OS_lab2实验报告

思考题

Thinking 2.1 访存地址

由于在实际程序中，**访存、跳转**等指令以及用于取指的 PC 寄存器中的访存目标地址都是虚拟地址,以及**指针**解引用的地址也是虚拟地址，所以，C语言中指针中存储的变量和MIPS中lw和sw指令使用的地址都是虚拟地址。

Thinking 2.2 链表

使用宏来实现链表，避免了代码的重复，提高了代码的可读性和可维护性，使链表操作更加简洁，便于理解。同时，宏可以避免手动编写重复的代码，减少因为人为错误导致的问题，提高程序的可靠性和稳定性。

- **单向链表**：由于只能通过链表中的前一项去访问后一项，所以在向链表中某一项之前插入或删除的时候，需要从head开始遍历，而向后插入和删除的时候则不需要。
- **双向链表**：可以直接访问链表中的前一项和后一项，所以基于链表中某一项的前后进行插入和删除的时候不需要遍历，可以直接操作。但是由于不知道链表的结尾，所以向结尾处插入或者删除的时候，需要遍历整个链表。
- **循环链表**：由于循环链表有指向链表结尾的指针，所以在双向链表的基础上，它能够直接向结尾处插入或者删除。

Thinking 2.3 Page_list

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

选C。`pp_ref`是页对应物理内存的引用次数，`pp_link`是该页面对应的链表项，从`le_next`和`le_prev`可以索引到链表中的前后。`Page_list`里面应该一个指向某一个页面的指针。

Thinking 2.4 地址空间

- 操作系统在实现多进程的时候，会为每一个进程分配一个页表，通过页表来控制虚拟地址和物理地址的映射关系，在不同的进程中，同一个虚拟地址可能会映射到不同的物理地址上，ASID可以用来区分不同的地址空间，防止在多进程中虚拟地址被映射到错误的物理地址上。
- 根据文档中的描述，ASID对应39-32位，一共8位，所以可以容纳256个不同的地址空间。

Thinking 2.5 tlb_invalidate

- `tlb_invalidate`函数调用`tlb_out`，`tlb_out`是`tlb_invalidate`函数的功能主体。
- `tlb_invalidate`函数实现删除特定虚拟地址在TLB中的旧表项。
- ```
  LEAF(tlb_out)
  .set noreorder
      mfc0    t0, CP0_ENTRYHI
      // 将EntryHi中的信息存入t0，以便于函数结束时恢复
      mtc0    a0, CP0_ENTRYHI
      // 将需要删除的旧表项的key写入EntryHi
      nop
      /* Step 1: Use 'tlbp' to probe TLB entry */
      /* Exercise 2.8: Your code here. (1/2) */
      tlbp
      // 查找对应表项，并将索引写入Index
      nop
      /* Step 2: Fetch the probe result from CP0.Index */
      mfc0    t1, CP0_INDEX
      // 将Index中的旧表项索引写入t1
  .set reorder
      bltz    t1, NO_SUCH_ENTRY
      // 判断有没有查到TLB表项，没有则跳转到最后
  .set noreorder
      mtc0    zero, CP0_ENTRYHI
      mtc0    zero, CP0_ENTRYLO0
      mtc0    zero, CP0_ENTRYLO1
      // 将EntryHi,EntryLo0,EntryLo1写入0，方便后续用tlbwi指令删除对应TLB表项
      nop
      /* Step 3: Use 'tlbwi' to write CP0.EntryHi/Lo into TLB at CP0.Index  */
      /* Exercise 2.8: Your code here. (2/2) */
      tlbwi
      // 根据索引，将EntryHi,EntryLo0,EntryLo1的值写入对应的TLB表项，实现表项的删除
  .set reorder

  NO_SUCH_ENTRY:
      mtc0    t0, CP0_ENTRYHI
      // 恢复EntryHi的值
      j       ra
  END(tlb_out)
  ```

Thinking 2.6 x86体系结构中的内存管理机制

x86架构的内存管理

x86采用**段页式地址映射**，主要包括了**分段**和**分页**两种方式。

在x86架构中内存被分为三种形式，分别是**逻辑地址**，**线性地址**和**物理地址**。通过分段可以将逻辑地址转换为线性地址，而通过分页可以将线性地址转换为物理地址。

逻辑地址由两部分构成，一部分是**段选择器**，一部分是**偏移**。段选择符存放在段寄存器中，如CS（存放代码段选择符）、SS（存放堆栈段选择符）、DS（存放数据段选择符）和ES、FS、GS（一般也用来存放数据段选择符）等；偏移与对应段描述符（由段选择符决定）中的基地址相加就是线性地址。

OS提供逻辑地址，之后的分段操作x86的CPU会自动完成，并找到对应的线性地址。
  ```

从线性地址到物理地址的转换也是CPU自动完成的,转换使用的多级页表也由OS提供。

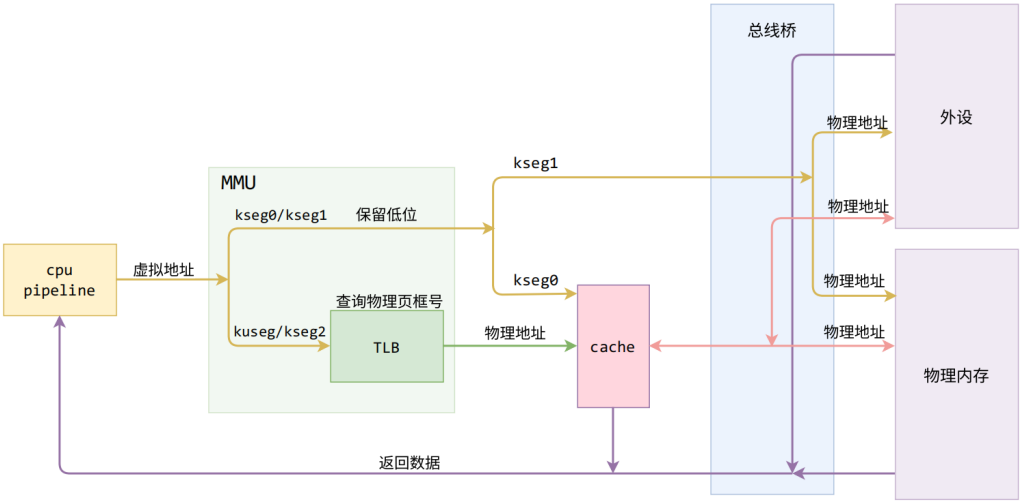
在x86架构中，分段是强制的，并不能关闭，而分页是可选的。

x86和MIPS内存管理机制的不同

- 地址管理方式
  - x86采用了分段和分页结合的内存管理机制
  - MIPS主要采用基于页的内存管理
- 页表结构
  - X86使用**多级页表结构**，将整个虚拟地址空间划分为多个层级的页表，这样可以有效管理大内存空间
  - MIPS通常使用**两级页表结构**，较为简单直接
- 逻辑地址不同
  - MIPS地址空间32位
  - x86支持64位逻辑地址，同时可以转换为32位地址

难点分析

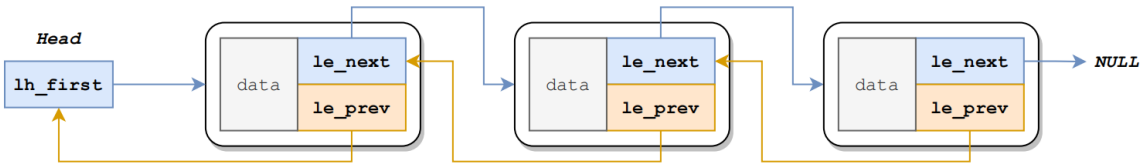
本次实验中，我认为的难点主要是**物理内存管理（链表法）**、**虚拟内存管理（二级页表）**、**TLB清除与重填。**（好吧，感觉整个实验都是难点）



从这张**cpu-tlb-memory**关系图中，我们能看出来，**MMU**在虚拟地址和物理地址之间的映射中起到了关键作用。物理内存被分为了一个个大小相等的物理页框，便于和同等大小的虚拟页面相映射。对于`kseg0/kseg1`这样高2G空间的地址，MMU可以**直接映射**到相应的物理地址，而对于用户态下的`kuseg`中的地址，MOS使用**二级页表**来管理虚拟地址和物理地址之间的映射，当然，**TLB**在其中也起到了快速查找物理页框的作用。

物理内存管理

物理内存被分成了一个个的页框，在MOS中，一共有npage个Page结构体，每个结构体对应一个物理页框。为了动态管理空闲的页框，我们使用链表宏来构建一个链表，存储着没有被分配的页框。

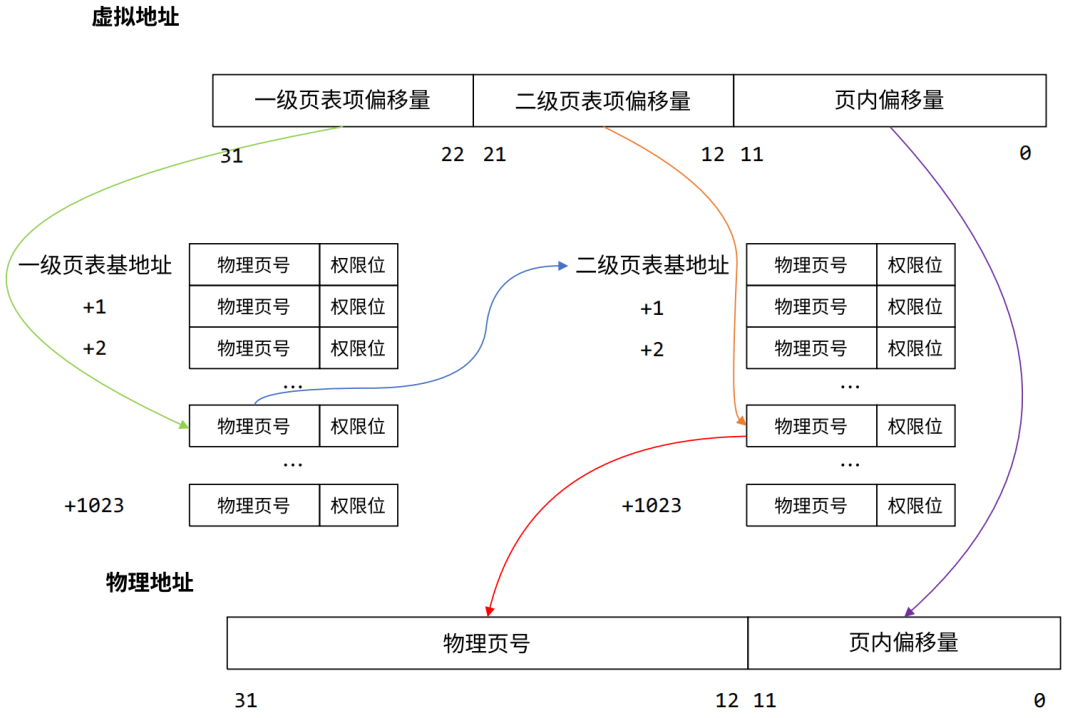


可以看出，实验中使用的是双向链表。在`include/queue.h`中有各种各样对链表操作的宏。在链表宏的基础上，以下`page`开头的一系列函数被用来进行物理内存的管理：

```
void page_init(void); // 初始化物理内存管理
page_alloc(struct Page **pp); // 分配物理页面
page_decref(struct Page *pp); // 减少物理页面引用
page_free(struct Page *pp); // 回收物理页面到空闲页面链表
// ...
```

虚拟内存管理

二级页表的变换机制：



实验体会

感觉lab2真的好难，内容又多又杂乱，指导书反反复复看了好几遍。各种各样的函数、宏、定义都需要比较熟悉才能充分地理解。

lab2的主要内容是将操作系统内核启动之后，对内存进行初始化，初始化物理内存，建立二级页表用于完成虚拟内存和物理内存之间的映射，填写TLB。做完lab2，我对MIPS的内存管理机制有了更深刻的认识。

在实验中，我再一次意识到学会读代码的重要性，实验中的任务只是让我们去补全一些代码，因此，我们首先应该读懂整个函数的功能和实现流程，对于函数中需要调用的其他函数的功能有一定的了解，然后填补空缺，串联起整个函数。整个过程刚开始我觉得是比较困难的，因为我总感觉无从下手，后来我不断地研究注释和指导书，理解函数的结构，最终才顺利地完成了实验。