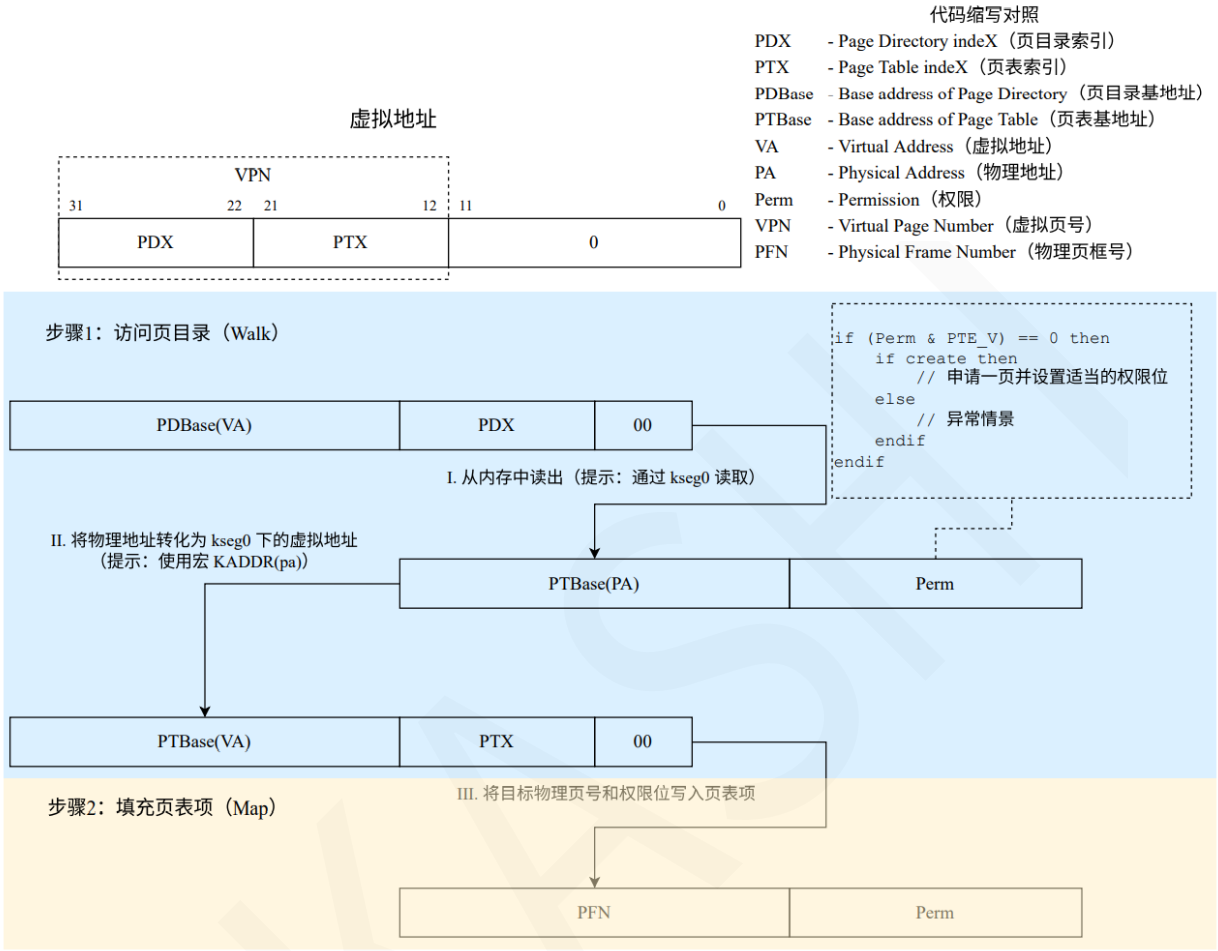
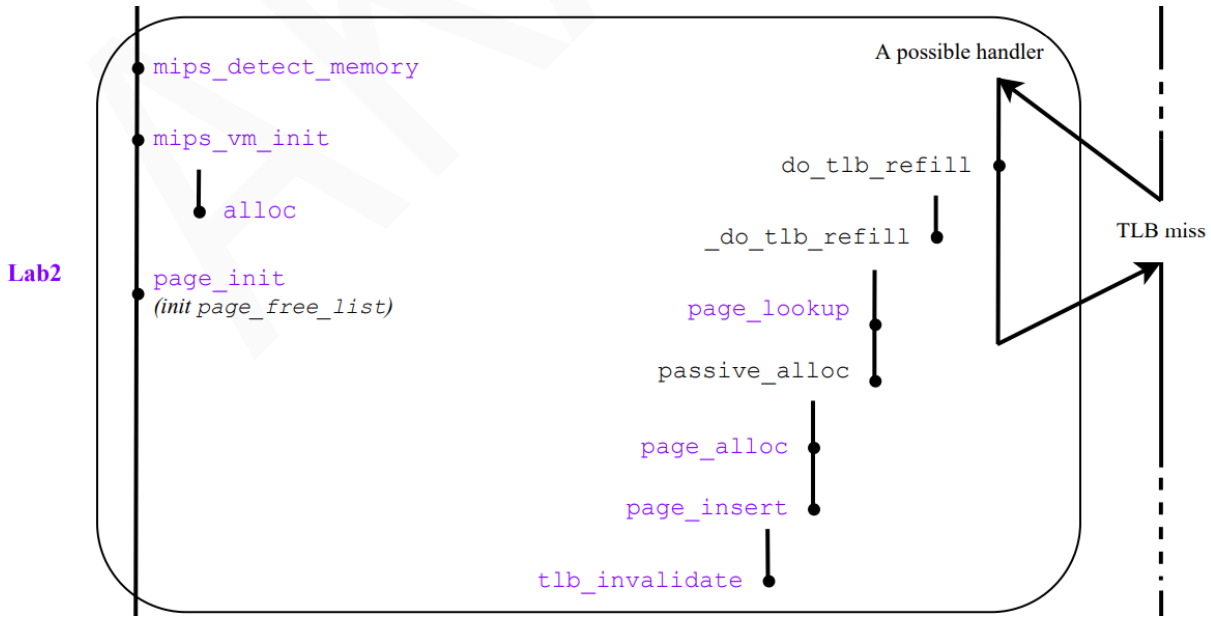
内存管理

页式内存管理


代码缩写对照
PDX - Page Directory indeX（页目录索引）
PTX - Page Table indeX（页表索引）
PDBase - Base address of Page Directory（页目录基地址）
PTBase - Base address of Page Table（页表基地址）
VA - Virtual Address（虚拟地址）
PA - Physical Address（物理地址）
Perm - Permission（权限）
VPN - Virtual Page Number（虚拟页号）
PFN - Physical Frame Number（物理页框号）
虚拟地址
VPN
31
22
21
12
11
0
PDX
PTX
0
步骤1：访问页目录（Walk）
if (Perm & PTE_V) == 0 then
if create then
// 申请一页并设置适当的权限位
else
// 异常情景
endif
endif
PDBase(VA)
PDX
00
I. 从内存中读出（提示：通过 kseg0 读取）
II. 将物理地址转化为 kseg0 下的虚拟地址
（提示：使用宏 KADDR(pa)）
PTBase(PA)
Perm
PTBase(VA)
PTX
00
步骤2：填充页表项（Map）
III. 将目标物理页号和权限位写入页表项
PFN
Perm


Lab2
mips_detect_memory
mips_vm_init
alloc
page_init
(init page_free_list)
A possible handler
do_tlb_refill
_do_tlb_refill
page_lookup
passive_alloc
page_alloc
page_insert
tlb_invalidate
TLB miss

定义与声明

```
//include/pmap.h
LIST_HEAD(Page_list, Page);
typedef LIST_ENTRY(Page) Page_LIST_entry_t;

struct Page {
    Page_LIST_entry_t pp_link; /* free list link */
    u_short pp_ref; /* 页引用次数 */
};
```

全局变量

```
static u_long memsize; // 物理内存大小
u_long npage; // 物理页框数，包括了全部的物理空间

Pde *cur_pgdir; // 当前页目录的内核虚地址

struct Page *pages; // 指向页控制块数组，该数组在内核堆上
static u_long freemem; // 已使用的内存，内核虚地址

struct Page_list page_free_list; // 空闲页的页控制块队列
```

函数

```
//kern/pmap.c
// 获取物理地址空间大小，并计算页框数
void mips_detect_memory(void);
// 调用 alloc 申请页控制块的内存空间
void mips_vm_init(void);
// 各种初始化
void mips_init(void);
/* 初始化页式内存管理，在此之后 alloc 将被弃用
初始化空闲页控制块链表
将 freemem 对其到页
将 freemem 以下的虚拟地址对应的物理地址对应的页控制块引用自增
将 freemem 以上的虚拟地址对应的物理地址对应的页控制块插入空闲页链表
*/
void page_init(void);
// 依靠 freemem 分配内存
void *alloc(u_int n, u_int align, int clear);

// 从页空闲链表中获取一个页控制块，将其对应的页放置在pp指向的位置上，并初始化这个页的数据，成功返回0，失败返回-E_NO_MEM
int page_alloc(struct Page **pp);
// 将一个引用为0的页的页控制块插入空闲链表头
void page_free(struct Page *pp);
/* 静态函数，仅能在当前文件中使用。给定全空间虚地址，在给定的页目录中寻找/创建相应的页目录项及页表。计算出指向相应页目录项的指针
```

```
检查页目录项是否有效，无效且 create 非零则申请物理页创建页表，如果无效且 create 为零则将*ppte置为NULL并返回零
在相应的页表中找到对应的页表项，将其地址赋给*ppte
*/
static int pgdir_walk(Pde *pgdir, u_long va, int create, Pte **ppte);
// 减少页控制块引用，如果减为零则调用 page_free
void page_decref(struct Page *pp);
// 将pp控制块对应的页在 pgdir 上与 va 建立关联，会调用 pgdir_walk 寻找并判断 va 是否已经有映射，不需要 va 对齐到页，asid 的作用是帮助清除掉该进程在 va 处已经有的映射或是冲刷 TLB，失败返回 -E_NO_MEM
int page_insert(Pde *pgdir, u_int asid, struct Page *pp, u_long va, u_int perm);
// 找到一个虚拟地址映射到的页控制块作为返回值，如果ppte非空则将页表项指针放置在其指向的空间中。如果页表项（调用pgdir_walk查找）无效则返回空。在tlb重填时用到。va没有映射的话返回 NULL。
struct Page *page_lookup(Pde *pgdir, u_long va, Pte **ppte);
// 清除 va 所在的虚拟页在 pgdir 中的映射，并调用 tlb_invalidate 清除对应TLB项
void page_remove(Pde *pgdir, u_int asid, u_long va);
// 调用 tlb_out 实现TLB表项的冲刷
void tlb_invalidate(u_int asid, u_long va) {
    tlb_out(PTE_ADDR(va) | (asid << 6));
}
```

内联函数

```
//include/pmap.h
//提供Page结构体指针，返回该结构体在页控制块数组中的下标，即页框号
static inline u_long page2ppn(struct Page *pp);

//提供Page结构体指针，返回该Page所管辖的物理页起始地址
static inline u_long page2pa(struct Page *pp);

//提供物理地址，返回管辖该页框的Page结构体指针，并进行越界检查，pa无需对齐
static inline struct Page *pa2page(u_long pa);

//给定Page结构体指针，返回其在内核地址空间上映射的虚拟地址
static inline u_long page2kva(struct Page *pp);

//给定页目录指针和虚拟地址，返回其物理地址，如果页目录项或者页表项无效，返回-1
static inline u_long va2pa(Pde *pgdir, u_long va);
```

地址转换宏

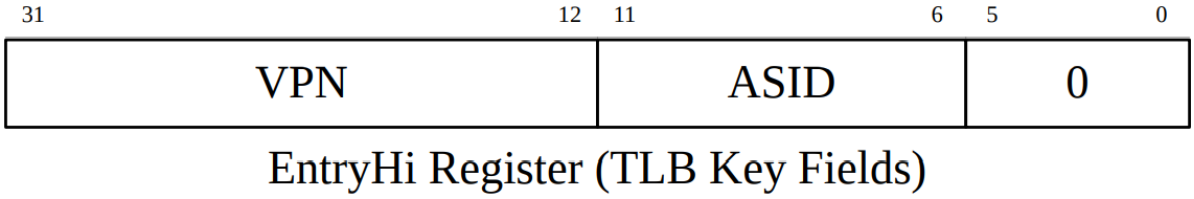
```
//include/mmu.h
#define PDX(va) ((((u_long)(va)) >> 22) & 0x03FF)
#define PTX(va) ((((u_long)(va)) >> 12) & 0x03FF)
#define PTE_ADDR(pte) ((u_long)(pte) & ~0xFFF) //从页表项中取物理页地址，或从虚拟地址中获取虚拟页号

// 根据地址获取页框号
#define PPN(va) (((u_long)(va)) >> 12)
#define VPN(va) (((u_long)(va)) >> 12)

//内核地址空间转换宏
#define PADDR(kva) //转物理
#define KADDR(pa) //转虚拟
```

权限

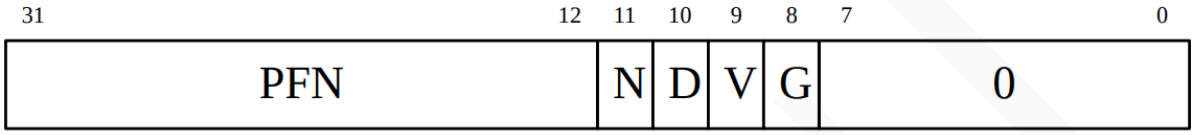
```
//include/mmu.h
// 全局位（Global bit），用于指示是否对该页表项进行全局性的匹配。当这个标志位被设置时，在TLB匹配时只考虑VPN字段（虚拟页面号），而忽略ASID字段（地址空间ID）。
#define PTE_G 0x0100

// 用于指示页表项所映射的物理页是否有效。如果这个标志位被设置为0，表示该页表项所映射的物理页无效，访问该页表项会触发TLB异常。exception (TLBL/TLBS).
#define PTE_V 0x0200

// 脏位（Dirty bit），用于指示该页表项所映射的物理页是否可写。如果这个标志位被设置为0，表示该页表项所映射的物理页只读，写该页表项会触发TLB修改异常。
#define PTE_D 0x0400

// 不可缓存位（Non-cacheable bit），用于指示该页表项所映射的物理页是否可被缓存。如果这个标志位被设置为1，表示该页表项所映射的物理页不可被缓存。（不能通过cache访问）
#define PTE_N 0x0800

// 写时复制位（Copy On Write），该标志位用于实现写时复制技术，用于共享物理页。如果这个标志位被设置为1，表示该页表项所映射的物理页是只读的，当对这个物理页进行写操作时，会触发一个页面异常，系统会将这个物理页复制一份，然后修改新的物理页，将它映射到当前进程的页表中。
// Copy On Write. Reserved for software, used by fork.
#define PTE_COW 0x0001

// 共享页面位（Library bit），该标志位也用于实现写时复制技术，用于共享物理页。如果这个标志位被设置为1，表示该页表项所映射的物理页是只读的，当进行写操作时，会触发一个页面异常，然后操作系统会复制一份物理页，并将它映射到当前进程的页表中，同时这个物理页可以被其他进程共享。将在Lab6中用于实现管道
// Shared memmory. Reserved for software, used by fork.
#define PTE_LIBRARY 0x0004
```

TLB

每一个 TLB 表项都有 64 位，其中高 32 位是 Key，低 32 位是 Data。



EntryHi Register (TLB Key Fields)



EntryLo Register (TLB Data Fields)

EntryHi、EntryLo 都是 CP0 中的寄存器，他们只是分别对应到 TLB 的 Key 与 Data， 并不是 TLB 本身。BadVaddr 保存引发地址异常的虚拟地址。Index TLB 读写相关需要用到该寄存器。Random 随机填写 TLB 表项时需要用到该寄存器。

Key（EntryHi）：

- VPN：Virtual Page Number
  - 当 TLB 缺失（CPU 发出虚拟地址，在 TLB 中查找物理地址，但未查到）时， EntryHi 中的 VPN 自动（由硬件）填充为对应虚拟地址的虚页号。
  - 当需要填充或检索 TLB 表项时，软件需要将 VPN 段填充为对应的虚拟地址。
- ASID：Address Space IDentifier
  - 用于区分不同的地址空间。查找 TLB 表项时，除了需要提供 VPN，还需要提供 ASID（同一虚拟地址在不同的地址空间中通常映射到不同的物理地址）。

Data（EntryLo）：

- PFN：Physical Frame Number
  - 软件通过填写 PFN，随后使用 TLB 写指令，才将此时的 Key 与 Data 写入 TLB 中。
- NDGV：权限位，见上

```
tlbr ; 以 Index 寄存器中的值为索引，读出 TLB 中对应的表项到 EntryHi 与 EntryLo。
tlbwi ; 以 Index 寄存器中的值为索引，将此时 EntryHi 与 EntryLo 的值写到索引指定的 TLB 表项中。前需要nop。
tlbwr ; 将 EntryHi 与 EntryLo 的数据随机写到一个 TLB 表项中（此处使用 Random 寄存器来“随机”指定表项，Random 寄存器本质上是一个不停运行的循环计数器）。前需要nop。
tlbp ; 根据 EntryHi 中的 Key（包含 VPN 与 ASID），查找 TLB 中与之对应的表项，并将表项的索引存入 Index 寄存器（若未找到匹配项，则 Index 最高位被置 1）。前后需要nop。
```

```
; kern/tlb_asm.S
; 本质：填写 CP0 相关寄存器，使用 TLB 相关指令
```

```
/* 被 tlb_invalidate 调用，清除 PTE_ADDR(va) | (asid << 6) 对应的TLB表项
将参数写入 EntryHi 进行检索
如果查找到表项则通过已写入 Hi 和 Lo 的0来将该表项置零，没有则跳过
回复 EntryHi 的内容
*/
LEAF(tlb_out)

/* void do_tlb_refill(void) 异常处理函数，调用_do_tlb_refill完成TLB重填
分别从 CP0_BADADDR 和 CP0_ENTRYHI 中获取失配虚地址和asid
将其作为参数传递给_do_tlb_refill完成充填
将返回值（Pte的内容）赋给 CP0_ENTRYLO0
*/
NESTED(do_tlb_refill, 0, zero)
```

```
// kern/tlbex.c

/* 用户进程 TLB 重填
循环调用 page_lookup 在当前的 cur_pgdir 中搜索va 的映射，如果没有找到就调用
passive_alloc(va, cur_pgdir, asid) 申请物理页，建立va的映射
返回 Pte 项本身的数据而不是指针
*/
Pte _do_tlb_refill(u_long va, u_int asid);

/* 为用户进程建立地址映射
被动页表(passive page table)，用于用户进程的地址空间
如果va在UTEMP和USTACKTOP之外，引发内核崩溃
申请页和插入页失败，引发内核崩溃
*/
static void passive_alloc(u_int va, Pde *pgdir, u_int asid);
```