操作系统

练习**1**：实现 **first-fit** 连续物理内存分配算法（需要编程）

在实现first fit 内存分配算法的回收函数时，要考虑地址连续的空闲块之间的合并操作。提示:

在建立空闲页块链表时，需要按照空闲页块起始地址来排序，形成一个有序的链表。可能会

修改default\_pmm.c中的default\_init，default\_init\_memmap，default\_alloc\_pages，

default\_free\_pages等相关函数。请仔细查看和理解default\_pmm.c中的注释。

答：

default\_init不用修改

具体代码

static void default\_init(void) {

list\_init(&free\_list);

nr\_free = 0;

}

default\_init\_memmap ：用于构建空闲页链表

具体修改后：

static void default\_init\_memmap(struct Page \*base, size\_t n) {

assert(n > 0);

struct Page \*p = base;

for (; p != base + n; p ++) {

assert(PageReserved(p));

p->flags = 0；

SetPageProperty(p);

p->property = 0;

set\_page\_ref(p, 0);//清空引用

list\_add\_before(&free\_list, &(p->page\_link));

}

nr\_free += n;

base->property=n;

}

default\_alloc\_pages ：用于为进程分配空闲页；

default\_init\_memmap，主要就是从空闲页块的链表中去遍历，找到第一块大小大于n的块，然后分配出来，把它从空闲页链表中除去，然后如果有多余的，把分完剩下的部分再次加入会空闲页链表中即可。

练习**2**：实现寻找虚拟地址对应的页表项（需要编程）

通过设置页表和对应的页表项，可建立虚拟内存地址和物理内存地址的对应关系。其中的

get\_pte函数是设置页表项环节中的一个重要步骤。此函数找到一个虚地址对应的二级页表项

的内核虚地址，如果此二级页表项不存在，则分配一个包含此项的二级页表。本练习需要补

全get\_pte函数 in kern/mm/pmm.c，实现其功能。请仔细查看和理解get\_pte函数中的注释。

答：

get\_pte(pde\_t \*pgdir, uintptr\_t la, bool create) {

pde\_t \*pdep = &pgdir[PDX(la)];

if (!(\*pdep & PTE\_P)) {

struct Page \*page;

if (!create || (page = alloc\_page()) == NULL) { // (3) check if creating is needed, then alloc page for page table

return NULL;

}

set\_page\_ref(page, 1);

uintptr\_t pa = page2pa(page);

memset(KADDR(pa), 0, PGSIZE);

\*pdep = pa | PTE\_U | PTE\_W | PTE\_P;

}

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

}

pde\_t 全称为page directory entry，也就是一级页表的表项（注意：pgdir实际不是表项，而是一级页表本身，pgdir给出页表起始地址。）

pte\_t 全称为page table entry，表示二级页表的表项。

uintptr\_t 表示为线性地址，由于段式管理只做直接映射，所以它也是逻辑地址。

PTE\_U: 位3，表示用户态的软件可以读取对应地址的物理内存页内容

PTE\_W: 位2，表示物理内存页内容可写

PTE\_P: 位1，表示物理内存页存在

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（需要编程）

当释放一个包含某虚地址的物理内存页时，需要让对应此物理内存页的管理数据结构Page做

相关的清除处理，使得此物理内存页成为空闲；另外还需把表示虚地址与物理地址对应关系

的二级页表项清除。请仔细查看和理解page\_remove\_pte函数中的注释。为此，需要补全在

kern/mm/pmm.c中的page\_remove\_pte函数。

答：

static inline void

page\_remove\_pte(pde\_t \*pgdir, uintptr\_t la, pte\_t \*ptep) {

if (\*ptep & PTE\_P) { //判断页表中该表项是否存在

struct Page \*page = pte2page(\*ptep);

if (page\_ref\_dec(page) == 0) { //判断是否只被引用了一次

free\_page(page); //如果只被引用了一次，那么可以释放掉此页

}

\*ptep = 0; //如果被多次引用，则不能释放此页，只用释放二级页表的表项

tlb\_invalidate(pgdir, la); //更新页表

}

}

