Week10 Report in Class (Fri56)

11911839 聂雨荷

Q1

请参照Sv32的地址转换过程，写出Sv39的转换过程

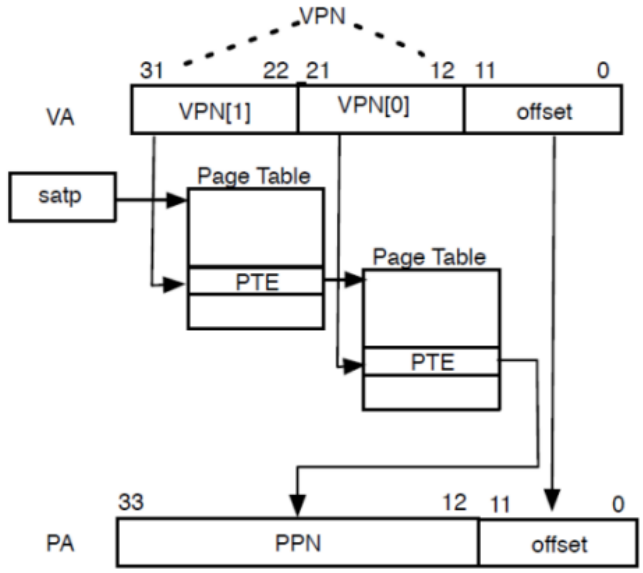


图 10.14：Sv32 中地址转换过程的图示。

当在 satp 寄存器中启用了分页时，S 模式和 U 模式中的虚拟地址会以从根部遍历页表的方式转换为物理地址。图 10.14 描述了这个过程：

1. satp.PPN 给出了一级页表的基址，VA[31:22]给出了一级页号，因此处理器会读取位于地址(satp. PPN × 4096 + VA[31: 22] × 4)的页表项。
2. 该 PTE 包含二级页表的基址，VA[21:12]给出了二级页号，因此处理器读取位于地址(PTE. PPN × 4096 + VA[21: 12] × 4)的叶节点页表项。
3. 叶节点页表项的 PPN 字段和页内偏移（原始虚址的最低 12 个有效位）组成了最终结果：物理地址就是(LeafPTE. PPN × 4096 + VA[11: 0])

Sv39的转换过程

1. satp.PPN 给出了一级页表的基址，VA[38:30]给出了一级页号，因此处理器会读取位于地址(satp.PPN × 4096 + VA[31:22] × 8)的页表项
2. 该PTE包含二级页表的基址，VA[29:21]给出了二级页号，因此处理器会读取位于地址(PTE.PPN × 4096 + VA[29:21] × 8)的页表项
3. 该PTE包含三级页表的基址，VA[20:12]给出了三级页号，因此处理器会读取位于地址(PTE.PPN × 4096 + VA[20:12] × 8)的叶节点页表项
4. 叶节点页表项的PPN字段和页内偏移（原始虚址的最低12个有效位VA[11:0]）组成了最终结果：物理地址就是(LeafPTE.PPN × 4096 + VA[11:0])

Q2

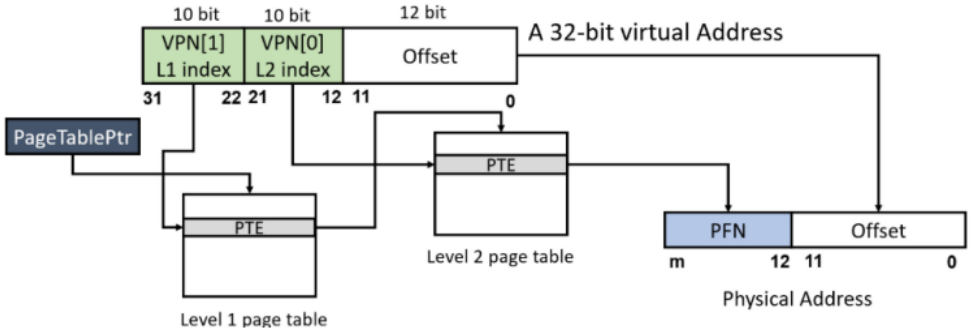
一个巨页的大小是多少，如何计算得出巨页的大小?

如果第二季页表项为叶节点页表项，则该页表项的VA[20:0]位将表示一个整体的偏移 量，我们称之为巨页。它的大小为 $2^{20} = 1\text{MB}$。

我们是通过计算他整体的偏移量大小来得到巨页的大小的。一个页的大小等于$2^{\#\text{offset}}$。

Q3

一个4GB的内存空间使用下图所示的二级页表，页面大小为4KB，PTE大小为PTE_size。一个需要在高虚拟地址空间，中虚拟地址空间，和低虚拟地址空间分别使用4MB的连续空间的进程至少需要多大的页表空间?



二级页表中，第二级页号 `VPN[0]` 为 10 个 bit，因此可以表示 2^{10} 个不同的页面，每个页面大小为4KB，故一个二级页表可以指向 2^{10}×4KB=4MB的物理空间。

一个进程需要使用高虚拟地址空间的4MB连续空间，中地址空间的4MB连续空间和低虚拟地址空间的4MB了连续空间，可以知道这三部分空间属于不同的第二级页表指向的空间。故我们需要 3 个第二级页表。除此之外我们还需要 1 个第一级页表存储这 3 个二级页表的PTE。第一页表和第二页表的大小均为 2^{10} × PET_size，所以我们需要 4 × 2^{10} × PTE_size = 2^{12} × PTE_size 大小的页表空间。

Q4

`static inline void *page2kva(struct Page *page)` 的作用是什么?

```
static inline void *page2kva(struct Page *page)
{
    return KADDR(page2pa(page));
}

static inline uintptr_t page2pa(struct Page *page) {
```

```
    return page2ppn(page) << PGSHIFT;
}

static inline ppn_t page2ppn(struct Page *page) {
    return page - pages + nbase;
}

/* *
* KADDR - takes a physical address and returns the
corresponding kernel virtual address. It panics if you pass
an invalid physical address.*/
#define KADDR(pa)
\
    ({
\
        uintptr_t __m_pa = (pa);
\
        size_t __m_ppn = PPN(__m_pa);
\
        if (__m_ppn >= npage) {
\
            panic("KADDR called with invalid pa %08lx",
__m_pa);\
        }
\
        (void *)(__m_pa + va_pa_offset);
\
    })
```

将以一个指向 page 结构体的指针先转换为对应的物理地址，然后转换成内核的虚拟地址。

整体上是通过一个 page 结构体求出它对应的虚拟地址。