**操作系统**

# 虚拟化CPU

# 虚拟化内存

## 直接使用物理内存的遇到的问题

**问题1**：进程地址空间不隔离。由于程序都是直接访问物理内存，所以恶意程序可以随意修改别的进程的内存数据，以达到破坏的目的。有些非恶意的，但是有bug的程序也可能不小心修改了其它程序的内存数据，就会导致其它程序的运行出现异常。这种情况对用户来说是无法容忍的，因为用户希望使用计算机的时候，其中一个任务失败了，至少不能影响其它的任务。

**问题2**：内存使用效率低。在A和B都运行的情况下，如果用户又运行了程序C，而程序C需要20M大小的内存才能运行，而此时系统只剩下8M的空间可供使用，所以此时系统必须在已运行的程序中选择一个将该程序的数据暂时拷贝到硬盘上，释放出部分空间来供程序C使用，然后再将程序C的数据全部装入内存中运行。可以想象得到，在这个过程中，有大量的数据在装入装出，导致效率十分低下。

**问题3**：程序运行的地址不确定。当内存中的剩余空间可以满足程序C的要求后，操作系统会在剩余空间中随机分配一段连续的20M大小的空间给程序C使用，因为是随机分配的，所以程序运行的地址是不确定的。

## 解决方法

### 二 分段

为了解决上述问题，人们想到了一种变通的方法，就是增加一个中间层，利用一种间接的地址访问方法访问物理内存。按照这种方法，程序中访问的内存地址不再是实际的物理内存地址，而是一个虚拟地址，然后由操作系统将这个虚拟地址映射到适当的物理内存地址上。这样，只要操作系统处理好虚拟地址到物理内存地址的映射，就可以保证不同的程序最终访问的内存地址位于不同的区域，彼此没有重叠，就可以达到内存地址空间隔离的效果。

当创建一个进程时，操作系统会为该进程分配一个4GB大小的虚拟进程地址空间。之所以是4GB，是因为在32位的操作系统中，一个指针长度是4字节，而4字节指针的寻址能力是从0x00000000~0xFFFFFFFF，最大值0xFFFFFFFF表示的即为4GB大小的容量。与虚拟地址空间相对的，还有一个物理地址空间，这个地址空间对应的是真实的物理内存。如果你的计算机上安装了512M大小的内存，那么这个物理地址空间表示的范围是0x00000000~0x1FFFFFFF。当操作系统做虚拟地址到物理地址映射时，只能映射到这一范围，操作系统也只会映射到这一范围。当进程创建时，每个进程都会有一个自己的4GB虚拟地址空间。要注意的是这个4GB的地址空间是“虚拟”的，并不是真实存在的，而且每个进程只能访问自己虚拟地址空间中的数据，无法访问别的进程中的数据，通过这种方法实现了进程间的地址隔离。那是不是这4GB的虚拟地址空间应用程序可以随意使用呢？很遗憾，在Windows系统下，这个虚拟地址空间被分成了4部分：NULL指针区、用户区、64KB禁入区、内核区。应用程序能使用的只是用户区而已，大约2GB左右(最大可以调整到3GB)。内核区为2GB，内核区保存的是系统线程调度、内存管理、设备驱动等数据，这部分数据供所有的进程共享，但应用程序是不能直接访问的。

人们之所以要创建一个虚拟地址空间，目的是为了解决进程地址空间隔离的问题。但程序要想执行，必须运行在真实的内存上，所以，必须在虚拟地址与物理地址间建立一种映射关系。这样，通过映射机制，当程序访问虚拟地址空间上的某个地址值时，就相当于访问了物理地址空间中的另一个值。人们想到了一种分段(Sagmentation)的方法，它的思想是在虚拟地址空间和物理地址空间之间做一一映射。比如说虚拟地址空间中某个10M大小的空间映射到物理地址空间中某个10M大小的空间。这种思想理解起来并不难，操作系统保证不同进程的地址空间被映射到物理地址空间中不同的区域上，这样每个进程最终访问到的

物理地址空间都是彼此分开的。通过这种方式，就实现了进程间的地址隔离。还是以实例说明，假设有两个进程A和B，进程A所需内存大小为10M，其虚拟地址空间分布在0x00000000到0x00A00000，进程B所需内存为100M，其虚拟地址空间分布为0x00000000到0x06400000。那么按照分段的映射方法，进程A在物理内存上映射区域为0x00100000到0x00B00000，，进程B在物理内存上映射区域为0x00C00000到0x07000000。于是进程A和进程B分别被映射到了不同的内存区间，彼此互不重叠，实现了地址隔离。从应用程序的角度看来，进程A的地址空间就是分布在0x00000000到0x00A00000，在做开发时，开发人员只需访问这段区间上的地址即可。应用程序并不关心进程A究竟被映射到物理内存的那块区域上了，所以程序的运行地址也就是相当于说是确定的了。 图二显示的是分段方式的内存映射方法。

操作系统管理内存的机制——为什么要设置虚拟内存？ - 神之子 - 研究生了没

图二 分段方式的内存映射方法

这种分段的映射方法虽然解决了上述中的问题一和问题三，但并没能解决问题二，即内存的使用效率问题。在分段的映射方法中，每次换入换出内存的都是整个程序，这样会造成大量的磁盘访问操作，导致效率低下。所以这种映射方法还是稍显粗糙，粒度比较大。实际上，程序的运行有局部性特点，在某个时间段内，程序只是访问程序的一小部分数据，也就是说，程序的大部分数据在一个时间段内都不会被用到。基于这种情况，人们想到了粒度更小的内存分割和映射方法，这种方法就是分页(Paging)。

### 三 分页

分页的基本方法是，将地址空间分成许多的页。每页的大小由CPU决定，然后由操作系统选择页的大小。目前Inter系列的CPU支持4KB或4MB的页大小，而PC上目前都选择使用4KB。按这种选择，4GB虚拟地址空间共可以分成1048576个页，512M的物理内存可以分为131072个页。显然虚拟空间的页数要比物理空间的页数多得多。

在分段的方法中，每次程序运行时总是把程序全部装入内存，而分页的方法则有所不同。分页的思想是程序运行时用到哪页就为哪页分配内存，没用到的页暂时保留在硬盘上。当用到这些页时再在物理地址空间中为这些页分配内存，然后建立虚拟地址空间中的页和刚分配的物理内存页间的映射。

下面通过介绍一个可执行文件的装载过程来说明分页机制的实现方法。一个可执行文件(PE文件)其实就是一些编译链接好的数据和指令的集合，它也会被分成很多页，在PE文件执行的过程中，它往内存中装载的单位就是页。当一个PE文件被执行时，操作系统会先为该程序创建一个4GB的进程虚拟地址空间。前面介绍过，虚拟地址空间只是一个中间层而已，它的功能是利用一种映射机制将虚拟地址空间映射到物理地址空间，所以，创建4GB虚拟地址空间其实并不是要真的创建空间，只是要创建那种映射机制所需要的数据结构而已，这种数据结构就是页目和页表。

当创建完虚拟地址空间所需要的数据结构后，进程开始读取PE文件的第一页。在PE文件的第一页包含了PE文件头和段表等信息，进程根据文件头和段表等信息，将PE文件中所有的段一一映射到虚拟地址空间中相应的页(PE文件中的段的长度都是页长的整数倍)。这时PE文件的真正指令和数据还没有被装入内存中，操作系统只是根据PE文件的头部等信息建立了PE文件和进程虚拟地址空间中页的映射关系而已。当CPU要访问程序中用到的某个虚拟地址时，当CPU发现该地址并没有相相关联的物理地址时，CPU认为该虚拟地址所在的页面是个空页面，CPU会认为这是个页错误(Page Fault)，CPU也就知道了操作系统还未给该PE页面分配内存，CPU会将控制权交还给操作系统。操作系统于是为该PE页面在物理空间中分配一个页面，然后再将这个物理页面与虚拟空间中的虚拟页面映射起来，然后将控制权再还给进程，进程从刚才发生页错误的位置重新开始执行。由于此时已为PE文件的那个页面分配了内存，所以就不会发生页错误了。随着程序的执行，页错误会不断地产生，操作系统也会为进程分配相应的物理页面来满足进程执行的需求。

分页方法的核心思想就是当可执行文件执行到第x页时，就为第x页分配一个内存页y，然后再将这个内存页添加到进程虚拟地址空间的映射表中,这个映射表就相当于一个y=f(x)函数。应用程序通过这个映射表就可以访问到x页关联的y页了。

## 为什么虚拟化内存

* 虚拟内存可以结合磁盘和物理内存的优势为进程提供看起来速度足够快并且容量足够大的存储；
* 虚拟内存可以为进程提供独立的内存空间并引入多层的页表结构将虚拟内存翻译成物理内存，进程之间可以共享物理内存减少开销，也能简化程序的链接、装载以及内存分配过程；
* 虚拟内存可以控制进程对物理内存的访问，隔离不同进程的访问权限，提高系统的安全性；

# 线程的上下文切换

当CPU从执行一个线程切换到执行另外一个线程的时候，它需要先存储当前线程的本地的数据，程序指针等，然后载入另一个线程的本地数据，程序指针等，最后才开始执行。这种切换称为“上下文切换”(“context switch”)。CPU会在一个上下文中执行一个线程，然后切换到另外一个上下文中执行另外一个线程。