Report-Project8

付昊源
517021910753

2019 年 6 月 13 日

1 Designing a Virtual Memory Manager

1.1 程序设计思想

在这一 project 中，我们需要模拟一个虚拟内存到物理内存的管理器。

从虚拟内存向物理内存的转换逻辑如下：对于虚拟内存，我们只关心其 0 至 15 位，其中 0-7 位为页内偏移量，8 至 15 位为页号，向物理内存转换时，页内偏移量不变，而页号会由页表（也可能是 TLB）翻译为页框号，由此组合为物理内存。其中，如果页表中无法找到某页号所对应的页框号，就说明发生了页错误，需要由硬盘（在本 project 中为 BACKING_STORE.bin 文件）导入物理内存，并在页表中添加相应的转换。

此外，还有 TLB 的存在，TLB 相当于容量小的页表，因此搜索速度更快，它用于储存最近访问的页表，采用 LRU 算法进行替换，因此它处于不断地更新中。

对于页框数为 256 的物理内存，当一个页第一次被访问时，页错误必然出现，我们需要导入该页到物理内存中。之后如果再访问该页，则不会出现页错误，因为物理内存与虚拟内存一样大，不会发生页替换。但事实上，页框数会小于页数，也就是说物理内存小于虚拟内存，我们需要去处理页替换，这里我也使用了 LRU 算法进行页替换。

1.2 运行结果截图

由于需要翻译的虚拟地址有 1000 个，截图会过长，因此我在这里只截取结尾部分的翻译情况，并给出 TLB 命中率和页错误率。

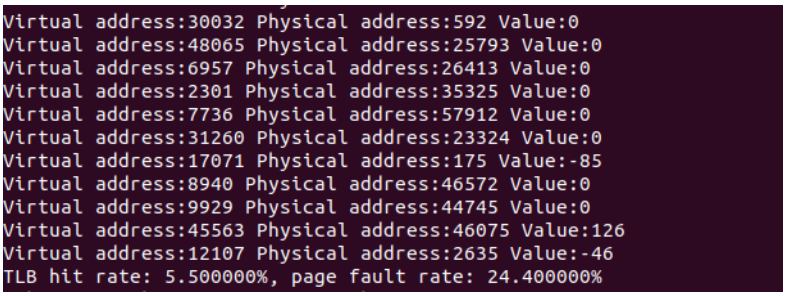

图 1: Physical memory with 256 page frames

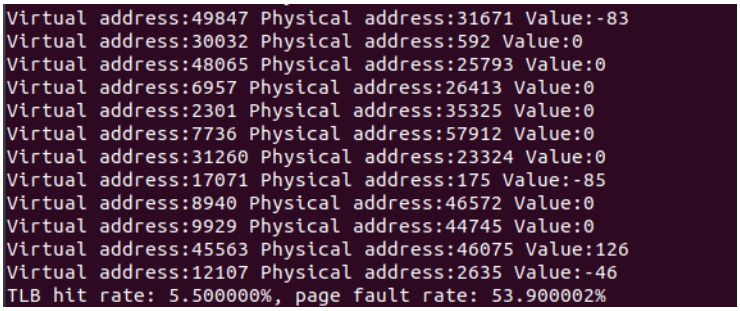

图 2: Physical memory with 128 page frames

1.3 核心数据结构及代码解释

在这一实现中，我定义了两个结构体 tlb 和 pm，分别是 TLB 和物理内存。TLB 所包含的变量有系统时钟 systick，它所存有的页号 pageNumber 和与之对应的页框号 frameNumber。物理内存 pm 包含系统时钟 systick，它所对应的页框号 frameNumber 以及这一页框的所有数据 data[]。其中，systick 的标记用于确定 LRU 算法的替代目标。

物理内存、页表、TLB 的实现均通过数组，数组的元素个数也通过宏定义来确定（对于题目中所需改变页框数为 128，只需要在宏定义中进行改变）。在主函数中，我们通过读取 addresses.txt 这个文件获取待翻译的虚拟地址，由于我们只需要获取 0-7 位和 8-15 位，我们需要进行位操作，即移位和与运算，获取页号和偏移量。获取之后，首先我们要在 TLB 中进行查询，如果查到对应页表，注意更新 systick。

若 TLB 命中，只需要直接去物理内存中根据页框号和偏移量获取数据，

这一操作通过 getValue() 函数实现。同时，getValue() 函数会返回一个 bool 变量到 pageReplaceFlag 中，如果物理内存中找到了该页框号，则返回 false，反之返回 true，即我们需要进行页替换。

若 TLB 没有命中，则需要到页表中进行查询，同时需要使用 LRU 算法更新 TLB，更新 TLB 通过 TLBReplace() 函数进行，其内容包括找到替换目标，即 TLB 中 systick 最小的一个元素，并将其页号、由页表翻译而来的页框号和此时的系统时间记录进去。

若页表中也没有查询到，则出现页错误。值得一提的是，页替换和页错误在算法的实现上并没有什么区别，因为我们都是找到物理内存中 systick 最小的元素进行替换，对于还没有使用的空元素，我们初始化 systick 为 0，一定是最小的，因此并不影响结果的正确性。不论是出现页错误、还是进行页替换，都由 pageReplace() 函数进行处理，函数中的操作包括找到物理内存中替换的目标，写入系统时钟、页框号以及从 BACKING_STORE.bin 文件读入一页的数据。同时，这个函数返回物理内存数组中存放该页的下标，这样可以避免主函数读取数据时再遍历物理内存，寻找页框号的系统开销。

以上是一次对一个虚拟内存的寻址逻辑，由于 addresses.txt 文件有 1000 个地址，因此这一循环会重复 1000 次，每次输出逻辑地址、物理地址和数据值。最后，根据之前的 TLB 命中数和页错误数，计算 TLB 命中率和页错误率，并输出结果。