OS拾贝-3 ： TLB重填流程

TLB 的重填过程由 do_refill 函数（lib/genex.S）完成，相关流程我们在介绍两级页表时已经有所了解，但当时并没有涉及 TLB 部分，加入 TLB 后，整个流程大致如下：

1. **确定此时的一级页表基地址：**mCONTEXT 中存储了当前进程一级页表基地址位于 kseg0 的虚拟地址；

   通过自映射相关知识，可以计算出对于每个进程而言，0x7fdff000 这一虚拟地址也同样映射到该进程的一级页表基地址，但是重填时处于内核 态，如果使用 0x7fdff000 则还需要额外确定当前属于哪一个进程，使用位于 kseg0 的虚拟地址可以通过映射规则直接确定物理地址。

2. 从 BadVaddr 中取出引发 TLB 缺失的虚拟地址, 并确定其对应的一级页表偏移量（高 10 位）；
3. 根据一级页表偏移量, 从一级页表中取出对应的表项：此时取出的表项由**二级页表基地址的物理地址与权限位组成；**
4. 判定权限位: 若权限位显示该表项无效（无 PTE_V ）, 则调用 page_out ，随后回到第一步；
5. 确定引发 TLB 缺失的虚拟地址对应的二级页表偏移量（中间 10 位），与先前取得的二级页表基地址的物理地址共同确认二级页表项的物理地址；
6. 将**二级页表项物理地址转为位于 kseg0 的虚拟地址**（高位补 1），随后页表中取出对应的表项：**此时取出的表项由物理地址与权限位组成**；
7. 判定权限位: 若权限位显示该表项无效（无 PTE_V），则调用 page_out ，随后回到第一步；（PTE_COW 为写时复制权限位，将在 lab4 中用到，此时将所有页表项该位视为 0 即可）
8. 将物理地址存入 EntryLo , 并**调用 tlbwr 将此时的 EntryHi 与 EntryLo 写入到 TLB 中**（EntryHi 中保留了虚拟地址相关信息）。

其中第 4 步与第 7 步均可能调用 **page_out 函数**（mm/pmap.c）来**处理页表中找不到对应表项的异常**

do-refill

```
    .extern tlbra                       # 外部引入这个变量tlbra
.set    noreorder                       # 禁止下重新排序
NESTED(do_refill,0 , sp)                # 定义do_refill函数
            //li    k1, '?'
            //sb    k1, 0x90000000
##1. 确定此时的一级页表基地址：mCONTEXT 中存储了当前进程一级页表基地址位于 kseg0 的虚拟地址;
            .extern mCONTEXT            # 外部引入mCONTENT变量

//this "1" is important
1:          //j 1b
            nop
            lw      k1,mCONTEXT         # 将 mCONTENT 的值存入 k1
            and     k1,0xfffff000       # 将 k1 的值与运算0xfffff000的结果存入k1(低12位置零)
##2. 从 BadVaddr 中取出引发 TLB 缺失的虚拟地址，并确定其对应的一级页表偏移量（高 10 位）；
            mfc0        k0,CP0_BADVADDR # 将 BADVADDR 寄存器中的值存入 k0
            srl     k0,20               # 将 k0 中的值右移20位
            and     k0,0xfffffffc       # 将 k0 的值低2位置零
# 此时，k0中存放的就是一级页表偏移量×4的结果，正好是页表项个数×页表项大小4字节，得到了实际上的地址偏移量
            addu        k0,k1           # 将 k0 + k1 的值存入k0
##3. 根据一级页表偏移量，从一级页表中取出对应的表项：此时取出的表项由二级页表基地址的物理地址与权限位组成;
            lw      k1,0(k0)            # 将此时k0中地址 存放的内容存入k1
```

```
# 此时 k1中就是一级页表项中存放的内容
##4. 判定权限位: 若权限位显示该表项无效（无 PTE_V ），则调用 page_out ，随后回到第一步;
            nop
                    move        t0,k1    # k1 -> t0
                    and     t0,0x0200    # PTE_V 0x0200，这里就是检查一级页表项的权限
位（有效位）
                    beqz        t0,NOPAGE    # 如果为0，说明一级页表项无效，那么跳转到
NOPAGE
                                             # 在NOPAGE中跳回 pageout
            nop
##5. 确定引发 TLB 缺失的虚拟地址对应的二级页表偏移量（中间 10 位），与先前取得的二级页表基地
址的物理地址共同确认二级页表项的物理地址;
            and     k1,0xfffff000        # k1 低12位置零
# 此时，k1中存放的就是对应的二级页表基地址的物理地址
            mfc0    k0,CP0_BADVADDR      # 将 BADVADDR 寄存器中的值存入 k0
            srl     k0,10                # 将 k0 中的值右移10位
            and     k0,0xfffffffc        # 将 k0 的值低2位置零
            and     k0,0x00000fff        # 将 k0 的值高20位置零
# 此时，k0中存放的就是二级页表偏移量×4的结果，正好是页表项个数×页表项大小4字节，得到了实际上的
地址偏移量

            addu        k0,k1            # 将 k0 + k1 的值存入k0
                                # 此时 k0 存放的就是二级页表项的物理地址
##6. 将二级页表项物理地址转为位于 kseg0 的虚拟地址（高位补 1），随后页表中取出对应的表项：此
时取出的表项由物理地址与权限位组成;
            or      k0,0x80000000        # 将 k0 的最高位 置 1
                                # 此时，k0中存放的就是二级页表项的虚拟地址
            lw      k1,0(k0)             # 将此时k0中地址 存放的内容存入k1
##7. 判定权限位: 若权限位显示该表项无效（无 PTE_V），则调用 page_out ，随后回到第一步;
（PTE_COW 为写时复制权限位，将在 lab4 中用到，此时将所有页表项该位视为 0 即可）
            nop
                    move        t0,k1    # k1 -> t0
                    and     t0,0x0200    # PTE_V 0x0200，这里就是检查二级页表项的权限
位（有效位）
                    beqz        t0,NOPAGE    # 如果为0，那么跳转到NOPAGE
                                             # 在NOPAGE中跳回 pageout
            nop
            move        k0,k1    # k1 -> k0
            and     k0,0x1       # PTE_COW 0x0001，这里就是检查二级页表项的权限位（写时
复制位）
            beqz        k0,NoCOW     # 如果为0，那么跳转到NOCOW
            nop
##8. 将物理地址存入 EntryLo ，并调用 tlbwr 将此时的 EntryHi 与 EntryLo 写入到 TLB 中
（EntryHi 中保留了虚拟地址相关信息）
            and     k1,0xffffbff     # 将k1（二级页表项）的第10位（即可写位PTE_R
0x0400）置零
                                     # 即现在的二级页表项是 只读
NoCOW:
            mtc0    k1,CP0_ENTRYLO0 # 将k1寄存器的值（即物理页号+权限位）存入 EntryLo
            nop
            tlbwr                    # 调用 tlbwr 将此时的 EntryHi 与 EntryLo 写入到
TLB 中

            j       2f               # 调转到下面的 2 标签处 f 即 forward
            nop
NOPAGE:
//3: j 3b
nop
            mfc0    a0,CP0_BADVADDR      # 将 BADVADDR 寄存器中的值存入 a0
            lw      a1,mCONTEXT          # 将 mCONTEXT 存入 a1
            nop

            sw      ra,tlbra             # 将 ra 寄存器中的值存入 tlbra 这个全局变量
            # 这里存放的 ra 是 do_refill 函数执行的下一步
            jal     pageout              # 调用 page_out函数，这里是C函数，
```

```
                                        # ra被写入返回地址，即 pageout 函数执行的下一步
                                        # C函数转为汇编后，会在结尾加一个jr ra 跳回来
            nop
//3: j 3b
nop
            lw      ra,tlbra            # 将 tlbra 这个全局变量的值存入 ra 寄存器中
            nop

            j   1b                      # 跳转回到 do_refill 函数的开始部分 1 标签处 b 即 back
2:          nop

            jr      ra          # 跳转回到 ra，注意这里的 ra 是 do_refill 函数执行的下一步
            nop
END(do_refill)
```