Project 4 Virtual Memory 设计文档

中国科学院大学
葛忠鑫
2020.12.12

1. 内存管理设计

（1）你设计的页表是几级页表，最大能索引到多大的物理空间？页表项的数据结构是什么？页表本身的数据结构是什么？ 你设计的页表需要几个物理页框保存？

设计的页表是一级页表，共 512 项。最大能索引 4MB 的空间。页表项按照 entylo 寄存器组织，并对应一个 TLB 项，包含奇偶两个映射，其数据结构如下：

```
// page table entry
// 31                                6 5  3  2   1   0
// +-----------------+---------------+----+---+---+---+
// |                PFN                |  C | D | V | G |
// +-----------------+---------------+----+---+---+---+
typedef struct PTE
{
    uint32_t entrylo0;
    uint32_t entrylo1;
} PTE_t;
```

页表本身是一个一维数组，以下标作为虚拟页号。页表放在 PCB 中，每个进程的页表大小为 4KB，需要一个物理页框进行存储。

（2）在 A-Core 中，你处理 TLB miss 的流程是怎样的？（for MIPS 同学）

当在 TLB 中找不到项与欲引用的地址映射匹配时发生 TLB refill 例外，当虚拟地址引用与 TLB 中某一项相匹配，但该项标记为无效时，发生 TLB invalid 例外。

TLB miss 具体情况有：无 VPN 匹配的 TLB 项、VPN 匹配但 G 位不为 1 且 ASID 不和当前 PID 相等、以及有匹配项但 V 位为 0。

处理流程：

a) 利用 tlbp 指令查找更新 index 寄存器，并根据 index 寄存器的 P 位判断需要 refill 还是 invalid。P 位为 1 表示没有找到对应的 TLB，此时进行 refill 例外处理。
b) 如果是 refill，就直接设置相应的 CP0 寄存器，然后写入 TLB。
c) 如果是 invalid，则去查询相应的页表项是否有物理页框对应，如果有，则直接将 TLB 设置为有效，如果没有，则需要分配物理页框，如果物理页框没有了，需要进行页替换。

具体实现代码如下：

```
void do_TLB_Refill()
{
    uint32_t entrylo0, entrylo1;
    uint64_t context = get_cp0_context();
```

```
    tlbp_operation();
    uint32_t index = 0;
    if (get_cp0_index() & 0x80000000) {
        // TLB refill
        set_cp0_index(index);
        index = (index + 1) % NUM_MAX_TLB;
    }
    else {
        // TLB invalid
        if(!(current_running->page_table[context >> 4 & (NUM
_MAX_PTE - 1)].entrylo0 & 0x2) ||
           !(current_running->page_table[context >> 4 & (NUM
_MAX_PTE - 1)].entrylo1 & 0x2)){
            do_page_fault();
        }
    }
    entrylo0 = current_running->page_table[context >> 4 & (N
UM_MAX_PTE - 1)].entrylo0;
    entrylo1 = current_running->page_table[context >> 4 & (N
UM_MAX_PTE - 1)].entrylo1;
    set_cp0_entrylo0(entrylo0);
    set_cp0_entrylo1(entrylo1);
    tlbwi_operation();
}
```

（3）设计或实现过程中遇到的问题和得到的经验（如果有的话可以写下来，不是必需项）

a. 在将用户栈实现到 mapped 区域后，一旦进入内核，要记得切换栈指针 sp 为内核栈。

b. 由于例外入口距离内核函数较远，jal 不能成功跳转，需要结合使用 dla 和 jr 指令完成。

```
da
NESTED(TLBexception_handler_entry, 0, sp)
TLBexception_handler_begin:
    dla k0, handle_tlb
    jr k0
TLBexception_handler_end:
END(TLBexception_handler_entry)
```

2. 缺页处理设计

（1）何时会发生缺页处理？你设计的缺页处理流程是怎样的？

缺页处理发生在 TLB 无效，且对应的页表项无效时（因为任务三中初始化页表为空）。

缺页处理流程：由于实验测试需要地址空间不大，尚未没有实现物理内存回收，所以从 0.5G 的物理地址后开始分配，并设置静态变量 PFN 指示空闲物理空间的起始。代

码如下：

```
1.  void do_page_fault()
2.  {
3.      static uint64_t PFN  = 0x20000;
4.      uint64_t context = get_cp0_context();
5.      current_running->page_table[context >> 4 & (NUM_MAX_PTE
    - 1)].entrylo0 = (PFN << 6) | (PTE_C << 3) | (PTE_D << 2) |
    (PTE_V << 1);// | PTE_G;
6.      PFN ++;
7.      current_running->page_table[context >> 4 & (NUM_MAX_PTE
    - 1)].entrylo1 = (PFN << 6) | (PTE_C << 3) | (PTE_D << 2) |
    (PTE_V << 1);// | PTE_G;
8.      PFN ++;
9.  }
```

（2）你使用什么数据结构管理物理页框，管理多少物理页框（例如管理哪些地址范围内的物理页框）？在缺页分配时，按照什么策略或原则进行物理页框分配？

一个静态变量。

（3）你的设计中是否有 pinned 的物理页框？若有，具体是保存什么内容的物理页框？

没有实现 pinned 的物理页框。

（4）设计或实现过程中遇到的问题和得到的经验（如果有的话可以写下来，不是必需项）

3. C-Core 设计（做 C-Core 的同学需要写）

尚未完成。

请至少包含以下内容

（1）你设计的操作系统通过页表访问的可用物理内存是多少？ swap 操作是由专门的进程完成么？

（2）你设计的页替换策略是怎样的，有什么优势和不足么？

（3）你设计的测试用例是怎样的？

（4）设计或实现过程中遇到的问题和得到的经验（如果有的话可以写下来，不是必需项）

4. 关键函数功能

参考文献

[1] 龙芯 GS264 处理器核用户手册