Lab2

Thinking

Thinking 2.1

> Thinking 2.1 请根据上述说明，回答问题：在编写的 C 程序中，指针变量中存储的地址被视为虚拟地址，还是物理地址？MIPS 汇编程序中 lw和sw 指令使用的地址被视为虚拟地址，还是物理地址？

- 在 C 语言程序中，指针变量中存储的地址为虚拟地址。
- MIPS 汇编程序中 lw 和 sw 指令使用的地址被视为虚拟地址。

Thinking 2.2

> Thinking2.2请思考下述两个问题：
> •从可重用性的角度，阐述用宏来实现链表的好处。
> •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。

- 各类结构体都可以使用 `queue.h` 的宏简化代码量，可读性强，简化代码量，易于维护。
- 单向链表插入和删除都是O(1)复杂度，但是对于任意第 `i` 个元素的插入和删除都要遍历一遍链表；二双向链表对第 `i` 个元素的插入和删除操作时间复杂度只有O(1)；循环链表与双向链表类似

Thinking 2.3

> Thinking2.3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。

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

- 正确的展开结构如上，向前的指针应当指向前一个结点的地址。

Thinking 2.4

> Thinking 2.4 请思考下面两个问题：
> • 请阅读上面有关TLB的描述，从虚拟内存和多进程操作系统的实现角度，阐述ASID的必要性。
> • 请阅读 MIPS 4Kc 文档《MIPS32® 4K™ Processor Core Family Software User's Manual》的 Section 3.3.1 与 Section 3.4，结合 ASID 段的位数，说明 4Kc 中可容纳不同的地址空间的最大数量。

- 由于在多线程系统中，对于不同进程，相同的虚拟地址各自占用不同的物理地址空间，所以同一虚拟地址通常映射到不同的物理地址。因此TLB中装着不同进程的页表项，ASID用于区别不同进程的页表项。

  没有ASID机制的情况下每次进程切换需要地址空间切换的时候都需要清空TLB。

- ASID6位，容纳64个不同进程。

Thinking 2.5

> Thinking2.5 请回答下述三个问题：
> •tlb_invalidate和tlb_out的调用关系？
> •请用一句话概括tlb_invalidate的作用。
> •逐行解释tlb_out中的汇编代码。

- `tlb_invalidate` 是TLB失效操作的高层封装函数（通常在C代码中实现），它会调用底层的 `tlb_out` 汇编函数来完成具体的TLB表项清除操作。
- `tlb_invalidate` 的作用是**强制清除TLB中指定虚拟地址和ASID对应的表项**，确保后续内存访问能重新加载最新的映射关系（例如在页表修改后避免脏缓存）。
- ```
  LEAF(tlb_out)              ; 声明函数入口
  .set noreorder             ; 禁用指令重排序（确保流水线顺序执行）
  mfc0    t0, CP0_ENTRYHI    ; 保存当前CP0_ENTRYHI（VPN+ASID）到t0，用于恢复
  mtc0    a0, CP0_ENTRYHI    ; 将参数a0（目标VPN+ASID）写入CP0_ENTRYHI
  nop                        ; 空操作（延迟槽，等待CP0更新）
  ```

Thinging 2.6

> Thinking 2.6 请结合 Lab2 开始的 CPU 访存流程与下图中的 Lab2 用户函数部分，尝试将函数调用与CPU访存流程对应起来，思考函数调用与CPU访存流程的关系。

函数调用与 CPU 访存流程的对应关系（从指令执行，栈操作，内存访问三个角度来分析）：

1. 函数调用前的参数传递时 CPU 动作：调用者（如 main 函数）将参数压入栈或存入寄存器（取决于调用约定）；访存操作：通过 push 或 mov 指令将参数写入栈内存（Store 操作）
2. 函数调用时的返回地址保存 CPU 动作：执行 call 指令时，CPU 自动将返回地址（下一条指令的 eip）压栈；访存操作：隐式的 push eip（Store 操作），修改栈指针 esp
3. 函数内部的局部变量访问时 CPU 动作：被调用函数（如 foo）通过栈指针 ebp 访问参数和局部变量，访存操作：
4. 加载参数：mov eax, [ebp+8]（Load 操作）
5. 存储局部变量：mov [ebp-4], eax（Store 操作）
6. 函数返回时的栈帧恢复时 CPU 动作：执行 ret 指令时，CPU 从栈中弹出返回地址到 eip，访存操作：隐式的 pop eip（Load 操作）
   函数调用对 CPU 访存流程的影响：
   栈内存的频繁访问：
7. 函数调用通过栈传递参数、保存返回地址和局部变量，导致大量的 Load/Store 操作；栈指针（esp/ebp）的修改是访存的核心
8. 访存局部性：栈操作具有空间局部性，CPU 缓存（Cache）能有效加速栈访问
9. 流水线冲突风险：频繁的栈内存访问可能导致数据冲突（如 push 和 pop 依赖 esp），需流水线停顿或乱序执行优化
  ```

Thinking 2.7

> Thinking 2.7 从下述三个问题中任选其一回答：
> • 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。
> • 简单了解并叙述LoongArch 中的内存管理机制，比较 LoongArch 与 MIPS 在内存管理上的区别。

X86 体系结构中的内存管理机制

1. 通过分段将逻辑地址转换为线性地址，通过分页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
2. 偏移与对应段描述符中的基地址相加就是线性地址，操作系统创建全局描述符表和提供逻辑地址，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是CPU自动完成的，转化时使用的Page Directory和Page Table等需要操作系统提供
   X86 和 MIPS 在内存管理上的区别
3. TLB不命中的处理：MIPS触发TLB缺失和充填，然后CPU重新访问TLB；x86硬件MMU索引获得页框号，直接输出物理地址，MMU充填TLB加快下次访问速度
4. 分页方式不同：一种MIPS系统内部只有一种分页方式；x86的CPU支持三种分页模式
5. 逻辑地址不同：MIPS地址空间32位；x86支持变长地址，逻辑地址为64位，同时提供转换为32位定址选项
6. 段页式的不同：MIPS同时包含了段和段页式两种地址使用方式，在x86架构的保护模式下的内存管理中，分段是强制的，并不能关闭，而分页是可选的

难点分析

- 本次实验难度陡然上升，我的 `exam` 都是想了很久才磨出来。到 `lab2` 了，逐渐进入操作系统的深层次部分，这要求我们对各个文件，各个函数，各个变量都要了解透彻。只有在课下把excercise好好做出来，并且理解透彻，才能快速掌握课上的题目要求。本次实验 `exam` 相较于往年多了反求虚拟地址，只要能够好好理解 `va` 和 `*pte` 之间的关系，还是能够做出来的。而题目的整体框架基本没变，就是要遍历出所有满足条件的页表项并取消映射，可以借助已有的 `page_remove` 来解决。题目整体的框架如下。

```
//已知Pde *pgdir
for (u_long i = 0; i < 1024; i++) { //遍历页目录的1024项
    Pde *pde = pgdir + i; //第i个页目录项对应的虚拟地址
    if ((*pde) & PTE_V) { //第i个页表有效
        for (u_long j = 0; j < 1024; j++) { //遍历第i个页表的1024项
            Pte *pte = (Pte*)KADDR(PTE_ADDR(*pde)) + j; //第j个页表项对应的虚拟地址
            if ((*pte) & PTE_V)) { //第j个页有效
                // 将满足要求的映射取消并计数
            }
        }
    }
}
```

- 上述问题有几个关键点，一个是怎么遍历所有页表项，这一点在上述代码中已经给出。还有就是怎么用过页表项来求 `va` ，只要正确理解物理页号和偏移量即可。最后是有效位，权限位的检查，这些检查方式大同小异，好好看作业就可以。

实验体会

- 通过Lab2的学习与实践，我对操作系统的内存管理机制有了更立体的理解。在实现物理内存检测、页表管理和TLB处理的过程中，深刻体会到虚拟内存背后的工程复杂性——从物理页帧的初始化分配，到多级页表的自映射设计，每一个函数背后都是硬件特性与软件逻辑的精密配合。编写 `page_alloc` 时感受到物理内存的稀缺性管理，调试 `pgdir_walk` 时领悟到虚拟地址到物理地址的转换如同解谜，而实现 `tlb_out` 则让我意识到硬件缓存（TLB）对性能的关键作用。
- 操作系统的启动是一层套一层慢慢向上累加。在对页面操作时，我们用到了两种函数，一种是在内核刚刚启动的时候，这一部分内存通常用于存放内存控制块和进程控制块等数据结构，只能使用基础的 `alloc` ，一种则是在为用户创造环境之后按根据用户申请页面的需要进行操作。这体现了操作系统启动中每一步都是紧密相连的。