Lab2实验报告

Thinking

2.1

CPU运行程序时，会发出地址，进行内存读写操作。在计组课设中，CPU 直接发送物理地址，这是为了简化内存操作，让大家关注CPU内部的计算与控制逻辑。而在操作系统课设中，R3000 CPU 只会发出虚拟地址。

请你根据上述说明，回答问题：

- 在我们编写的程序中，指针变量中存储的地址是虚拟地址还是物理地址？**虚拟地址**
- MIPS 汇编程序中lw, sw使用的是虚拟地址还是物理地址？**虚拟地址**

2.2

- 请从可重用性的角度，阐述用宏来实现链表的好处。**用宏实现链表可以在C语言中使用泛型，使得多种结构类似的数据结构可以复用这一链表宏函数。**
- 请你查看实验环境中的 /usr/include/sys/queue.h，了解其中单向链表与循环链表的实现，比较它们与本实验中使用的双向链表，分析三者在插入与删除操作上的性能差异。

本实验中的双向链表：删除操作O(1)，插入链表头O(1) 插入链表尾O(n)，插入某元素前后O(1)

单向链表：删除表头O(1) 删除表中一个元素O(n) 插入到某元素前后O(1) 插入链表头O(1) 插入链表尾O(n)

循环链表：插入 O(1) 删除O(1)

2.3

请阅读 `include/queue.h` 以及 `include/pmap.h`, 将 `Page_list` 的结构梳理清楚，选择正确的展开结构。

```
A:
struct Page_list{
    struct {
        struct {
            struct Page *le_next;
            struct Page **le_prev;
        }* pp_link;
        u_short pp_ref;
    }* lh_first;
}
```

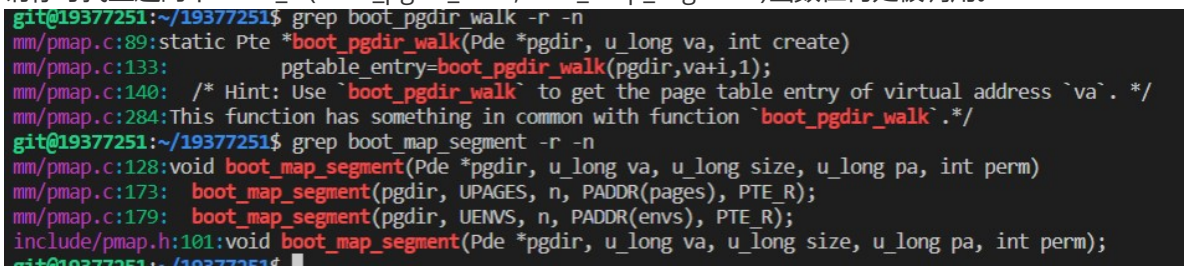
```
B:
struct Page_list{
    struct {
        struct {
            struct Page *le_next;
            struct Page **le_prev;
        } pp_link;
        u_short pp_ref;
    } lh_first;
}
```

```
C:
struct Page_list{
    struct {
        struct {
            struct Page *le_next;
            struct Page **le_prev;
        } pp_link;
        u_short pp_ref;
    }* lh_first;
}
```

易得Page_list中包含Page结构体，而Page结构体中成员变量都不包含指针，因此A排除。Page_list是一个链表结构，因此是C。

2.4

请你寻找上述两个 boot_* (boot_pgdir_walk,boot_map_segment)函数在何处被调用。



boot_pgdir_walk在boot_map_segment中被调用

boot_map_segment在mips_vm_init中被调用

2.5

请你思考下述两个问题：

- 请阅读上面有关 R3000-TLB 的叙述，从虚拟内存的实现角度，阐述 ASID 的必要性

  用于区分不同进程的地址空间（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上），且在一定机制下可以实现对大于32位的虚拟地址空间的映射

- 请阅读《IDT R30xx Family Software Reference Manual》的 Chapter 6，结合 ASID 段的位数，说明 R3000 中可容纳不同的地址空间的最大数量

  1<<6 64个

2.6

请你完成如下三个任务：

- tlb_invalidate 和 tlb_out 的调用关系是怎样的?

  tlb_invalidate中调用tlb_out

- 请用一句话概括 tlb_invalidate 的作用

  将当前虚拟地址对应于tlb中的项清零，更新tlb

- 逐行解释 tlb_out 中的汇编代码

```
LEAF(tlb_out)
//1: j 1b
nop     // 先把HI的值保存到k1
    mfc0    k1,CP0_ENTRYHI
    // a0是传递进函数的参数 HI储存虚拟地址空间和标志位
    mtc0    a0,CP0_ENTRYHI
    nop
    // tlbp指令 查询HI中虚拟地址是否在TLB中
    // 有匹配则把匹配的index写入INDEX寄存器
    // 无匹配INDEX寄存器最高位置1 变成负数
    tlbp
    nop
    nop
    nop
    nop
    mfc0    k0,CP0_INDEX
    // 读出INDEX 如果<0 则虚拟地址不在TLB中 进入NOFOUND        bltz    k0,NOFOUND

    nop
    // 如果>=0 则虚拟地址在TLB中 进入下面部分 把HI LO0寄存器置0
    mtc0    zero,CP0_ENTRYHI
    mtc0    zero,CP0_ENTRYLO0
    nop
    // 前面已经把index写入了INDEX寄存器
    // 写INDEX寄存器中索引的TLB条目
    tlbwi
NOFOUND:
    // 恢复HI
    mtc0    k1,CP0_ENTRYHI
    // 直接返回
    j       ra
    nop
END(tlb_out)
```

2.7

在现代的 64 位系统中，提供了 64 位的字长，但实际上不是 64 位页式存储系统。假设在 64 位系统中采用三级页表机制，页面大小 4KB。由于 64 位系统中字长为 8B，且页目录也占用一页，因此页目录中有 512 个页目录项，因此每级页表都需要 9 位。因此在 64 位系统下，总共需要 3 × 9 + 12 = 39 位就可以实现三级页表机制，并不需要 64 位。现考虑上述 39 位的三级页式存储系统，虚拟地址空间为 512 GB，若记三级页表的基地址为 PTbase ，请你计算：

- 三级页表页目录的基地址

  PTbase|(PTbase>>9)|(PTbase>>18)

- 映射到页目录自身的页目录项(自映射)

  PTbase|(PTbase>>9)|(PTbase>>18)|(PTbase>>27)

2.8

任选下述二者之一回答：

- 简单了解并叙述 X86 体系结构中的内存管理机制，比较 X86 和 MIPS 在内存管理上的区别。
- 简单了解并叙述 RISC-V 中的内存管理机制，比较 RISC-V 与 MIPS 在内存管理上的区别。

x86：

x86架构的内存管理机制分为两部分：分段机制和分页机制。分段机制为程序提供彼此隔离的代码区域、数据区域、栈区域，从而避免了同一个处理器上运行的多个程序互相影响。

分页机制实现了传统的按需分页、虚拟内存机制，可以将程序的执行环境按需映射到物理内存。此外，分页机制还可以用于提供多任务的隔离。

分段机制和分页机制都可以通过配置，支持简单的单任务系统、多任务系统或共享内存的多处理器系统。需要强调的一点是，处理器无论在何种运行模式下都不可以禁止分段机制，但是分页机制却是可选选项。

X86 采用段页式的内存管理机制，先通过分段单元将逻辑地址转换为线性地址，然后通过分页单元，将线性地址转换为物理地址，其中分页单元也是通过二级页表来进行实现。且 X86 通过页地址拓展（PAE）实现了对于大于4GB内存地址的管理

X86 通过段页式进行管理，相对于 mips 而言多了一层逻辑地址到线性地址的映射，且 mips 并没有实现页地址拓展（PAE）机制

mips使用分页机制

实验难点

实验的第一个难点是链表宏的理解，链表宏是为了实现泛型链表而实现的，可以根据不同的数据类型实现效果相同的链表，其中包含了链表基本的操作（例如 初始化 根据节点插入 删除首项）。

pp_link是链表实现的依据，其中包含指向前后的两个指针，但是指向前一个的指针并不能获取前一项的数据，仅能改变前一项的指向，因此不完全算是一个双向链表，但是这样做好处就是可以让链表头的类型不再与链表内部节点的类型一样，方便链表头的处理，此外还可以简化节点的删除操作，减小时间成本。

实验的第二个难点是虚实地址转换机制，在代码中我们使用到到的地址都是虚拟地址，需要转化到物理地址，内核在kseg0段，这一段的地址映射是连续的并且不经过MMU，在mmu.h中定义了地址转换中最关键的两个函数是KADDR与PADDR，通过±ULIM(0x80000000)±ULIM(0x80000000)实现映射。以这两个宏函数为核心，建立了一系列（内核）虚拟地址、物理地址、页结构体之间相互转换的（宏）函数：PPN、VPN、page2ppn、page2pa、pa2page、page2kva。

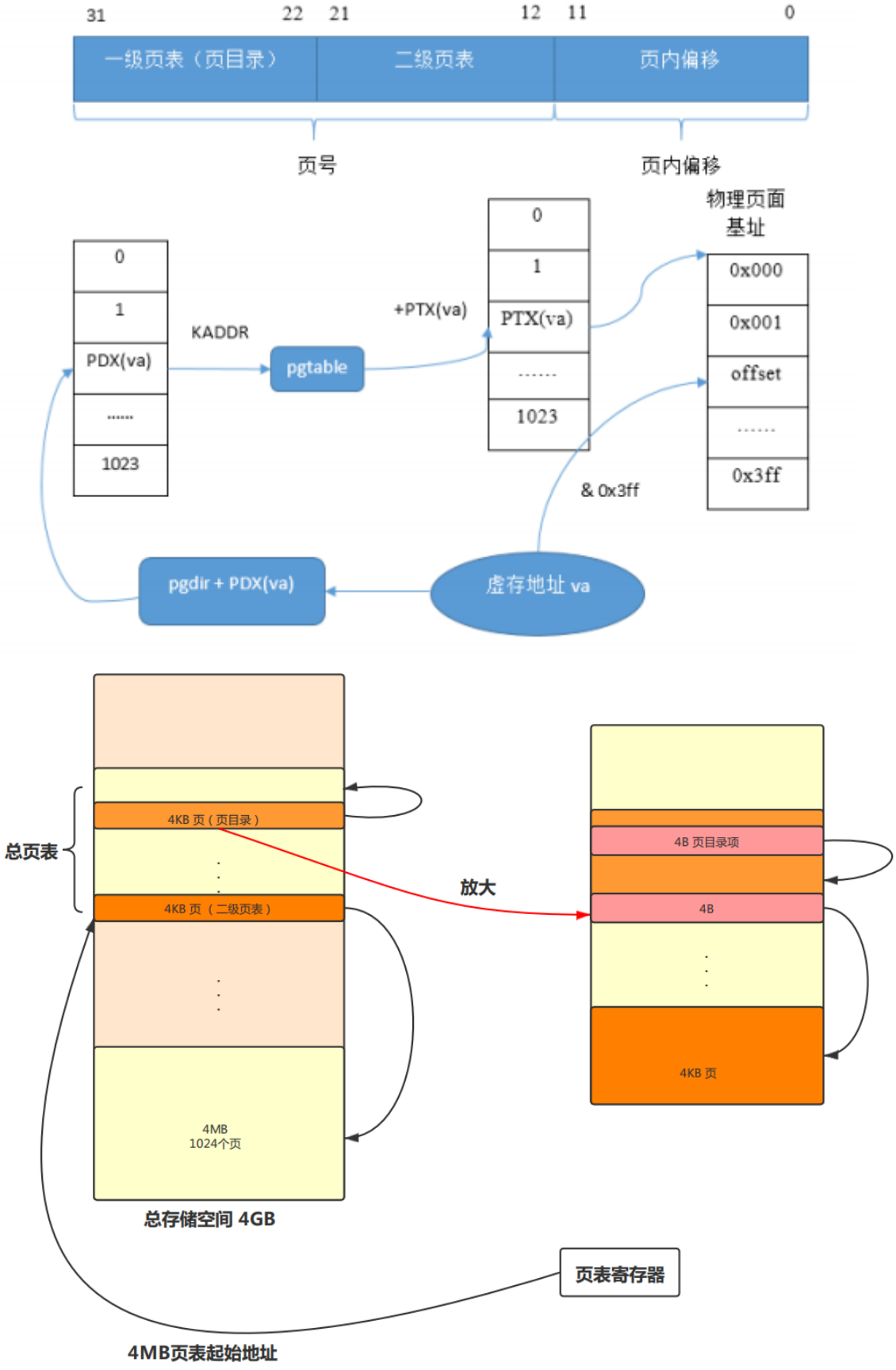
exercise中地址的转换过程：

Page --> pa --> kva:找到page在结构体数组的索引，左移12位得到页的物理地址，加0x8000_0000得到内核虚拟地址。

kva --> pa --> Page:内核虚拟地址减0x8000_0000得到物理地址，根据物理地址计算PPN，pages数组的第PPN个就是kva对应的page结构体。

还有一个难点是虚拟存储和页表自映射

自映射的原理可以通过下图进行理解



体会与感想

lab2-1的难点在于对链表宏的理解和虚拟内存的简要理解，链表部分主要应该知道每个链表宏的作用，这就需要通读一遍代码了解（知道有这个函数）和理解（知道怎么用）宏函数，这样就能在需要的时候不自己造轮子。虚拟内存部分则需要对各类转换函数有一定理解，这样在能够在需要的时候调用正确的函数，比如page2kva page2pa等。

exam中是实现一个页保护机制，对页结构体引入一个status成员变量后就很好实现。后续Extra中的内存分配机制实质上是对树形结构的考察，需要用到树节点的分裂与合并，在不考虑时间复杂度的情况下较好实现。

lab2-2主要考察对自映射的理解，exam是一个对二级页表的遍历，Extra是根据遍历结果进行赋值。

lab2整体课下测试编码量不大，对链表宏、页和页表自映射理解较好后便可实现，两次任务各花费了4-6h。课上测试是针对页、内存分配机制和页表自映射的考察。

通过lab2拓展了c实现泛型的方法，了解了页、页表自映射的知识。

指导书反馈

希望能够对函数内注释进行一定的解释，对所给的代码框架做一定的解释，虽然这样会减轻完成实验的负担，但是会导致知识一点不会但是能通过测试。

残留难点

2-2-Extra中更改页表项指向的物理地址部分不知道为何出错，导致没能通过，可惜。