Lab2实验报告

思考题部分

Thinking 2.1

无论在编写的C程序中的指针变量，还是MIPS汇编程序的lw,sw都使用的是虚拟地址。

Thinking 2.2

- 用宏来实现链表，可以根据需求创建不同数据类型的链表，在使用相关函数时只需要给出需求的数据类型，可重用性较高。
- 单向链表由于只有指向后一个元素的指针，因此在官方定义的宏函数中没有INSERT_BEFORE

  单向链表的REMOVE方法整体上与其他两者性能相当。

  循环链表不论从插入还是删除操作的角度都和实验中的双向链表接近。

Thinking 2.3

C

Thinking 2.4

- 由于同一虚拟地址在可能在不同的进程中使用，并对应不同的页框号；如果TLB不支持ASID，则每次切换进程时都要将TLB中的所有内容刷新，才能保证不同进程得到正确的地址转换结果。而有了ASID，就可以在切换进程时不用无效化所有的TLB内容，提高了效率。
- ASID对应了EntryHi的第11-6共6位，因此R3000可以容纳2^6=64个不同的地址空间。

Thinking 2.5

- 由tlb_invalidate在pmap.c的定义可知，tlb_invalidate调用了tlb_out。
- 将asid号地址空间的va对应的TLB中的页表项无效化，使下一次访问(asid,va)对应的页表项是触发TLB重填异常。

```
LEAF(tlb_out)                           //声明tlb_out为叶函数，不会调用其他函数
.set noreorder                          //不允许汇编器重排指令提高性能
    mfc0    t0, CP0_ENTRYHI             //将原有的HI的值先保存到t0中
    mtc0    a0, CP0_ENTRYHI             //将传入的旧表项的Key写入HI
    nop                                 //解决数据冒险的nop
    tlbp                                //根据旧表项的key查找TLB中对应的表项并将索引写入
index寄存器中
    nop                                 //解决数据冒险的nop
    mfc0    t1, CP0_INDEX               //将index的值写到t1中
.set reorder                            //允许汇编器对指令进行重新排序来提高性能
    bltz    t1, NO_SUCH_ENTRY           //通过与0比较判断有没有查到(没查到最高位被置了1，代
表负数)
.set noreorder                          //不允许汇编器重排指令提高性能
    mtc0    zero, CP0_ENTRYHI           //找到了，则将TLB的内容刷新掉(全部置0)
```

```
    mtc0    zero, CP0_ENTRYLO0          //同上
    nop
    tlbwi                               //将EntryHi和EntryLo的内容（0）写入index对应的
TLB条目
.set reorder                            //允许汇编器对指令进行重新排序来提高性能

NO_SUCH_ENTRY:                          //没有找到
    mtc0    t0, CP0_ENTRYHI             //恢复原来HI寄存器中的值
    j       ra                          //跳转返回
END(tlb_out)                            //结束
```

Thinking 2.6

x86的内存管理机制：（参考：https://blog.csdn.net/jiangwei0512/article/details/63687977，http://www.c114.com.cn/tech/169/a1171076.html，https://blog.csdn.net/cybertan/article/details/5884438）

x86架构中内存被分为三种形式，分别是逻辑地址（Logical Address），线性地址（Linear Address）和物理地址（Physical Address）。通过分段机制将逻辑地址转换为线性地址，通过分页机制将线性机制转换为物理地址。其中分段机制在x86中无法被禁用，是强制执行的，而分页机制是可选的。

分段机制将内存划分为一段一段由起始地址和长度表述的段。段的结构可以和程序的内容相结合，比如程序可以简单分为代码段，数据段和栈段。

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。

差别：

- 如上，x86系统的段页式内存管理，MIPS主要采用页式内存管理。
- 在x86系统中，CR3寄存器也称为页目录基地址寄存器（Page-Directory Base Register，PDBR），存放着页目录的物理地址。一个进程在运行前，必须将其页目录的基地址存入CR3，而且，页目录的基地址必须对齐到4KB页边界。而在MIPS系统中，是通过其他方式实现的。

Thinking A.1

$PD_{base}=PT_{base}|(PT_{base}>>9)|(PT_{base}>>18)$

$PDE_{self\text{-}mapping}=PT_{base}|(PT_{base}>>9)|(PT_{base}>>18)|(PT_{base}>>27)$

难点分析

2.4.1 链表宏

Page_list

用LIST_HEAD建立起的名为Page-list的结构体,实际上可以作为一个Page链表的表头，内容lh_first指针指向第一个页控制块。

```
struct Page_list{
    struct Page *lh_first;
}
```

对queue.h中部分宏定义的解释

```
LIST_INIT(head)         //将head中的lh_first指向null,head在本实验中是*Page_list,即指向Page链表表头的指针。
LIST_NEXT(elm, field)   //在本实验中用于返回在链表中指向elm的下一个页控制块的指针
LIST_INSERT,LIST_REMOVE相关的宏定义功能就和其名字基本相同。
```

2.4.2 页控制块（Struct Page）

页控制块是一个与物理页框一一对应的，存储着对应页辅助信息的一个结构体。内容包含一个pp_link结构体和对应页框的引用次数pp_ref。

pp_link结构体内容包含一个页控制体的指针next, next指向连续的下一个页结构体，prev则指向上一个页结构体的le_next。

```
struct Page{
    struct pp_link{
        struct Page *le_next;
        struct Page **le_prev;
    }
    u_short pp_ref;
}
```

与页控制块相关的宏定义函数如下：

```
u_long page2ppn(struct Page *pp) //获取页控制块对应的物理页框号
u_long page2pa(struct Page *pp)  //获取页控制块对应页的物理地址
u_long page2kva(struct Page *pp) //获取页控制块对应页的虚拟地址（仅限kseg0范围内，利用KADDR（）将获取的物理地址转化为虚拟地址）
struct Page *pa2page(u_long pa)  //根据某一页的物理地址获取该页对应的页控制块
```

page_alloc的部分个人疑惑

单从Lab2的Page_alloc方法来看，这种从申请页面的方式并不能体现虚拟地址的扩展功能，感觉实际操作的只有初始化的那些物理页，实际上是指导书阅读不细。指导书中提到了，在 MOS 中，对这一过程进行了简化，一旦物理页全部被分配，进行新的映射时并不会进行任何的页面置换，而是直接返回错误，没有涉及到物理页的页面置换。

2.5.1 两级页表机制

常用宏

```
PDX(va) //获取va的31-22位(对应一级页表偏移量)
PTX(va) //获取va的21-12为(对应二级页表偏移量)
```

pgdir_walk

给定虚拟地址，在给定的页目录中获取对应页表项的函数。该函数完整体现了从虚拟地址获取二级页表项地址的整个过程，也展示了两级页表机制的运作机理。

```
static int pgdir_walk(Pde *pgdir, u_long va, int create, Pte **ppte) {
    Pde *pgdir_entryp;
    struct Page *pp;
    int allocres;
    pgdir_entryp=pgdir+PDX(va);                                    //将给定的页目录基地址加上一级页表的偏移量获取对应一级页表项的地址
    if(((*pgdir_entryp)&PTE_V)==0){                                //该页表项内容无效，则要进行处理(创建/赋予NULL值)
        if(create){
            if((allocres=page_alloc(&pp))<0){                      //申请物理页面失败了，没有空闲的物理页面
                return allocres;
            }
            *pgdir_entryp=page2pa(pp)|PTE_D|PTE_V;                 //获取刚申请到的物理页的物理地址并附加有效位，将其作为整体赋予该页表项。
                                                                   //这一申请的物理页面是新的二级页表所在页
            (pp->pp_ref)++;                                        //对应物理页面的引用次数++
        }else{
            *ppte=NULL;                                            //如果不申请新页面，则*ppte也就是二级页表项的指针直接无效化
            return 0;
        }
    }
    *ppte=(Pte*)(KADDR(PTE_ADDR(*pgdir_entryp)))+PTX(va);          //此时pgdir_entryp是一个一级页表项的指针，利用PTE_ADDR获取对应二级页表的物理地址再转成虚拟地址后，再加上偏移，才是对应二级页表项的地址
    return 0;
}
```

2.5.2 虚拟地址到物理页面的映射

page_insert

```
int page_insert(Pde *pgdir, u_int asid, struct Page *pp, u_long va, u_int perm) {
    Pte *pte;
    int pgdirwalkres;
    pgdir_walk(pgdir, va, 0, &pte);                                //尝试获取该va对应的页表项
    if (pte && (*pte & PTE_V)) {                                   //如果页表项非空且有效的话，检查是否映射到目标页控制块
        if (pa2page(*pte) != pp) {
            page_remove(pgdir, asid, va);                          //如果不是的话，解除该va对这个页控制块的引用，如果没有引用就添加到freelist
        } else {
```

```
            tlb_invalidate(asid, va);                           //如果是的话，因为要修改
有效位为perm,所以还需要将TLB的对应的内容flush掉
            *pte = page2pa(pp) | perm | PTE_V;                  //修改页表项内容即可，不
需要重新建立页表项。
            return 0;
        }
    }
    tlb_invalidate(asid, va);
    if((pgdirwalkres=pgdir_walk(pgdir, va, 1, &pte))<0){
        return pgdirwalkres;
    }
    *pte = page2pa(pp)|perm|PTE_V;                              //修改页表项内容为pp的
页框号加设置的有效位
    (pp->pp_ref)++;
    return 0;
}
```

实验体会

1. 个人C语言基础存在一定的薄弱之处。在本次实验中出现了大量的指针和结构体嵌套，在自己阅读指导书并尝试搞清楚结构体之间的层次，不同指针指代的内容时，经历了相当比较痛苦的过程。但同时，操作系统的代码也是对这种薄弱之处的训练，在理清本次实验的过程中，也确实对其有一定的提升。
2. 多文件，多函数，多宏定义，汇编与C语言混合使用的复杂性。操作系统牵扯到了软件和硬件的沟通，也因此会有汇编和C语言的共同使用。同时这种多文件之间的相互调用大大增加了编程的复杂性，要搞懂一个宏定义的内容需要浏览多个宏定义，要搞明白一个文件的代码也要浏览多个文件。即使在我使用了vscode来进行多文件的同时浏览，还是遇到了一定的困难。
3. 不同于计组，计组理论课后期讲的TLB, Cache, 虚存，主存等内容和实验的完成关系不够强，理论和实验略有脱节。但是操作系统课程不同，操作系统理论课与实验的关系结合紧密，页式内存，自映射的思想在实践中均有体现。理论课是实验的指导，实验是对理论的加深，这种关系是更加合理的。