

实验 2：物理内存管理

黄伟健 P14206019

练习 0：填写已有实验

答：利用 Ubuntu 系统中的 meld 工具，针对 Lab1 中曾经更改的几个文件，分别对照更正既有代码。

练习 1：实现 first-fit 连续物理内存分配算法

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 defaultpmm.c 中的 defaultinit, defaultinitmemmap, defaultallocpages, defaultfreepages 等相关函数。请仔细查看和理解 default_pmm.c 中的注释。

```
#define free_list (free_area.free_list)
#define nr_free (free_area.nr_free)
/* 初始化 free_list 同时为 nr_free 置零 */
static void
default_init(void) {
    list_init(&free_list); //free_list 记录空的块
    nr_free = 0; //nr_free 为当前空块总数
}
/*函数用来初始化一块大小为 PGSIZE*n 的物理内存区域
调用过程依次为 kern_init --> pmm_init-->page_init-->init_memmap-->
pmm_manager->init_memmap
根据 page_init 函数传参，具体为某个连续地址的空闲块的起始页和页个数，建立空闲块的双向链表*/
static void
default_init_memmap(struct Page *base, size_t n) {
    assert(n > 0); //确认 n 的合理性
    struct Page *p = base; //页指针
    for (; p != base + n; p++) {
        assert(PageReserved(p)); //有效页
        p->flags = 0; //有关有效性标记，参见 pmm.c
        SetPageProperty(p); //设置有效
        p->property = 0; //当前页为空且不为第一页则 property 为零
        set_page_ref(p, 0); //由于 p 此时是 free 状态，ref 也应置零
        list_add_before(&free_list, &(p->page_link)); //链接 free_list
    }

    base->property = n; //以 base 为首用 property 表示内存块 page 总数
    nr_free += n; //更新物理内存空闲块总数
}
/* 函数用来实现 first fit，输入所需的块大小 n，返回相应第一个 page 的地址 */
static struct Page *
default_alloc_pages(size_t n) {
    assert(n > 0); //检查 n 的合理性
    if (n > nr_free) { //若 n 太大超过现存空闲块总数，则返回代表无法满足的空指针
        return NULL;
    }
    list_entry_t *le, *len;
    le = &free_list;
    while((le=list_next(le)) != &free_list) { //依次遍历 free_list
        struct Page *p = le2page(le, page_link); //得到对应页指针
        if(p->property >= n){ //检查是否是足够大的块
            int i;
            for(i=0;i<n;i++){ //设置已分配的内存空间
                len = list_next(le);
```

```

struct Page *pp = le2page(le, page_link);
SetPageReserved(pp);
ClearPageProperty(pp);
/* 设置 PG_reserve=1, PG_property=0 */
list_del(le); //删除已申请的页
le = len;
}
if(p->property>n){ //若仍有剩余则将剩余块加回去
(le2page(le,page_link))->property = p->property - n;
}
ClearPageProperty(p);
SetPageReserved(p);
nr_free -= n; //更新空块总数
return p; //返回目标指针
}
}
return NULL; //遍历后找不到符合条件的块则返回空指针
}
/* 函数参数是指向一个 page 物理地址的指针, 以及需要 free 的 pages 的个数, 主要功能是将以
指针 page 开始的 n 块 page 的内存释放, 并将其加入到 free_list 中*/
static void
default_free_pages(struct Page *base, size_t n) {
assert(n > 0); //检查 n 的合理性
assert(PageReserved(base)); //检查 base 的合理性

list_entry_t *le = &free_list; //得到 free_list 的表头
struct Page *p; //找到插入位置的页指针
while((le=list_next(le)) != &free_list) { //从高到低遍历找到合适的插入位置
p = le2page(le, page_link); //得到对应页的指针
if(p>base){
break;
}
}
//插入链表中并更新相关标记, 如 p->property 等
for(p=base;p<base+n;p++){
list_add_before(le, &(p->page_link));
}
base->flags = 0; //重置 p->ref, p->flags
set_page_ref(base, 0);
ClearPageProperty(base);
SetPageProperty(base);
/* 设置 PG_reserve=1, PG_property=0 */
base->property = n; //设置页大小
p = le2page(le,page_link) ;
if( base+n == p ){
base->property += p->property;
p->property = 0;
}
le = list_prev(&(base->page_link)); //合并块
p = le2page(le, page_link);
if(le!=&free_list && p==base-1){ //检查是否块之间是否可以合并
while(le!=&free_list){
if(p->property){
p->property += base->property;
base->property = 0;
break;
}
le = list_prev(le);
p = le2page(le,page_link);
}
}
nr_free += n; //增加空块总数
return ;

```

}

练习 2：实现寻找虚拟地址对应的页表项

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 `getpte` 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 `getpte` 函数 in `kern/mm/pmm.c`，实现其功能。请仔细查看和理解 `getpte` 函数中的注释。`getpte` 函数的调用关系图如下所示：

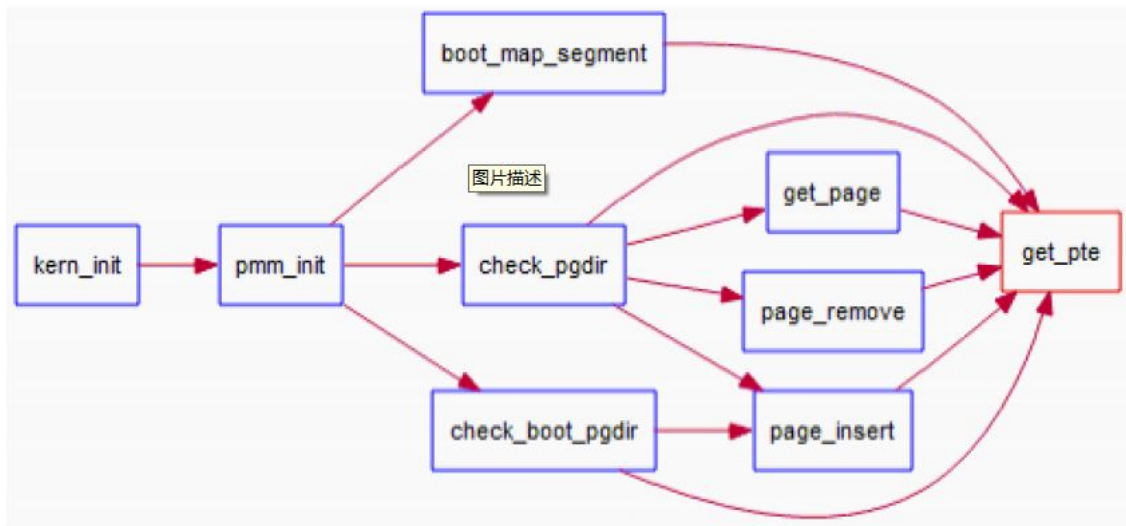


图 1 `getpte` 函数的调用关系图

```
//get_pte - get pte and return the kernel virtual address of this pte for
// la
// - if the PT contains this pte didn't exist, alloc a page for PT
// parameter:
// pgdir: the kernel virtual base address of PDT
// la: the linear address need to map
// create: a logical value to decide if alloc a page for PT
// return vaule: the kernel virtual address of this pte
pte_t *
get_pte(pde_t *pgdir, uintptr_t la, bool create) {
/* LAB2 EXERCISE 2: 12307130244
*
* If you need to visit a physical address, please use KADDR()
* please read pmm.h for useful macros
*
* Maybe you want help comment, BELOW comments can help you finish the code
*
* Some Useful MACROs and DEFINES, you can use them in below implementation.
* MACROs or Functions:
* PDX(la) = the index of page directory entry of VIRTUAL ADDRESS la.
* KADDR(pa) : takes a physical address and returns the corresponding kernel
virtual address.
* set_page_ref(page,1) : means the page be referenced by one time
* page2pa(page) : get the physical address of memory which this (struct Page
*) page manages
* struct Page * alloc_page() : allocation a page
* memset(void *s, char c, size_t n) : sets the first n bytes of the memory
area pointed by s
* to the specified value c.
* DEFINES:
* PTE_P 0x001 // page table/directory entry flags bit : Present
* PTE_W 0x002 // page table/directory entry flags bit : Writeable
* PTE_U 0x004 // page table/directory entry flags bit : User can access
*/
#if 0
```

```

pde_t *pdep = NULL; // (1) find page directory entry
if (0) { // (2) check if entry is not present
// (3) check if creating is needed, then alloc page for page table
// CAUTION: this page is used for page table, not for common data page
// (4) set page reference
uintptr_t pa = 0; // (5) get linear address of page
// (6) clear page content using memset
// (7) set page directory entry's permission
}
return NULL; // (8) return page table entry
#endif

```

找到一个虚地址对应的二级页表项的内核虚地址，若不存在则分配一个包含此项的二级页表

```

pde_t *pdep = &pgdir[PDX(la)]; //寻找一级页表项，获得相应指向页目录表项的指针
if (!(*pdep & PTE_P)) { //检查该页目录表项对应的 page 是否存在
struct Page *page;
if (!create || (page = alloc_page()) == NULL) { //检查是否需要建立二级页表，若
不存在则分配空间
return NULL;
}
set_page_ref(page, 1); //设置 page_ref
uintptr_t pa = page2pa(page); //获取线性地址，即物理地址
memset(KADDR(pa), 0, PGSIZE); //没有映射故需清空新申请的表，初始化该 page 所有位置
*pdep = pa | PTE_U | PTE_W | PTE_P; //设置控制页，将页表的物理地址处理后赋值给目
录表项
}
return &((pte_t *)KADDR(PDE_ADDR(*pdep)))[PTX(la)]; //返回二级页表项
/* *pdep 为 pdep 所指向的页目录表内容，存储了所指向的页表首地址及控制信息
PDE_ADDR(*pdep) 为所指向页表首物理地址，KADDR 获得了页表首虚拟地址
通过强制类型类型转换 (pte_t *) 强制转换成执行页表的首地址
通过 [PTX(la)] 索引获得对应 page 中的 Page table entry
最后利用 & 操作符取其地址即为所求的 pte */
}

```

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

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 `pagerremovepte` 函数中的注释。为此，需要补全在 `kern/mm/pmm.c` 中的 `pagerremovepte` 函数。`pagerremovepte` 函数的调用关系图如下所示：

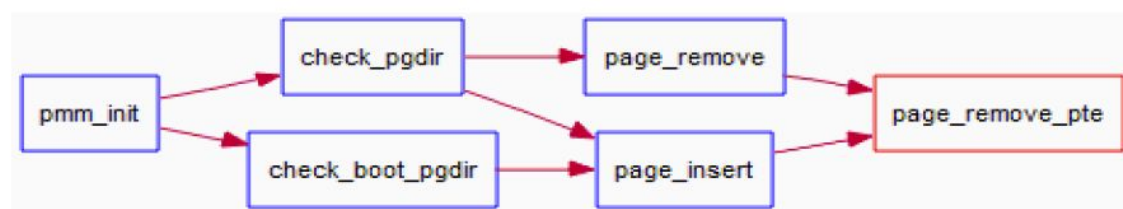


图 2 `pagerremovepte` 函数的调用关系图

```

//page_remove_pte - free an Page struct which is related linear address la
// - and clean(invalidate) pte which is related linear address la
//note: PT is changed, so the TLB need to be invalidate
/* 释放一个包含虚地址的物理内存页时，需要将管理此物理内存页的数据结构 page 做相应清除处
理，使得物理内存页变为空闲状态，另外还需要将记录虚地址与物理地址对应的二级页表项清除*/
static inline void
page_remove_pte(pde_t *pgdir, uintptr_t la, pte_t *ptep) {
/* LAB2 EXERCISE 3: 12307130244
*
* Please check if ptep is valid, and tlb must be manually updated if mapping
is updated

```

```

*
* Maybe you want help comment, BELOW comments can help you finish the code
*
* Some Useful MACROs and DEFINES, you can use them in below implementation.
* MACROs or Functions:
* struct Page *page pte2page(*ptep): get the according page from the value
of a ptep
* free_page : free a page
* page_ref_dec(page) : decrease page->ref. NOTICE: ff page->ref == 0 , then
this page should be free.
* tlb_invalidate(pde_t *pgdir, uintptr_t la) : Invalidate a TLB entry, but
only if the page tables being
* edited are the ones currently in use by the processor.
* DEFINES:
* PTE_P 0x001 // page table/directory entry flags bit : Present
*/
#if 0
if (0) { //(1) check if page directory is present
struct Page *page = NULL; //(2) find corresponding page to pte
//(3) decrease page reference
//(4) and free this page when page reference reaches 0
//(5) clear second page table entry
//(6) flush tlb
}
#endif
if (*ptep & PTE_P) { //检测对应二级页表项是否存在
struct Page *page = pte2page(*ptep); //搜索相应一级页
if (page_ref_dec(page) == 0) { //page->ref==1
free_page(page); //释放页
}
*ptep = 0; //清除二级页表项指针, 将 ptep 所指向内容清空
tlb_invalidate(pgdir, la); //删除 tlb 中的缓冲项
}
}

```