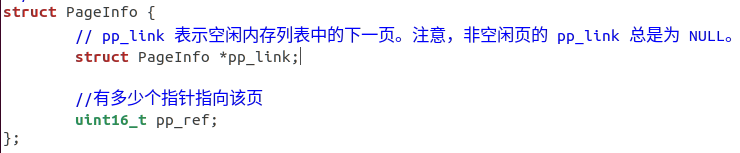
3.1 物理页管理

操作系统必需跟踪哪些物理 RAM 是空闲的，哪些正在使用。这个练习主要编写物理页面分配器。它利用一个 PageInfo 结构体组成的链表记录哪些页面空闲，每个结构体对应一个物理页。

因为页表的实现需要分配物理内存来存储页表，在虚拟内存的实现之前，我们需要先编写物理页面分配器。



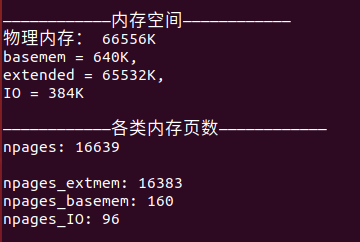
*（memlayout.h中PageInfo的定义）*

在文件 kern/pmap.c 中，内核刚开始运行时就会调用函数mem\_init()，对整个操作系统的内存管理系统进行一些初始化的设置。

【步骤一】进入mem\_init()，第一步是调用子函数i386\_detect\_memory()来检查现在系统中有多少可用的总内存空间以及三个部分内存空间各自多大，并根据设置的PGSIZE划分页，在此函数中计算总页数和三个部分各自的页数。

JOS把整个物理内存空间划分成三个部分：

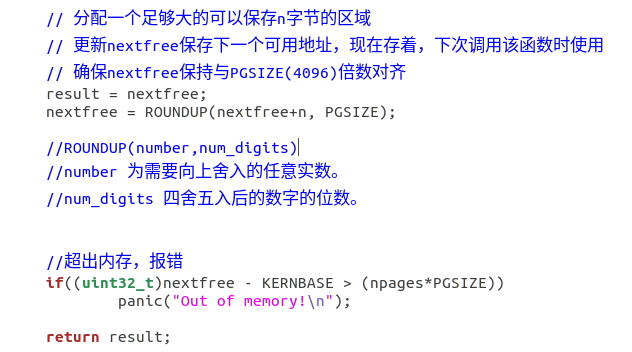
1. Basemen：从0x00000~0xA0000，可用
2. IO hole：0xA0000~0x100000，不可用，主要被用来分配给外部设备了。
3. Extmen：0x100000~0x，可用，最重要的内存区域。



*（i386\_detect\_memory()运行结果）*

【步骤二】（1）调用函数boot\_alloc函数分配一块大小为PGSIZE的、紧跟着操作系统内核bss之后的虚拟内存，用于存放操作系统页表。操作系统之后工作在虚拟内存模式下时，需要这个页目录表进行地址转换（虚拟地址→物理地址）。

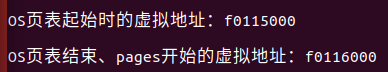
其中boot\_alloc函数可以用于分配虚拟内存，其参数是要分配的虚拟内存字节大小，当输入0时可以查询当前使用的虚拟内存尾。这个函数的核心思想是维护一个静态变量nextfree，里面存放着下一个可以使用的空闲内存空间对应的虚拟地址。第一次进入该函数时需要对nextfree进行初始化。注意：分配的内存大小需要使用ROUNDUP函数使之与PGSIZE保持倍数对齐。



*（bootalloc()函数补充的代码）*

（2）初始化一个物理内存中的指针kern\_pgdir指向上述操作系统页表。

（3）调用memset函数清空操作系统页表区域虚拟内存。



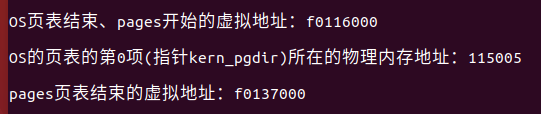
*（通过信息打印可知，OS页表结束地址-OS页表起始地址=PGSIZE）*

【步骤三】为刚刚创建的页目录表添加第一个页目录表项，占用页表的第0个位置。表项内容即指针kern\_pgdir的真实地址。UVPT的定义是一段虚拟地址的起始地址，0xef400000，从这个虚拟地址开始，存放的就是这个操作系统的页表kern\_pgdir，所以我们必须把它和页表kern\_pgdir的物理地址映射起来，PADDR(kern\_pgdir)就是在计算kern\_pgdir所对应的真实物理地址。

C:\Users\Lenovo1\AppData\Roaming\Tencent\Users\40800931\QQ\WinTemp\RichOle\THBS]_I%}RKLBHXX3]Q@J0M.png

*（通过信息打印可知指针所在的真实物理地址）*

【步骤四】使用boot\_alloc函数分配一块大小为npages \* sizeof(struct PageInfo)的虚拟内存，用于存放一个struct PageInfo的数组，内存中的指针pages指向这块虚拟内存。OS通过这个数组来追踪所有内存页的使用情况。每一个PageInfo代表内存当中的一页。PageInfo有2个属性：1、当前页是否被占用。2、指向下一个页的指针。之后调用memset函数清空对应区域内存。



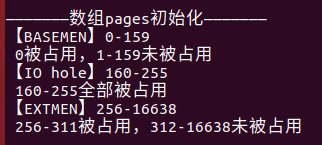
*（为pages分配虚拟内存）*

注意：通过信息打印发现，pages页表结束时的内存地址减pages开始时的内存地址并不完全地等于npages \* sizeof(struct PageInfo)，而是略大于，原因是在boot\_alloc函数中分配虚拟内存时，需要注意与PGSIZE倍数对齐，故对分配的虚拟内存大小向上适当多分配。

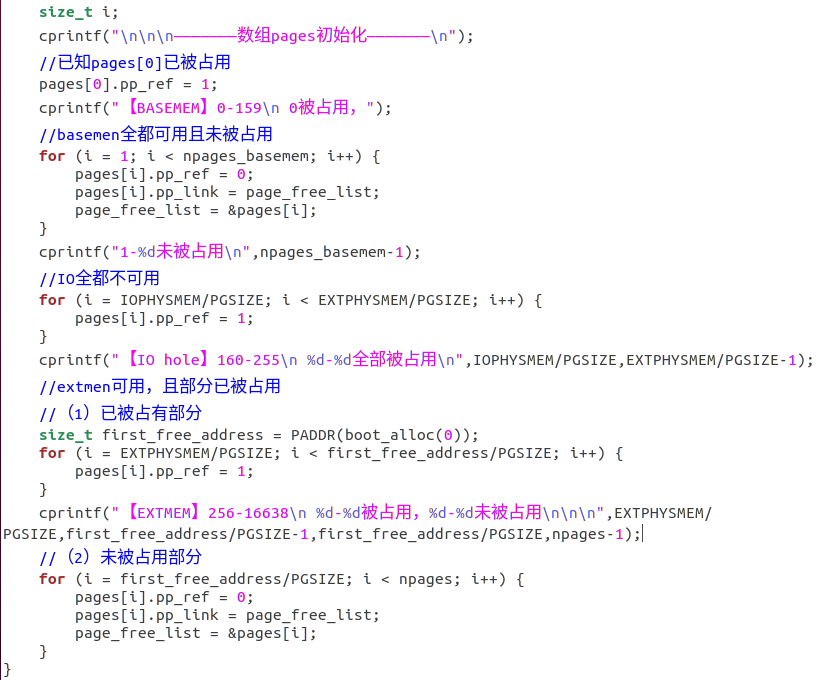
【步骤五】调用函数page\_init()初始化pages和page\_free\_list两个struct pageinfo的链表。其中pages\_free\_list链表中存放着所有空闲页的信息，而pages中存放着所以空闲\非空闲页的信息。

jos把整个物理内存空间划分成三个部分，即BASEMEM、IO HOLE、EXTMEM。

根据已知basemem的第一项已被占用，而basemem的其他页未被占用。而IO HOLE由于主要被用来分配给外部设备而不可用，故相当于所有页均被占用。EXTMEM在步骤四中前一部分页被占用，部分空闲。



*（pages初始化结果）*



*（page\_init()补充代码截图）*

【步骤六】调用check\_page\_free\_list() 和 check\_page\_alloc()测试以上物理页分配器。

check\_page\_free\_list()检查page\_free\_list链表的所谓空闲页是否真的都是合法的、空闲的。当输入参数为1时，这个函数要在检查前先进行一步额外的操作，对空闲页链表free\_page\_list进行修改，经过page\_init，free\_page\_list中已经存放了所有的空闲页表，但是他们的顺序是按照页表的编号从大到小排列的。当前操作系统所采用的页目录表entry\_pgdir（不是kern\_pgdir）中，并没有对大编号的页表进行映射，所以这部分页表我们还不能操作。但是小编号的页表，即从0号页表开始到1023号页表，已经映射过了，所以可以对这部分页表进行操作。那么check\_page\_free\_list(1)要完成的就是把这部分页表对应的PageInfo结构体移动到free\_page\_list的前端，供操作系统现在使用。剩下的操作就是对free\_page\_list进行检查。

check\_page\_alloc()函数的功能是检查page\_alloc()，page\_free()两个子函数是否能够正确运行。

作业要求之一是实现page\_alloc()函数，通过注释我们可以知道这个函数的功能是分配一个物理页。而函数的返回值就是这个物理页所对应的PageInfo结构体。所以这个函数的大致步骤应该是：

　　1. 从free\_page\_list中取出一个空闲页的PageInfo结构体

　　2. 修改free\_page\_list相关信息，比如修改链表表头

　　3. 修改取出的空闲页的PageInfo结构体信息，初始化该页的内存



*（page\_alloc()函数补充代码）*

同样，作业要求实现page\_free()，根据注释可知，这个方法的功能就是把一个页的PageInfo结构体再返回给page\_free\_list空闲页链表，代表回收了这个页。主要完成以下几个操作：

　　1. 修改被回收的页的PageInfo结构体的相应信息。

2. 把该结构体插入回page\_free\_list空闲页链表



*（page\_free函数补充代码）*