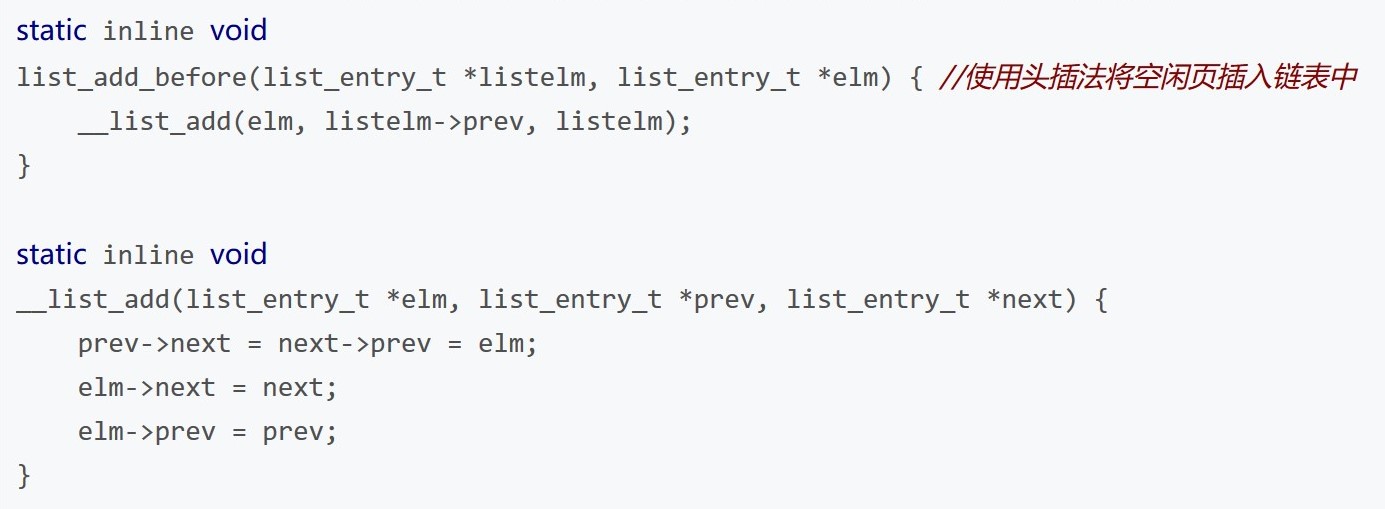
**uCore操作系统Lab2实验报告**

1711291 信息安全 李季航

---- 一份提满了愚蠢和细节问题的实验报告

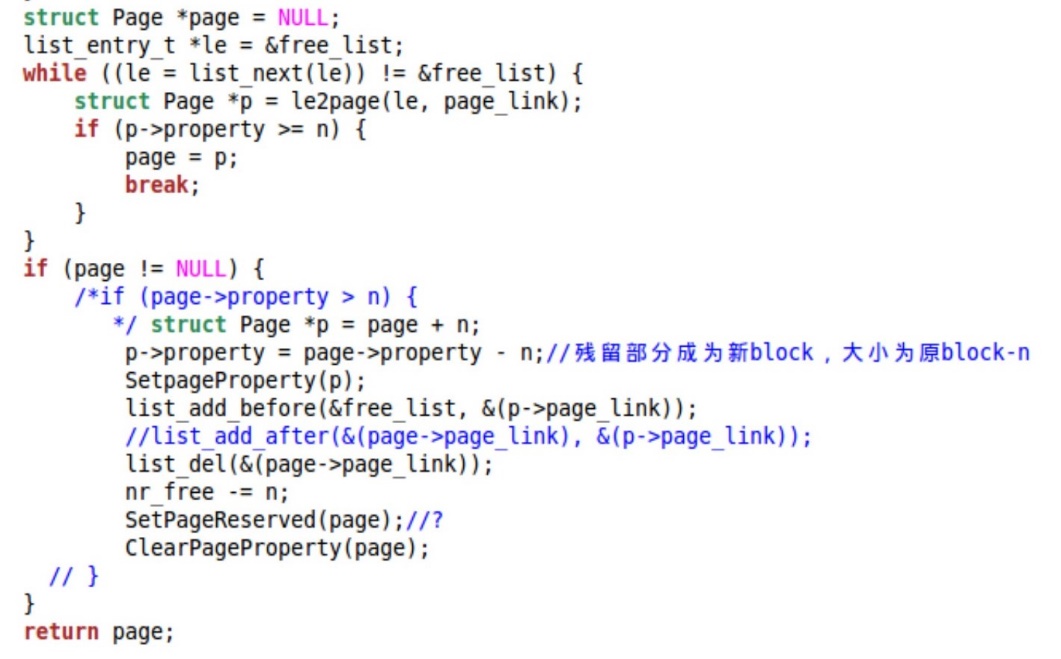
1. **练习1：实现first-fit连续物理内存分配算法**
2. default\_init\_memmap
   1. 为什么要把list\_add改为list\_add\_before？根据定义来看，把base的链接指针插入free\_list前的位置不会导致free\_list起始位置改变吗？



原本这是个困扰了我很久的问题，直到我在思考为什么会有“(le=list\_next(le))!=&free\_list”这个循环终止条件的时候醍醐灌顶意识到这是个双向链表。。。。在free\_list前插入实际就是。。某种意义上插在了链表尾。。。也正是因为成环，所以可以根据判断是否再次回到头来终止循环。。啊我可能是个傻子

1. default\_alloc\_pages

这是我一上来写出来的情况，存在几个问题



* 1. SetPageProperty、PageReserved是什么关系？ 在n大小块被找到后需不需要SetPageReserved？
     1. SPP设置property位，表示被存放到free\_list中了；SPR表示该页已经被内存申请占用了。
     2. PR检查reserved or not
     3. 不知道该不该反正我设了SPR并且改了base(i.e. page)->property=n，表示被挑出来了一个大小为n的空间，防止如果要用的话base->property还是原来大于等于n的另一个值。
  2. 为什么存在page->property>n的二次确认？

这也是往后写才想到。。因为有可能空block的大小正好和所需大小相等，此时就尽管分配而不需要把残留部分设置为新的一块。。之前判断的是是否有大于等于所需大小的块。。。。

* 1. 为什么这里是add\_after？

因为这是一个按内存地址顺序存储的list，必须要把p放到原来page的后面相邻位置正确！

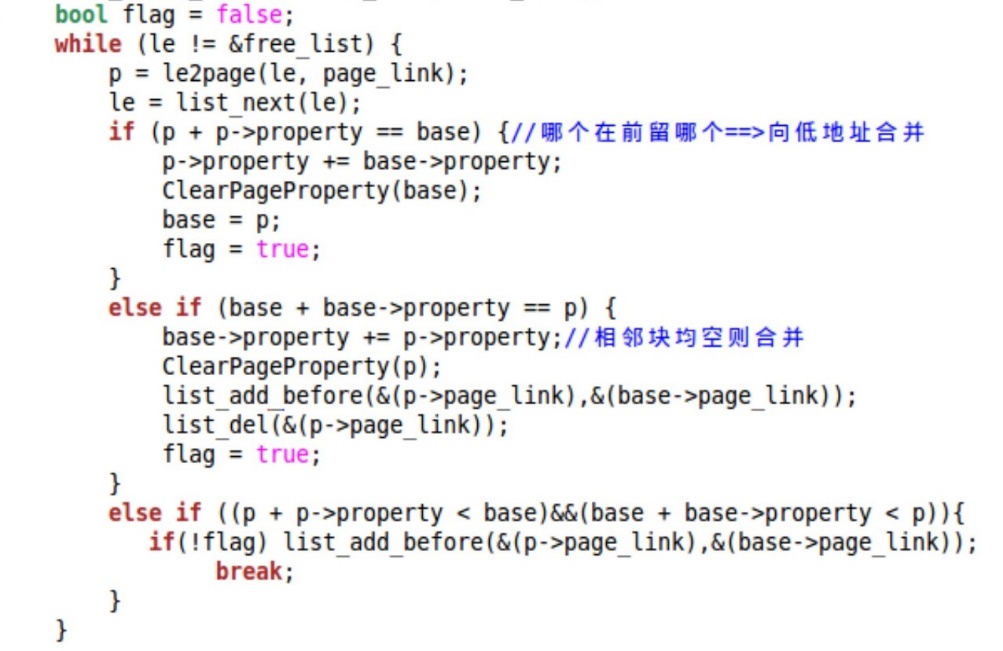
由于改完和答案有些不一样所以把自己的放上来。。

****

1. default\_free\_pages
   1. 最后的list\_add不多余吗既然前面删除的是被合并的block头？

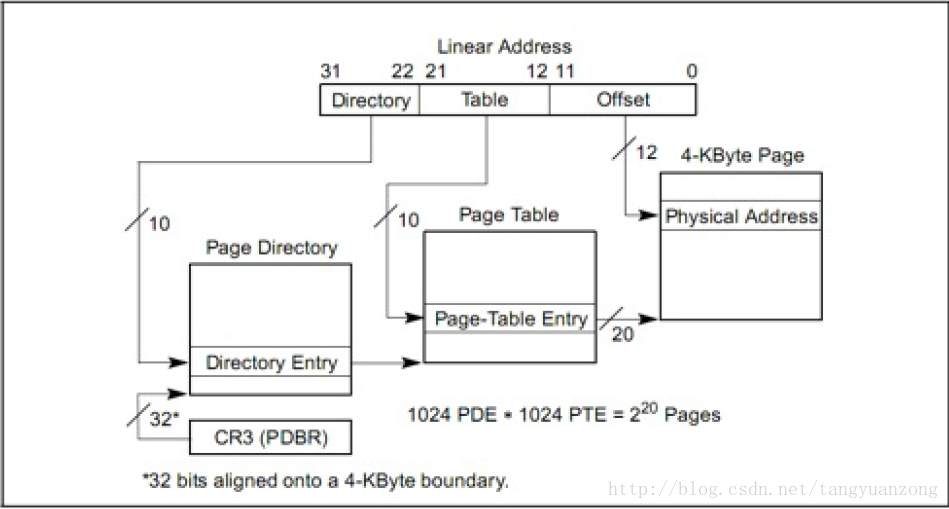
问了朋友^v^，原本base是被占用的，所以前面才会需要一个循环释放，然后找的是base前后是否有相邻空间的空块，base本身并不是空的所以最后要加入到list里。我觉得我需要睡觉了。。。。这都什么愚蠢的问题

* 1. 答案最后又做了一次循环给这块合并出来的新块找位置，感觉不太需要再循环一次，做了一点优化。可以在超过base位置时，如果并未完成合并则插入base，如果合并过则直接终止循环。

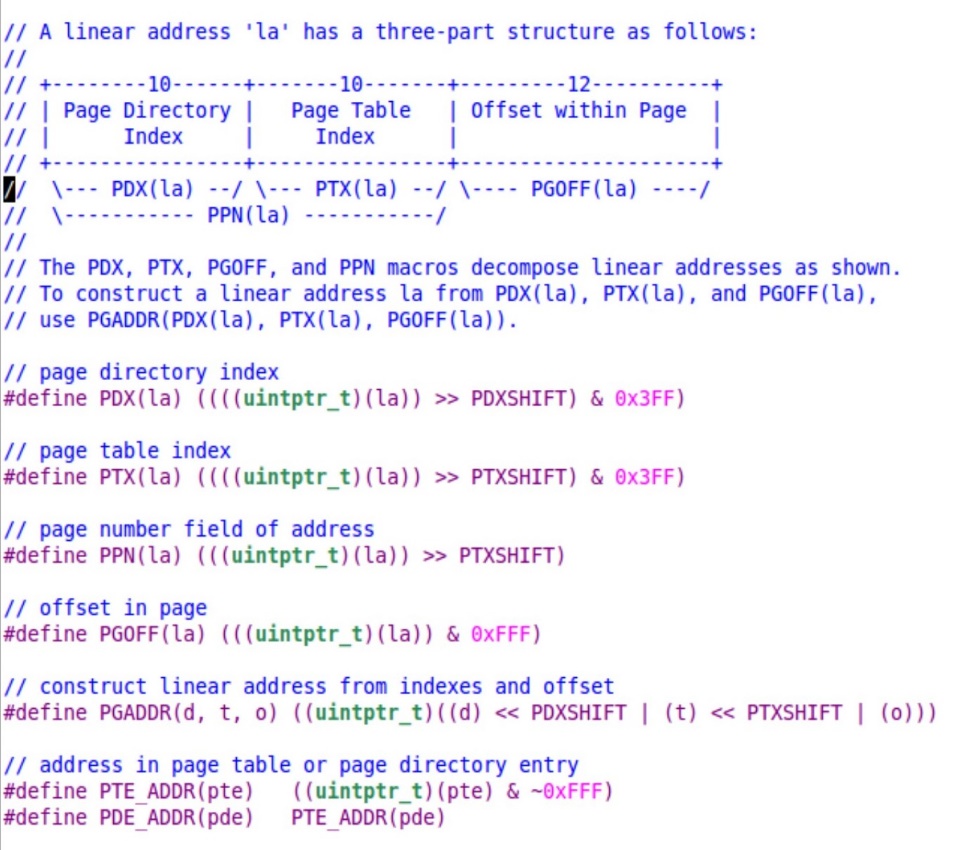
****

1. **练习2：实现寻找虚拟地址对应的页表项**

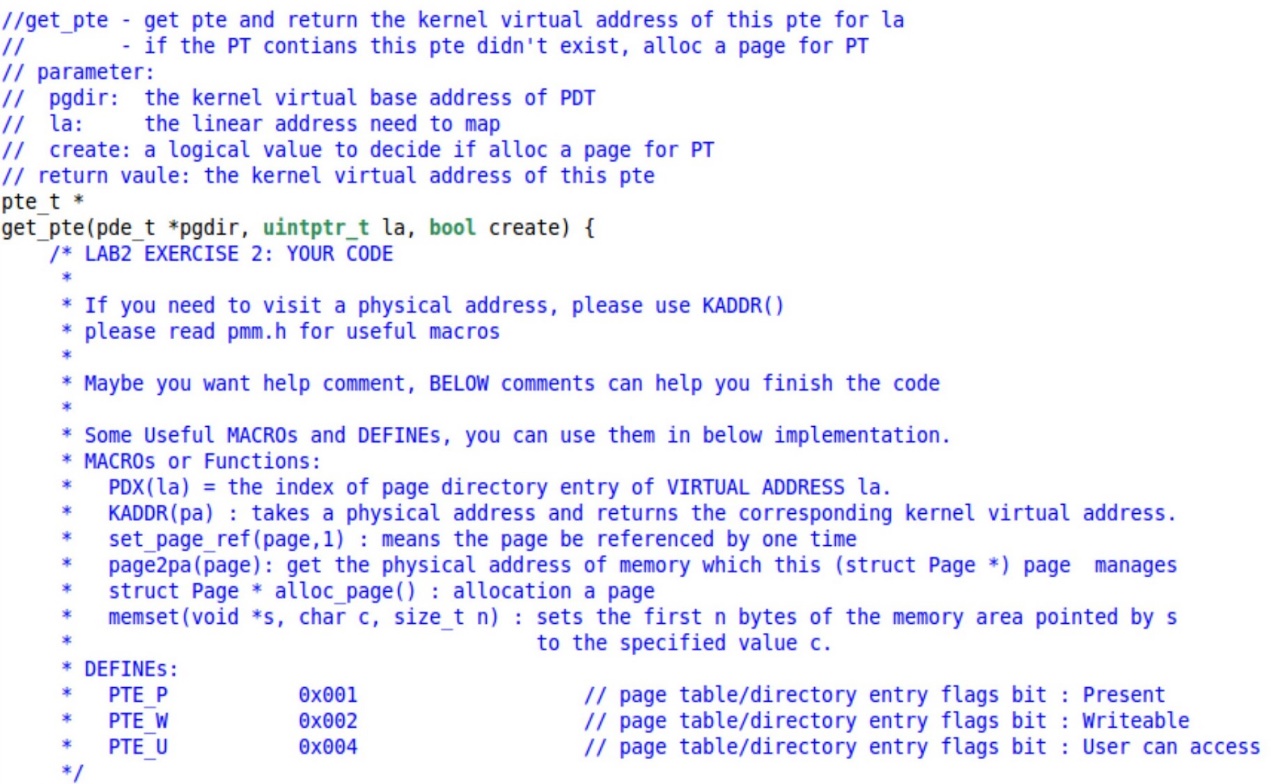
**找到了一张虚拟地址寻址物理地址逻辑的清晰示意图：**



**结合一下mmu.h里对la的注释，寻址的过程就很清晰了：**

****

**有必要再上一下这道题的函数注释。。**

****

1. 设置引用只能设1而不能+1吗？如果是create出来的page应该set，不是create出来的是不是应该+1？

在找

“static inline void set\_page\_ref(struct Page\* page,int val){

page->ref = val;

}”

时发现了一个

“static inline int page\_ref\_inc(struct Page\* page){

page->ref += 1;

return page->ref;

}”

觉得应该用这个吧？

然后又请教了前面提到的同学他友好地提醒了我这个函数内部存在的两次判断：第一次判断PTE\_P是否置位，如果这个页存在就不会进大的body不存在改reference次数的问题。页在以前未被创建的前提下才存在判断需不需要create的问题，i.e.不需要create≠被创建过，而是没被创建也不create^v^ 感觉又是一个愚蠢的问题呢！所以质疑答案的尝试再次失败！答案万岁！然后顺便去确认了一下alloc\_page确实可能返回NULL。。

1. 宏定义PGSIZE也太难找了。。。这要不看答案怎么知道咋设？？不过或许熟悉了各种配置文件是干啥使的就会找了？
2. \*pdep = pa | PTE\_U | PTE\_W | PTE\_P;

页目录项内容 = (页表起始物理地址 &0x0FFF) | PTE\_U | PTE\_W | PTE\_P

把这个新创建的页设置为可读、可写、存在

1. return是真不会写。。

return &((pte\_t \*)KADDR(PDE\_ADDR(\*pdep)))[PTX(la)];

i.KADDR(PDE\_ADDR(\*pdep))部分由页目录项地址得到关联的页表物理地址，再转成虚拟地址

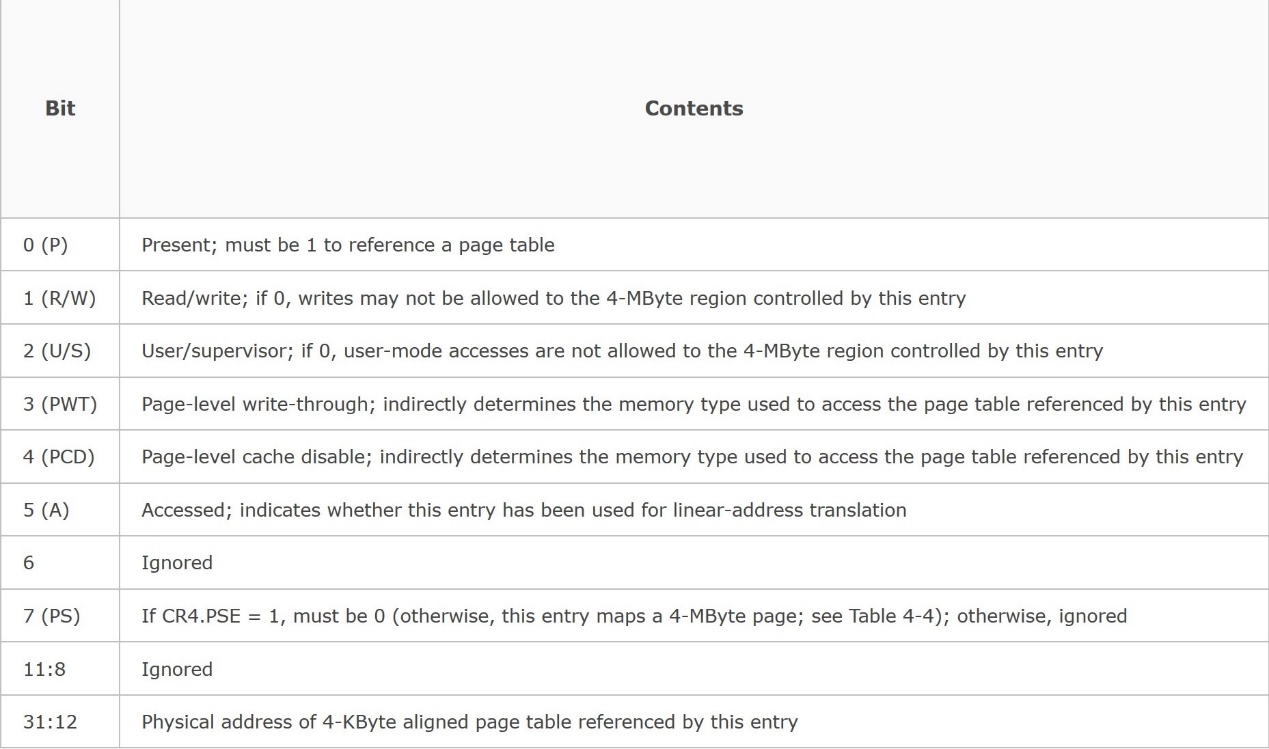
ii. PTX(la)部分返回虚拟地址la的页表项索引

iii.整体返回的是虚拟地址la对应的页表项入口地址

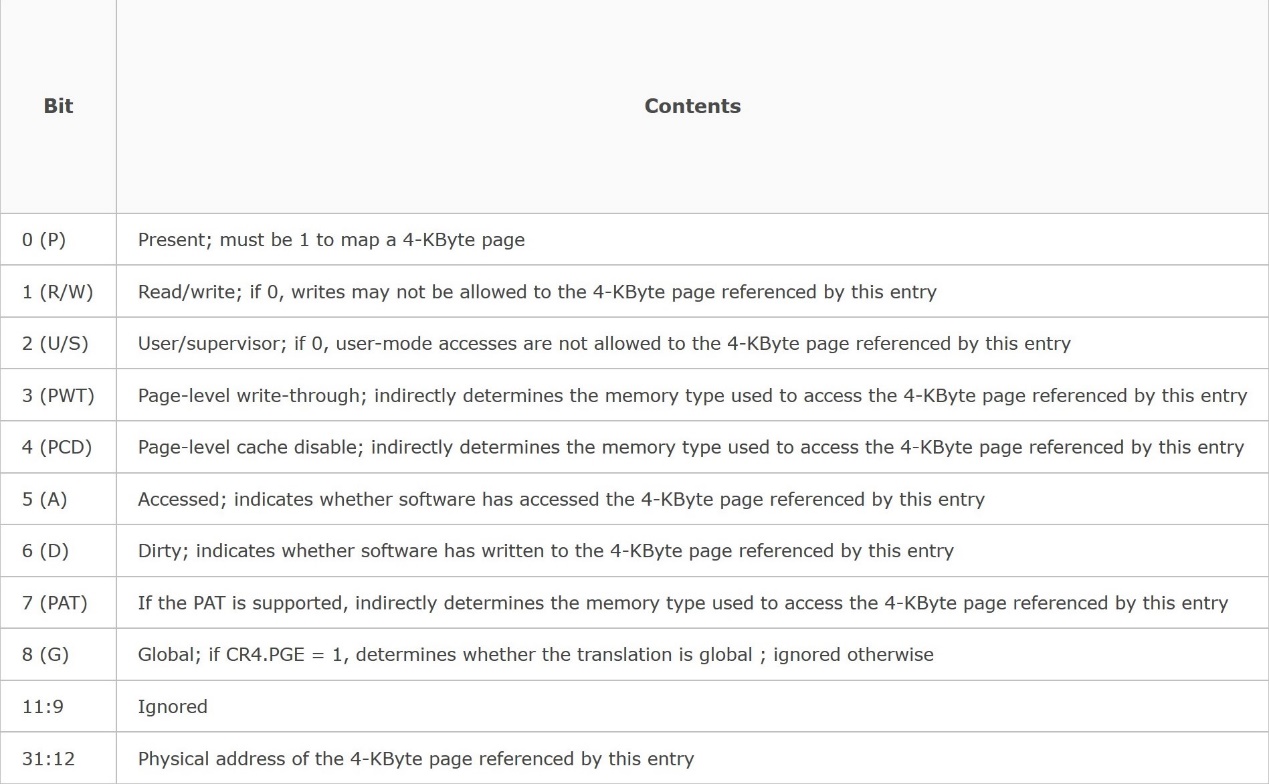
下面是实验指导的问题：

1. PDE和PTE每个组成部分的含义和对ucore的潜在用处。

PDE：



PTE:



1. 访存过程中出现页访问异常时，硬件都要做什么？
   * 1. 将引发页访问异常的地址保存在cr2寄存器中，给出errorCode

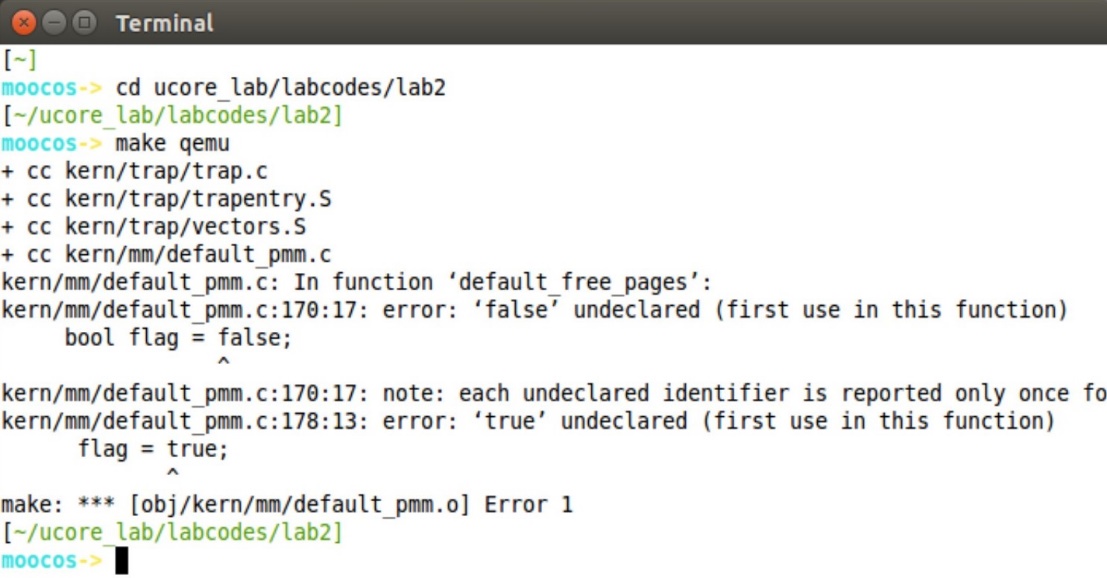
控制寄存器（CR0～CR3）控制和确定处理器的操作模式以及当前执行任务的特性：CR0含有控制处理器操作模式和状态的系统控制标志；CR1保留不用；CR2存储导致页错误的[线性地址](https://baike.baidu.com/item/%E7%BA%BF%E6%80%A7%E5%9C%B0%E5%9D%80/9013682)；CR3存储页目录表[物理内存](https://baike.baidu.com/item/%E7%89%A9%E7%90%86%E5%86%85%E5%AD%98)基地址。

* + 1. OS把错误代码存在trapframe的tf\_err
    2. 中断，调用异常处理例程

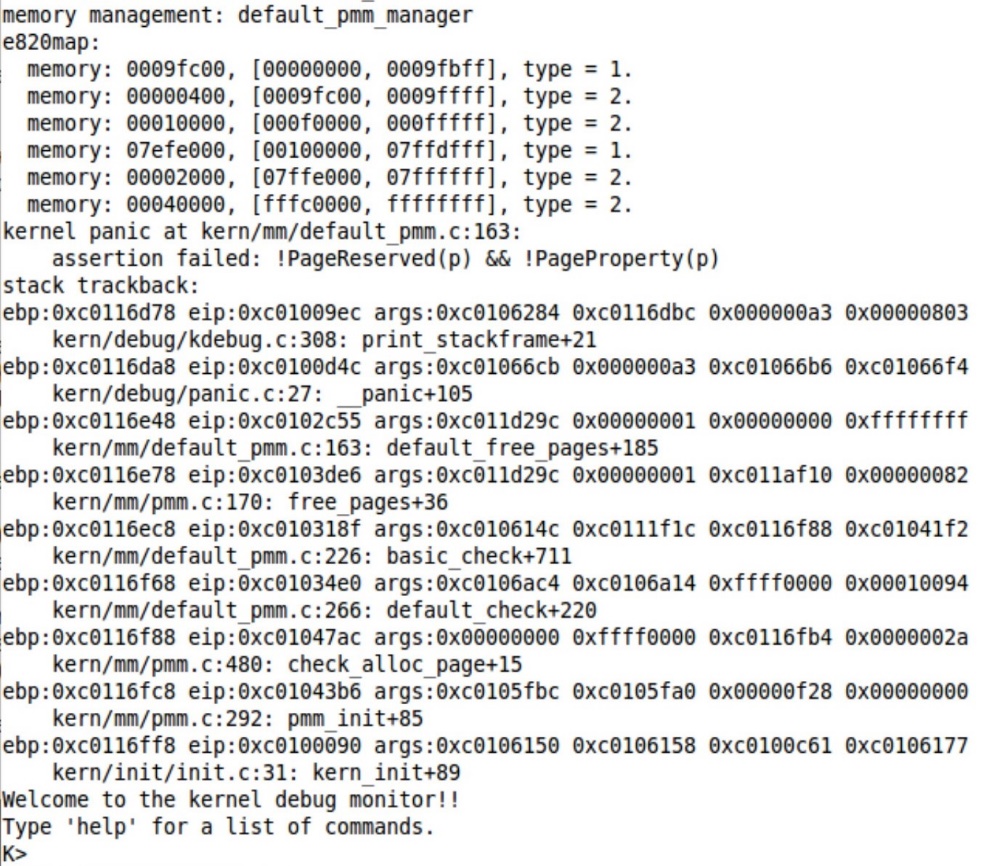
1. **练习3：释放某虚地址所在的页并取消对应二级页表项的映射**

**emmm好像没啥想说的我们直接进入测试**

**make qemu意外发现在ucore里bool型的false和true未定义不能直接用。。：**

****

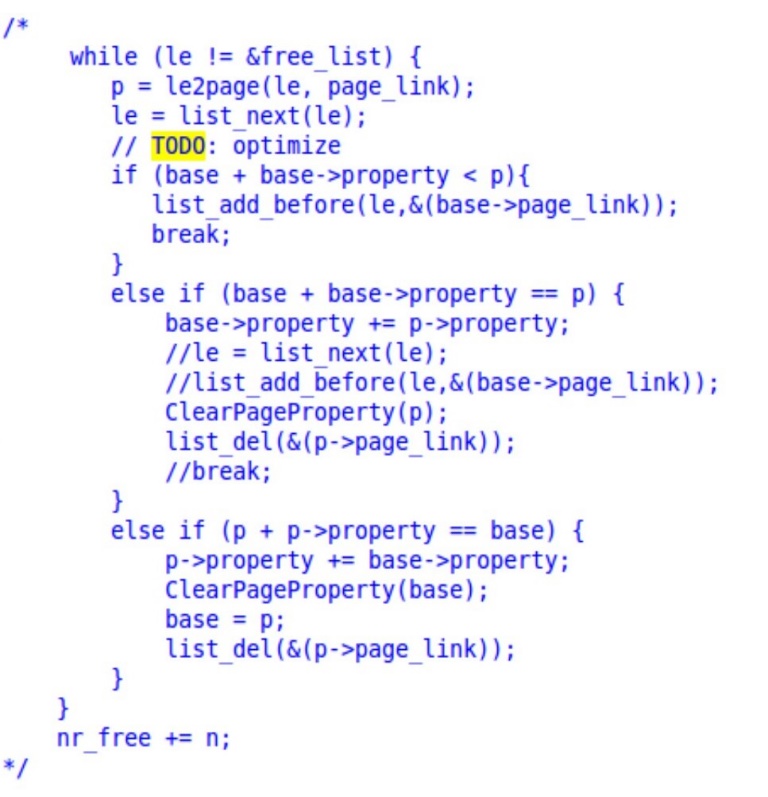
**绝望地发现自己需要debug：**

****

当我把PR和PP两个分开来assert之后发现问题出在PageReerved上。。然后开始反思自己都做了什么。。。想到前面自己加的SetPageReserved，注释掉了这个问题就没了。。或许allocate page只管allocate不管后续这个返回的page是否真的被占用了吗所以不能设置Reserved？

然后又出现了一个bug说assertion failed: (p0 = alloc\_page()) != NULL

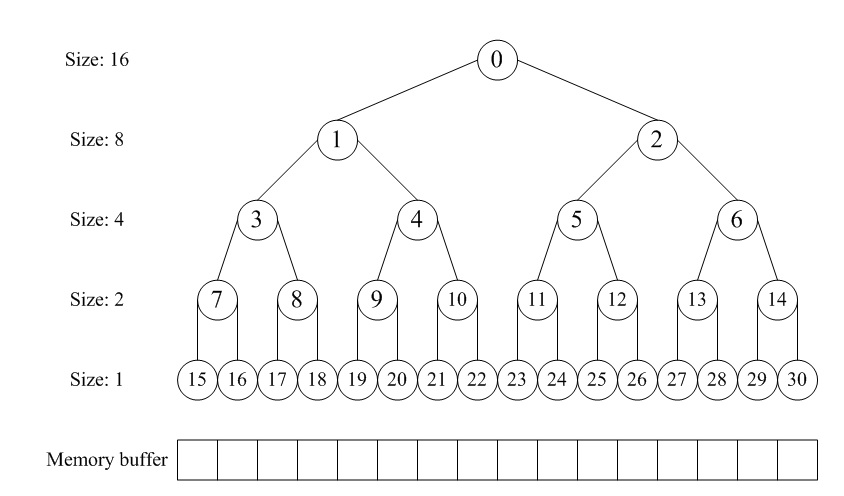
我改了好久，一开始的确实有问题，但最后这个改的版本真的看不出问题在哪！！！我很愤怒！不改了！



1. **Challenge 1：buddy system（伙伴系统）分配算法**

不太会在linux里实现。。。就纸上谈兵一下我的想法吧。。

和给的样例有所异同。相同的是很赞同它二叉树的数据结构选择，不同的是想要在二叉树的基础上维护一个链表。借用一下样例给的图：





假设黑笔断开的位置以下是目前还不需要拆分的内存，维护一个内存各块头部连接而成的双向链表。类似现有的default\_alloc\_page，维护一个指向头部（当前节点15）的指针。

每个节点记录：节点（内存块）的大小；向内存高地址（逆时针）指向的下一内存块；向低地址（顺时针）指向的下一内存块；指向父节点的指针；是否被reserved。

分配时：由低地址向高地址遍历；

回收时：优先尝试向高地址合并，再尝试向低地址合并。相邻内存块不能跨父辈节点合并。

1. **Challenge 2：任意大小的内存单元slub分配算法**