换	1. 一级页表覆盖到全部虚拟地址空间，二级页表在需要时创建 2. 建立多级页表的目的在于建立索引，以便不用浪费主存空间区存储无用的页表项，也不用盲目地顺序式查找页表项 3. 页表寄存器存放的是一级页表起始物理地址 很多层参与，时间上开销大，加入TLB，提高地址的转换速度
具有快表的地址变换	• 快表是相联存储器(Translation Lookaside Buffer，TLB) • 快表也叫页表缓存，转址旁路缓存 • 快表专门存放程序最常访问的页表项的Cache • 快表位于CPU芯片中，用于加速地址变换的过程 • CPU芯片中，封装了MMU(内存管理单元) • MMU用来完成地址转换和TBL的访问与交互

结构图

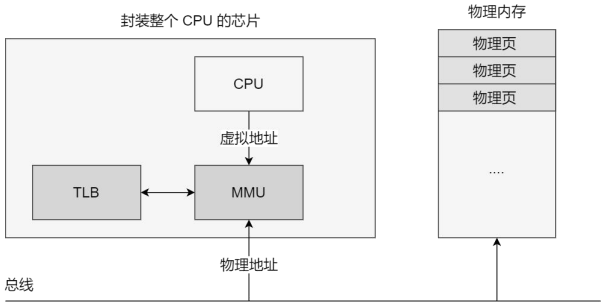
基本地址变换



两级页表地址变换



具有快表的地址变换



虚拟内存的实现

常考点

虚拟存储概念相关	• 虚拟存储器的最大容量是由计算机的地址结构决定的，与主存容量和外存容量没有关系 • 虚拟存储技术基于程序的局部性原理，局部性越好，虚拟存储系统越能更好地发挥作用 • 虚拟存储技术只能基于非连续分配技术 • 使用覆盖，交换方法可以实现虚拟存储
页面置换算法相关	• 无论采用什么页面置换算法，每种页面第一次访问时不可能在内存中，必然发生缺页，所以缺页次数≥不同的页号数量
缺页中断相关	• 请求分页存储器中，页面尺寸增大，存放程序需要的页帧数减少，缺页中断次数也会减少 • 影响缺页中断的时间有：缺页率，磁盘读写时间，内存访问时间 • 缺页中断可能执行的操作：置换页面，分配内存（不会进行越界出错处理） • 缺页处理过程中可能执行的操作：修改页表，分配页框，磁盘I/O（内存没有页面，需要从外存读入）
其他	• 系统调用是由用户进程发起的，请求OS的服务 • 创建新进程可以通过系统调用完成 • 可以加快虚实地址转换的操作：增大快表TLB的容量，让页表常驻内存 • 当磁盘利用率很高，但是CPU利用率不高时，改进CPU利用率的操作：增大内存的容量，减少多道程序的度数

基本概念

传统存储管理方式的特征	一次性	• 作业必须一次性全部装入内存，才能开始运行
	驻留性	• 作业被装入内存后，就一直驻留在内存中，直到作业结束 • 运行中的进程会因等待I/O而被阻塞，可能处于长期等待状态
局部性原理	时间局部性	• 程序中的某条指令一旦执行，不久后该指令可能再次运行;出现的原因是程序中存在着大量的循环结构
	空间局部性	• 程序在一段时间内所访问的地址，可能集中在一定的范围内
虚拟存储器	定义	• 系统为用户提供的一个比实际内存容量大得多的存储器
	特征	• 多次性 = 即只需将当前运行的那部分程序和数据装入内存即可开始运行【最重要的特征】 • 对换性 = 即作业无需一直常驻内存，要用时换入，不要用时换出 • 虚拟性 = 从逻辑上扩充内存的容量【最重要的目标】
虚拟内存的实现	方式（离散分配）	1. 请求分页存储管理 2. 请求分段存储管理 3. 请求段页式存储管理
	需要的东西	• 一定的硬件支持，一定容量的内存和外存 • 页表/段表机制作为主要的数据结构 • 中断机制，当程序要访问的部分还未调入内存时，产生中断 • 地址变换机构

请求分页管理方式

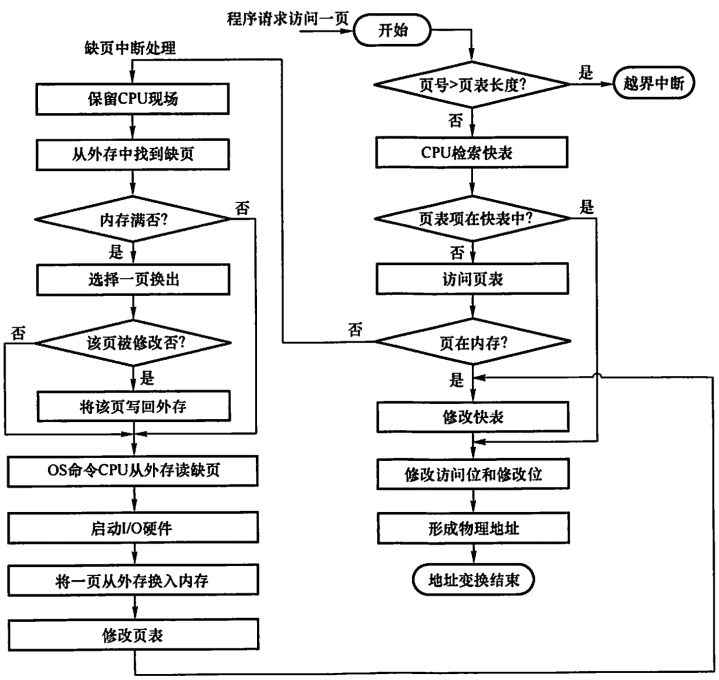
<table>
<tr><td>特点</td><td colspan="3">• 只要求将当前一部分页面装入内存，便可启动作业运行，不需要一次全部装入
• 在作业执行的过程中，当访问的页面不存在时，再通过调页功能将其调入</td></tr>
<tr><td>相比基本分页管理增加的功能</td><td colspan="3">• 请求调页功能：将要用的页面调入内存【调入】
• 页面置换功能：将不用的页面换出到外存【调出】</td></tr>
<tr><td>页表的构成</td><td colspan="3">页号+页框号+状态位P+访问字段A+修改位M+外存地址L</td></tr>
<tr><td rowspan="4">页面机制新增四个字段</td><td>状态位/合法位P</td><td>标记该页是否已被调入内存中</td><td>供程序访问时参考，用于判断是否触发缺页异常</td></tr>
<tr><td>访问字段A</td><td>记录本页在一段时间内被访问的次数</td><td>供置换算法换出页面时参考</td></tr>
<tr><td>修改位M</td><td>标识该页在调入内存后是否被修改过</td><td>当页面被淘汰时，若页面数据没有修改，则不用写回外存</td></tr>
<tr><td>外存地址L</td><td>用于指出该页在外存上的地址，通常是物理块号</td><td>供写回外存和从外存中调入此页时参考</td></tr>
<tr><td rowspan="3">缺页中断</td><td>定义</td><td colspan="2">• 缺页是在CPU执行某条指令过程中，进行取指令或读写数据时发生的一种故障，是内中断或者叫做异常</td></tr>
<tr><td>产生时间</td><td colspan="2">• 每当要访问的页面不在内存中时，便产生一个缺页中断，请求OS将所缺的页调入内存
• 缺页中断是访存指令引起的，说明所要访问的页面不在内存中
• 进行缺页中断处理并调入所要访问的页后，访存指令应该重新执行</td></tr>
<tr><td>特点</td><td colspan="2">• 在指令执行期间而非一条指令执行完后产生和处理中断信号
• 一条指令在执行期间，可能产生多次缺页中断</td></tr>
<tr><td>地址变换机构新增功能</td><td colspan="3">1. 产生和处理中断信号
2. 从内存中换出一页</td></tr>
<tr><td>地址变换过程</td><td colspan="3"> 

图 3.21　请求分页中的地址变换过程</td></tr>
</table>

页框分配（进程准备执行时，由OS决定给特定进程分配几个页框）

驻留集是什么	驻留集 = 给一个进程分配的物理页框（也叫做物理块）的集合	
驻留集的大小	1. 分配给进程的页框越少，驻留在内存的进程就越多，CPU的利用率就越高 2. 进程在主存中的页面过少，缺页率相对较高 3. 分配的页框过多，对进程的缺页率没有大的影响	
分配策略	**固定分配局部置换**	物理块固定，缺页时先换出一个线程再调入所缺页
	可变分配全局置换	物理块可变，缺页时增加物理块再调入所缺页
	可变分配局部置换	物理块可变，若不频繁缺页则用局部置换，频繁缺页再用全局置换
	• 对各进程进行固定分配时页面数不变，不可能出现全局置换	
物理块调入算法	1. 平均分配算法 2. 按比例分配算法 3. 优先权分配算法	
调入页面的时机	预调页策略 = 运行前的调入	• 主要用于进程的首次调入，由程序员指出应先调入哪些页
	请求调页策略 = 运行时的调入	• 调入的页一定会被访问，策略易于实现 • 每次仅调入一页，增加了磁盘I/O开销
请求分页系统外存组成	• 存放文件的文件区【采用离散分配方式】 • 存放对换页面的对换区【采用连续分配方式】 • 对换区的磁盘I/O速度更快	
从何处调入页面	1. 系统拥有足够的对换区空间 2. 系统缺少足够的对换区空间 3. UNIX方式	
如何调入页面	情况1：所访问的页面不在内存时→缺页中断→无空闲物理块→决定淘汰页→调出页面→调入所缺页面 情况2：所访问的页面不在内存时→缺页中断→有空闲物理块→调入所缺页面	

分页和分段的比较

	英文全名	被淘汰的页面	特点
最佳置换算法OPT	Optimal replacement	以后永不使用的	• 基于队列实现的 • 该算法无法实现 • 只能用于评价其他算法
先进先出置换算法FIFO	First In First Out	在内存中驻留时间最久的页面	• 会出现Belady异常（分配的物理块数增大但页故障数不减反增） • 性能差，但实现简单
最近最久未使用置换算法LRU	Least Recently Use	最近最长时间未访问过的页面	• 性能好，但实现复杂 • 需要寄存器和栈道硬件支持 • 堆栈类算法 • 耗费高因为要对所有页排序
时钟置换算法CLOCK	Clock	根据顺序找第一个访问位为0的页面，当指针指向访问位为1的页面时，先把访问位置0，再继续寻找	• FIFO和LRU的结合 • 改进型CLOCK算法需要使用位和修改位

其余零散知识点

抖动/颠簸	**定义**	• 在页面置换时，出现频繁的页面调度行为 • 所有页面置换策略都有可能造成抖动
	产生原因	• 系统中同时运行的进程太多—>分配给每个进程的物理块太少—>进程在运行时频繁出现缺页—>频繁的调动页面 • 主要原因是因为页面置换算法不合理
	解决方法	• 撤销部分进程 • 增加磁盘交换区大小和提高用户进程优先级都与抖动无关
工作集	**定义**	• 在某段时间间隔内，进程要访问的页面集合
	如何确定工作集	• 基于局部性原理，用最近访问过的页面来确认
	有什么作用	• 工作集反映了进程在接下来一段时间内很可能频繁访问的页面集合 • 为了防止抖动现象，要使分配给进程的物理块数>工作集大小
内存映射文件	**定义**	与虚拟内存有些相似，将磁盘文件的全部或部分内容与进程虚拟地址空间的某区域建立映射关系
	作用	可以直接访问被映射的文件，而不必执行文件I/O操作，也无需对文件内容进行缓存处理
	优点	适合用来管理大尺寸文件
虚拟存储器性能影响因素		1. 页面较大→缺页率较低→可以减少页表长度，但使得页内碎片增大 2. 页面较小→缺页率较高 →可以减少内存碎片，提高内存利用率 →使得页表过长，占用大量内存 3. 分配给进程的物理块数越多，缺页率就越低 4. 分配给进程的物理块数超过某个值时，对缺页率的改善并不明显 5. 好的页面置换算法可以使进程在运行过程中具有较低的缺页率 6. LRU，CLOCK将未来可能要用到的进程保存在内存中，可以提高页面的访问速度 7. 编写程序的局部化程度越高，执行时的缺页率越低 8. 存储和访问尽量使用系统的访问方式（如都按行存储就按行访问）