中国科学院大学
University of Chinese Academy of Sciences

虚存设计关键问题

中国科学院大学计算机与控制学院
中国科学院计算技术研究所

2021-11-10

虚存设计

- 设计目标
  - 保护
    - 隔离进程的错误
  - 虚拟化
    - 用磁盘来扩展物理内存的容量
    - 方便用户编程（进程地址空间从0-max）

虚存设计

- 虚存设计时要考虑的问题
  - 虚实映射
    - 映射机制：分段、分页
    - 映射加速：TLB、TLB缺失处理、 TLB表项多少
    - 映射开销：页大小和页表大小
  - 按需加载和页替换
    - 缺页处理
    - 页替换：替换算法、颠簸、 pin/unpin
    - 后备存储（swap）
  - 虚存使用
    - 清零
    - 写时复制
    - 共享内存

内容提要

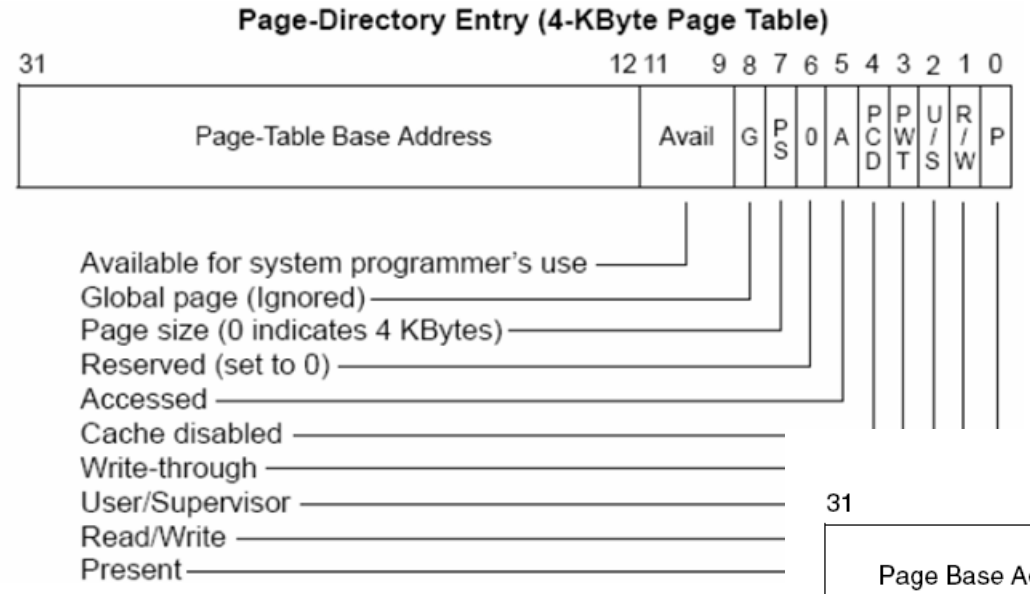
- 虚实映射设计问题
  - 页大小
- 页替换设计问题
  - 颠簸与工作集
  - 后备存储
  - pin/unpin
- 页使用设计问题
  - 清零
  - 写时复制 (Copy-on-write)
  - 进程间共享内存
- UNIX和Linux的虚存机制

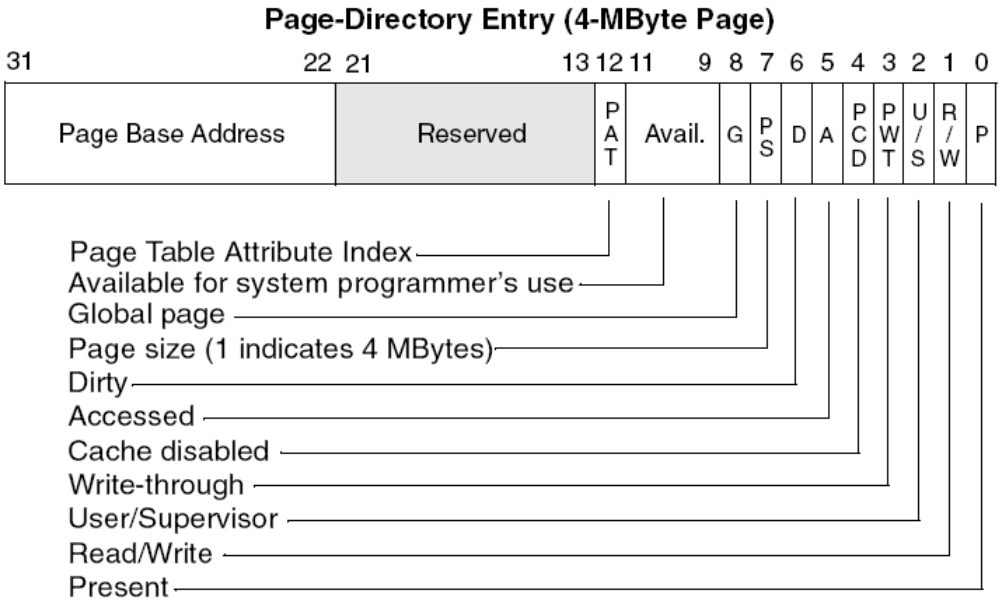
页大小

- 大页
  - 好处：页表小 & 磁盘I/O高效
  - 不足：内部碎片
    - 进程执行不需要的部分也放进了内存
    - 页可能不满
- 小页
  - 好处：减少内部碎片
  - 不足：页表大
    - 占用更多的内存
    - 加载时间更长
    - 查找虚页的时间更长
  - 不足：磁盘I/O不高效
    - 进程加载、页替换

例子：x86的页目录项格式

- 4KB和4MB页时的目录表项





内容提要

- 虚实映射设计问题
  - 页大小
- 页替换设计问题
  - 颠簸与工作集
  - 后备存储
  - pin/unpin
- 页使用设计问题
  - 清零
  - 写时复制 (Copy-on-write)
  - 进程间共享内存
- UNIX和Linux的虚存机制

颠簸

- 颠簸
  - 频繁发生缺页，运行速度很慢
  - 进程被阻塞，等待页从磁盘取进内存
  - 从磁盘读入一页~10ms，执行一条指令~几ns
- 虚存访问时间
  - 请求平均访问时间 = h × 内存访问时间 + (1-h) × 磁盘访问时间
  - h：应用访存请求中，直接在内存中访问的比例
  - 例：假设内存访问时间=100ns，磁盘访问时间=10ms, h=90%
    - 请求平均访问时间约是 **1 ms** !

颠簸

- 颠簸原因
  - 进程的工作集 > 可用的物理内存
  - 进程过多，即使单个进程都小于内存
  - 内存没有被很好地回收利用

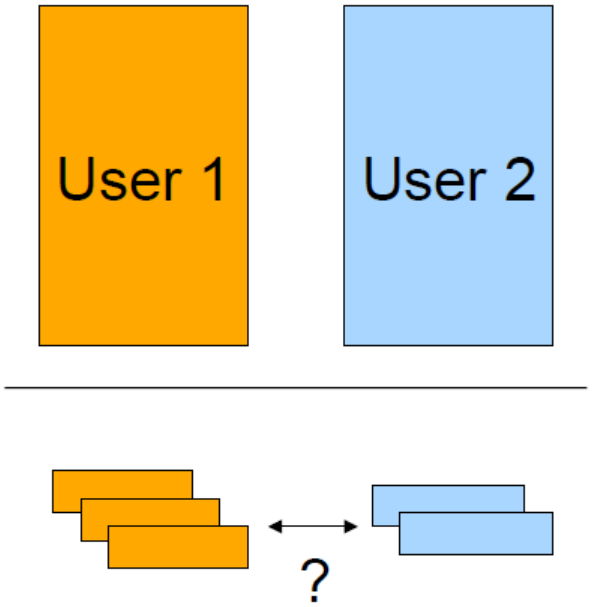
- 如何避免
  - 多进程时，哪些工作集放进内存
  - 多进程时，如何分配页

哪些工作集放进内存

- 进程分为两组
  - 活跃组：工作集加载进内存
  - 不活跃组：工作集不加载进内存
- 如何确定哪些进程是不活跃的
  - 等待事件（敲键盘、点击鼠标、…）
  - 等待资源
  - 典型方法是设定一个等待时间阈值，等待时间大于阈值的进程为不活跃进程
- 两种调度器
  - 长期调度器（准入调度器）决定
    - 哪些进程可以同时运行（CPU vs I/O，工作集之和，…）
    - 哪些是不活跃的进程，把它们换出到磁盘
    - 哪些是活跃进程，把它们换入到内存
  - 短期调度器（CPU调度器，执行CPU调度算法）决定
    - 把CPU分配给哪个活跃进程

在多大范围内分配页

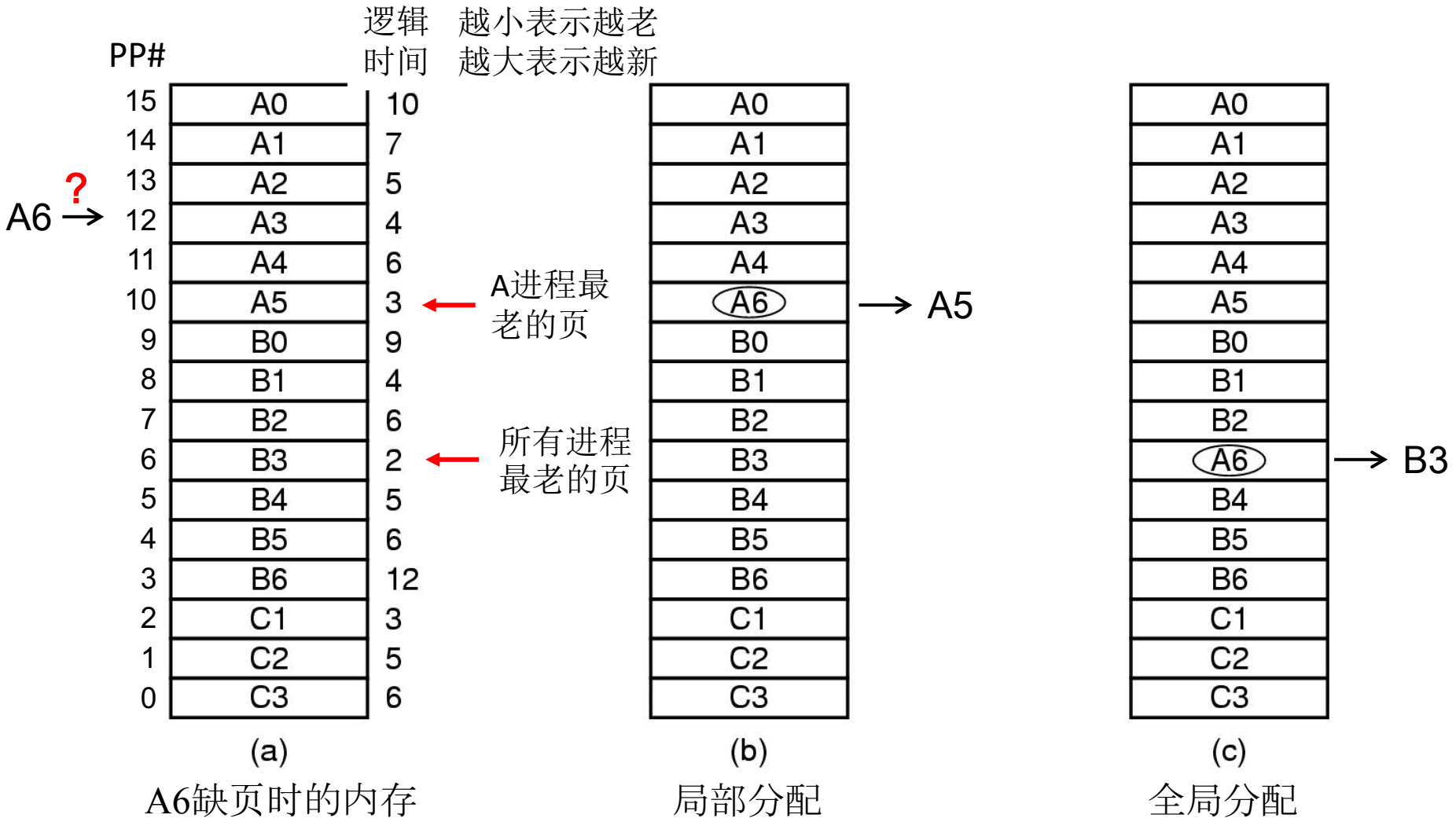
- 多个进程同时运行时，替换哪个进程的页框？



- 全局分配
  - 从所有进程的所有页框中选择
- 局部分配
  - 只从本进程（发生缺页的进程）的页框中选择

页分配

- 例子：局部分配 与全局分配



全局分配 vs. 局部分配

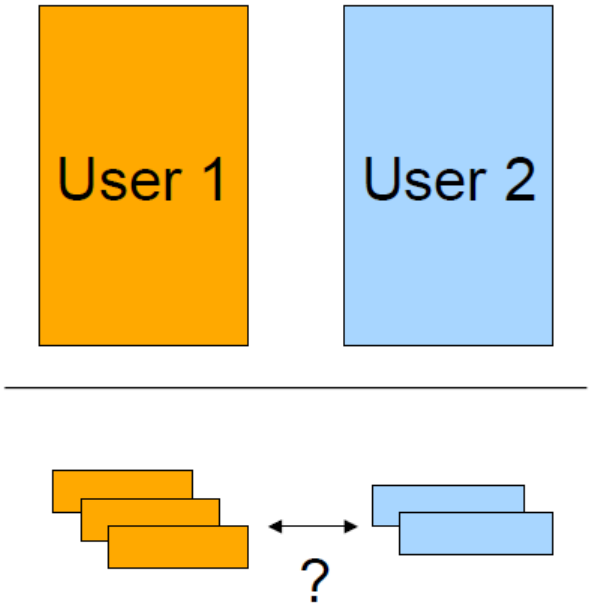
- 全局分配
  - 从所有进程的所有页框中选择
  - 可替换其它进程的页框
  - 每个进程运行期间，其内存大小是动态变化的
  - 好处：全局资源调配
  - 坏处：没有隔离，干扰其他进程/受其它进程的页替换干扰
    - 不能控制各个进程的内存使用量
- 局部分配
  - 只从本进程（发生缺页的进程）自己的页框中选择
  - 一个进程运行期间，固定其内存大小不变
    - 页框池：分配给进程的页框的集合，不同进程的页框池大小可不同
  - 好处：隔离，不影响其它进程
  - 坏处：不灵活
    - 进程使用内存频繁时会出现颠簸
    - 难以充分利用内存：每个进程对内存的需求不一样

平衡分配

- 局部分配 + 池大小动态调整
  - 每个进程有自己的页框池（pool）
  - 从自己的池中分配页，且从自己的工作集中替换页
  - 用一种机制来运行时动态调整每个池的大小
    - 初始大小 + 动态调整

局部

全局



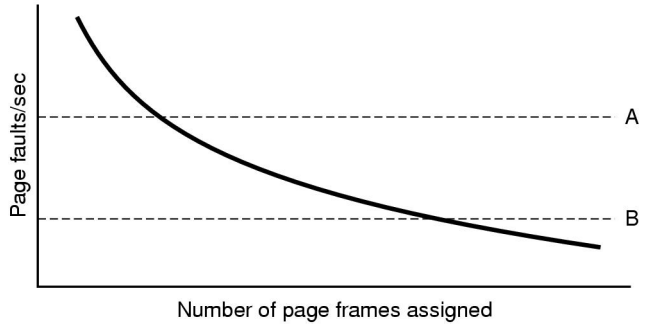
平衡分配

- 进程加载方式：进程换入时
  - 纯粹的按需加载页：加载慢
  - 预加载：先加载部分页 → 初始池大小
  - 初始池大小~工作集

- 初始池大小
  - 固定大小：所有进程都一样
  - 平均分配
  - 根据进程大小按比例分配

平衡分配

- 动态调整池大小：进程大小变化
  - PFF算法（page fault frequency）
    - 缺页率PFR：进程每秒产生多少次缺页
    - 对于大多数替换策略，PFR随分配给进程的内存增加而减少

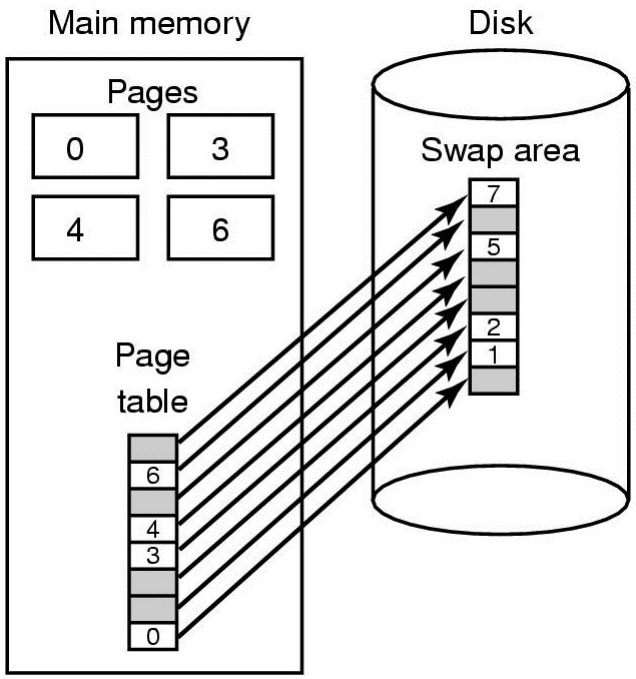


两个阈值：A和B
A为上限，B为下限

  - 根据进程的PFR来调整分配给它的内存量（pool size)
    - 当PFR高于A，就增加其内存（pool增大）
    - 当PFR低于B，就缩小其内存（pool减小）
  - PFR计算
    - 方法1：只看当前1秒钟内的缺页次数Ri
    - 方法2：求滑动均值
      - 例：$PFR_i = (R_i + PFR_{i-1})/2$

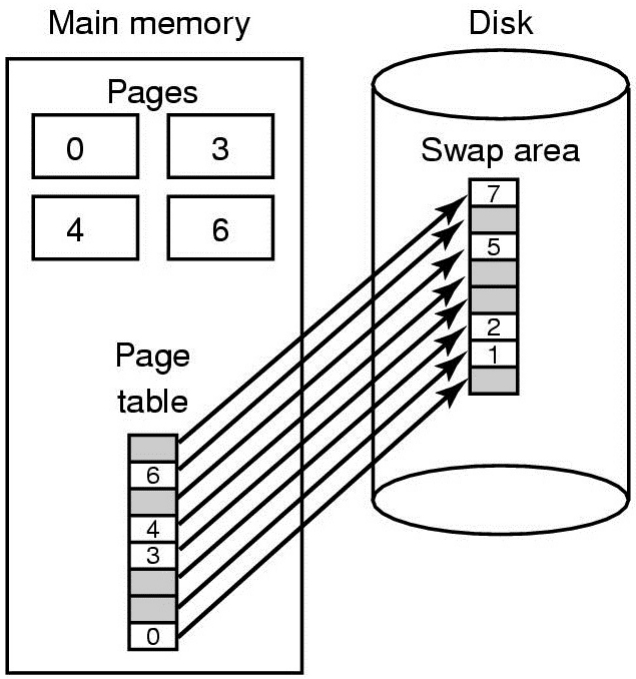
后备存储

- 交换区（swap area）
  - 在磁盘上
  - 专门用于存储进程换出页
  - 交换分区（swap partition）
    - 用专门的磁盘分区
  - 交换文件
    - 用一些文件
- 交换空间管理：两种方法
  - 静态分配
  - 动态分配



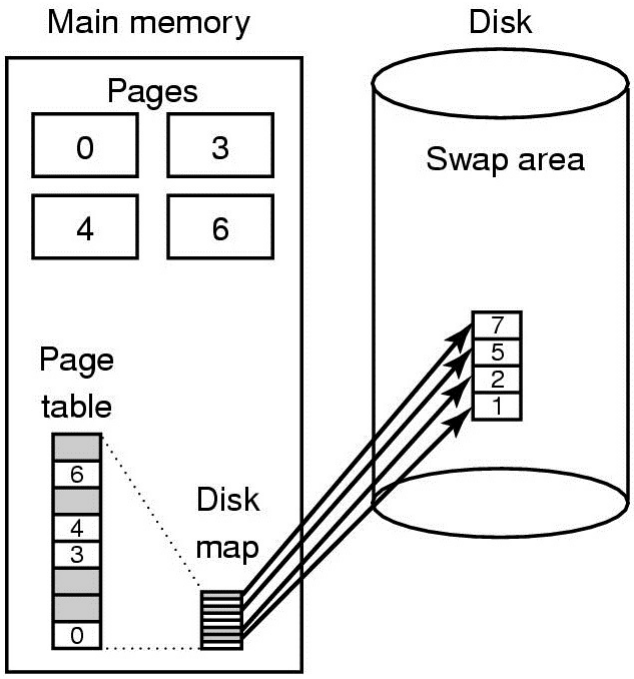
交换空间管理

- 静态分配
  - 创建进程时分配，进程结束时回收
  - 大小：进程映像（process image）
  - 进程控制块记录交换空间的磁盘地址
  - 绑定：一个虚存页←→一个磁盘页
    - 磁盘页称为shadow page
  - 初始化：两种方法
    1. 页换出：进程映像拷贝到交换区
    2. 页换入：进程映像加载进内存
  - 缺点
    - 难以增长：栈段、堆



交换空间管理

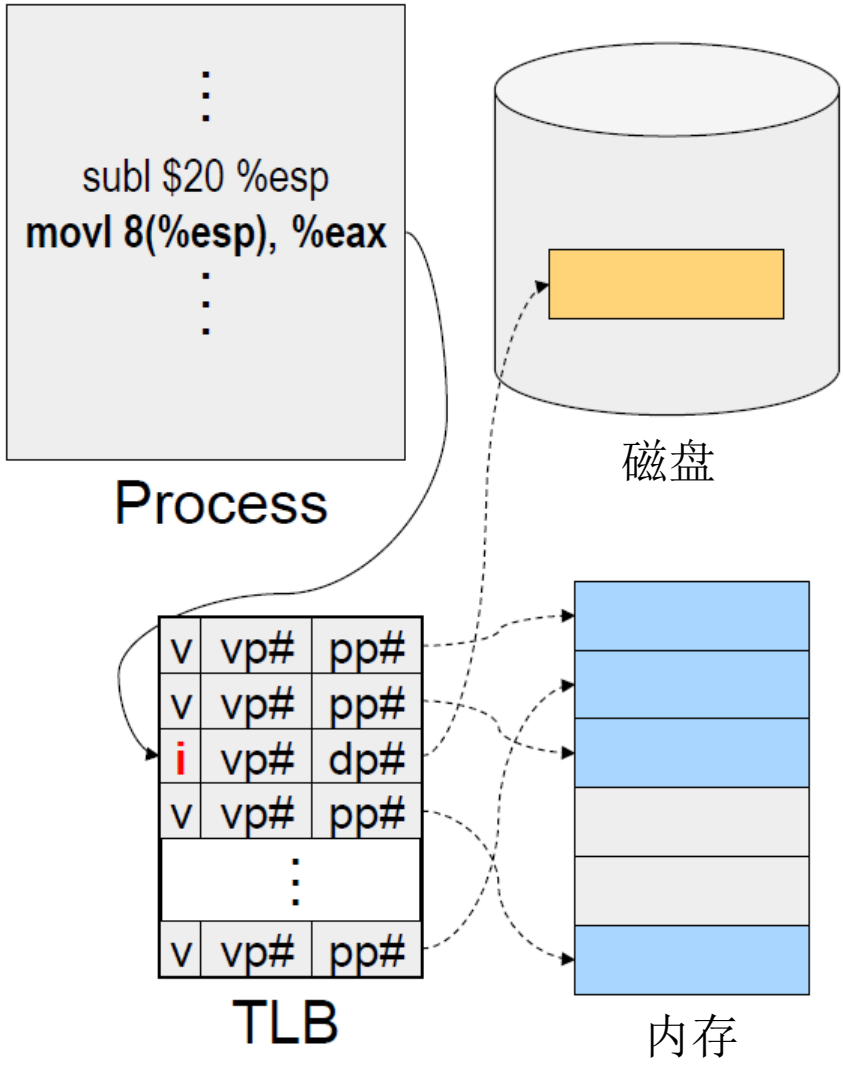
- 动态分配
  - 创建进程时不分配
  - 页换出时分配，页换入时回收
  - 虚页与磁盘页不绑定
    - 多次换出，分配不同的磁盘页
  - PTE中记录页的磁盘地址



- 一个优化：程序text段
  - 直接用磁盘上的可执行文件作为交换区，减少交换区大小
  - 换出时直接丢弃（只读的），减少不必要的写回

页地址转换

- PTE：虚页→页框和磁盘
  - 如果valid bit=1，物理页号pp#
  - 如果valid bit=0，磁盘页号dp#
- 换出
  - 将PTE和TLB置为无效
  - 将页拷贝到磁盘
  - 将磁盘页号填入PTE
- 换入
  - 找一个空闲页框（可能触发页替换）
  - 将页从磁盘拷贝到这个页框中
  - 将页框号填入PTE中，并将PTE置为有效



钉住页 (pin/unpin)

- 为什么需要？
  - 有些页需要频繁访问，换出后又会被换入，影响系统性能
  - 数据在内外存间进行传输时，需要传输的页框不能被换出，否则CPU会把新内容写入这些页框
- 系统调用接口
  - pin：把虚页钉在内存，使它们不会被换出
  - unpin：取消pin，使它们可以被换出
- 如何设计？
  - 用一个数据结构来记录所有钉住的页
  - 换页算法在替换页时检查该数据结构
    - 如果页被钉住，则不替换它，重新再选择一页
  - 如果整个内核全部在物理内存中，还需要提供pin和unpin调用吗？

模拟PTE的控制位

- 模拟R位：用有效标志位(V位)
  - 用一个预留位记录页的实际有效情况
  - 将页的V位置成无效，即使它们是有效的
  - 访问任何一页，产生page fault，进入缺页处理程序
    - 如果它在内存 (预留位=1)，将V位置为1 (即R位为1)，将页加入LRU链
    - 如果它不在内存 (预留位=0)，进行页替换
    - 重新执行该指令
- 模拟M位：用读写标志位 (R/W)
  - 用一个预留位来记录页的实际读写允许情况
  - 将页的R/W位设置为read-only (RO)
  - 在对页写时，产生page fault，进入缺页处理程序
  - Write fault：对RO的页写导致的fault
    - 如果它不为RO (预留位)，把页的R/W位置为1，表示可读写（即M位为1）
    - 重新执行该指令

内容提要

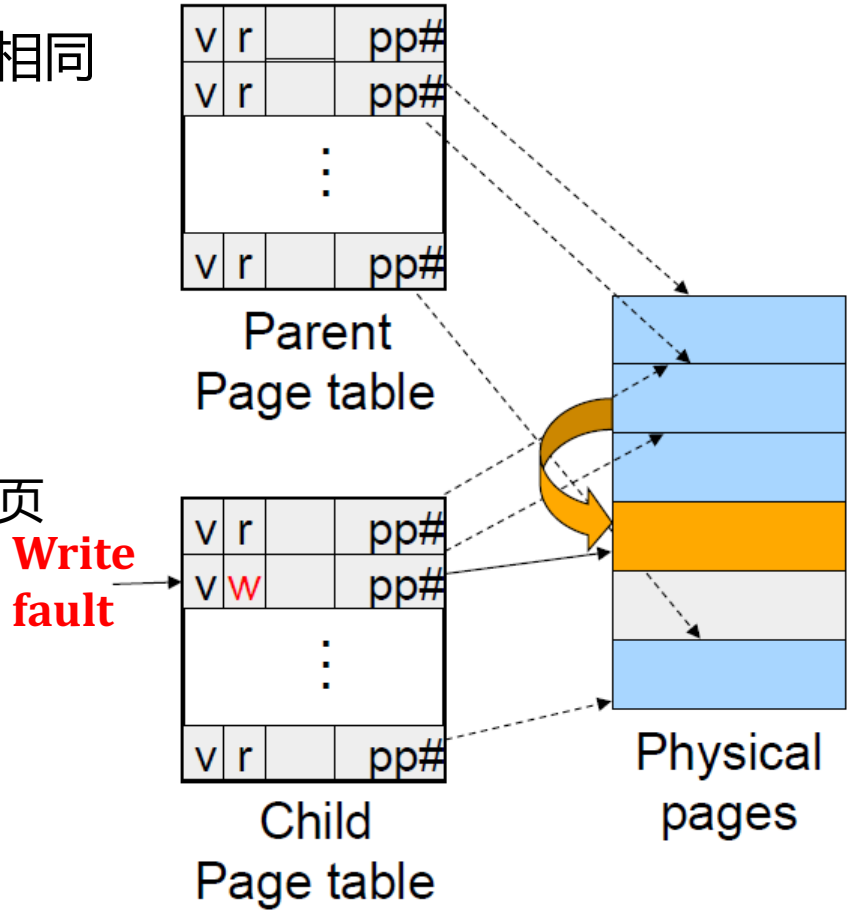
- 虚实映射设计问题
  - 页大小
- 页替换设计问题
  - 颠簸与工作集
  - 后备存储
  - pin/unpin
- 页使用设计问题
  - 清零
  - 写时复制 (Copy-on-write)
  - 进程间共享内存
- UNIX和Linux的虚存机制

清零页

- 将页清零
  - 把页置成全0
  - 堆和栈的数据都要初始化
- 如何实现
  - 对于堆或栈段中的页，当它们第一次发生page fault时，将它们清零
  - 有一个专门的线程来做清零
- 不做页清零可能引起安全问题

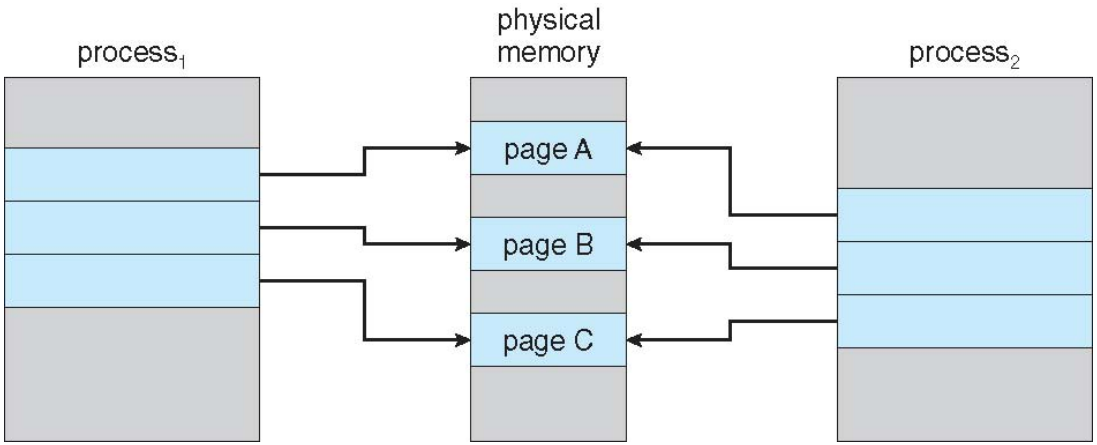
写时复制（Copy-On-Write）

- 该技术用于创建子进程（fork系统调用）
- 原理
  - 子进程的地址空间使用其父进程相同的映射
  - 将所有的页置成 read-only
  - 将子进程置成就绪
  - 对于读，没有问题
  - 对于写，产生page fault
    - 修改PTE，映射到一个新的物理页
    - 将页内容全部拷贝到新物理页
    - 重运行发生缺页的指令

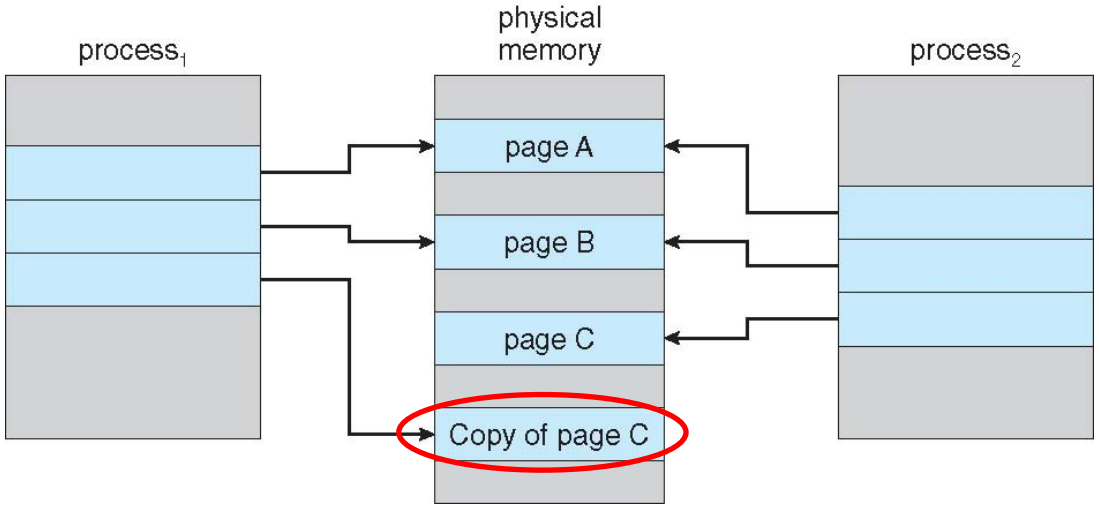


写时复制（Copy-On-Write）

- 示例



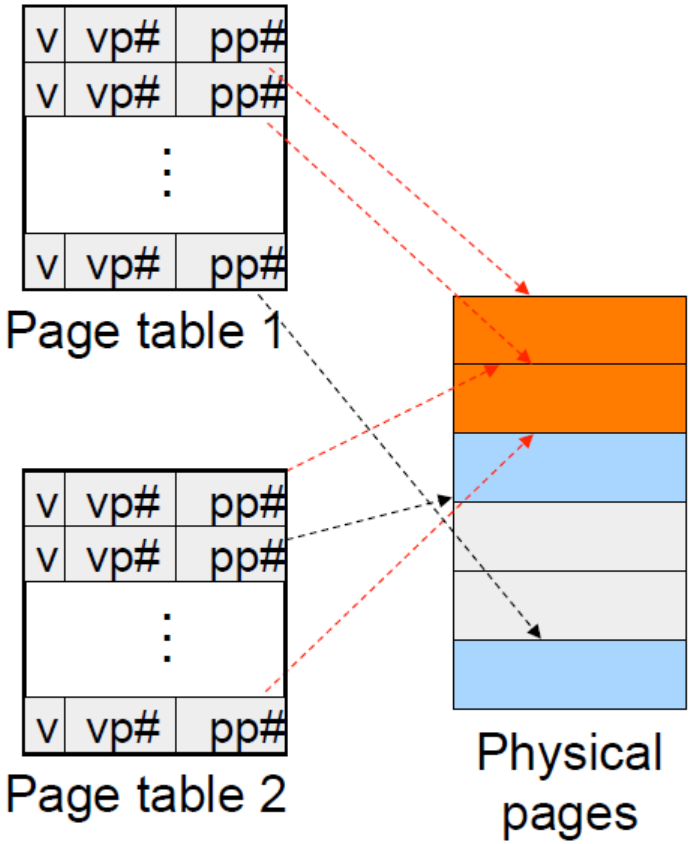
进程P1 fork一个子进程P2



进程P1修改page C

进程间共享内存

- 两个进程的页表共享一些物理页
  - 进程间通信
  - 进程间数据共享
- API
  - 共享内存的系统调用
    - shm_get, shm_ctl, …
- 涉及问题
  - 共享页的换入换出
    - 影响多个进程
    - pin和unpin共享页
  - 有共享页的进程的工作集
    - 共享页优先进入工作集
  - 有共享页的进程结束

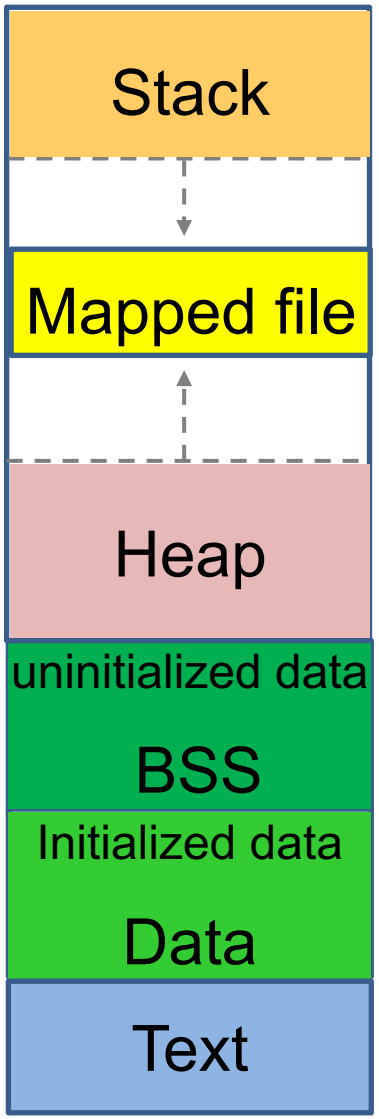


内容提要

- 页替换设计问题
  - 颠簸与工作集
  - 后备存储
  - pin/unpin
- 虚实映射设计问题
  - 页大小
- 页使用设计问题
  - 清零
  - 写时复制 (Copy-on-write)
  - 进程间共享内存
  - 分布式共享内存
- UNIX和Linux的虚存机制

UNIX的地址空间

- Text段：只读 & 大小不变
- 数据段
  - 初始化数据
  - 未初始化数据：BSS（Block Started by Symbol）
  - brk区用于增长或缩小（Heap）
- 栈段
- 内存映射文件 (memory mapped file)
  - 将一个文件映射进虚存
  - mmap(addr, len, prot, flags, fd, offset)
  - unmap(addr, len)
  - 像访问内存一样访问文件

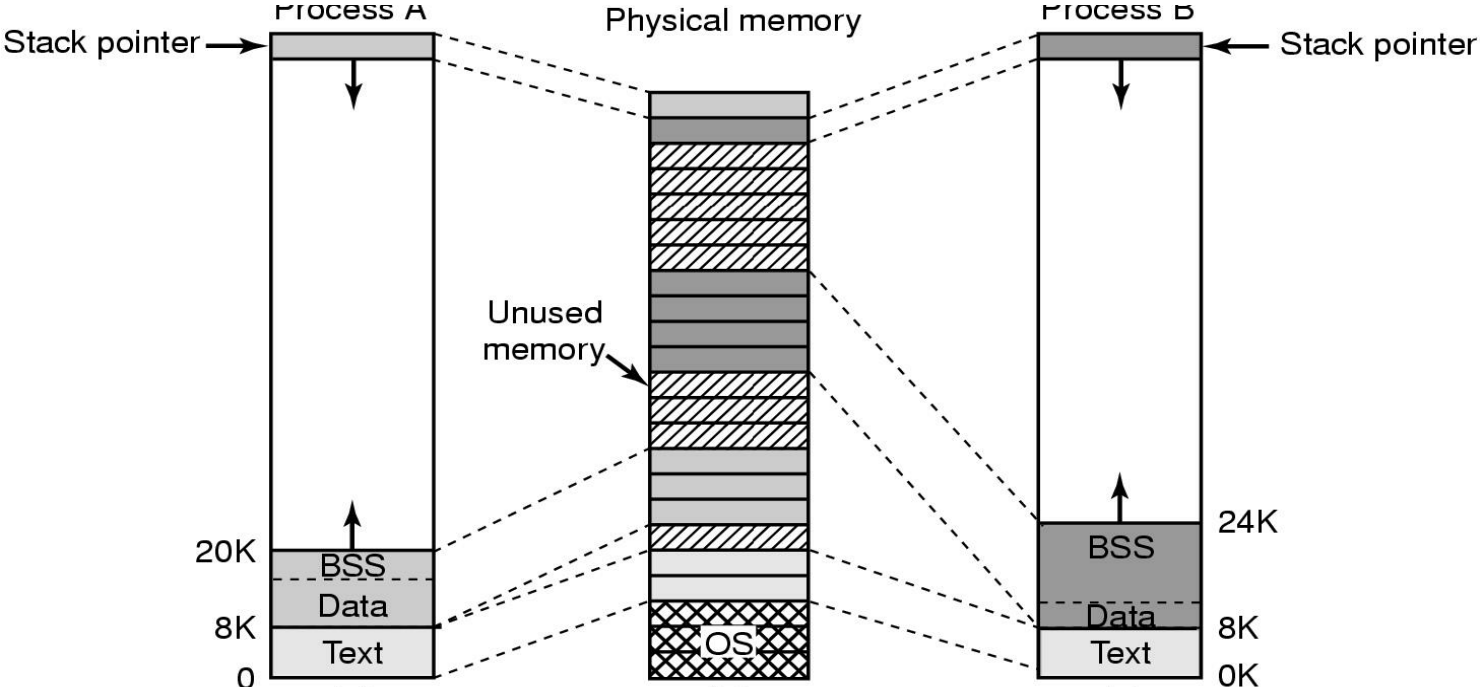


BSD4的虚存

- 物理内存划分
  - Core map（pin）：所有页框的描述信息
  - 内核（pin ）
  - 其它页框：给用户进程
- 页替换
  - 运行换页守护进程（page daemon)，直到有足够的空闲页
  - 早期BSD使用原始的Clock替换算法（有第二次机会的FIFO）
  - 后来的BSD使用双表针的Clock算法
  - 如果page daemon不能得到足够的空闲页，则运行swapper
    - 寻找idle时间超过20秒及以上的进程
    - 最大的4个进程

Linux的地址空间

- 32-位机的地址空间
  - 3GB用户空间
  - 1GB内核空间（用户态不可见）
- 栈段从3GB位置向下增长
  - 初始时保存进程的环境变量和命令行参数
- 数据段：
  - 大小可变：brk vs. malloc
  - BSS：未初始化的全局变量
    - 页加载时初始化为0
    - zero page: 避免分配大量全0的物理页

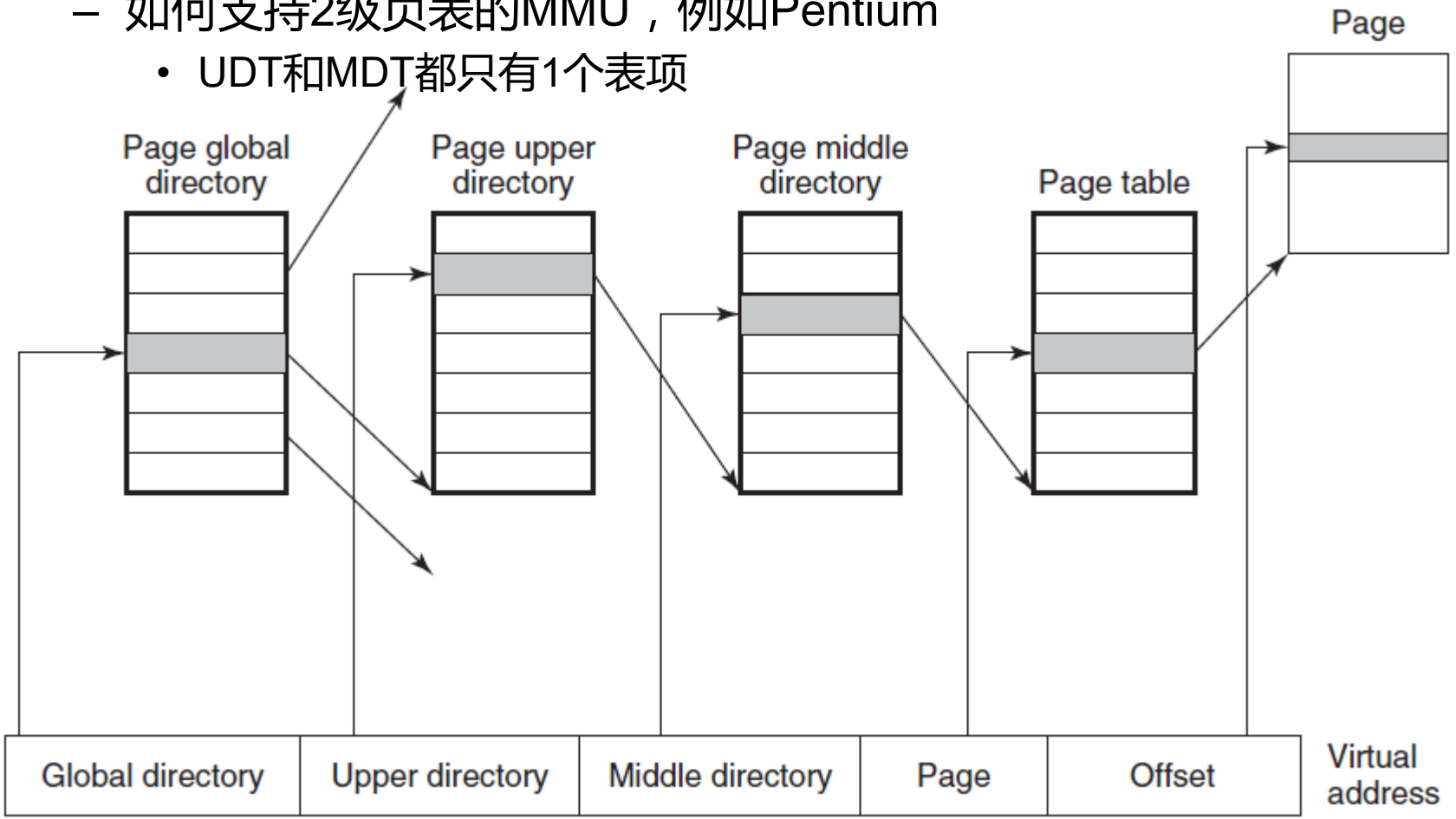


系统调用

System call	Description
s = brk(addr)	Change data segment size
a = mmap(addr, len, prot, flags, fd, offset)	Map a file in
s = unmap(addr, len)	Unmap a file

Linux的地址转换

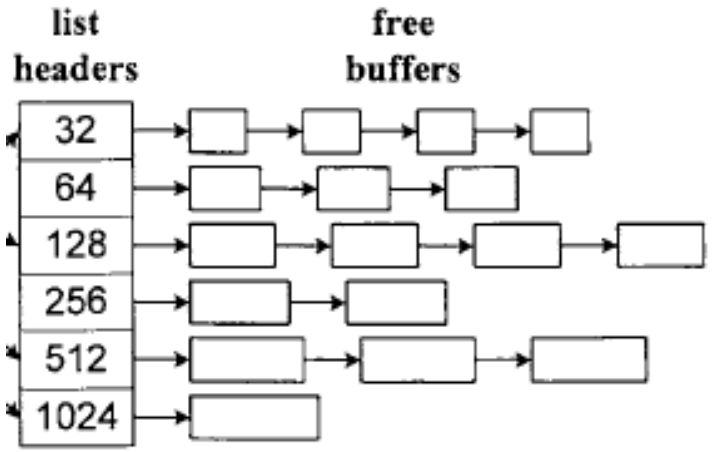
- 2.6.11及以后的Linux使用4级页表
  - 虚地址划分为5段：GDT、UDT、MDT、PT和页内偏移
  - 如何支持2级页表的MMU，例如Pentium
    - UDT和MDT都只有1个表项



Linux的物理页分配

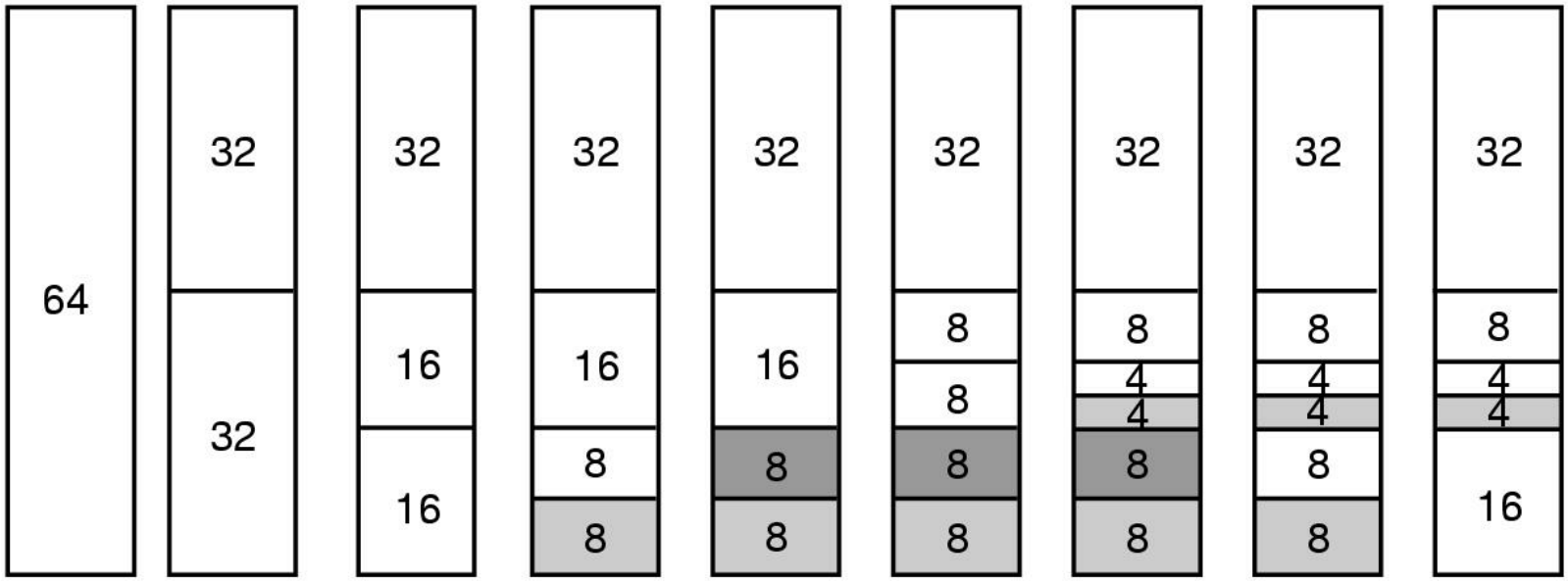
- 伙伴算法
  - 初始时：只有1个段，含全部可用空间
  - **分配空间**，m页：找size>=m且size最小的段，假设其size为n
  - 如果n>=2m，则将该段对分为两个size=n/2的段，
  - 重复对分，直到n/2 < m =< n
  - **释放空间**，m页：若释放段的”伙伴”也是空闲的，则合并它们，重复合并

- 数据结构
  - 空闲链表数组：每条链上的页大小为2的幂



Linux的物理页分配

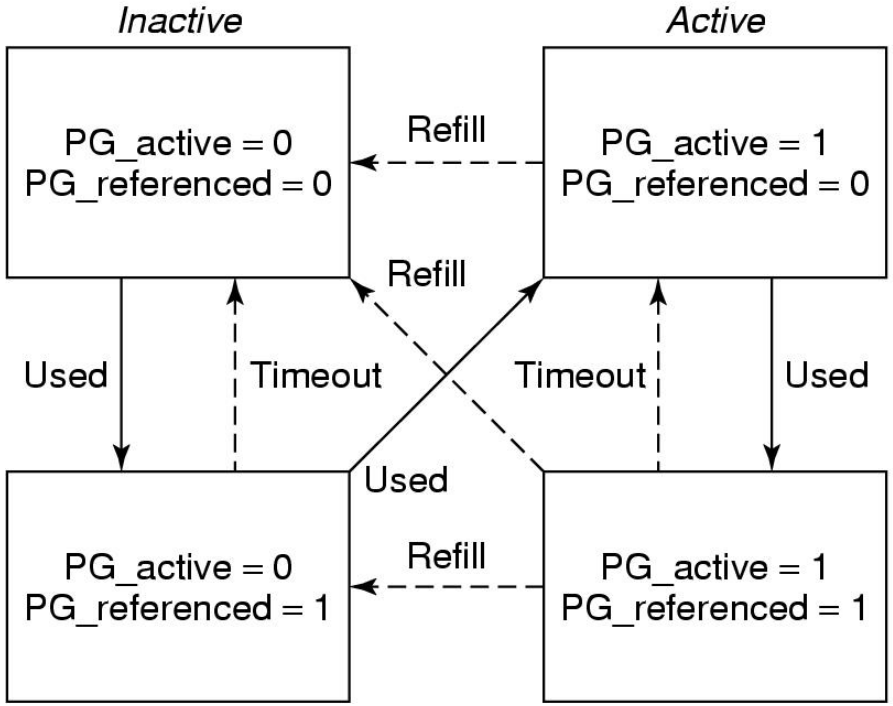
- 大块空间分配采用伙伴算法
  - 例子：请求1，分配8页，(d)完成，3次对分
  - 请求2，分配8页，(e)完成
  - 请求3，分配4页，(g)完成，1次对分
  - 请求4，释放8页，(h)完成
  - 请求5，释放8页，(i)完成，1次合并



Linux的页替换

- 方法
  - 保持一定数量的空闲页
  - 文件缓存 (page cache) 使用Clock算法
  - 未使用的共享页使用Clock算法
  - 用户进程的内存使用改进的Clock算法

- 改进的Clock算法
  - 两条链
  - Active list：所有进程的工作集
  - Inactive list：回收的候选页
  - Refill是将页从active list移动到inactive list



总结：内存管理

- 虚存：进程地址空间是独立的、连续的
  - 隔离 & 保护 & 方便程序员
- 地址转换：虚地址→物理地址，保护&错误隔离
  - 每个内存访问都要进行，由硬件MMU完成
  - TLB：专门硬件加速地址转换 (查页表)
- 地址映射
  - 分段、分页
  - 页表结构：用于地址转换
  - 页表项（PTE）：PP# & 控制位 V、R、M、R/W、C
  - 大地址空间：分段+分页、多级页表、反向页表
- 访存操作在正常情况下全部是硬件完成
  - 只有在缺页、无权限等异常情况下，才需要OS参与

总结：内存管理

- 按需加载页
  - 一个进程并不是每时每刻都需要所有的页
  - 频繁使用的页放内存，其它的页放磁盘
  - 要访问的页不在内存，透明地将它加载进内存 → OS缺页处理
  - 加载页时内存可能 没有空闲页框 → 进行页替换
- 缺页处理
  - 分配空闲页框 或 替换一个页框
  - 写回页 & 加载页（磁盘I/O）
  - 刷新TLB、修改页表项、加载TLB
- 页替换算法
  - FIFO & Second chance
  - Clock
  - LRU
  - 工作集