页表自映射机制

前言

笔者在写LAB2时便对该问题不解，拖到LAB4写完才有功夫想起来解决这个问题。在LAB2时，我以为的自映射机制是指页目录的一项映射到他自身，1024个二级页表中有一张是页目录，而把这个“自映射”的前提**整个页表在虚拟空间是连续的，且有一个固定的起始虚拟地址值**想的理所当然，当然后面我很快发现了这一问题，然而一直想不通究竟如何实现，写LAB3为进程块初始化虚拟空间时，有这么一条语句：

```
e->env_pgdir[PDX(UVPT)] = PADDR(e->env_pgdir) | PTE_V;
```

解释说是实现了自映射，可笔者仍不解，为什么单靠一句话就将页表变成“连续”的呢?

特别鸣谢：guo-hy哥哥的博客

虚实转化

在我们完成实验所采用的R3000上，软件访存的虚拟地址会先被 MMU 硬件映射到物理地址，随后使用物理地址来访问内存或其他外设。虚实转化这一步是由硬件帮我们进行，页表填充等工作都是OS在辅助进行，硬件只需要知道一个信息即可，即**页目录物理地址**，在x86架构下存储在CR3寄存器中，在本实验中由于并未涉及硬件，笔者也不是很清楚，但总之硬件一定会得知这个信息，并根据高十位偏移量找到二级页表的物理地址，接着去访问对应的二级页表，再根据中十位偏移量找到需要寻找的物理地址，加上页偏移量，即完成了虚实转化。

然而这些过程对于软件是不可见的(内核态下可以通过e->pgdir访问)，换句话说，用户无法直接访问到页目录，乃至二级页表，这便造成了一定的局限性，我们有没有方法令用户也能通过一个确定的虚拟地址(UVPT)访问到页表呢，这便需要自映射机制了。

自映射机制

1. UVPT = 0x7fc0 0000，后22位皆为0，是4MB对齐的地址
2. 在页表初始化时，e->env_pgdir[PDX(UVPT)] = PADDR(e->env_pgdir) | PTE_V，这句话的含义为**页目录的第511(UVPT前十位)项中填充的物理页号是页目录自身的物理页号。**

以上两个条件是核心中的核心。

在用户虚拟空间，页表被存储在[UVPT,UVPT+4MB)之中，当我们访问UVPT时，硬件会找到页目录物理首地址并找到511项，因此我们访问到的是页目录自身的物理地址(页目录也是特殊的二级页表)，接着由于中十位偏移量为0，我们会访问到第一张二级页表的物理地址，**即使二级页表之间物理地址是不连续的，即使甚至第一张二级页表都还没有对应的物理页面，在用户看来，我们是通过UVPT这一虚拟地址访问到了一个看似连续的二级页表的首地址。**

我们同样可以通过 UVPT + 虚拟页号 的方式访问到任何想访问的页表项。值得注意的是，UVPT在这里是int*的指针，因此虚拟页号在地址计算上需*4才可(虚拟页号有22位偏移)。由于4MB对齐的限制，UVPT高十位永远不变，因此高十位将**永远**访问到页目录，根据中十位的偏移找到页目录中对应的二级页表的物理地址，再根据页

偏移找到想要访问的二级页表对应的页表项。这样我们便**使用一段连续的虚拟地址，访问到了页表的各项，实现了自映射。**

再来观察UVPD = UVPT + UVPT >> 10，计算方式就不多赘述。经过上述流程，不难发现，高十位访问到了页目录，中十位又访问到了页目录，页偏移量为0，因此最后的效果为**访问到了页目录的起始地址。**

记录下来以后豁然开朗，萦绕心头多年的乌云终于消散。