Lab2实验报告

一、实验思考题

Thinking 2.1

指针变量存储的是虚拟地址；lw，sw使用的是虚拟地址

Thinking 2.2

用宏来实现链表：可以生成多类型的链表，间接具备了泛型的功能，重用性好；

链表形式	插入 & 删除
实验用：双向链表	无需遍历，从给出的链表项即可完成操作，较为灵活
环境：单向链表	需要获取插入位置前的链表项才可插入，删除时还需遍历查找删除项前一项
环境：循环链表	同单向链表，但稍微灵活些（从尾到头）

Thinking 2.3

C

Thinking 2.4

boot_pgdir_walk：在boot_map_segment中被调用

boot_map_segment：在mips_vm_init中被调用

Thinking 2.5

ASID的必要性：不同进程间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，所以需要ASID去区分该虚拟地址是那个进程的

ASID共6位，最多可容纳64个不同的地址空间（版本号只是增加了同时可运行的进程数）

Thinking 2.6

tlb_invalidate调用tlb_out

tlb_invalidate(Pde *pgdir, u_long va)使va对应的tlb表项失效

```
mfc0 k1, CP0_ENTRYHI          # 将EntryHi（TLB键值）存入$k1
mtc0 a0, CP0_ENTRYHI          # 将传入的参数写入EntryHi
nop                           # 解决tlbp相关的数据冒险
tlbp    # 根据EntryHi中键值（VPN+ASID）查找表项，并将其索引存入Index（失败则置首位1）
nop                           # 解决tlbp相关的数据冒险
mfc0 k0, CP0_INDEX            # 取出查找结果存入$k0
bltz k0, NOFOUND              # 若$k0小于0则跳转至NOFOUND（首位为1）
nop                           # 延迟槽
mtc0 zero, CP0_ENTRYHI        # EntryHi清零
mtc0 zero, CP0_ENTRYLO0       # EntryLo0清零
nop
tlbwi   # 以Index值为索引,将EntryHi EntryLo值（0）写回相应TLB表项中。
```

```
NOFOUND:
mtc0 k1, CP0_ENTRYHI        # 恢复原EntryHi键值

j ra                        # 函数返回
nop
```

Thinking 2.7

三级页表页目录基地址：PDbase = PTbase + (PTbase >> 9)

映射到页目录自身的页目录项：PDbase + (PTbase >> 27)

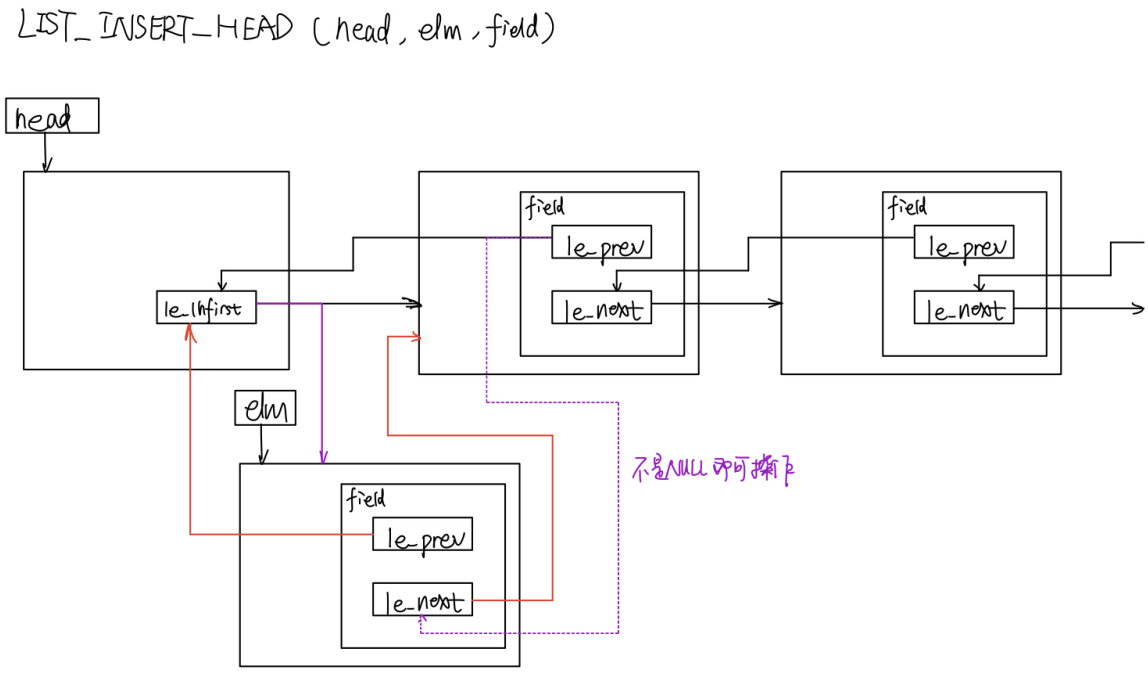
Thinking 2.8

行为	mips	x86
TLB miss	触发TLB Refill异常，交由软件内核进行TLB充填	直接由硬件MMU从CR3中获取地址；并进行TLB填充
转换失败的地址	存入BadVAddr	存入CR2

二、实验难点

1.Page/链表结构理解

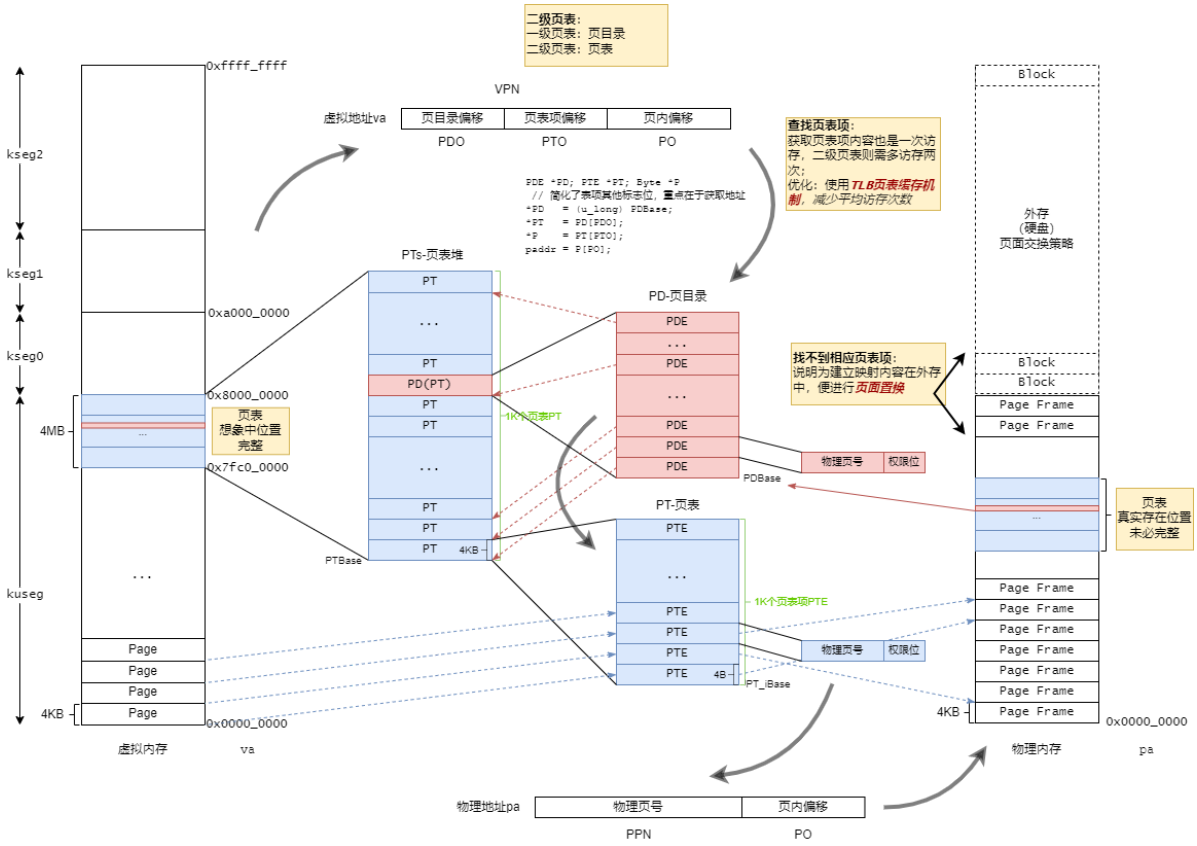
当时在理解时画的一个图，感觉这样想就行



> 真是难嘞，感觉到现在，整个内存管理的细节图景我还是不很明了，牵连进程管理相关部分也很模糊。主要就是地址映射相关部分

2.二级页表结构/地址映射相关理解

看自映射时画的，但现在觉得不太对，很多理解有偏差，加急理解了要



3.tlb重填落到代码处的具体操作理解

4.繁多的函数，相关功能的熟悉

专门写某文件的解读文档会有助益

三、体会与感想 & 建议

> 当时边做lab2边写的感想。实在是来不及了马上就24:00了，没时间修正斟酌了，相关建议可能很短视幼稚，可能源于我自己的理解问题。
>
> 但总体来看，这个指导书我实在是看不来，总感觉没有完整的图景，介绍比较跳越割裂，初学者看起来累；指导书内容和所给代码不匹配，代码注释写的不完善会误导初学者漏写功能...

内存管理我的软肋qwq，太蠢了。。

6.ROUND(a,n) 好妙的样子，没看懂

```
#define ROUND(a, n)   (((((u_long)(a))+(n)-1)) & ~((n)-1))
```

7.page控制块那里，没有完整的图景，也不知道pages怎么得来的，写得很虚空。

9.freemem的含义搞错

10.指导书的文案让人理解为maxpa不用写。

11.**令人在意**：LIST_INSERT_TAIL的一处写错，导致了循环进入main.c直至内存分配完毕

12.饶舌的文案：static Pte *boot_pgdir_walk(Pde *pgdir, u_long va, int create)，它返回一级页表基地址 pgdir 对应的两级页表结构中，va 这个虚拟地址所在的二级页表项

13.病句，boot_map_segment(..)处：同时为相关页表项的权限为设置为 perm。-> 将， 为删去

14.pgdir_walk函数中if(create)内，若page分配成功后应该有++page->pp_ref； （后文的page_insert中有）

15.似没有说明白tlb_out函数的位置和形式，得自己推断在tlb_asm.S中

16.“经过以上分析，tlb_invalidate 的重要性更得以体现——如果页表内容变化而 TLB未更新，则可能访问到错误的物理页面。”-TLB重填中。可是上面一句也没讲和tlb_invalidate的关系（之前讲的tlb_invalidate已经忘了，甚至如果前面实验代码看马虎点，我们连tlb_invalidate什么意思都不知道），这句就显得没头没脑；

17.接上，还有前面好像在讲do_refill函数，转眼下面就去给tlb_out函数添指令？几乎不知道两者联系

18.boot_map_segment中：“提示：需要填写的部分为物理页框号 pa + i，不需要使用宏进行转换 ”：pa是物理地址，pa+i也是物理地址，虽然内容上和物理页框号相同，可以不用宏转换，但使用PTE_ADDR(pa+i)在语义上会更贴切一些吧；pa会对齐嘛

19.有些代码不够好看。page_remove中对pp_ref的操作可以用之前写的page_decref函数而不是再写一遍；page相关的函数中的变量可以统一名称和声明顺序（`struct Page *ppage; Pde *pgdir; Pte *pgtable_entry;`之类的），可以便于归纳理解；一些判断的模式（`if ((r = page_alloc(&p))`）可以统一一下