1.查看default_pmm.c会发现，default_init，default_init_memmap，default_alloc_pages， default_free_pages等相关函数均已经实现，但是仔细观察会发现，这几个函数的实现有点问题，所有的链表插入操作都没有注意插入的位置，于是乎，我们来一个个的纠正：

首先是default_init_memmap：
default_init_memmap每次只是复制插入一个节点，在pmm.c的page_init中：

```
    for (i = 0; i < memmap->nr_map; i ++) {
        uint64_t begin = memmap->map[i].addr, end = begin + memmap->map[i].size;
        if (memmap->map[i].type == E820_ARM) {
            if (begin < freemem) {
                begin = freemem;
            }
            if (end > KMEMSIZE) {
                end = KMEMSIZE;
            }
            if (begin < end) {
                begin = ROUNDUP(begin, PGSIZE);
                end = ROUNDDOWN(end, PGSIZE);
                if (begin < end) {
                    init_memmap(pa2page(begin), (end - begin) / PGSIZE);
                }
            }
        }
    }
```

即会按照之前探测出来的内存的顺序，依次调用default_init_memmap，从而构建出整个列表，既然是按照从小到大的顺序插入的，那么我们就要把

```
    list_add(&free_list, &(base->page_link));
```

改成

```
    list_add_before(&free_list, &(base->page_link));
```

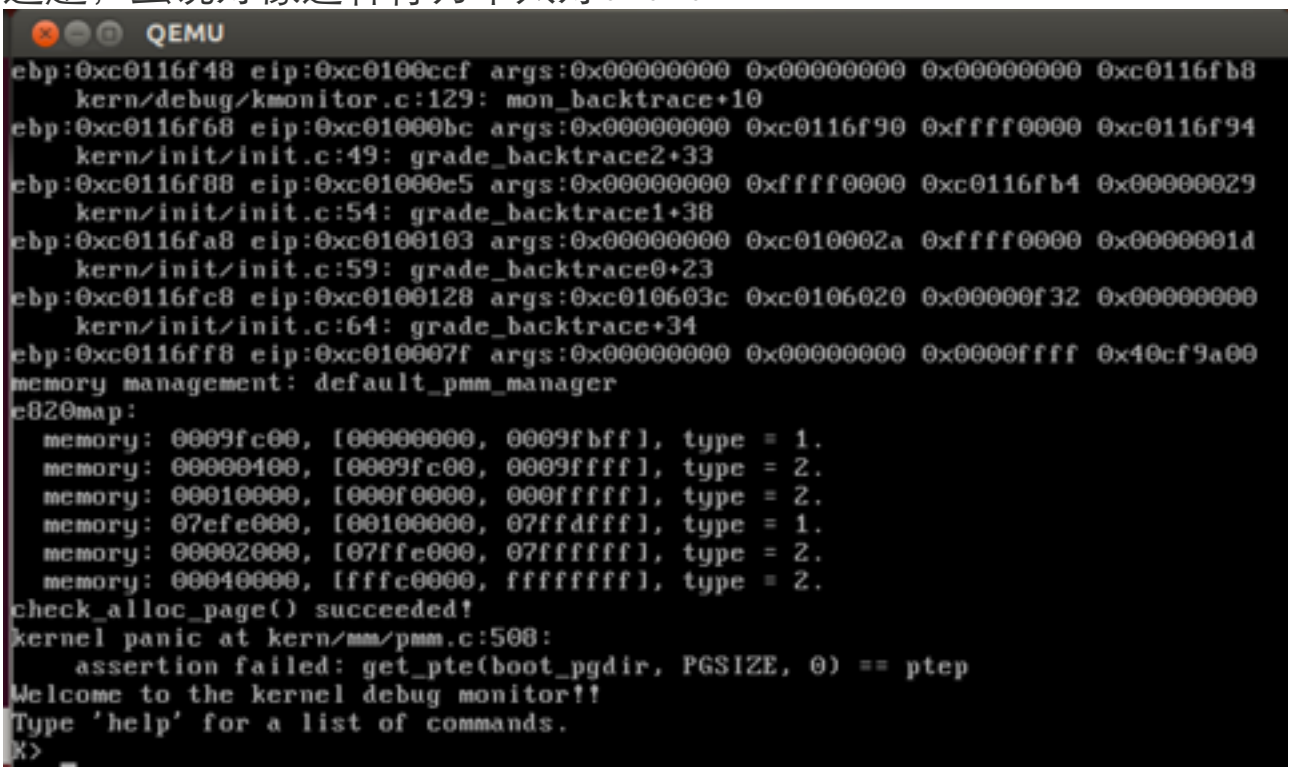
以保证最后的顺序是从小到大的

然后是default_alloc_pages：
它会将找到的节点先删除，再将剩余部分插入，那么我们就记录一下删除前的位置，最后再插入到相同位置即可。

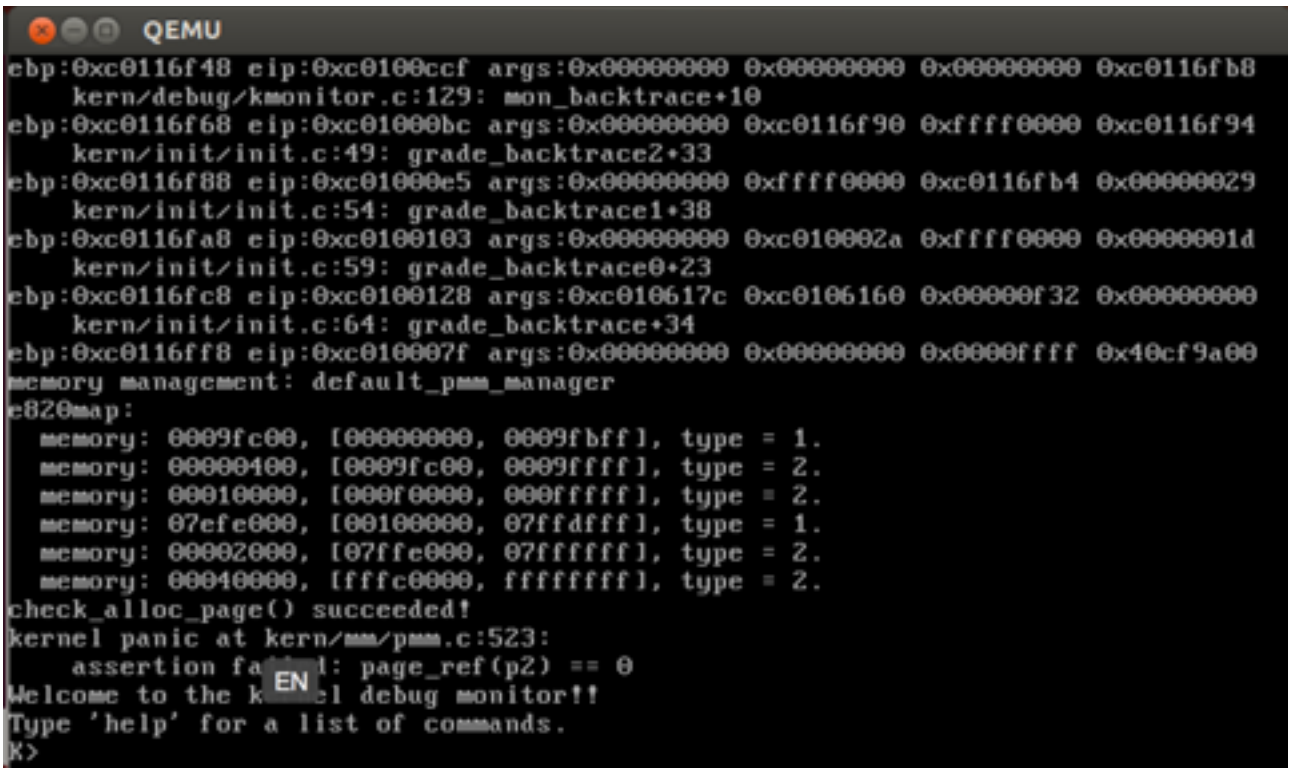
最后是default_free_pages：

free的时候我们需要找到对应的位置，然后插入，而需要合并的情况只会是发生在其对应位置的前面一个或者后面一个，鉴于这样修改对源程序改动较大，此处我们不修改其合并部分的代码，只在其最后找到需要插入的正确位置并插入即可。

PS：话说这题如果直接修改list.h中add的实现似乎可以更方便的解决这题，虽说好像这种行为不太好。。。



2.
首先找出对应的二级页表在页目录表中的指针：
pde_t *pdep = pgdir + PDX(la);
然后检查其PTE_P位以确认该二级页表是否存在，若不存在，则根据create决是否新建。
对于新建的page，由于alloc page的时候并没有设置好引用次数，故需要在此时手动调整ref为1。
对于新建的页，需要将其清0，注意清0时的首地址是该page对应的虚拟地址。
接着要给此页目录表项设置权限为PTE_U | PTE_W | PTE_P。
最后返回需要的二级页表项即可，由于存储在页目录表中的地址是带有PTE_U | PTE_W | PTE_P权限的，故需要在mmu.h中找到相应的宏PDE_ADDR来得到去掉此权限后的正确地址，然后要注意的是在返回的时候调整为虚拟地址。



3.
有了上一题的基础，这一题就显得很简单了，一步步的按照注释来写就好，所有需要的函数定义全都在注释中给出来了。
首先检测PTE_P位来确认要移除的二级页表项是有效的，然后将此页的ref次数减一，如果减为0了要记得对其进行free。
最后将二级页表中此项清0以从二级页表中将此项溢出，并通过tlb_invalidate刷新tlb。