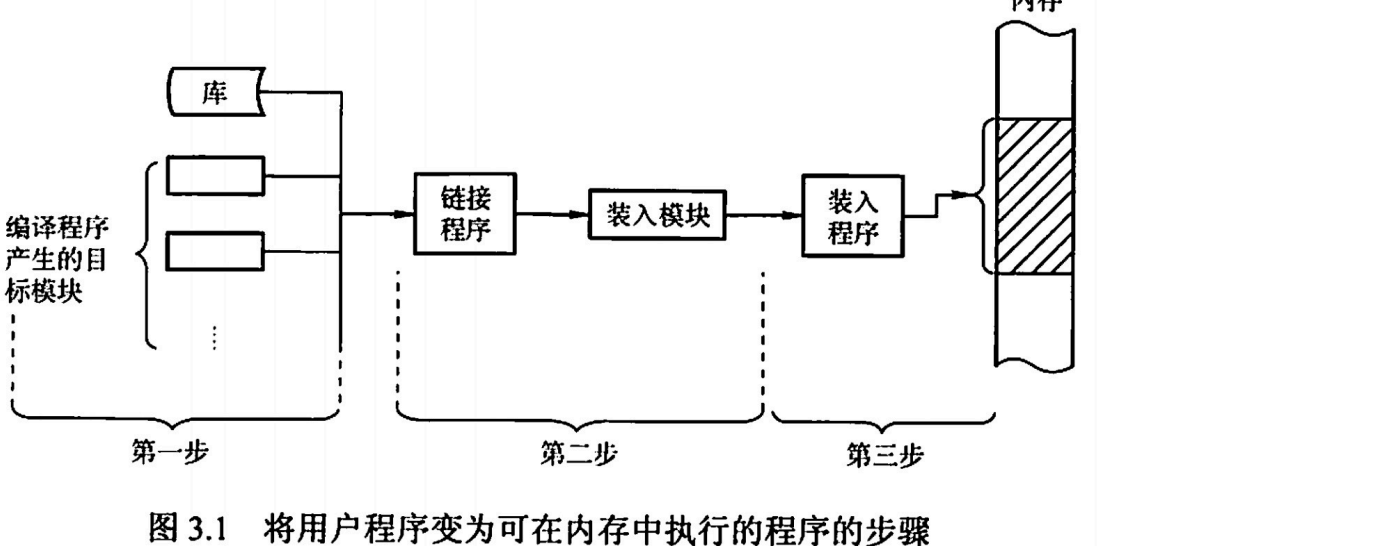


内存管理概念

基本概述	
内存管理是什么?	<ul style="list-style-type: none">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	<ul style="list-style-type: none">目的是为了更好支持多道程序并发执行方便用户提高内存利用率
内存管理的功能	<div>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</div> <div>2.地址转换 存储管理将逻辑地址转换为物理地址</div> <div>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</div> <div>4.内存共享 允许多个进程访问内存的同一部分</div> <div>4.存储保护 保证多道作业在各自的存储空间运行, 互不干扰</div>
内存管理的分配	<div>1.连续分配 单一连续分配——「资源复用和多道OS」---->固定分区分配——「为了固定大小不同的程序」---->动态分区分配</div> <div>2.不连续分配 分段存储管理---->分页存储管理---->段页存储管理</div>

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	 <p>图 3.1 将用户程序变为可在内存中执行的程序的步骤</p>
step1: 编译	编译是由编译程序将用户源代码编译成若干目标模块
step2: 链接	链接是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	<ul style="list-style-type: none">静态链接动态链接运行时动态链接
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	<ul style="list-style-type: none">静态装入: 在编程时把物理地址计算好可重定位装入: 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, 装入后有可能换出

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
特点	物理地址空间是指内存中物理单元的集合
	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

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	<div>1.代码段<ul style="list-style-type: none">代码段是只读的, 可以被多个进程共享</div> <div>2.数据段<ul style="list-style-type: none">程序运行时加工处理的对象, 包括全局变量和静态变量</div> <div>3.进程控制块PCB<ul style="list-style-type: none">存放在系统区, OS通过PCB控制和管理进程</div> <div>4.堆<ul style="list-style-type: none">用来存放动态分配的变量【动态的】</div> <div>5.栈<ul style="list-style-type: none">用来实现函数调用的【动态的】</div>

内存保护

目的	确保每个进程都有一个单独的内存空间
方法	<div>方法1: 在CPU中设置一对上下限寄存器, 判断CPU访问的地址是否越界</div> <div>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)</div> <ul style="list-style-type: none">重定位寄存器/基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

内存共享

概念	<ul style="list-style-type: none">只有只读区域的进程空间可用共享源代码/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	<div>1.段的共享</div> <div>2.基于共享内存的进程通信 (第二章的同步互斥)</div> <div>3.内存映射文件</div>

内存管理方法

方法一: 连续分配

定义	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
特点	<ul style="list-style-type: none">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
碎片	<ul style="list-style-type: none">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
分类	<div>1.单一连续分配</div> <div>2.固定分区分配</div>
定义	<div>在此方法下, 内存分为两个区<ul style="list-style-type: none">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</div> <ul style="list-style-type: none">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
优点	<ul style="list-style-type: none">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)
缺点	<ul style="list-style-type: none">只能用于单用户单任务的OS有内部碎片存储利用率极低
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
动态分配算法	特点
首次适应算法	最简单, 效果最好, 速度最快
邻近适应算法	比首次适应算法差
最佳适应算法	性能很差, 会产生最多的外部碎片
最坏适应算法	可能导致没有可用的大内存块, 性能差

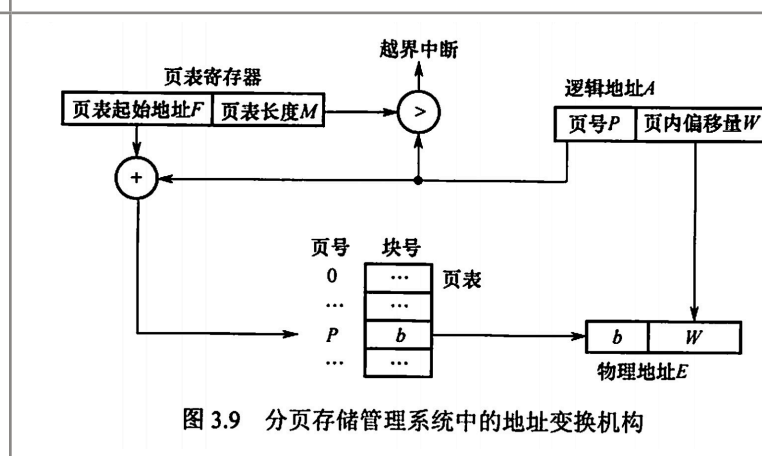
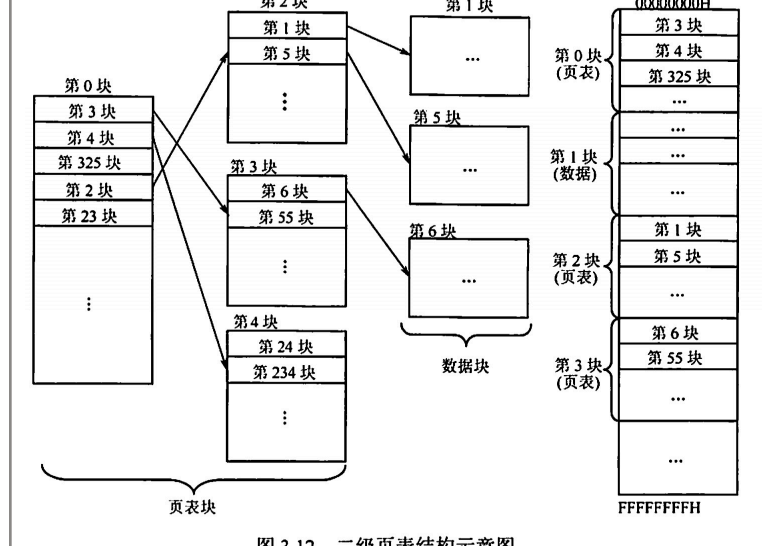
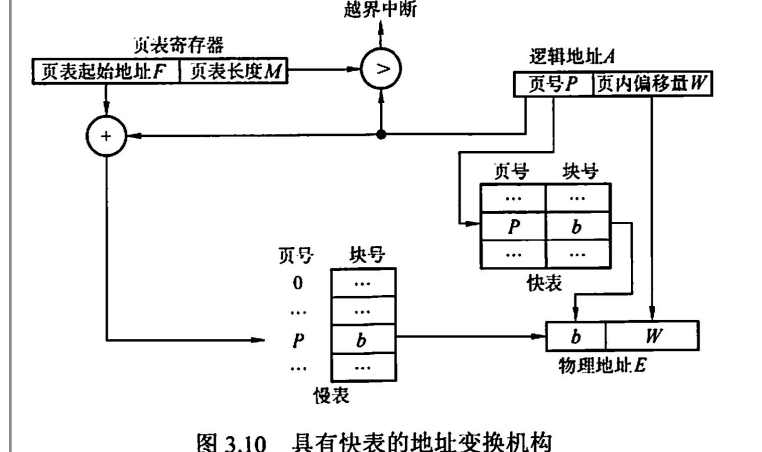
方法二: 分段

概念	<ul style="list-style-type: none">分段 = 进程由若干个逻辑分段组成, 不同的段有不同的属性, 用分段的方式把进程进行分离段表 = 一张逻辑空间与内存空间映射的表每段的长短不同---->无法整除得段号, 无法求得段内偏移---->段号, 段内偏移显式给出
分段管理地址空间是二维的	
段的共享与保护	<ul style="list-style-type: none">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<ul style="list-style-type: none">存储控制保护地址越界保护
分段的优点	<ul style="list-style-type: none">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
分段的缺点	<ul style="list-style-type: none">会产生外部碎片内存交换的效率低
分段的主要特点	<ul style="list-style-type: none">满足程序员/用户方便编程分段共享分段保护动态链接动态增长
地址变换	<ul style="list-style-type: none">物理地址 = 段基址 + 偏移量【可结合计组第三章内存管理学习】

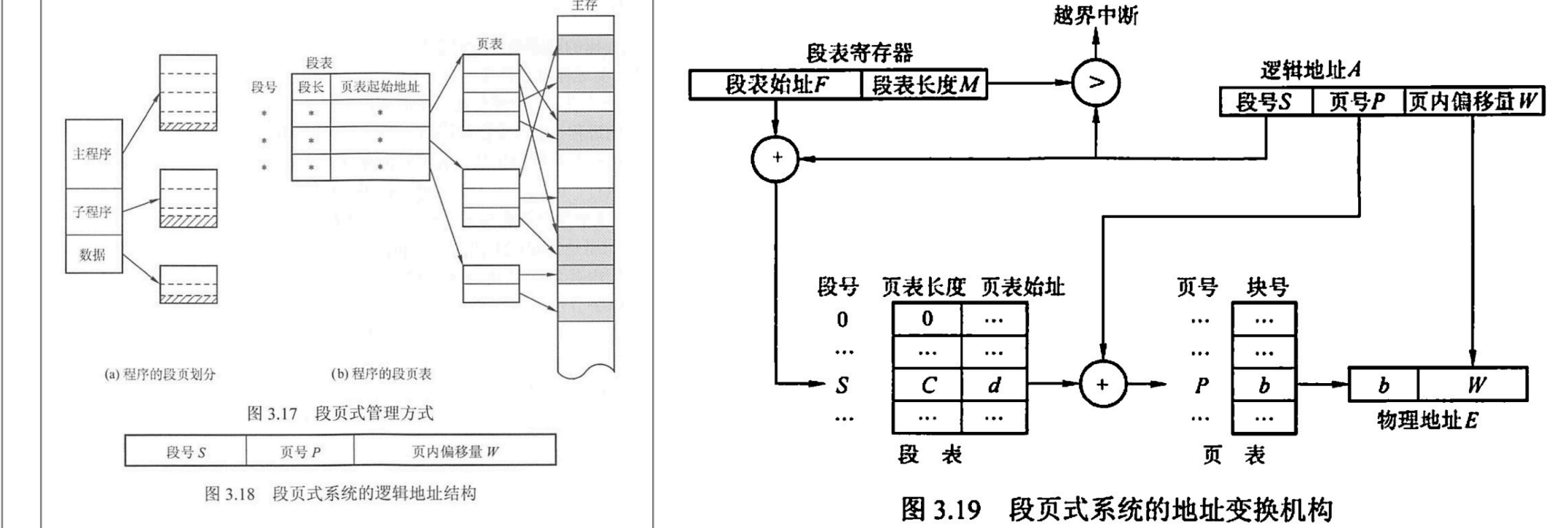
方法三: 分页

概念	<ul style="list-style-type: none">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 = 进程中的块<ul style="list-style-type: none">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 = 页号+页内偏移量W<ul style="list-style-type: none">地址结构决定了虚拟内存的寻址空间有多大页表 = 记录页面在内存中对应的物理块号<ul style="list-style-type: none">页表的起始地址放在页表基址寄存器PTBR中
分页的优点	<ul style="list-style-type: none">能有效地提高内存利用率
分页的缺点	<ul style="list-style-type: none">会产生内部碎片
分页的特点	<ul style="list-style-type: none">逻辑地址分配按页分配物理地址分配按内存块分配所有进程都有一张页表, 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, 页表存在内存中分页是面向计算机的

分页的地址变换

基本知识变换	<ul style="list-style-type: none">虚拟地址与物理地址通过页表来映射, 页表存放在内存中进程访问的虚拟地址在页表中查不到时, 系统产生一个缺页异常只有在程序运行时, 需要用到对应虚拟内存里面的指令和数据时, 再加载到物理内存里面去分页产生的页表过大, 使用多级页表, 解决空间上的问题	 <p>图 3.10 分页存储管理系统中的地址变换机构</p>
两级页表地址变换	<ul style="list-style-type: none">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, 也不用盲目地顺序查找页表项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	 <p>图 3.12 二级页表结构示意图</p>
具有快表的地址变换	<ul style="list-style-type: none">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), 用来完成地址转换和TLB的访问与交互	 <p>图 3.10 具有快表的地址变换机构</p>

方法四: 段页

实现方法	<div>step1: 将程序划分为多个有逻辑意义的段【分段】</div> <div>step2: 对分段划分出来的连续空间, 再划分固定大小的页【分页】</div>	
地址变换	<ul style="list-style-type: none">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	 <p>图 3.18 段页式存储管理地址变换机构</p>
得到物理地址的3次内存访问	<div>1.访问段表, 得到页表起始地址</div> <div>2.访问页表, 得到物理页号</div> <div>3.将物理页号与页内偏移移组合, 得到物理地址</div>	

虚拟内存的实现

基本概念

传统存储管理方式的特征	<ul style="list-style-type: none">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: 作业被转入内存后, 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	<ul style="list-style-type: none">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<ul style="list-style-type: none">出现的原因是程序中存在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	<div>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</div> <div>特征:<ul style="list-style-type: none">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】</div>
虚拟内存的实现	<div>方式(离散分配)</div> <div>1.请求分页存储管理</div> <div>2.请求分段存储管理</div> <div>3.请求段页式存储管理</div> <div>支持:<ul style="list-style-type: none">都需要一定的硬件支持: 一定容量的内存和外存页表/段表机制, 作为主要的数据库结构中断机制, 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</div>

请求分页管理方式

- 相比基本分页管理增加的功能
 - 请求调页功能: 将要用的页面调入内存
 - 页面置换功能: 将不用的页面换出到外存

硬件支持

页面机制新增四个字段	<ul style="list-style-type: none">驻留集 = 给一个进程分配的物理页框的集合状态位/合法位P访问字段A修改位M外存位置
地址变换机构新增的功能	<ul style="list-style-type: none">1.产生和处理中断信号2.从内存中换出一页

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	<ul style="list-style-type: none">驻留集 = 给一个进程分配的物理页框的集合
驻留集的大小	<div>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</div> <div>2.进程在主存中的页面越少, 缺页率相对较高</div> <div>3.分配的页框过多, 对进程的缺页率没有大的影响</div>
分配策略	<div>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</div> <div>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</div> <div>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</div>
物理块调入算法	<div>1.平均分配算法</div> <div>2.按比例分配算法</div> <div>3.优先权分配算法</div>
调入页面的时机	<div>预调页策略 = 运行前的调入</div> <div>请求调页策略 = 运行时的调入</div> <ul style="list-style-type: none">主要用于进程的首次调入, 由程序员指出应先调入哪些页调入的页一定会被访问, 策略易于实现但每次仅调入一页, 增加了磁盘I/O开销
请求分页系统外存组成	<ul style="list-style-type: none">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	<div>1.系统拥有足够的对换区空间</div> <div>2.系统缺少足够的对换区空间</div> <div>3.UNIX方式</div>

页面置换算法 (即选择调出页面的算法)【后续补充问题】

- 时钟置换算法CLOCK: FIFO和LRU的结合

	最佳置换算法OPT	先进先出置换算法FIFO	最久未使用置换算法LRU
被淘汰的页面	以后永不使用的/最长时间内不再被访问的	在内存中驻留时间最久的页面	最久最长时间未访问过的页面
特点	<ul style="list-style-type: none">基于队列实现的该算法无法实现, 只能用于评价其他算法	<ul style="list-style-type: none">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	<ul style="list-style-type: none">性能好, 但实现复杂需要寄存器和栈道硬件支持堆栈算法耗费高因为要对所有页排序
例题			