# Lab2实验报告

## 思考题

### Thinking 2.1

- 指针变量存储的地址是虚拟地址
- MIPS汇编程序中 `lw` 和 `sw` 指令使用的地址也是虚拟地址

### Thinking 2.2

- 从可用性的角度，使用宏来进行链表的实现，一方面可以规范系统内所有链表的格式以及表达形式，易于编码与链表的维护；另一方面可以不限制链表的类型，只要有相同的格式即可；另外可以降低链表指针与链表内容见不必要的耦合性，维护数据安全；同时也降低了维护难度
- 插入和删除上的差异

  单向链表插入时需要修改被插入项和自身的指针，插入头部的时候需要修改自身的指针，同时修改链表头部指向自己，在插入时需要从 head 开始遍历，消耗量大，而其他的双向和循环链表有宏，可以进行快速插入；而在删除方面，也需要首先定位到前一个元素。双向链表可以直接取 出，而单向链表仍然需要从 head 遍历到前一位，复杂度 `O(n)` 循环链表同理，由于循环链表的实现也是一个单指针链表，所以在插 入和删除时仍然无法避免循环，复杂度 `O(n)`

### Thinking 2.3

- 选 **C**
- 由 `LIST_HEAD` 可以知道最外层结构体中仅有一个指向 `Page` 的结构体指针，内层为 `Page` 结构体，含有一个 `pp_ref` 引用次数变量以及一个 `LIST_entry`，其中存在一个指向下一个 `Page` 的指针以及指向上一个 `Page` 的 `le_next` 的指针

### Thinking 2.4

- ASID用来标记当前虚拟地址所属的进程编号，区分不同进程的虚拟空间，如果访问的是其他程序的虚拟地址，通常会指向一个错误的物理地址。因此有了ASID，能够区分不同的请求虚拟地址，确定虚拟地址的来源，得到正确的物理地址
- ASID为 0 ~ 7 共 8 位，因此可容纳 $2^8 = 256$ 个不同的地址空间

### Thinking 2.5

- `tlb_invalidate` 函数调用了汇编函数 `tlb_out`

```
13 void tlb_invalidate(u_int asid, u_long va) {
14         tlb_out((va & ~GENMASK(PGSHIFT, 0)) | (asid & (NASID - 1)));
15 }
```

- 删除特定虚拟地址在 TLB 中的旧表项

```
LEAF(tlb_out)
.set noreorder # 禁止代码优化
        mfc0    t0, CP0_ENTRYHI # 从CP0_ENTRYHI寄存器取值至t0
        mtc0    a0, CP0_ENTRYHI # 将a0的值存入CP0_ENTRYHI中
        nop
        tlbp # 查找TLB表项，将索引存入index寄存器
        nop
        mfc0    t1, CP0_INDEX # 将索引取值至t1寄存器
.set reorder # 优化代码
        bltz    t1, NO_SUCH_ENTRY # 索引为空，退出
.set noreorder # 禁止代码优化
        mtc0    zero, CP0_ENTRYHI # 赋值清空
        mtc0    zero, CP0_ENTRYLO0 # 赋值清空
        mtc0    zero, CP0_ENTRYLO1 # 赋值清空
        nop
        tlbwi # 清空index对应的TLB项
.set reorder

NO_SUCH_ENTRY:
        mtc0    t0, CP0_ENTRYHI # 把CP0_ENTRYHI的值存回去
        j       ra
END(tlb_out)
```

## Thinking 2.6

- CPU发出访存指令
  - **对应函数调用**：当用户程序执行访存指令（如 `lw`、`sw`）时，CPU 会发出虚拟地址。
  - **流程**：用户程序通过指针解引用或直接访存指令触发 CPU 发出虚拟地址。
- 查询TLB
  - **对应函数调用**：硬件自动查询 TLB，检查目标虚拟地址是否在 TLB 中。
  - **流程**：
    - 若命中（TLB Hit），直接获取物理地址并完成访存。
    - 若未命中（TLB Miss），触发 TLB 缺失异常，进入内核异常处理流程。
- TLB确实异常处理

- **对应函数调用**：内核通过 `tlb_refill` 异常处理函数处理 TLB Miss。
- **流程**：
  1. 内核根据虚拟地址的 `PDX`（页目录索引）和 `PTX`（页表索引）查找两级页表。
  2. 调用 `pgdir_walk` 函数：
     - 若二级页表不存在且 `create` 参数为真，调用 `page_alloc` 分配新页表。
     - 将新页表的物理地址填入页目录项，并设置权限位。
  3. 调用 `page_lookup` 检查页表项是否有效：
     - 若无效（PTE_V 为 0），调用 `page_alloc` 分配新物理页，并调用 `page_insert` 填写页表项。
  4. 将页表项内容填入 `EntryLo0` 和 `EntryLo1`，通过 `tlbwr` 指令写入 TLB。
- 重新执行访存指令

## Thinking 2.7

问题一

- **X86 内存管理机制**
  - **分段机制**：x86 采用分段 + 分页的混合内存管理方式，逻辑地址通过段寄存器转换为虚拟地址。每个段由段描述符定义基址、界限和权限。
  - **分页机制**：线性地址通过多级页表（通常为 4 级：PML4 → PDPT → PD → PT）转换为物理地址。支持 4KB、2MB、1GB 等不同大小的页。使用 TLB 加速地址转换。
  - **TLB 管理**：x86 的 TLB 由硬件自动管理，但软件可以通过 `invlpg` 指令手动刷新 TLB 项。
- **区别**
  - MIPS不支持分段机制，x86可以
  - MIPS位两级页表，x86为4级页表
  - TLB方面，MIPS完全由软件管理，x86由硬件自动管理，软件可以干预
  - ASID方面，MIPS明确支持，而x86部分支持

## Thinking A.1

- `PTbase | PTbase >> 9 | PTbase >> 18`
- `PTbase | PTbase >> 9 | PTbase >> 18 | PTbase >> 27`

# 实验难点

1. 深入理解虚拟内存和物理内存的关系，不只是转换关系，更多的是思想上的关系，每部分代码到底对应的是哪个 seg 的空间，需要如何进行转换，页控制块如何控制页面分配等
2. 对于链表宏的理解和使用，作为广为接收和使用的链表宏方式，需要多多操作才能深入理解其优美之处
3. 本次实验文本和文件过多，且关系错综复杂，需要好好整理和反复阅读，从而加深记忆以及理解，看一次就多一份深入的感受！

# 体会与感想

本次实验开始之前感觉没什么难度，是不是和上学期的机组差不多，结果开始的时候慢慢就不对劲了，怎么函数这么多，宏这么多，虚拟地址和物理地址上怎么申请了这么多奇奇怪怪的东西，都是放哪儿的，如何映射的。第一遍学完之后有点懵，和舍友讨论了好久，特别是这两天OS课程也在学，但讲得又不是基于MIPS的操作系统，搞得有点懵了

之后又花了一次连续的时间来过一遍，进行笔记的二次总结，也算是终于理清了！

当然，肯定还会有第三遍的，深入理解操作系统才是学习实验的目标！