# Lab2 Memory Management

**Part 1: Physical Page Management**

1 allocator

Record free / allocated

Processes sharing each allocated pages

Routine to allocate and free pages

2 vm

Va -> pa

Set up MMU

Boot\_alloc

n大于0时，allocate相应大小（4K对齐后）的空间，并利用kaddr、paddr两个函数进行输入检查，返回kva。n等于0时，返回下一页的地址，而不allocate任何空间。N小于0时，返回null。

Mem\_init

申请n个Page大小的空间，并初始化为0。

page\_init

Physical page 0, IO hole标记为1（意思是永远有一个进程在引用该页），即永远不能被free和allocated。剩余的base memory和extented memory标记为0，且加入free list准备被allocate。此外，已经被allocated的页，例如pages数组所占的页，应当也被标记为allocated。

Alloc\_page

如果没有空页则返回null，如果flag要求清零，则将新allocate的页用memset设为全零，然后将free list的第一页返回，free list指向下一个空页。

**Part 2: Virtual Memory**

Question1

此处应该为uintptr\_t，因为对于程序而言只可见虚拟地址

Pgdir\_walk

用拿到的va的PDX段索引相应的页，若不存在且创建，则allocate一个新的页，修改相应参数，将该页的物理地址记录在PDE中。

然后根据va的PTX段在PDE指向的页中找到对应的PTE，将其指针从物理地址（即记录的值除去标记位）转换为内核虚拟地址返回。

Boot\_map\_region

遍历size覆盖的页的数目，用区间内的va找到对应的PTE，将对应的物理地址写入。

Page\_lookup

使用va获得对应的pte，将它指向的物理地址（记录的地址用pa2page删掉后面的标记位）转换为页返回。

根据注释，如果pte\_store不为0，还需要将该pte记录在store中。即该函数返回值为需要的页，pte中保存了对应的pte。

Page\_remove

调用lookup找到对应的页，将对应的page的ref减一，如果为零则free（在函数page\_decref中）。将对应的tlb禁用。将对应的pte置为0.

Page\_insert

用目标va知道到对应的pte（pgdir\_walk)，如果不存在则返回错误。然后将page的ref加一，如果va对应的pte已经存在，就将它对应的page remove，然后将新的page保存其中。

Exercise 5

使用boot\_map\_region将Upages之上的虚拟地址与对应的物理地址映射。

Question2

Entry --- Base VA --- Points to

1023 --- 4G - 4M = 0xffc00000 --- Page table for top 4MB of phys memory

960 ~ 1022 --- 0xfxx00000 -- Page table for phys memory above kernbase

958 --- 0xefbf8000 --- kernal stack

957 --- 0xef800000 --- Cur. page table

956 --- 0xef000000 --- R0 Pages

Question3

pde和pte中存在权限位，地址翻译的时候MMU会进行检查，当前mode是否用对应的访问权限。

Question4

最大可支持4MB UPAGES，最多可以有4MB / sizeof(struct Page) \* 4K = 2G 物理空间， 由Virtual Memory Map的设计决定。

Question5

2G 物理内存

= 2G / 4K = 512K 个PTE （页）

= 512K / （PGSIZE / sizeof(pte\_t)) = 512 个pte page

< 1 个 PDE

共需要512K个 struct page保存page信息，占用512K \* 8B = 4M

共需要512页保存pte，占用512 \* 4K = 2M，

需要一页保存PDE，占用4K。

Question6

在执行如下汇编，jmp的时候修改了eip

mov $relocated, %eax

jmp \*%eax

在low eip可以继续执行是因为，entry\_pgdir也被映射到了0~4M的虚拟空间，仍然可以正常访问。

修改eip必要的原因是，之后加载了kern\_pgdir，而entry\_pgdir不会再被使用。代码使用的都是高处的虚拟地址，如果使用非常低的地址用来翻译成物理地址的话，会找不到对应的物理地址。