

Lab2

班级: 222111

学号: 22373340

姓名: 詹佳博

思考题

Thinking 2.1

- 虚拟地址; 虚拟地址。

Thinking 2.2

请从可重用性的角度, 阐述用宏来实现链表的好处。

- C语言中**没有泛型**这个概念。在学习 Java 的时候, 我了解到**泛型**能够让我们更加面向需求的来运用相关的数据结构类型。结合大一上数据结构的经历, 我经常会碰到要使用不同数据类型的链表、栈等, 这个时候就得**定义多个结构体**, 甚至操作函数代码也得**重复写多遍**。
- **用宏实现链表**, 可以巧妙的实现“泛型”, 也可以提高代码的效率。

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 in 插入与删除操作上的性能差异。

- 单向链表在环境中分为有尾和无尾。单向链表的插入操作的时间复杂度为 $O(1)$, 删除操作的时间复杂度为 $O(n)$;
 - `SLIST` 为单向无尾链表。进行插入操作时只需要将新插入节点的指针 `field.sle_next` 指向要插入的位置的下一个节点, 并将要插入的上一个节点的指针 `field.sle_next` 指向它即可; 删除时跟插入操作基本相反。
 - `STAILQ` 为单向有尾链表。它的删除与插入操作相比 `SLIST` 多了对尾部的操作, 在处理时多了 `if` 是尾部的判断, 其余基本相同。
- `CIRCLEQ` 为循环链表。它的结构相比单向链表多一个最后一个节点的指针指向头节点。循环链表的插入操作的时间复杂度为 $O(1)$, 删除操作的时间复杂度为 $O(n)$;
- 双向链表的插入操作的时间复杂度为 $O(1)$, 删除操作的时间复杂度为 $O(1)$;

Thinking 2.3

- C:

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

Thinking 2.4

- ASID唯一标识每个进程，并用于为该进程提供地址空间保护，当TLB尝试解析VPN时，它确保当前正在运行的进程的ASID与VPN关联的ASID匹配。如果ASID不匹配，则将该尝试视为TLB未命中。ASID 使得操作系统能够在不同进程之间快速切换，同时保持各自的地址映射信息。
- ASID是一个8位的字段。这意味着理论上 有 2^8 即 256 个不同的地址空间，也即最大数量为 256。

Thinking 2.5

- `tlb_invalidate` 调用了 `tlb_out`
- `tlb_invalidate` 函数实现更新页表中虚拟地址对应的页表项的同时，将 TLB 中对应的旧表项无效化。

- ```
LEAF(tlb_out)
.set noreorder
 mfc0 t0, CP0_ENTRYHI # 从CP0取EntryHi数据，写入t0
 mtc0 a0, CP0_ENTRYHI # 往CP0的EntryHi写入数据a0
 nop # 待前面指令退出流水线
 tlbp # 根据EntryHi中的Key查找对应的旧表项，将表项的索引存入Index
 nop # 待前面指令退出流水线
 mfc0 t1, CP0_INDEX # 从CP0取Index数据，写入t1
.set reorder
 bltz t1, NO_SUCH_ENTRY # 如果Index数据小于0，则不存在条目，跳转到NO_SUCH_ENTRY
.set noreorder
 mtc0 zero, CP0_ENTRYHI #
 mtc0 zero, CP0_ENTRYLO0 # 存在条目的情况下，向EntryHi，EntryLo0和EntryLo1写入0
 mtc0 zero, CP0_ENTRYLO1 #
 nop # 待前面指令退出流水线
 tlbwi # 将EntryHi、EntryLo0和EntryLo1中的0写入索引指定的表项
.set reorder
NO_SUCH_ENTRY:
 mtc0 t0, CP0_ENTRYHI # 往CP0的EntryHi写入数据t0
 j ra # 返回
END(tlb_out)
```

## Thinking A.1

- 三级页表的基地址为  $PT_{base}$ ，则中间页表基地址为  $(PT_{base} \gg 12) * 8 + PT_{base} = PT_{base} \gg 9 + PT_{base}$ ，则三级页表页目录基地址为  $(PT_{base} \gg 9 + PT_{base}) \gg 9 + PT_{base}$ ，也就是  $PT_{base} \gg 18 + PT_{base} \gg 9 + PT_{base}$
- 中间页表基地址为  $PT_{base} \gg 9 + PT_{base}$ ，三级页表页目录基地址为  $PT_{base} \gg 18 + PT_{base} \gg 9 + PT_{base}$ ，位移距离为  $PT_{base} \gg 18$ ，故而有  $(PT_{base} \gg 18) / 4K = PT_{base} \gg 30$  项。故而映射到页目录自身的页目录项（自映射）为  $(PT_{base} \gg 18 + PT_{base} \gg 9 + PT_{base}) + PT_{base} \gg 30 * 8$ ，也就是  $PT_{base} \gg 27 + PT_{base} \gg 18 + PT_{base} \gg 9 + PT_{base}$

