Project 4 Virtual Memory 设计文档

中国科学院大学
[薛峰]
[2018/12/8]

1. 内存管理设计

（1）你设计的页表项包含哪些内容？页表本身使用什么数据结构保存？

PTE 结构体定义如右图所示

其包含的内容有：

VPN：虚页号；

PFN：物理页框号；

pid：占据改页表项的进程的 id 号；

valid：该页表项是否有效。

页表本身采用数组保存。

```
typedef struct {
  // design here
  uint32_t VPN;
  uint32_t PFN;

  pid_t pid;
  int8_t valid;

} PTE_t;
```

（2）任务 1 和任务 2 中各自初始化了多少个页表项，以及使用了多少个物理页框保存页表？

任务 1 不需要初始页表项，直接设置好 32 个 TLB 项，因此不需要物理页框保存页表；

任务 2 中我只使用了 1024 个页表项（页面大小为 4K，因此可用的虚拟地址空间为 0x0~0x800000），因为一个页表项大小为 16B，因此需要 4 个物理页框保存页表。

（3）任务 2 和任务 3 中，进程的用户态栈的起始地址各是多少，栈空间各是多大？

任务 2 中，用户栈的起始地址为 0xa0f00000；任务 3 中，用户栈的起始地址为 0x7ffffff0，两个任务的栈空间都分配为 4k。

```
#define STACK_TOP_K  0xa0f00000
#define STACK_TOP_U  0x7ffffff0

#define STACK_SIZE   0x1000
```

（4）任务 1 和任务 2 中你设计的操作系统实际通过页表可以访问到的物理内存有多大？

任务 1 中，固定了 32 个 TLB 项，每个 TLB 项对应两个物理页框，因此可以访问 64 个物理页框，每个页框大小为 4K，因此可访问的物理内存大小为 256K。任务 2 中，使用了 1024 个页表项，对应 2K 个页表项，因此，可访问的物理空间为 8M。

（5）TLB miss 何时发生？在任务 2 中，你处理 TLB miss 的流程是怎样的？

要访问的虚拟地址不在 unmapped 区，且 tlb 里没有这个进程的这个虚拟页对应的 TLB 表项，这时就会产生 TLB miss 例外。

流程：首先遍历页表，找到该进程的这个虚拟页对应的 TLB 表项，随后用 VPN 和 PID 设置 CP0_ENTRYHI 寄存器在 TLB 里寻找对应的表项，若没找到说明是 TLB 充填例外，则将新的页表项在 TLB 中随意找一个位置填充，若找到，则说明是 TLB 无效例外，则将新的页表项填充到该位置上。

（6）设计或实现过程中遇到的问题和得到的经验。

遇到的问题：

任务一较为简单，因此写代码的时候未遇到什么问题（跳过任务二，直接做的任务三）。

得到的经验：

通过任务一的实现，我了解的 MIPS 是如何实现 TLB 的管理的，之前上体系结构理论课的时候老师有讲过 CP0_EntryHi、EntryLo 等寄存器的功能和 TLB 的管理，但是当时没有听懂。通过这次试验，我才真正了解其 TLB 机制。

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，此处的物理页分配策略是什么？

如果页表中找不到该进程的虚地址对应的页表项时，就会发生缺页处理。

流程：首先从页表项找到一个空页表项，再从物理内存中找到一个空的页框，将该虚拟地址对应的页表项与该物理页框建立映射，即将该页表项的 VPN 设置为该虚拟地址的 VPN 号，将 PFN 设置为该页框号，pid 设置为该进程的 id 号，并且 valid 设置为 1。因为一个 TLB 项对应两个页表项，因此一次需要处理两个页表项。

分配策略：从低地址的页框开始查找，将找到的第一个空闲的页框进行分配。

（2）你设计中哪些页属于 pinning pages？你实现的页替换策略是怎样的？

我的设计中没有 pinning pages；

替换策略：设置一个 static 变量 id，初始化为 0，将该变量设置为 index 的值，每次向 tlb 中填充一个页表项的时候该变量加 1，这样可以循环替换 TLB 中的 32 个项。

（3）设计或实现过程中遇到的问题和得到的经验。

遇到的问题：

一、开始的时候不知道如何让硬件比较 PID，并且最开始发现硬件在进行 TLB 操作的时候，并没有关心 TLB 表项中的 ASID 域。后来询问同学后知道，需要将 TLB 表项中的 G 设置为 0，这样硬件才会比较 TLB 表项中的 AISD 域是否和 EntryHi 寄存器中的 ASID 相同。

二、刚开始分配栈帧的时候，会报 TLB 例外，此时 29 号寄存器已经是虚拟地址，进入例外处理程序之后，28 号寄存器并没有更新，因此还会发生例外，所以会不断循环。因此，在进入到 0x80000000 时，需要将 pcb 中内存态的 29 号寄存器的值赋给 29 号寄存器。

得到的经验：

通过该实验，我最大的收获是知道了操作系统是如何实现内存保护。

3. 关键函数功能

（1）TLB 例外处理入口函数：

功能：先将 pcb 中内核态的 29 号寄存器赋给 29 号寄存器，随后跳转的 0x80000180.

```
.global TLBexception_handler_begin
.global TLBexception_handler_end

NESTED(TLBexception_handler_entry, 0, sp)
TLBexception_handler_begin:

  la  k0, current_running
  lw  k0, 0(k0)
  lw    $29, OFFSET_REG29(k0)

  li    k0, 0x80000180
  jr    k0
TLBexception_handler_end:
END(TLBexception_handler_entry)
```

（1）TLB 例外处理程序，tlb_helper()。

功能：处理 TLB 例外，包括 TLB 充填例外，TLB 无效例外，缺页。其处理流程已在之前描述过。

```
void tlb_helper()
{
  static id = 0;

  int index;
  int i, P;
  uint32_t VPN = (current_running->user_context.cp0_badvaddr & 0xffffe000) >> 12;
  uint32_t PID = current_running->pid;

  for(i = 0; i < PTE_NUMBER; i++)
  {
    if( PTE[i].VPN == VPN && PTE[i].pid == PID && PTE[i].valid == 1 )
      break;
  }

  if(i == PTE_NUMBER ) // page fault
    i = handle_page_fault(VPN);

  index = search_TLB(PTE[i].VPN << 12 | PID);
  P = index >> 31;

  if(P)  // not found
  {
    id ++;
    id = id%31;
    set_cp0_Index( id );
  }

  set_EntryLo0(PTE[i  ].PFN << 6 | 0x16);
  set_EntryLo1(PTE[i+1].PFN << 6 | 0x16);
  set_PageMask();

  return;
}
```

（2）int handle_page_fault(uint32_t VPN)函数

功能：从页表中寻找一个空的页表项，从物理页框中找到一个空闲的页框，并将其建立映射关系。返回该页表项的 index。

```
int handle_page_fault(uint32_t VPN)
{
  int i, j;
//  static x = 1;
  for(i = 0; i < PTE_NUMBER; i++)
  {
    if( PTE[i].valid == 0)
    break;
  }
  if(i == PTE_NUMBER)
  {
    printk("Error: There is no PTE to allocate\n");
    while(1);
  }

  for(j = 0; j < PF_NUMBER ; j++)
  {
    if( PF[j] == 0)
      break;
  }

  if(j == PF_NUMBER)
  {
    printk("Error: There is no PF to allocate\n");
    while(1);
  }

  PTE[i].VPN   = VPN;
  PTE[i].PFN   = j;
  PTE[i].pid   = current_running->pid;
  PTE[i].valid = 1;

  PTE[i+1].VPN   = VPN + 1;
  PTE[i+1].PFN   = j + 1;
  PTE[i+1].pid   = current_running->pid;
  PTE[i+1].valid = 1;

  PF[j  ] = 1;
  PF[j+1] = 1;

  // For Debug
/*  vt100_move_cursor(x, 6);
  printk("PF: i=%d, j=%d",i, j);
  x += 15;*/
  return i;
}
```