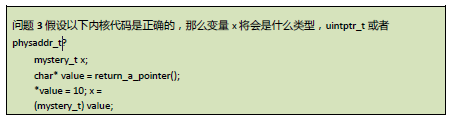
问题3：



答：由于这里使用了 \* 操作符解析地址，所以变量x应该是uintptr\_t类型。

作业4：

有关知识：

物理地址(physical address)  
用于内存芯片级的单元寻址，与处理器和CPU连接的地址总线相相应。  
——这个概念应该是这几个概念中最好理解的一个。可是值得一提的是，尽管能够直接把物理地址理解成插在机器上那根内存本身，把内存看成一个从0字节一直到最大空量逐字节的编号的大数组，然后把这个数组叫做物理地址。可是其实，这仅仅是一个硬件提供给软件的抽像，内存的寻址方式并非这样。所以，说它是“与地址总线相相应”，是更贴切一些。只是抛开对物理内存寻址方式的考虑，直接把物理地址与物理的内存一一相应，也是能够接受的。或许错误的理解更利于形而上的抽像。

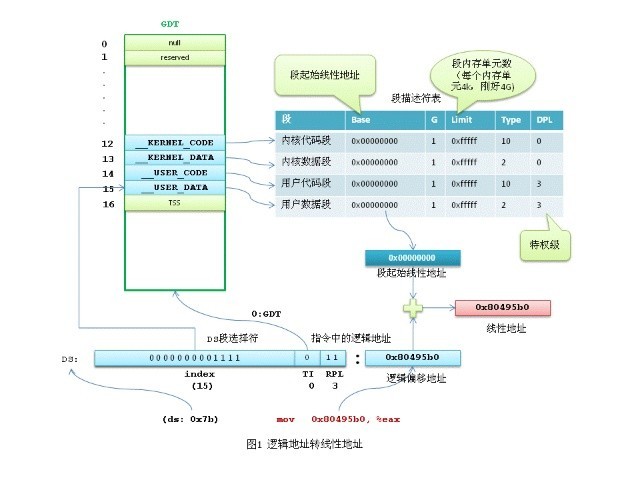
虚拟内存(virtual memory)  
这是对整个内存（不要与机器上插那条对上号）的抽像描写叙述。

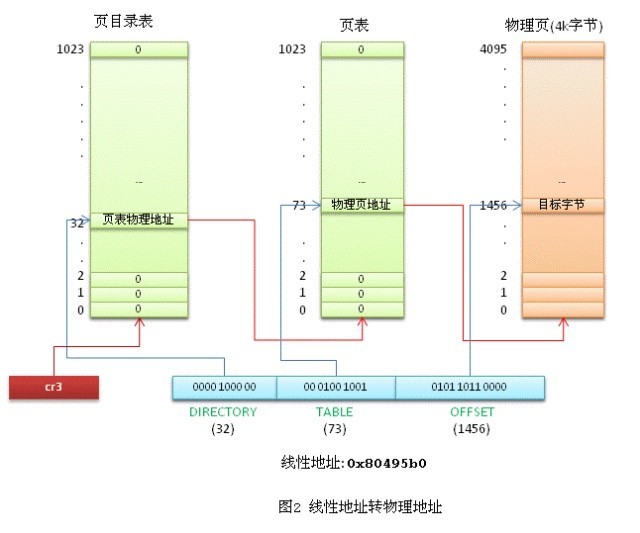
它是相对于物理内存来讲的，能够直接理解成“不直实的”，“假的”内存。比如，一个0x08000000内存地址。它并不正确就物理地址上那个大数组中0x08000000 - 1那个地址元素。  
之所以是这样。是由于现代操作系统都提供了一种内存管理的抽像，即虚拟内存（virtual memory）。进程使用虚拟内存中的地址，由操作系统协助相关硬件，把它“转换”成真正的物理地址。

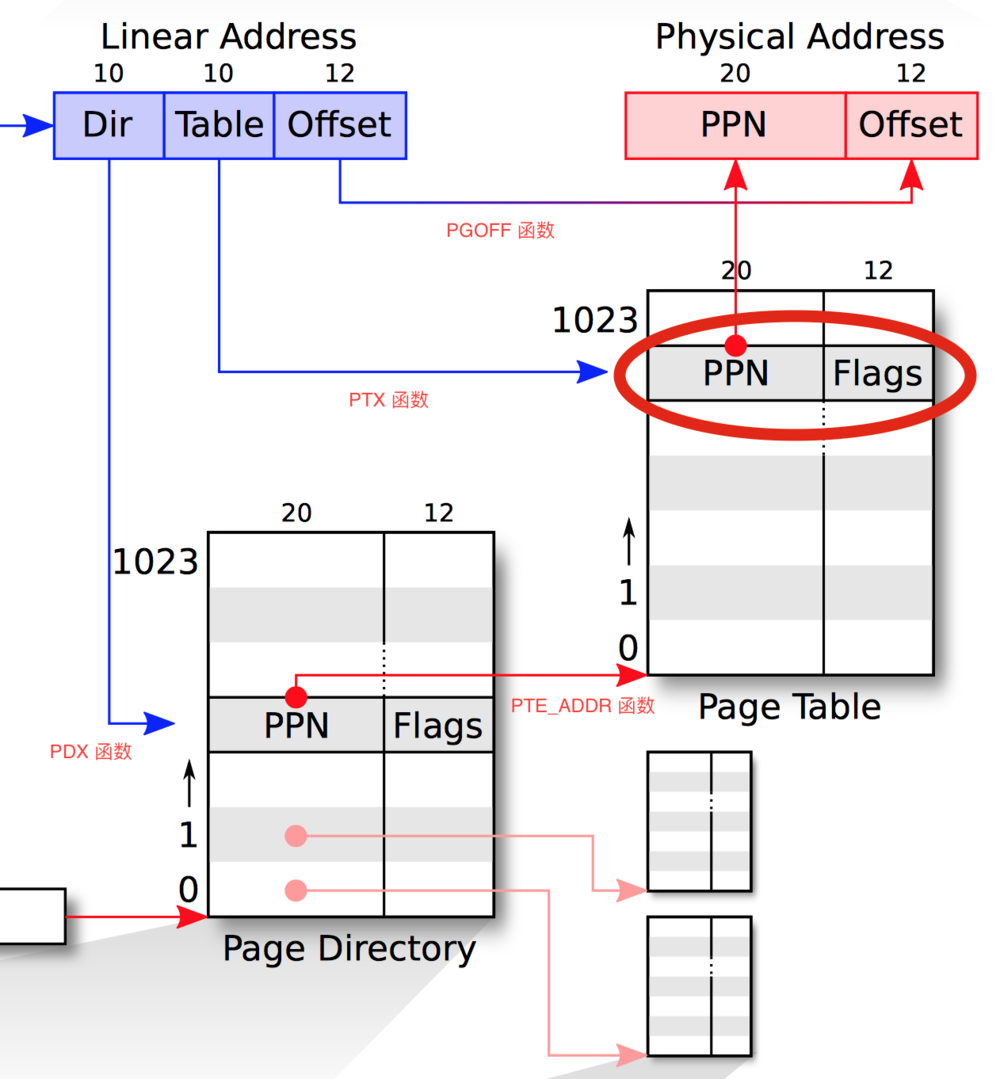
这个“转换”。是全部问题讨论的关键。  
有了这种抽像。一个程序，就能够使用比真实物理地址大得多的地址空间。

甚至多个进程能够使用同样的地址。不奇怪。由于转换后的物理地址并不是同样的。  
——能够把连接后的程序反编译看一下，发现连接器已经为程序分配了一个地址，比如，要调用某个函数A，代码不是call A，而是call 0x0811111111 ，也就是说，函数A的地址已经被定下来了。没有这种“转换”，没有虚拟地址的概念，这样做是根本行不通的。  
  
逻辑地址(logical address)  
Intel为了兼容，将远古时代的段式内存管理方式保留了下来。

逻辑地址指的是机器语言指令中，用来指定一个操作数或者是一条指令的地址。以上例，我们说的连接器为A分配的0x08111111这个地址就是逻辑地址。。“一个逻辑地址，是由一个段标识符加上一个指定段内相对地址的偏移量。表示为 [段标识符：段内偏移量]，也就是说。上例中那个0x08111111，应该表示为[A的代码段标识符: 0x08111111]，这样。才完整一些”  
  
线性地址(linear address)或也叫虚拟地址(virtual address)  
跟逻辑地址类似，它也是一个不真实的地址，假设逻辑地址是相应的硬件平台段式管理转换前地址的话，那么线性地址则相应了硬件页式内存的转换前地址。







首先完成pgdir\_walk函数，函数原型 pgdir\_walk(pde\_t \*pgdir, const void \*va, int create)，该函数的功能在注释中解释道：

**给定一个页目录表指针 pgdir ，该函数应该返回线性地址va所对应的页表项指针。**

　　所以在这里我们应该完成以下几个步骤：

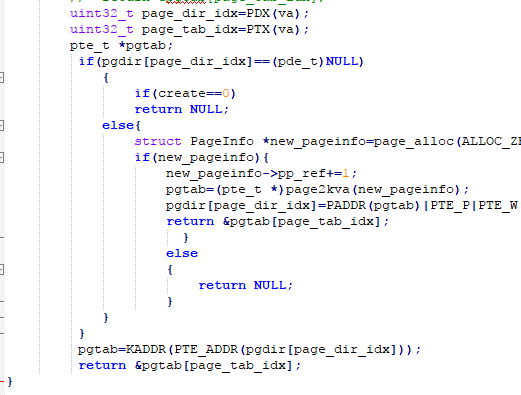
　　1. 通过页目录表求得这个虚拟地址所在的页表页对于与页目录中的页目录项地址 dic\_entry\_ptr。

　　2. 判断这个页目录项对应的页表页是否已经在内存中。

　　3. 如果在，计算这个页表页的基地址page\_base，然后返回va所对应页表项的地址 &page\_base[page\_off]

　　4. 如果不在则，且create为true则分配新的页，并且把这个页的信息添加到页目录项dic\_entry\_ptr中。

　　　　5. 如果create为false，则返回NULL。

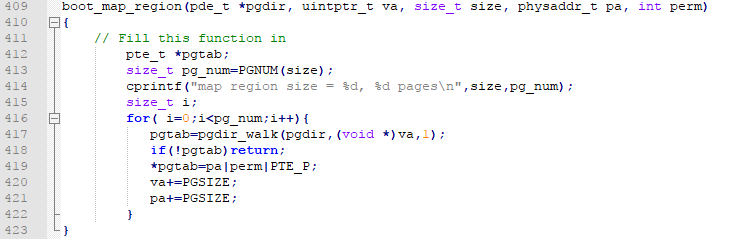


接下来完成boot\_map\_region函数，函数原型 static void boot\_map\_region(pde\_t \*pgdir, uintptr\_t va, size\_t size, physaddr\_t pa, int perm)，这个函数的功能在注释中被这样解释：

　　　　把虚拟地址空间范围[va, va+size)映射到物理空间[pa, pa+size)的映射关系加入到页表pgdir中。这个函数主要的目的是为了设置虚拟地址UTOP之上的地址范围，这一部分的地址映射是静态的，在操作系统的运行过程中不会改变，所以这个页的PageInfo结构体中的pp\_ref域的值不会发生改变。

　　　　这个函数要完成的步骤如下：

1. 需要完成一个循环，在每一轮中，把一个虚拟页和物理页的映射关系存放到响应的页表项中。直到把size个字节的内存都分配完。



接下来再继续查看page\_insert()，函数原型如下 page\_insert(pde\_t \*pgdir, struct PageInfo \*pp, void \*va, int perm)，功能上是完成：把一个物理内存中页pp与虚拟地址va建立映射关系。

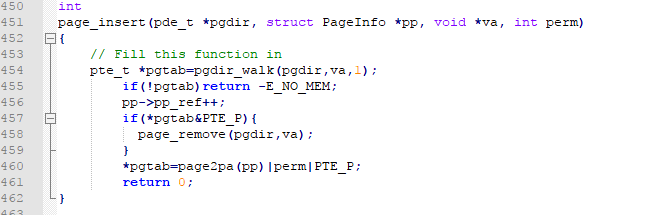
　　　　这个函数的主要步骤如下：

　　　　1. 首先通过pgdir\_walk函数求出虚拟地址va所对应的页表项。(4)

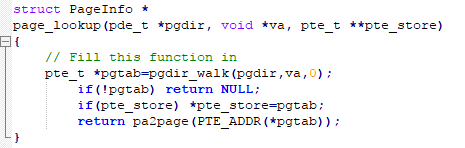
　　　　2. 修改pp\_ref的值。(8)

　　　　3. 查看这个页表项，确定va是否已经被映射，如果被映射，则删除这个映射。(9-13)

　　　　4. 把va和pp之间的映射关系加入到页表项中。(14-15)



page\_lookup 函数



最后一个就是page\_remove函数，它的原型是：void page\_remove(pde\_t \*pgdir, void \*va)，功能就是把虚拟地址va和物理页的映射关系删除。

　　　　注释里面还提示了要注意的几个细节：

　　　　1. pp\_ref值要减一

　　　　2. 如果pp\_ref减为0，要把这个页回收

　　　　3. 这个页对应的页表项应该被置0

