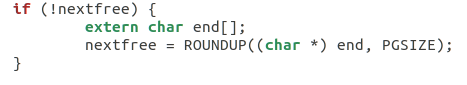
**Lab 2**

**2.1** **Physical Page Management**

这一部分主要是对pmap.c中的几个函数进行补全：boot\_alloc()、mem\_init() 、page\_init()、page\_alloc()、page\_free()



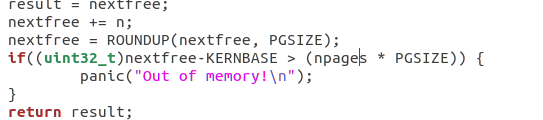
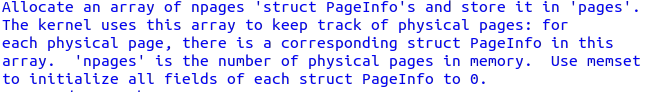
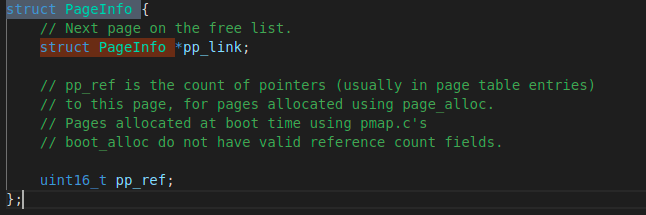
图1

图2

**boot\_alloc()：**需要分配n个字节的空闲内存，如果n为0则直接返回下一个空闲页，如图1所示，其中end指向bss段的末尾，ROUNDUP是用于进行对齐的。我们需要实现的是对n不为0的情况进行分配内存，代码如图2所示，更新result和nextfree再进行判断即可

图3

**mem\_init()：**按照图3说明的提示，给pages分配npages个struct PageInfo大小的内存，再用memset将其初始化为0

图4

**page\_init()：**如图4所示，在memlayout.h中定义的PageInfo结构体，由两个变量组成，pp\_link指向下一个空闲页，pp\_ref指的是对该页面的引用计数，pp\_link为0则表示该页面正在被使用。整个物理内存空间被分成三个部分：

base memory：区域为[PGSIZE, npages\_basemem \* PGSIZE)，这部分是可用的；

IO hole：区域为[IOPHYSMEM, EXTPHYSMEM)，这部分是不可用的；

extended memory：区域为[EXTPHYSMEM, ...)，这部分有些正在被使用，有些没有被使用。

page\_init()这个函数去标记所有空闲的页，更新page\_free\_list，因此代码如图5所示

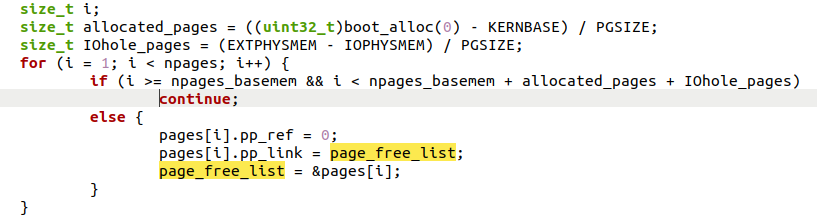
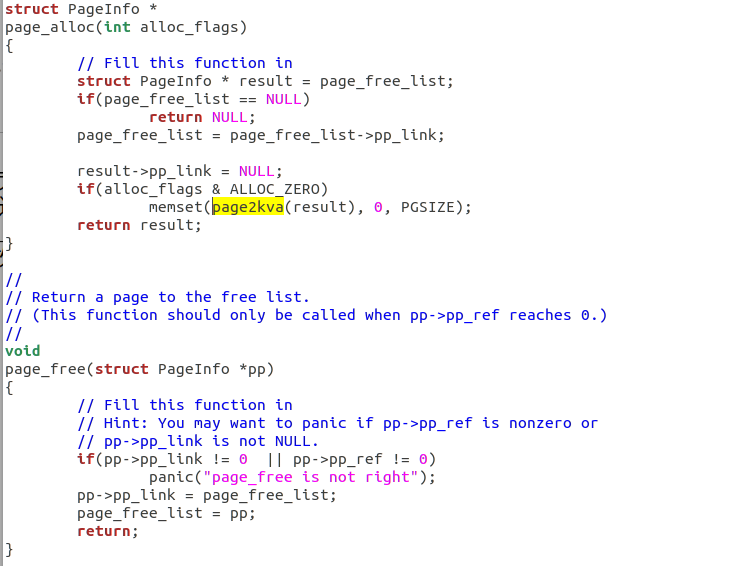
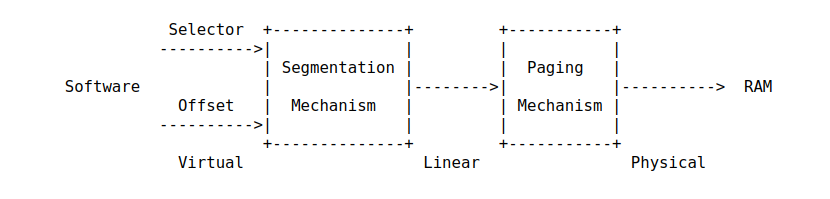


图5

**page\_alloc()和page\_free() ：**这两个是对页的分配和释放函数，整个空闲页面的结构类似与一个链表，而page\_free\_list指向最近的一个可用的空闲页。另外，根据两个函数的注释，page\_alloc()中如果满足(alloc\_flags & ALLOC\_ZERO)条件则将返回的物理地址页都填充0，page\_free()中要对释放的页面的pp\_link和pp\_ref进行判断。代码如图6所示。

图6

**2.2、****Virtual Memory**

 图7

如图7所示，虚拟地址是由两部分组成，一个是段选择子，另一个是段内偏移。线性地址指的是通过段地址转换机构把虚拟地址进行转换之后得到的地址。而物理地址是分页地址转换机构把线性地址进行转换之后得到的真实的内存地址，C语言程序中的指针的值其实是虚拟地址中段内偏移部分的值。

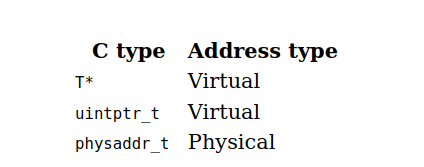


图8

JOS源文件中的地址有两种情况（如图8所示）：

uintptr\_t – 表示虚拟地址

physaddr\_t – 表示物理地址

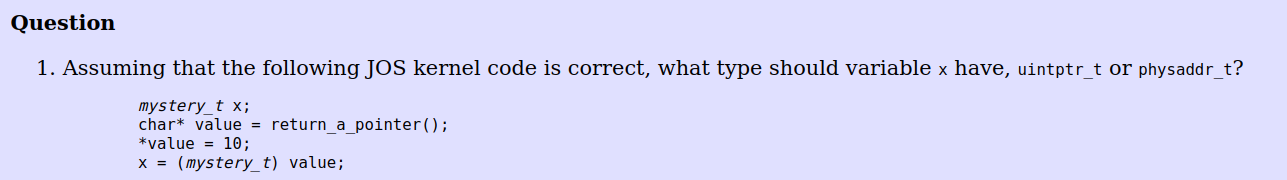
图9

图9中的问题：value是指针，因此x是uintptr\_t类型的值

**Exercise 4. In the file kern/pmap.c, you must implement code for the following functions.**

**pgdir\_walk()**

**boot\_map\_region()**

**page\_lookup()**

**page\_remove()**

**page\_insert()**

**check\_page(), called from mem\_init(), tests your page table management routines. You should make sure it reports success before proceeding.**

1. **pgdir\_walk()：**对于给定的一个页目录表指针pgdir，该函数要返回线性地址va所对应的页表项指针。大致思路是先找到页目录项地址，再通过va得到页表项，其中还需要进行一些判断条件。代码如图10所示

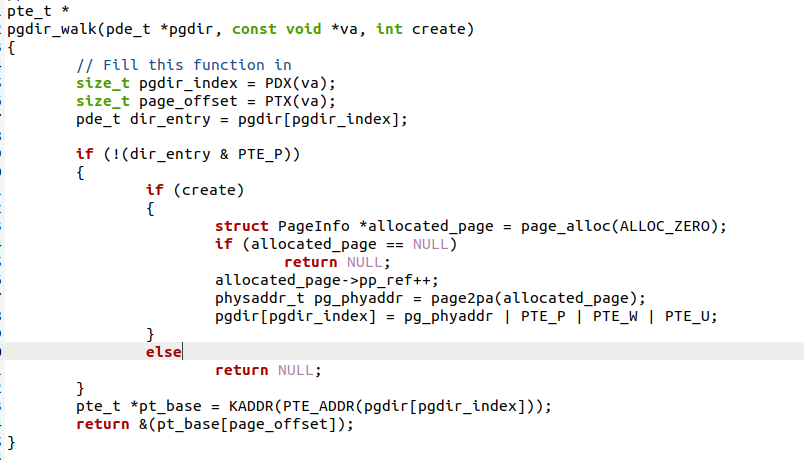
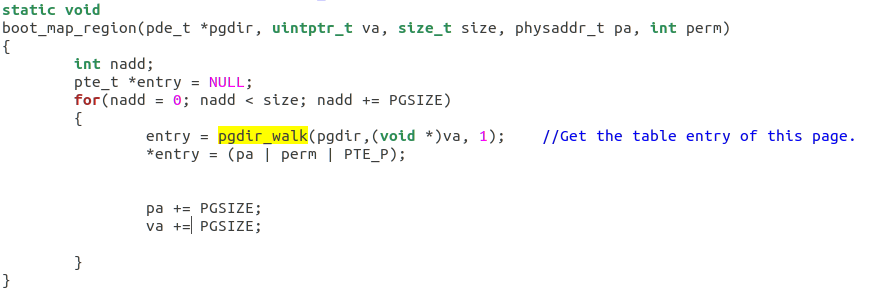


图10

2.boot\_map\_region()：这个函数的作用是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，size是PGSIZE的整数倍，设置循环即可，如图11所示

图11

3.page\_lookup():该函数返回虚拟地址va对应的页，如果参数pte\_store不为0，则将该页面存储到pte\_store中，比较关键的是通过PTE\_ADDR获得页的地址，再通过pa2page获得对应的页，代码如图12所示

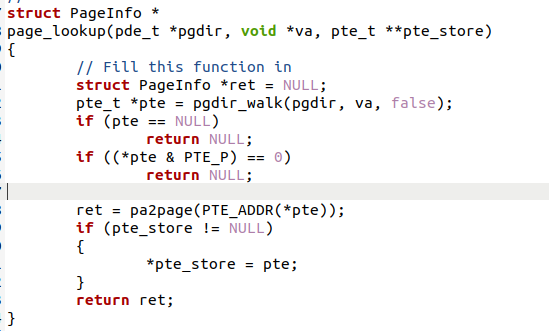


图12

4.page\_remove()：取消虚拟地址va对物理页的映射，需要对物理页的引用计数-1，如果引用计数为0，则释放该页，同时将tlb置为非法的，代码如图13所示

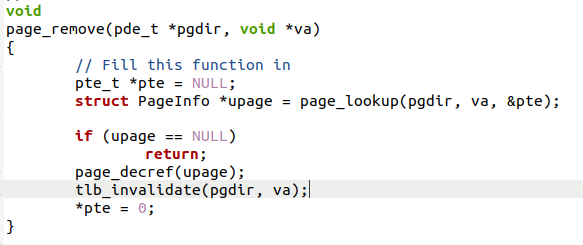


图13

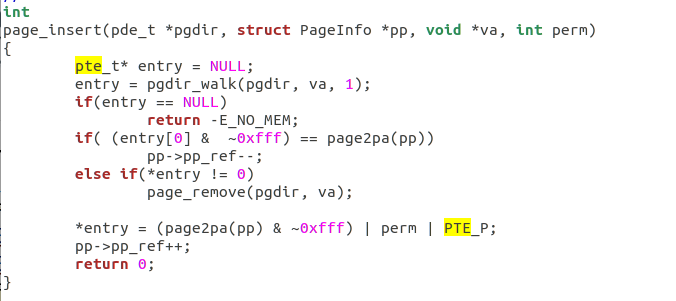
5.page\_insert()：将物理页pp映射到虚拟地址va上，同时将页表项的权限设置为'perm|PTE\_P'，有三种情况：

a.没有物理页映射到va上，直接映射pp到va上即可

b.有物理页映射到va上，但该页不是pp，需要将该页remove再进行映射

c.有物理页映射到va上，且该页是pp，则设置权限后返回即可

代码如图14所示

图14

2.3、Kernel Address Space

Exercise 5 Fill in the missing code in mem\_init() after the call to check\_page().

需要实现的部分是将一些地址范围映射到新的页目录项中，调用之前写过的boot\_map\_region函数即可。

首先是pages数组映射到线性地址UPAGES上，大小为一个PTSIZE。如图15所示

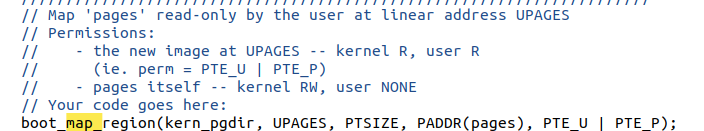


图15

然后是将bootstack的物理地址映射到[KSTACKTOP-KSTKSIZE, KSTACKTOP)上，这部分是内核的堆栈区域，内核空间可以读写，用户没有权限访问，如图16所示

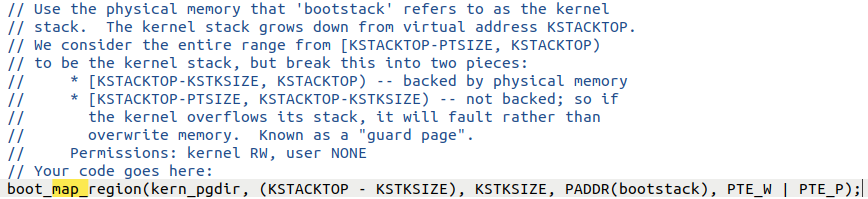


图16

最后是将整个内核空间进行映射，将虚拟地址范围[KERNBASE, 2^32)映射到[0, 2^32 – KERNBASE)，这部分也是内核空间可以读写，用户没有权限访问，如图17

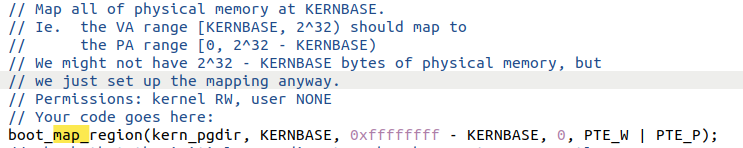


图17

接下来是几个问题：

3.We have placed the kernel and user environment in the same address space. Why will user programs not be able to read or write the kernel's memory? What specific mechanisms protect the kernel memory?

用户程序不能去随意修改内核中的代码，数据，否则可能会破坏内核，造成程序崩溃。操作系统通过分页机制去保护内核地址，将页表项中的User位置0，用户程序就不能访问这个页。

4.What is the maximum amount of physical memory that this operating system can support? Why?

UPAGES有4MB的大小管理pages，所以共有4MB/8B=0.5M个页面，每个页大小4kb，所以一共为2G的物理内存。

5.How much space overhead is there for managing memory, if we actually had the maximum amount of physical memory? How is this overhead broken down?

存放所有的PageInfo需要4MB，存放页目录表kern\_pgdir需要4KB，还需要存放当前的页表2MB。所以总的开销就是6MB + 4KB。

6.Revisit the page table setup in kern/entry.S and kern/entrypgdir.c. Immediately after we turn on paging, EIP is still a low number (a little over 1MB). At what point do we transition to running at an EIP above KERNBASE? What makes it possible for us to continue executing at a low EIP between when we enable paging and when we begin running at an EIP above KERNBASE? Why is this transition necessary?

在entry.S文件中有一个指令 jmp \*%eax，这个指令要完成跳转，就会重新设置EIP的值，把它设置为寄存器eax中的值，而这个值是大于KERNBASE的，所以就完成了EIP从小的值到大于KERNBASE的值的转换。在entry\_pgdir这个页表中，也把虚拟地址空间[0, 4MB)映射到物理地址空间[0, 4MB)上，所以当访问位于[0, 4MB)之间的虚拟地址时，可以把它们转换为物理地址。

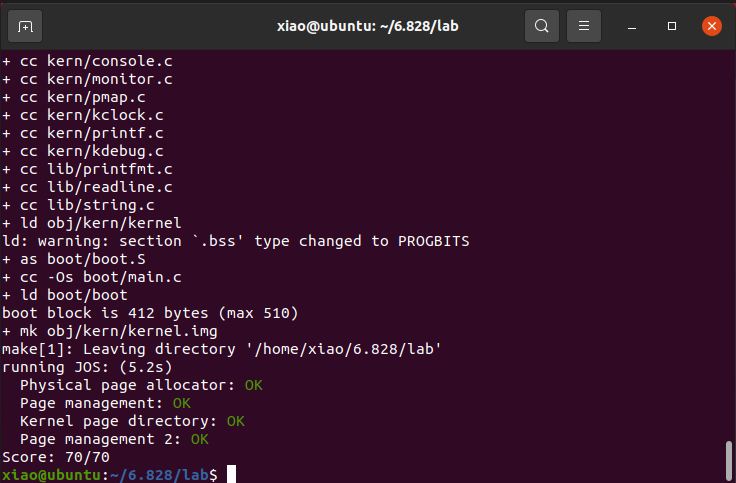


图18

图18是最后的结果