第四章　存储管理

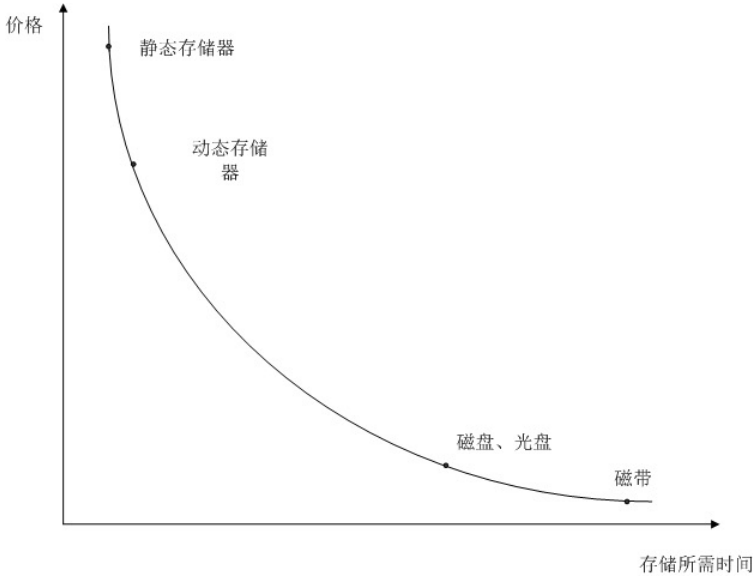
翁楚良

https://chuliangweng.github.io

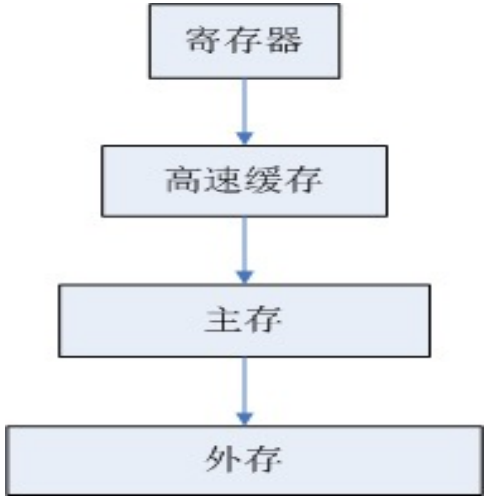
2023 春　ECNU

层次化存储体系结构

- 计算机的存储体系
  - 少量的非常快速、昂贵、易变的的高速缓存（cache）
  - 若干兆字节的中等速度、中等价格、易变的主存储器（RAM）
  - 数百兆或数千兆字节的低速、廉价、不易变的磁盘(外存)组成



- 操作系统的工作就是协调这些存储器的使用，其中管理存储器的部分程序称为存储管理器
  - 记录存储使用状况
  - 分配、回收存储资源
  - 数据的装入与写回



第四章 提纲

- 4.1 基本的内存管理
- 4.2 交换技术
- 4.3 虚拟存储管理
- 4.4 页面替换算法
- 4.5 页式存储管理的设计问题
- 4.6 段式存储管理
- 4.7 MINIX3进程管理器概述
- 4.8 MINIX3进程管理器实现

存储管理系统分类

- 在运行期间，进程需要在内存和磁盘之间换进换出的系统（交换和分页）和不需要换进换出的系统。

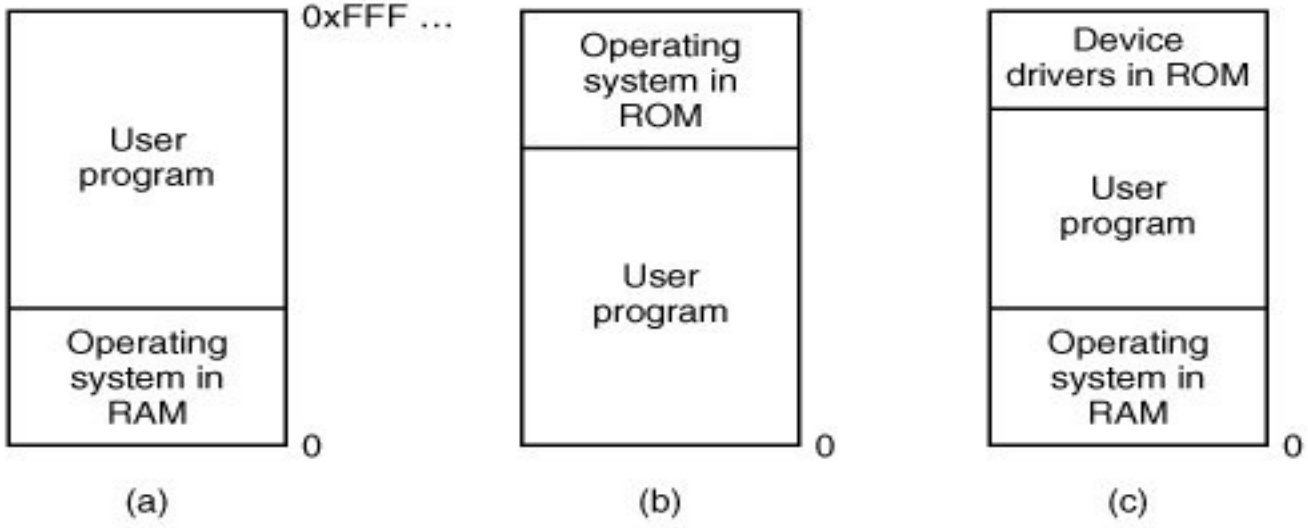
- 不需要换进换出的系统
  - 特点：进程被调入运行后，它将始终位于内存中，直至运行结束
    - 没有交换和分页的单道程序
    - 固定分区的多道程序

基本的存储管理

- 单道程序存储管理
- 多道程序存储管理
- 重定位和存储保护

没有交换或分页的单道程序

- 同一时刻只运行一道程序，应用程序和操作系统共享存储器
- 相应地，同一时刻只能有一个进程在存储器中运行。
  - 一旦用户输入了一个命令，操作系统就把需要的程序从磁盘拷贝到存储器中并执行它；在进程运行结束后，操作系统显示出一个提示符并等待新的命令。当收到新的命令时它把新的程序装入存储器，覆盖掉原来的程序。



早期的大型机小型机　　嵌入式系统　　早期PC机

基本的存储管理

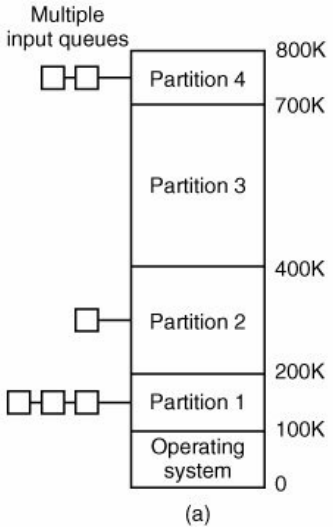
- 单道程序存储管理
- 多道程序存储管理
- 重定位和存储保护

多道程序(固定分区的系统)

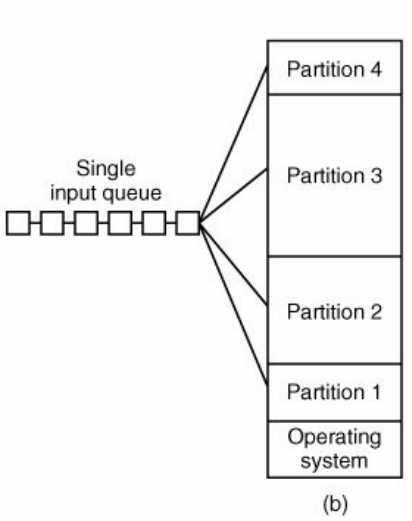
- 将内存划分为n个分区（可能不相等），分区的划分可以在系统启动时手工完成
  - 每个分区分别有一个运行队列
    - 当一个作业到达时，可以把它放到能够容纳它的最小的分区的输入队列中
  - 各分区共享同一个输入队列

IBM大型机的OS/360

由操作员在早晨设置好随后就不能再被改变的固定分区的系统



各分区有自己独立的输入队列



仅有单个输入队列的固定内存分区

基本的存储管理

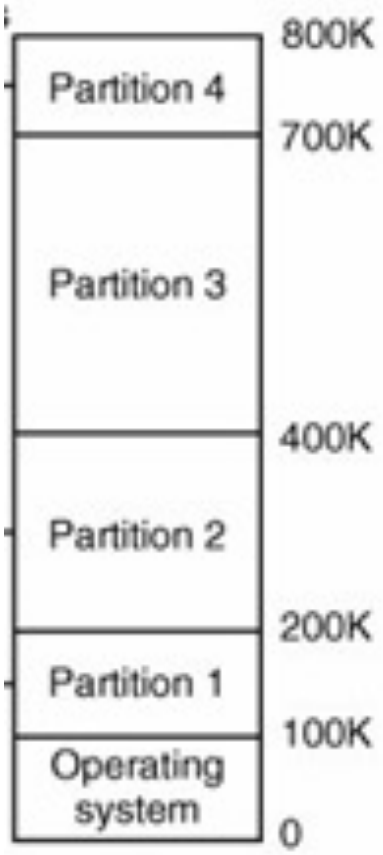
- 单道程序存储管理
- 多道程序存储管理
- 重定位和存储保护

重定位和保护

多道程序技术引发的问题???

- 重定位
  - 假设程序的第一条指令是调用在链接器产生的二进制文件中起始地址为100的一个过程。
  - 如果程序被装入分区1，这条指令跳转的目的地址将是绝对地址100，这会造成混乱，因为该地址在操作系统的内部。其实真正应该被调用的地址是100K+100。
  - 如果程序被装入分区2，它就应该去调用200K+100
- 保护
  - 在装入时重定位并没有解决保护问题，一个恶意的程序总可以生成一条新指令去访问任何它想访问的地址
  - 每个内存块分配4位的保护码，PSW中包含一个4位的密钥，若运行进程试图对保护码不同于PSW中密钥的主存进行访问，则由硬件引起一个陷入

一个既针对重定位又针对保护问题的解决方法是在机器中设置两个专门的寄存器，称为基址和界限寄存器



第四章 提纲

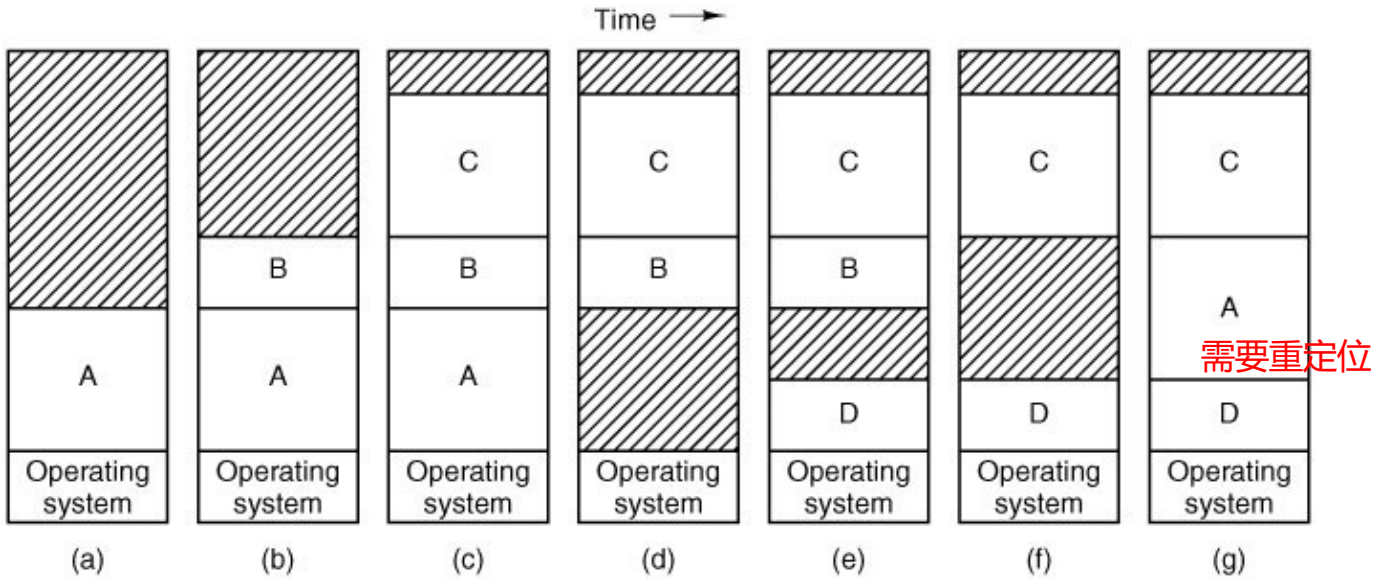
- 4.1 基本的内存管理
- 4.2 交换技术
- 4.3 虚拟存储管理
- 4.4 页面替换算法
- 4.5 页式存储管理的设计问题
- 4.6 段式存储管理
- 4.7 MINIX3进程管理器概述
- 4.8 MINIX3进程管理器实现

交换技术

虚拟存储器（virtual memory），使进程在只有一部分在主存的情况下也能运行。

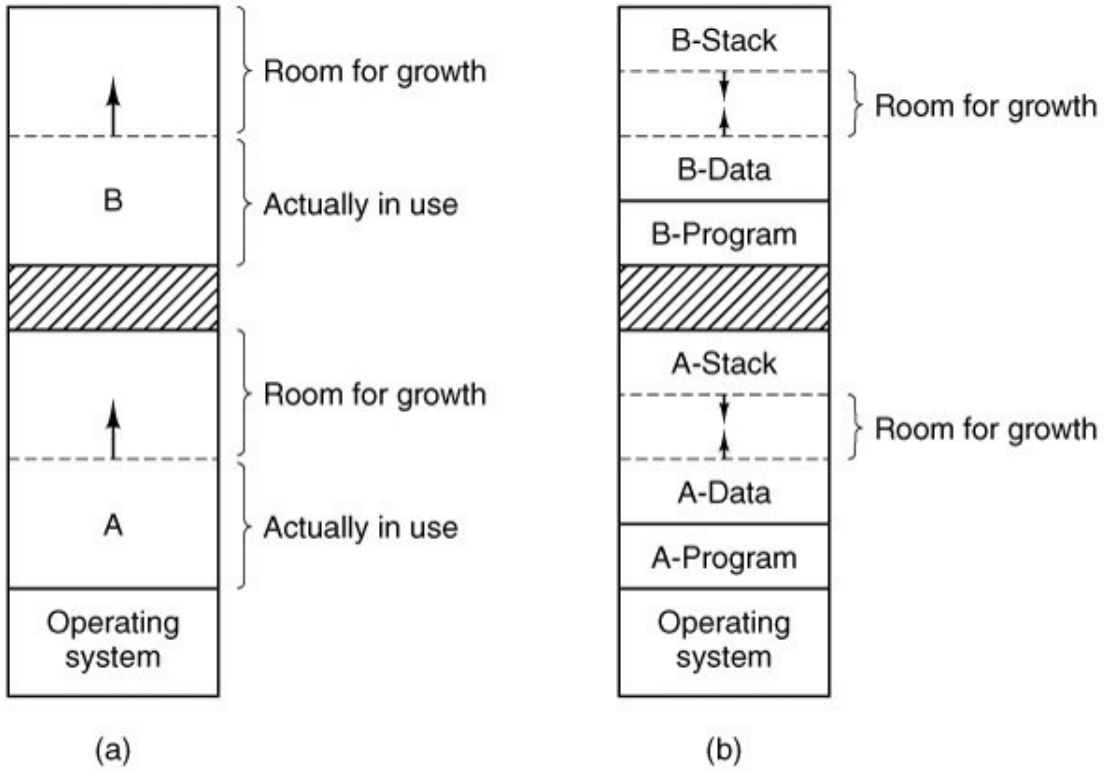
- 交换技术（swapping）
  - 把各个进程完整地调入主存，运行一段时间，再放回到磁盘上，过段时间再调入运行。

内存整理（memory compaction）
当交换在主存中生成了多个空洞时，可以把所有的进程向下移动至相互靠紧，从而把这些空洞结合成一大块



支持可变内存策略

- 进程分配内存固定，需要扩大内存时需将其移动到内存中一个足够大的空洞中
- 在进程被换进或移动时为其分配一点额外的内存

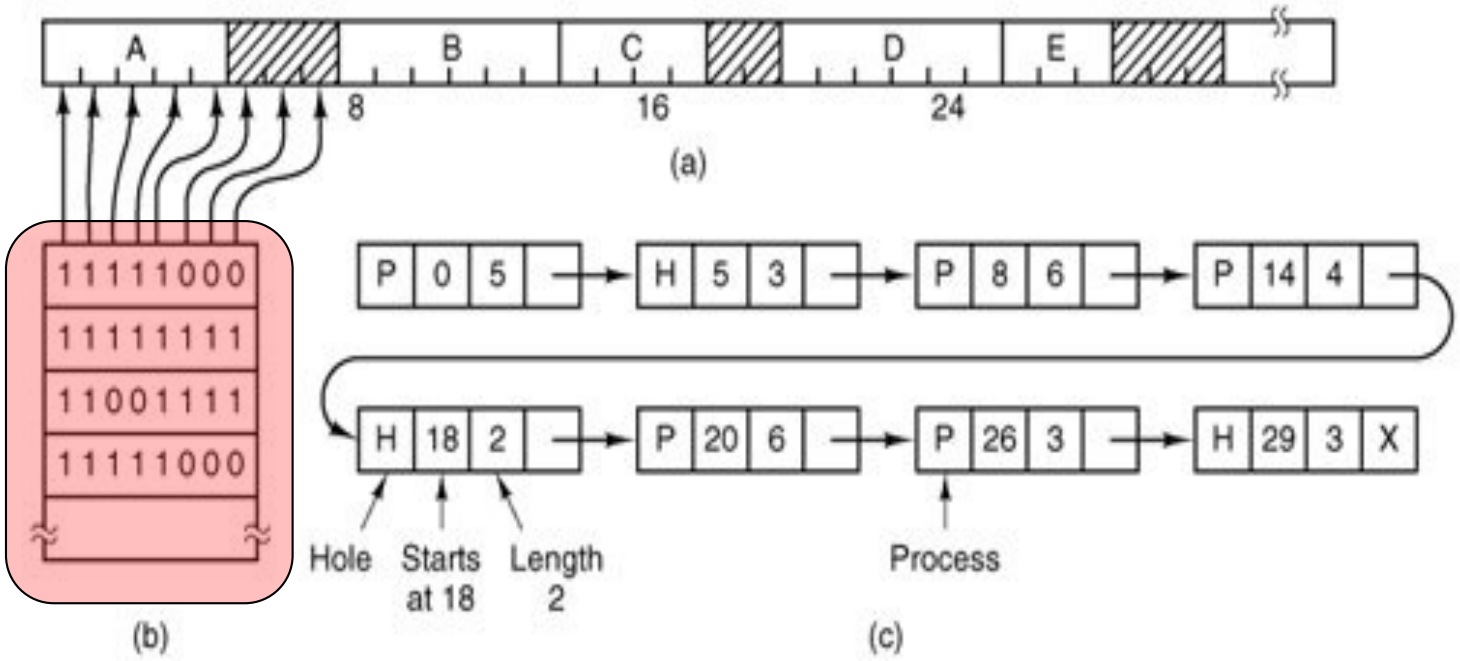


交换技术

- 进程被换入换出，内存动态分配，操作系统需要记录内存的使用情况
  - 基于位图的存储管理
  - 基于链表的存储管理

使用位图的内存管理

- 内存被划分为可能小到几个字或大到几千字节的分配单位，每个分配单位对应于位图中的一位，0表示空闲，1表示占用（或者反过来）。



位图的大小仅仅取决于内存和分配单位的大小

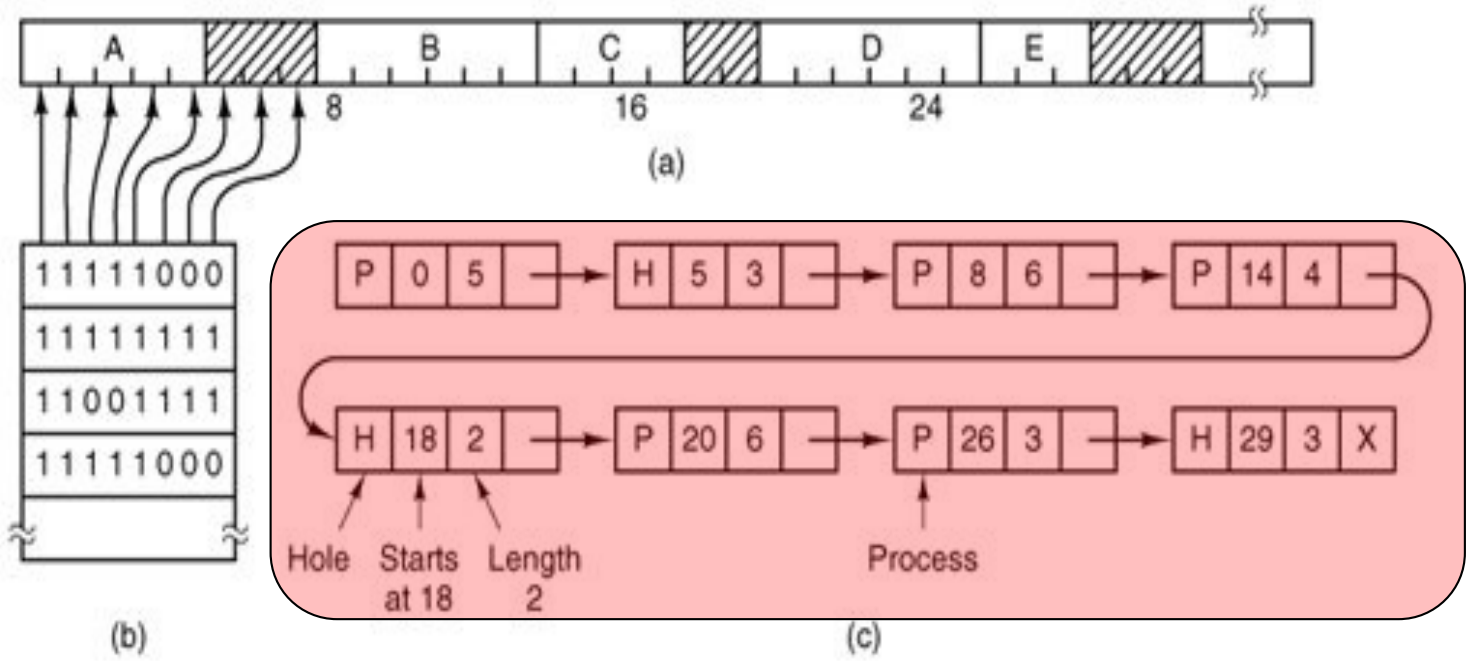
缺点：在位图中查找指定长度的连续0串是一个缓慢的操作

交换技术

- 进程被换入换出，内存动态分配，操作系统需要记录内存的使用情况
  - 基于位图的存储管理
  - 基于链表的存储管理

使用链表的内存管理

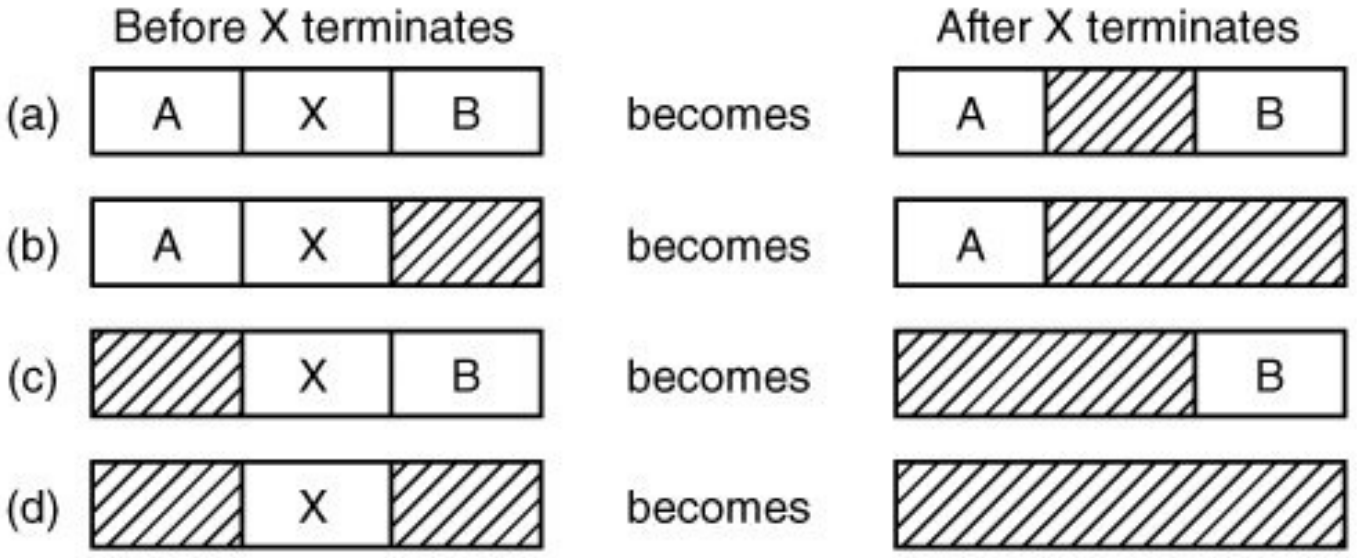
- 跟踪内存使用的另一个方法是维持一个已分配和空闲的内存段的链表



链表中的每一个表项都包含下列内容：指明是空洞(H)还是进程(P)的标志、开始地址、长度、和指向下一个表项的指针。

内存释放

- 一个要结束的进程一般有两个邻居（除非它是在内存的最低端或最高端），他们可能是进程也可能是空洞，这导致了图4-6所示的四种组合



内存分配算法

- 首次适配算法
  - 存储管理器沿着内存段链表搜索直到找到一个足够大的空洞，除非空洞大小和要分配的空间大小刚好一样，否则的话这个空洞将被分为两部分，一部分供进程使用，另一部分是未用的内存
- 下次适配
  - 每次找到合适的空洞时都记住当时的位置，在下次寻找空洞时从上次结束的地方开始搜索，而不是每次都从头开始
- 最佳适配算法
  - 试图找出最接近实际需要的大小的空洞，而不是把一个以后可能会用到的大空洞先使用
  - 最佳适配算法每次被调用时都要搜索整个链表，因此会比首次适配算法慢
- 最坏匹配算法
  - 在每次分配时，总是将最大的那个空闲区切去一部分，分配给请求者

内存分配算法

- 如果进程和空洞使用不同的链表，空洞链表可以按照大小排序以提高最佳适配的速度。
- 在最佳适配算法搜索由小到大排列的空洞链表时，当它找到一个合适的空洞时它就知道这个空洞是能容纳这个作业的最小的空洞，因此是最佳的，不需要象在单个链表的情况那样继续进行搜索。
- 当空洞链表按大小排序时，首次适配与最佳适配一样快，而下次适配则毫无意义。

快速适配法

- 为一些经常被用到长度的空洞设立单独的链表。
- 例，一个n个项的表，这个表的第一个项是指向长度为4K的空洞的链表的表头的指针，第二个项是指向长度为8K的空洞的链表的指针，第三个项指向长度12K的空洞链表，等等。
- 快速适配算法寻找一个指定大小的空洞是十分迅速的，但在一个进程结束或被换出时寻找它的邻接块以查看是否可以合并是非常费时间的

第四章 提纲

- 4.1 基本的内存管理
- 4.2 交换技术
- 4.3 虚拟存储管理
- 4.4 页面替换算法
- 4.5 页式存储管理的设计问题
- 4.6 段式存储管理
- 4.7 MINIX3进程管理器概述
- 4.8 MINIX3进程管理器实现

虚拟存储器

- 基本思想：操作系统把程序当前使用的那些部分保留在存储器中，而把其他部分保存在磁盘上

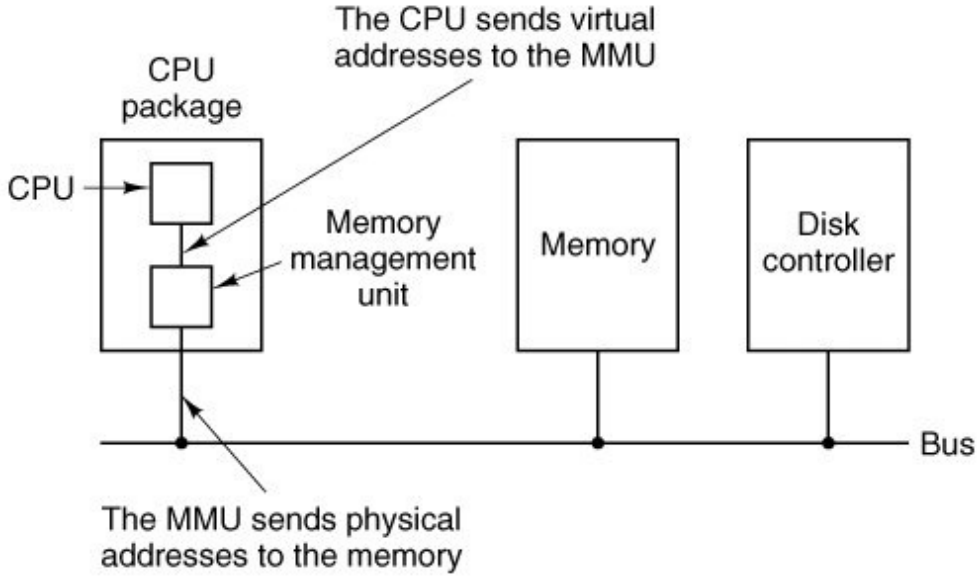
- 例
  - 一个16M的程序，通过仔细地选择在各个时刻将哪4M内容保留在内存中，并在需要时在内存和磁盘间交换程序的片段，那么就可以在一个4M的机器上运行。

虚拟存储管理

- 分页技术
- 页表
- 关联存储器TLB
- 反置页表

分页技术

- 虚地址空间被划分成称为页面（pages）的单位，在物理存储器中对应的单位称为页框（page frames），页和页框总是同样大小的。
- 由程序产生的地址被称为虚地址（virtual addresses），他们构成了一个虚地址空间（virtual address space）。
  - 在没有虚拟存储器的计算机上，虚地址被直接送到内存总线上，使具有同样地址的物理存储器字被读写
  - 在使用虚拟存储器的情况下，虚地址不是被直接送到内存总线上，而是送到存储管理单元(MMU)，它由一个或一组芯片组成，其功能是把虚地址映射为物理地址，

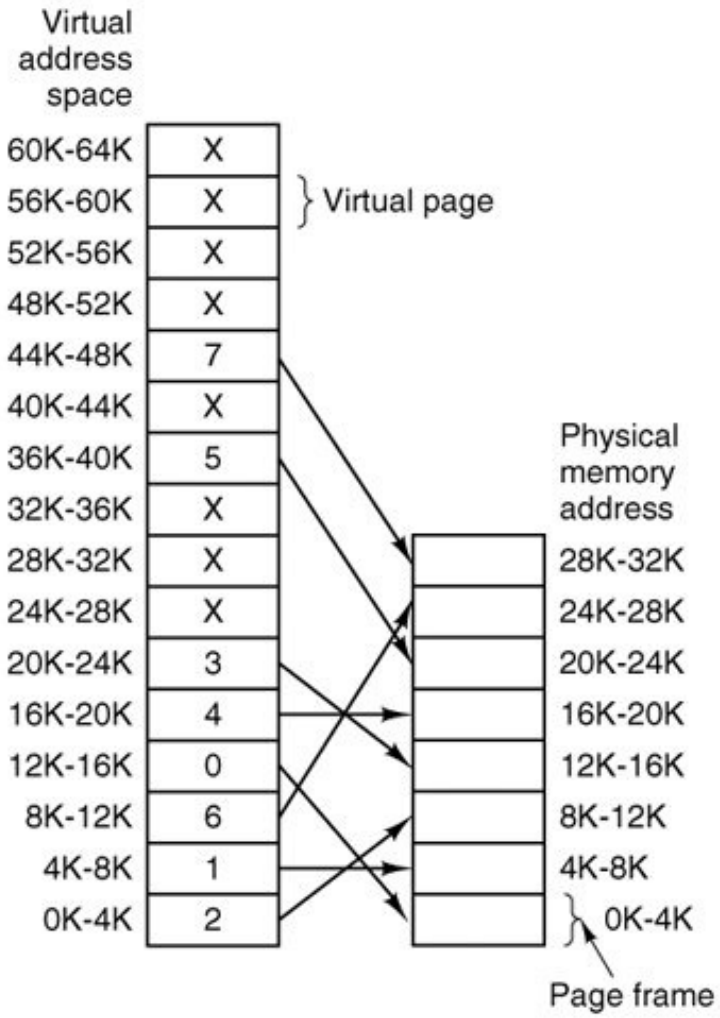


例

- 一台可以生成16位地址的计算机，地址变化范围从0到64K，这些地址是虚地址。
- 该台计算机只有32K的物理存储器，因此虽然可以编写64K的程序，他们却不能被完全调入内存运行。

MOVE REG, 0
MOVE REG, 8192

MOVE REG, 8192
MOVE REG, 24576

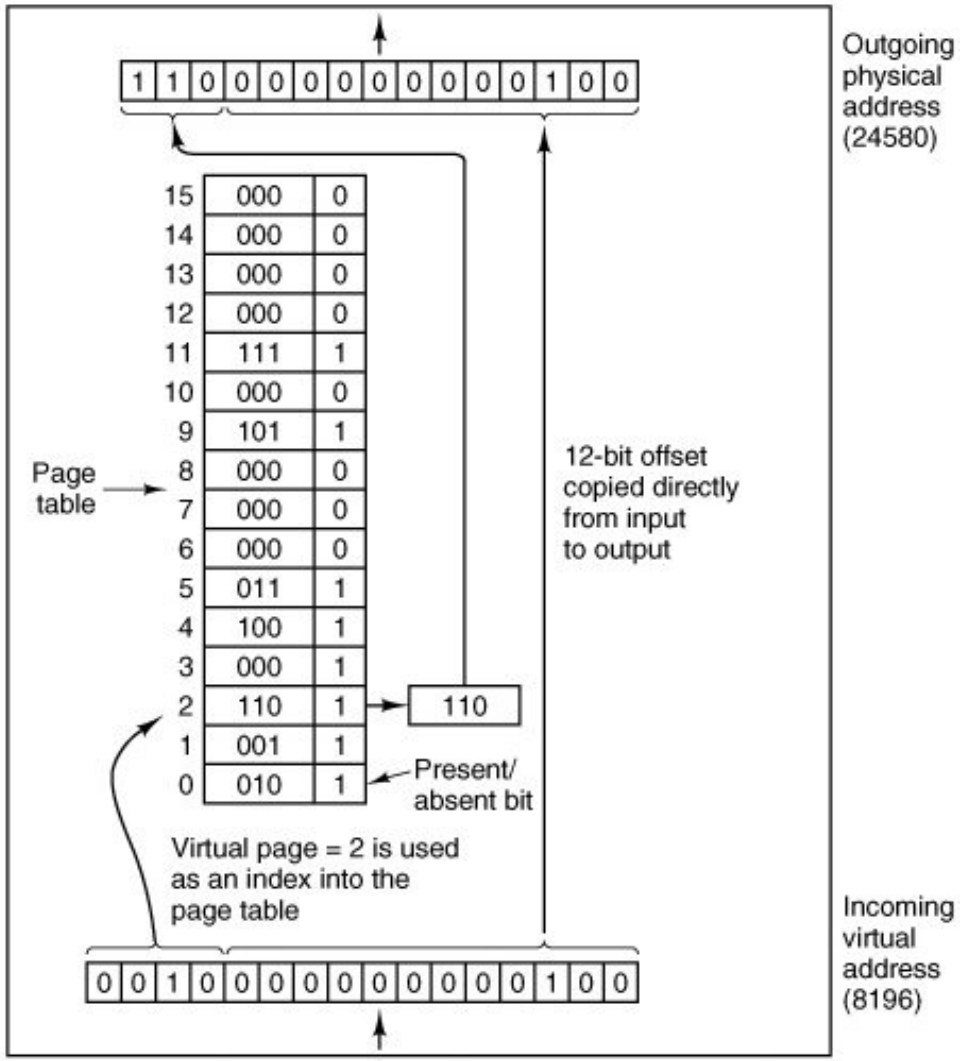


缺页故障

- 当访问未有映射的虚拟页，会引发陷入，这个陷入称为缺页故障
  - 操作系统找到一个很少使用的页框并把它的内容写入磁盘，随后把需引用的页取到刚才释放的页框中，修改映射，然后重新启动引起陷入的指令。

- 例，假设操作系统决定放弃页框1，那么它将把虚页8装入物理地址4K，并对MMU作两处修改：
  - 首先，它要标记虚页1为未映射，以使以后任何对虚地址4K到8K的访问都引起陷入；随后把虚页8对应表项的叉号改为1，
  - 因此在引起陷入的指令重新启动时，它将把虚地址32780映射为物理地址4108。

MMU内部结构



虚地址8196（二进制0010000000000100）

虚拟存储管理

- 分页技术
- 页表
- 关联存储器TLB
- 反置页表

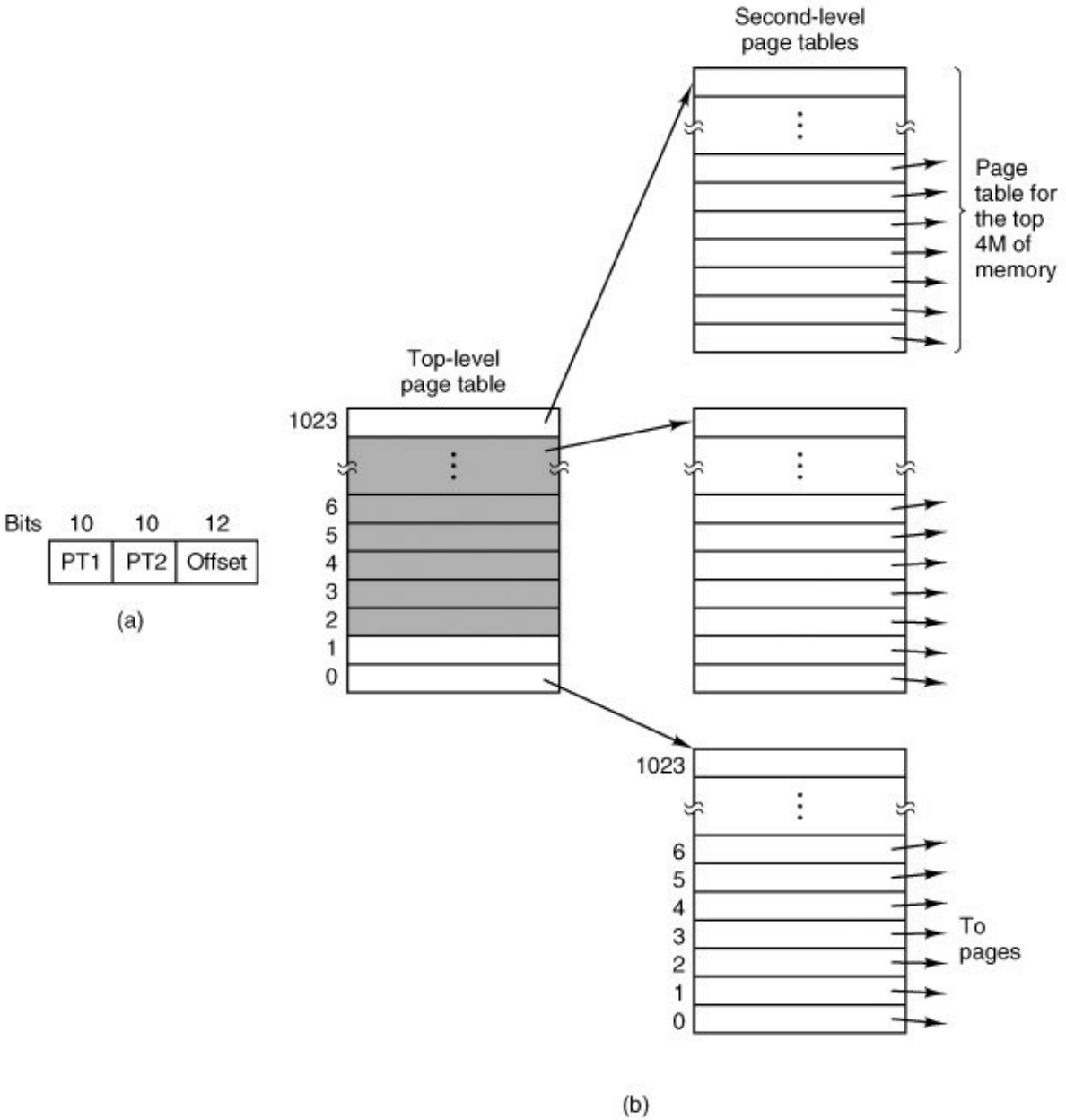
页表

- 虚地址被分成虚页号（高位）和偏移（低位）两部分，虚页号被用做页表的索引以找到该虚页对应的页表项，从页表项中可以找到页框号（如果有的话）。随后页框号被拼接到偏移的高位端，形成送往内存的物理地址。
- 从数学角度而言，页表是一个函数，它的参数是虚页号，结果是物理页框号。通过这个函数可以把虚地址中的虚页域替换成页框域，从而形成物理地址。

页表需要解决的两个主要问题
(1) 页表可能会非常大
(2) 地址映射必须十分迅速

多级页表(例)

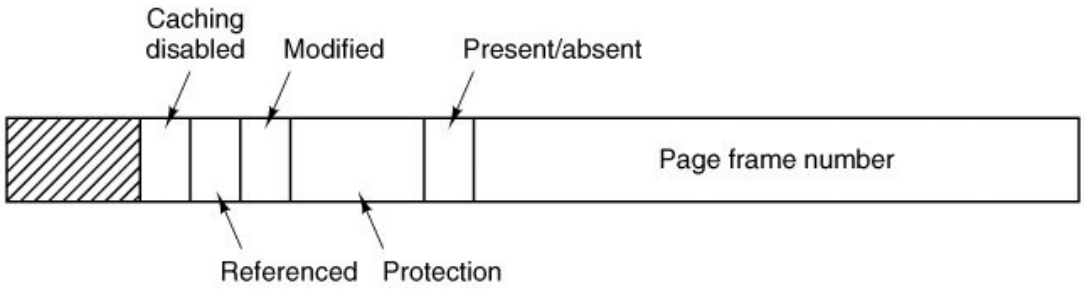
- 32位的虚地址划分为
  - 10位的PT1域
  - 10位的PT2域
  - 12位的偏移，相应地页长是4K，页面共有2^{20}个



多级页表优缺点

- 优点：避免将进程的所有页表项一直保存在内存中
- 缺点：需要多次访问内存，以查找页表
- 例：一个需要12兆字节的进程(地址空间大小为4GB)
  - 最低端是4兆程序正文，后面是4兆数据，顶端是4兆堆栈，在数据顶端上方和堆栈底端之间的上吉（千兆）字节的空洞根本没有使用
  - 虽然地址空间包含100万个页面(4GB/4KB=100万)，实际上只需要四个页表：顶级页表，0到4M、4M到8M和顶端4M的二级页表
  - 顶级页表中有1021个表项的Present/absent位都被设为0，使得访问他们时强制产生一个页面故障
    - 如果这种情况发生了，操作系统将注意到进程在试图访问一个不期望被访问的地址并采取适当的行动

页表项的结构



- 页框号，即物理页面号
- 有效位
  - 这一位是1时这个表项是有效的可以被使用，如果是0，表示这个表项对应的虚页现在不在内存中，访问这一位为0的页会引起一个页面故障。
- 保护位指出这个页允许什么样的访问。
  - 在最简单的形式下这个域只有一位，0表示读写，1表示只读。一个更先进的安排是使用三位，各位分别指出是否允许读、写、执行这个页。
- 修改位和访问位跟踪页的使用
  - 在一个页被写入时硬件自动设置修改位，此位在操作系统重新分配页框时是非常有用的，如果一个页已经被修改过（即它是“脏”的），则必须把它写回磁盘，否则只用简单地把它丢弃就可以了，因为它在磁盘上的拷贝仍然是有效的。此位有时也被称为脏(dirty)位，因为这反映了该页的状态。
  - 访问位在该页被引用时设置，不管是读还是写。它的值被用来帮助操作系统在发生页面故障时选择淘汰的页，不在使用的页要比在使用的页更适合于被淘汰。
- 禁止缓存位
  - 这个特性对那些映射到设备寄存器而不是常规内存的页面是非常重要的。

虚拟存储管理

- 分页技术
- 页表
- 关联存储器TLB
- 反置页表

TLB

- 在现代计算机系统中，一小部分页表的拷贝被保存在处理器芯片中，这部分页表是由最近访问页的页表项组成，这小部分页表是保存在转换后备缓冲缓存(TLB)中(通常在MMU中)。

当一个虚地址被送到MMU翻译时，硬件首先把它和TLB中的所有条目同时（并行地）进行比较。

如果找到了并且这个访问没有违反保护位，它的页框号将直接从TLB中取出而不用去查页表

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75