Project 4 Virtual Memory 设计文档

中国科学院大学
秦宏
2018.12.24

1. 内存管理设计

（1）页表项本身是一个结构体，页表项中包含了虚拟页号、物理页号、valid 位、占用该页表项的进程 pid 号，页表本身设计为一个页表指针，目前一共设计了 0x80000 项页表项。

（2）任务一初始化了 32 个 TLB 项，因此对应了 32 个页表项，同时产生了 32 个对应的物理页框相对应。其中虚拟页号从 0x00000000 开始，大小为 4KB，物理地址从 0x01000000 开始，大小 4KB。物理页框一共有 0x1000 个，因此任务二初始化了 0x1000 个页表项。

（3）任务 2 和任务 3 的用户态栈起始地址为 0x70000000,大小为 0x10000。一共在除了 pid=0 和 shell 进程外的 14 个 pid 项分配了 14 个用户栈。

（4）任务 1 中初始化了 32 个 TLB 项，自物理地址 0x01000000 开始，初始化了 32 个 4KB 大小的物理页框，共 128KB 大小。

任务 2 中初始化页表项，则页表项中容纳了 4K 个物理页框，即覆盖了 0x01000000 到 0x02000000 的物理地址空间，共 16MB。

（5）系统获得虚拟地址后在 TLB 项中查找 32 个 TLB 项后没有找到 TLB 项中存在对应虚拟页号的 TLB 项时发生 TLBmiss，在任务 2 中，发生 TLBmiss 是由于 TLB 项中没有对应虚拟地址的项，只需要在已有的页表项中找到与虚拟地址对应的页表项，将虚拟页号和固定分配的物理页号写入 TLB 项中即可，注意用 TLB 指令中的 tlbwr。

（6）

（i）其实任务书中并没有提到 tlbwr 指令的使用，我是询问同学才知道要使用 tlbwr 指令。之前一直以为是只需要使用 tlbwi 即可。

（ii）一开始做实验时同学推荐我跳过任务二来做，明白了任务二的含义后，发现确实没有做任务二的必要，如果是清楚了 TLB 的工作过程后。

（iii）CP0_ENTRYHI 寄存器写入数据时，前面的虚拟页号不是除以 2，这个卡了我好久，我是自己试了两种虚拟页号的存储方式才发现应该是虚拟页号的最后一位置 0 即可，不需要除以 2，这个是看任务书的时候理解错了。

2. 缺页处理设计

（1）任务三中并没有初始化页表项，因此发生 TLBmiss 后页表项中也查不到所需的页表项，这时候发生缺页，需要新建物理页与虚拟页的对应关系。缺页处理流程附加在 TLBrefill 处理中，流程主要为在页表项中找到虚拟页号对应的页表项，但是此时的 asid 值与当前 pid 不一致，原因是在进程调度切换时进行了对 entryhi 寄存器中 asid 的修改，导致再次切换回该进程时触发缺页，此时只需要再次修改 asid 就可以恢复 TLB 查找功能，无例外产生。

（2）存储页表项的内容可以设计在映射地址上，这时存储的页表项是一直存在且固定分配的，页替换策略采用先进先出的方式，由于物理页框的分配是递增的，当需要页替换时，选取物理页框最小的 TLB 项替换掉。Bonus 中的页替换并没有实现，初步想法是采用 FIFO 机制，将 TLB 项中的虚拟页内容不是移除，而是存入磁盘。

（3）实验中的过程设计还算顺利，只是之前 ENTRYHI 寄存器的写入一直卡了好长时间不知道如何处理，找不到错误发生的地方。

3. Bonus 设计

没有设计 bonus。

4. 关键函数功能

第一次分配 TLB 时的函数

```
void TLBrefill(void){

    uint32_t context = get_cp0_context();
    uint32_t index   = get_cp0_index();
    uint32_t entryhi = get_cp0_entryhi();
    uint32_t entrylo0 = get_cp0_entrylo0();
    uint32_t badvaddr = get_cp0_badvaddr();
    int i = 0;
    int j = 0;
    int k = 0;
    int temp = 0;
    int refill = 0;
    uint32_t asid = 0x000000ff & get_cp0_entryhi();
    uint32_t con_vir_addr = (context & 0x007ffff0)<<9;

    // debuginfo(1234);
    // printkf("\n%x \n%x \n%x \n%x
\n%x",badvaddr,context,index,entryhi,entrylo0);
    for (i = 0;(i < pte_number)&&(refill != 1);i = i + 2){
        // printk("\n%d",i);
        if((con_vir_addr ==
page_table[i].virtual_number)&&(page_table[i].valid == 1)){
                refill = 1;
                i = i - 2;// debuginfo(i);
            if(asid != page_table[i].pid){
                // debuginfo(badvaddr);
                printk("address overload %x",context);
                // while(1){}
            }
        }
        // printk("\n%d",i);
    }

    if(refill == 0){
        count++;
        for(i = 0;i < pte_number;i = i + 2){
```

```
            if((page_table[i].valid == 0)&&(page_table[i].virtual_number
== con_vir_addr)){
                // page_table[i].virtual_number = con_vir_addr;
                page_table[i].valid = 1;
                page_table[i+1].valid = 1;
                break;
            }
        }
        page_table[i].physical_number = phy_addr;
        page_table[i+1].physical_number = phy_addr + 0x1000;
        phy_addr = phy_addr + 0x2000;
        page_table[i].pid = (uint32_t)current_running->pid;
        page_table[i+1].pid = (uint32_t)current_running->pid;
    }
    // printk("%d",count);
    // printk("\n%x \n%d \n%x \n%x
\n%x",con_vir_addr,current_running->pid,i,page_table[i].virtual_number,
page_table[i+1].physical_number);
    // debuginfo(1234);

set_TLBrefill(page_table[i].virtual_number,page_table[i].physical_numbe
r>>6,current_running->pid);
    TLB_back();
```

分配对应的虚拟页号和物理页号。

```
    if(current_running->pid > 1)
        TLB_idchange(current_running->pid);
```

除了 shell 进程和 0 号进程之外的进程都需要进行 entryhi 寄存器的 asid 设置在 sched.c 中。

参考文献

[1] 无