huihongxiao GitBook: [master... Latest commit 7bbde8e on Oct 31, 2020 History

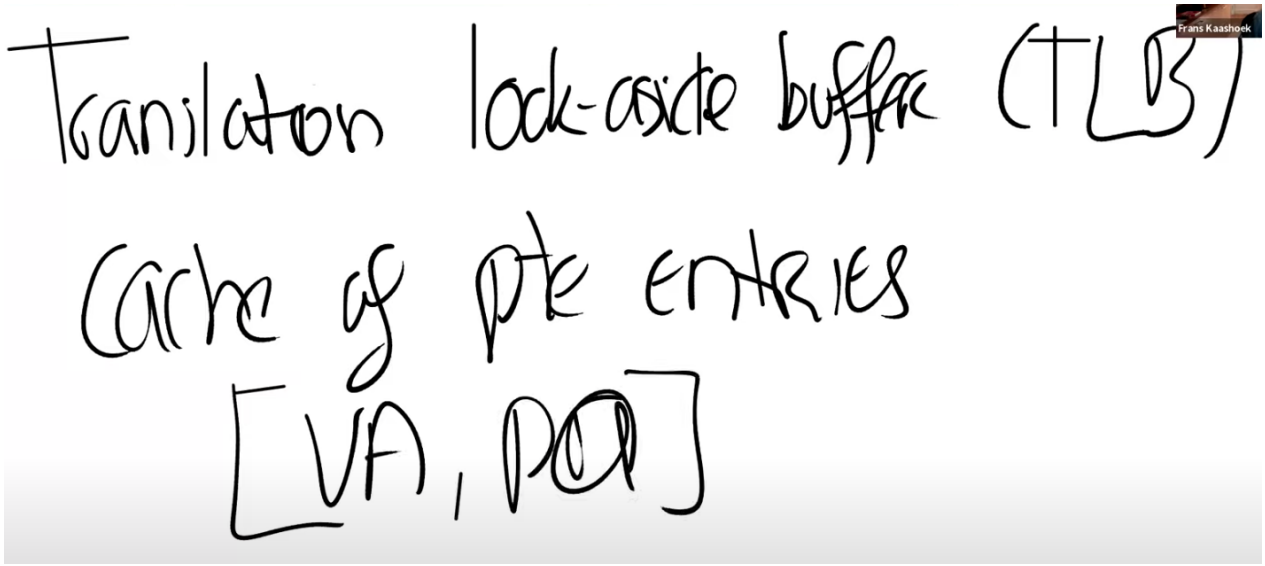
1 contributor

36 lines (26 sloc) 5.43 KB Raw Blame

4.4 页表缓存（Translation Lookaside Buffer）

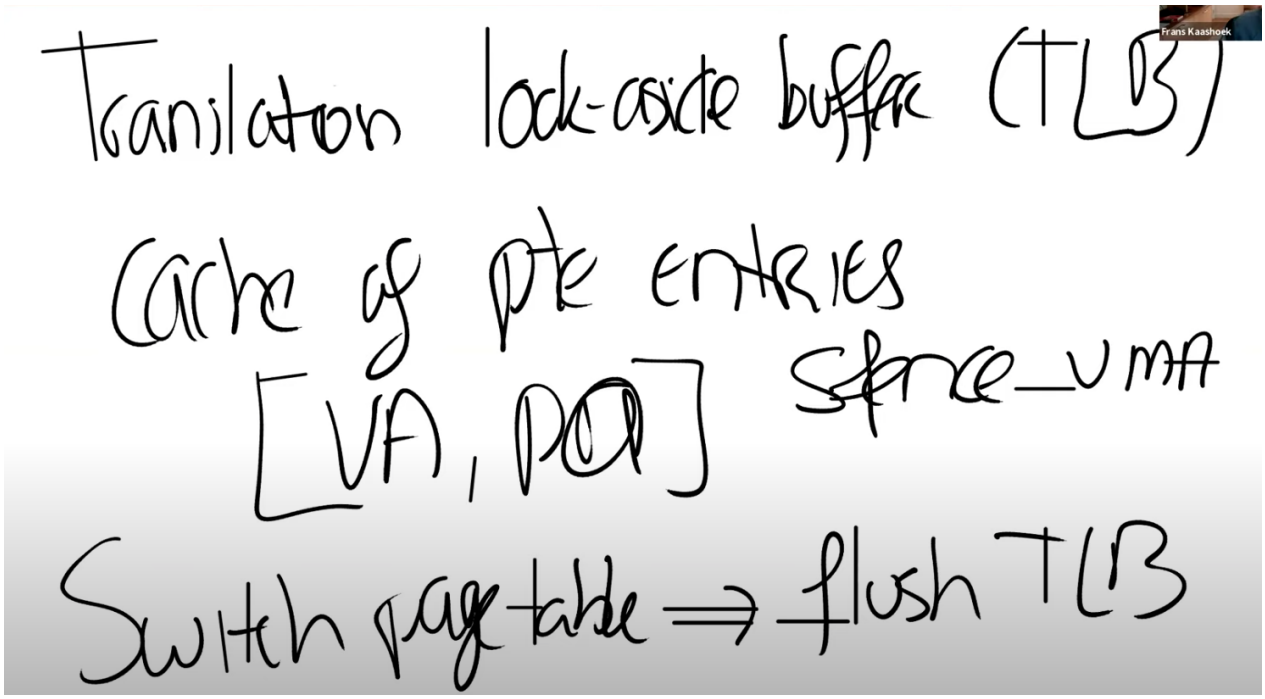
如果我们回想一下page table的结构，你可以发现，当处理器从内存加载或者存储数据时，基本上都要做3次内存查找，第一次在最高级的page directory，第二次在中间级的page directory，最后一次在最低级的page directory。所以对于一个虚拟内存地址的寻址，需要读三次内存，这里代价有点高。所以实际中，几乎所有的处理器都会对于最近使用过的虚拟地址的翻译结果有缓存。这个缓存被称为：Translation Lookside Buffer（通常翻译成页表缓存）。你会经常看到它的缩写TLB。基本上来说，这就是Page Table Entry的缓存，也就是PTE的缓存。

当处理器第一次查找一个虚拟地址时，硬件通过3级page table得到最终的PPN，TLB会保存虚拟地址到物理地址的映射关系。这样下一次当你访问同一个虚拟地址时，处理器可以查看TLB，TLB会直接返回物理地址，而不需要通过page table得到结果。



学生提问：前面说TLB会保存虚拟地址到物理地址的对应关系，如果在page级别做cache是不是更加高效?

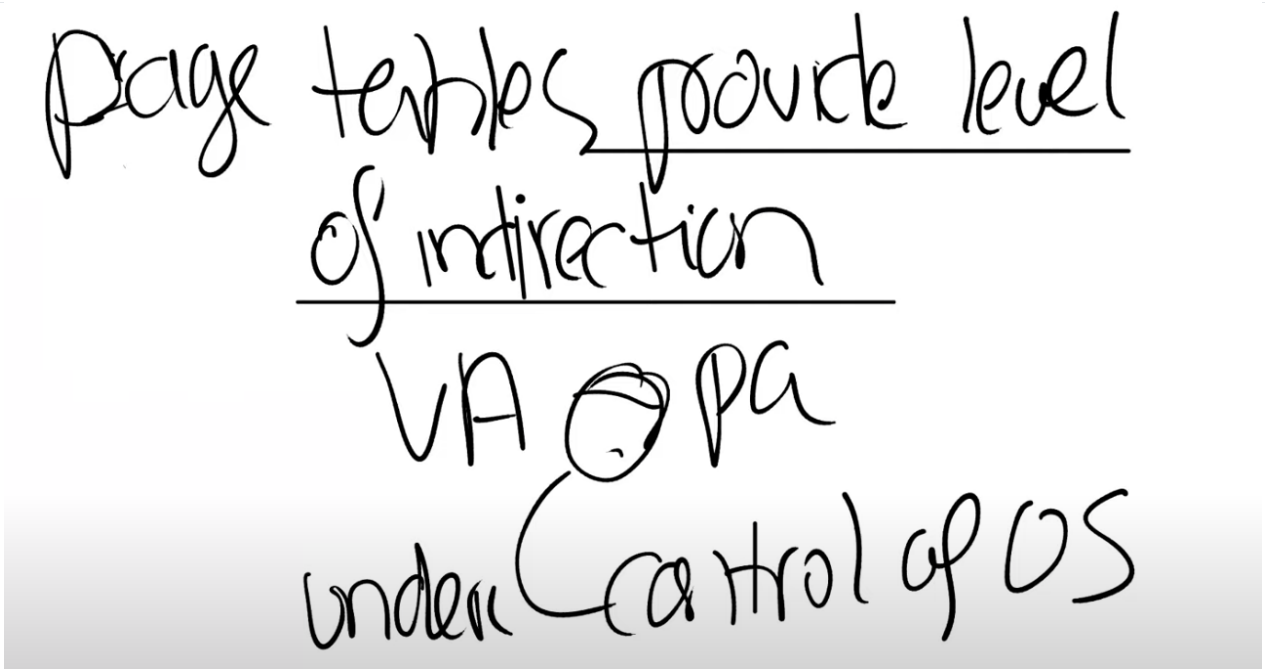
Frans教授：有很多种方法都可以实现TLB，对于你们来说最重要的是知道TLB是存在的。TLB实现的具体细节不是我们要深入讨论的内容。这是处理器中的一些逻辑，对于操作系统来说是不可见的，操作系统也不需要知道TLB是如何工作的。你们需要知道TLB存在的唯一原因是，如果你切换了page table，操作系统需要告诉处理器当前正在切换page table，处理器会清空TLB。因为本质上来说，如果你切换了page table，TLB中的缓存将不再有用，它们需要被清空，否则地址翻译可能会出错。所以操作系统知道TLB是存在的，但只会时不时的告诉操作系统，现在的TLB不能用了，因为要切换page table了。在RISC-V中，清空TLB的指令是sfence_vma。



学生提问：3级的page table是由操作系统实现的还是由硬件自己实现的?

> Frans教授：这是由硬件实现的，所以3级 page table的查找都发生在硬件中。MMU是硬件的一部分而不是操作系统的一部分。在XV6中，有一个函数也实现了page table的查找，因为时不时的XV6也需要完成硬件的工作，所以XV6有这个叫做walk的函数，它在软件中实现了MMU硬件相同的功能。
>
> 学生提问：在这个机制中，TLB发生在哪一步，是在地址翻译之前还是之后?
>
> Frans教授：整个CPU和MMU都在处理器芯片中，所以在一个RISC-V芯片中，有多个CPU核，MMU和TLB存在于每一个CPU核里面。RISC-V处理器有L1 cache，L2 Cache，有些cache是根据物理地址索引的，有些cache是根据虚拟地址索引的，由虚拟地址索引的cache位于MMU之前，由物理地址索引的cache位于MMU之后。
>
> 学生提问：之前提到，硬件会完成3级 page table的查找，那为什么我们要在XV6中有一个walk函数来完成同样的工作?
>
> Frans教授：非常好的问题。这里有几个原因，首先XV6中的walk函数设置了最初的page table，它需要对3级page table进行编程所以它首先需要能模拟3级page table。另一个原因或许你们已经在syscall实验中遇到了，在XV6中，内核有它自己的page table，用户进程也有自己的page table，用户进程指向sys_info结构体的指针存在于用户空间的page table，但是内核需要将这个指针翻译成一个自己可以读写的物理地址。如果你查看copy_in，copy_out，你可以发现内核会通过用户进程的page table，将用户的虚拟地址翻译得到物理地址，这样内核可以读写相应的物理内存地址。这就是为什么在XV6中需要有walk函数的一些原因。
>
> 学生提问：为什么硬件不开发类似于walk函数的接口？这样我们就不用在XV6中用软件实现自己的接口，自己实现还容易有bug。为什么没有一个特殊权限指令，接收虚拟内存地址，并返回物理内存地址?
>
> Frans教授：其实这就跟你向一个虚拟内存地址写数据，硬件会自动帮你完成工作一样（工作是指翻译成物理地址，并完成数据写入）。你们在page table实验中会完成相同的工作。我们接下来在看XV6的实现的时候会看到更多的内容。

在我们介绍XV6之前，有关page table我还想说一点。用时髦的话说，page table提供了一层抽象（level of indirection）。我这里说的抽象就是指从虚拟地址到物理地址的映射。这里的映射关系完全由操作系统控制。



因为操作系统对于这里的地址翻译有完全的控制，它可以实现各种各样的功能。比如，当一个PTE是无效的，硬件会返回一个page fault，对于这个page fault，操作系统可以更新 page table并再次尝试指令。所以，通过操纵page table，在运行时有各种各样可以做的事情。我们在之后有一节课专门会讲，当出现page fault的时候，操作系统可以做哪些有意思的事情。现在只需要记住，page table是一个无比强大的机制，它为操作系统提供了非常大的灵活性。这就是为什么page table如此流行的一个原因。