**实验报告**

**Lab 3**

姓名：田鑫

班级：信息安全

学号：19307110353

**Part A：Speed up system calls**

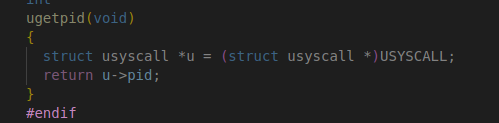
实验结果：

实验步骤：

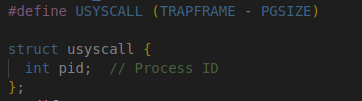
1、首先看pgtbltest中的代码，分析目的



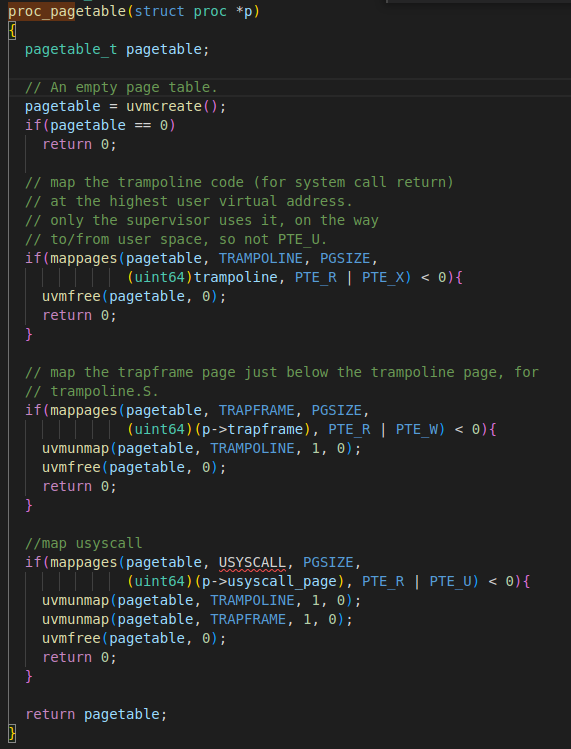
2、分析预先写的ugetpid



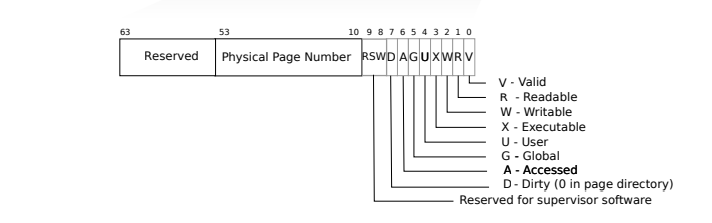
获取一个地址，从这个地址中取得pid



3、去哪里往USYSCALL中填PID的值？依据提示，找到proc\_pagetable,显然，在这里可以完成映射。依靠mappages函数。



然后来分析参数如何填。前几个参数很后理解，主要是最后一个，查找参考书发现，所以这里应该是可读且USER可访问，所以为PTE\_U、PTE\_R。



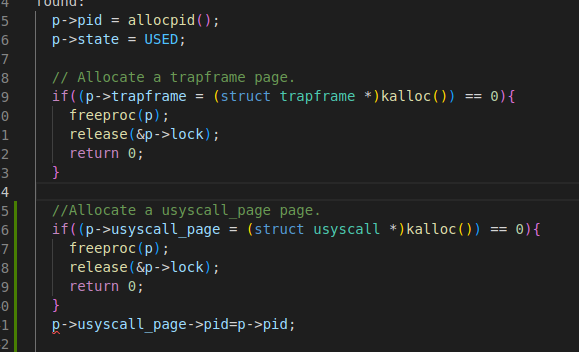




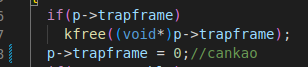
之后仿照上面这句话，来释放假如映射失败

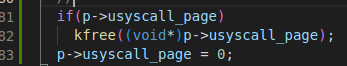
 （释放上面映射的）

4、依据提示分配与初始化page，完全仿写即可



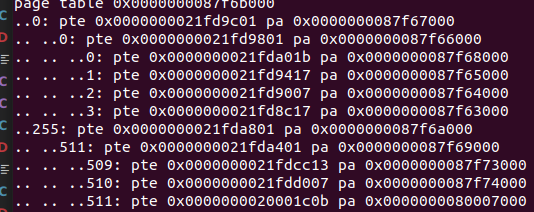
5、在freeproc()中释放

参考



**Part B：Print a page table**

实验结果：



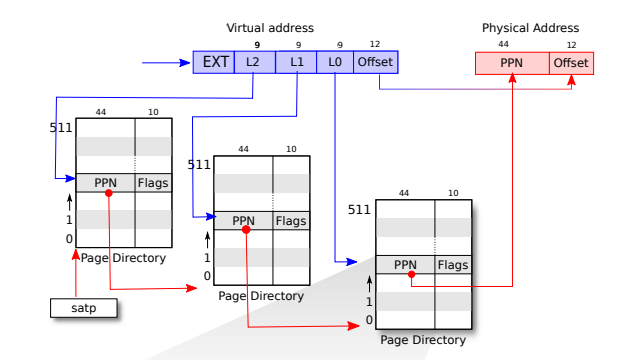
实验步骤：

1、先按要求在exec中加入一段：

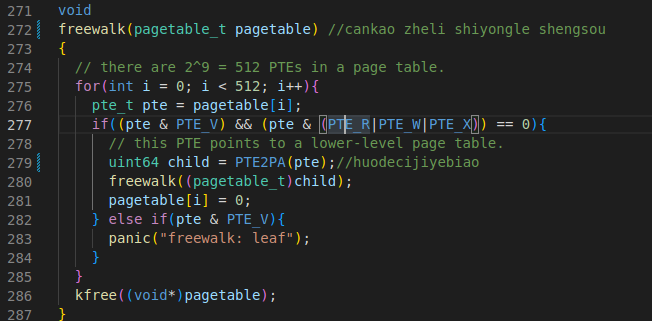


2、按提示在vm.c当中创建一个vmprint函数

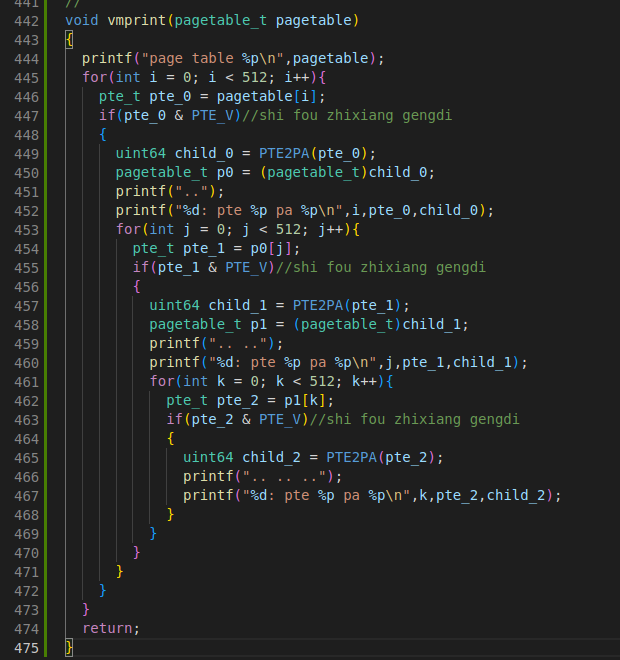
3、获取前置知识，主要来自参考书，知道最多三级页表，以及PTE指什么等。



4、参考freewalk实现函数



Freewalk这里是递归的，我们可以不用递归



三层循环足够

5、在defs.h中添加声明

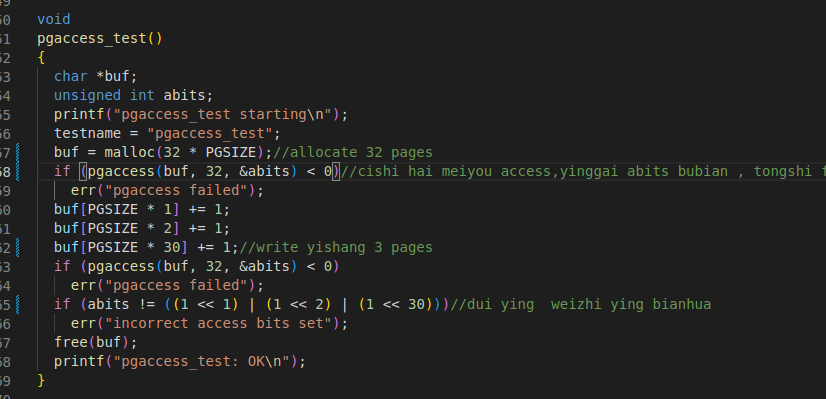
**Part C：Detect which pages have been accessed**

实验结果：

实验步骤：

1、先分析pgacess\_test:

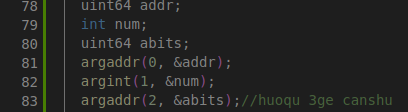
创建了一个buf，给他分配了32页，然后修改其中三页，看是否能返回正确的内容。



2、按提示在sysproc.c中创建函数。（虽然本来就有）

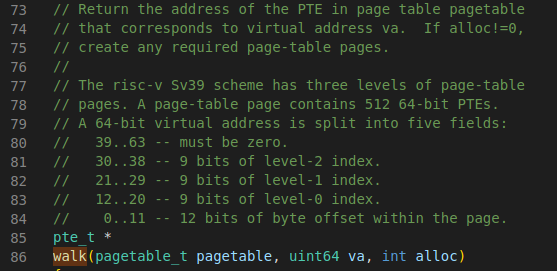
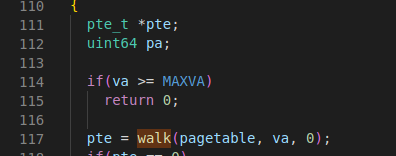
3、先按提示参考别的函数将参数获取

参考：

实现：

这里解释一下为什么有两个是argaddr获取地址，首先是传来的参数，buf是char类型指针，实际也就是地址。然后后是abits，传进的是&abits，所以还是地址

4、依照提示查看walk函数，以及用例

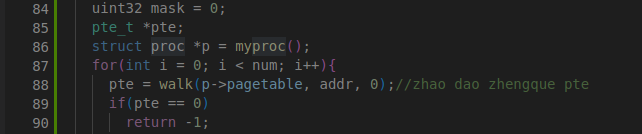
可以通过walk来获取pte

下一个问题是pagetable这个参数哪来？

其实上一次实验已经用过了，查看sysproc.c中的其他函数也能知道

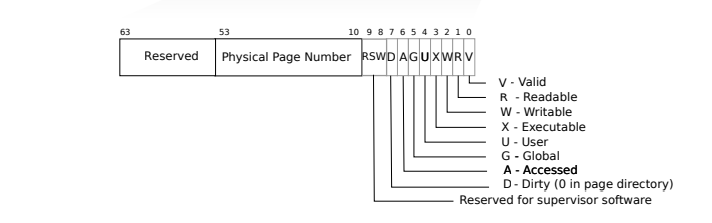


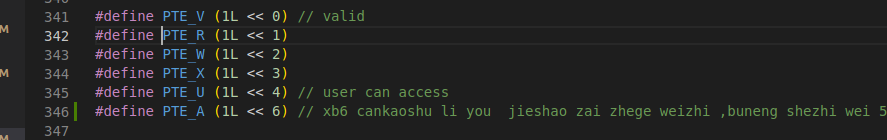
最终



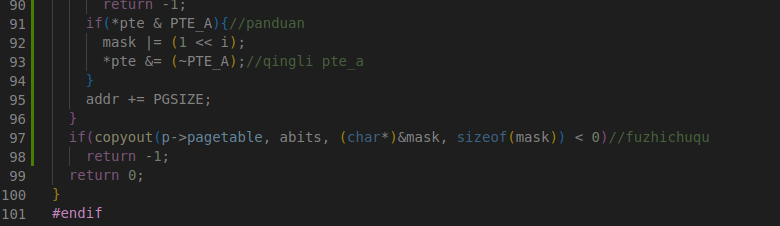
5、按照提示定义PTE\_A

这里我最开始想当然的定义在了第五位，后来查阅参考书发现，需要放在第六位。这个的赋值是交由CPU执行的





6、实现检测PTE\_A的功能



参照前两个实验即可，还需要使用copyout将其复制出去，然后clear。

问题回答：

1、在 Part A 加速系统调用部分，除了getpid()系统调用函数，你还能想到哪些系统调用函数可以如此加速？

Getpid是需要到内核获取一个只读的数，那么所有仅是从内核获取数据并且没有内核层面的修改都可以。但看了一圈系统调用，好像只有getpid能这样，其他的都不行。

2、虚拟内存有什么用处？

第一，虚拟内存可以使得进程对运行内存超过物理内存大小，因为程序运行符合局部性原理，CPU 访问内存会有很明显的重复访问的倾向性，对于那些没有被经常使用到的内存，我们可以把它换出到物理内存之外，比如硬盘上的 swap 区域。第二，由于每个进程都有自己的页表，所以每个进程的虚拟内存空间就是相互独立的。进程也没有办法访问其他进程的页表，所以这些页表是私有的，这就解决了多进程之间地址冲突的问题。第三，页表里的页表项中除了物理地址之外，还有一些标记属性的比特，比如控制一个页的读写权限，标记该页是否存在等。在内存访问方面，操作系统提供了更好的安全性。

3、为什么现代操作系统采用多级页表？

可以离散存储页表。节省页表内存空间。

4、简述 Part C 的 detect 流程。

首先是操作系统会在access到某个页的时候自动的将PTE\_A置1。

然后是sys\_pgaccess如何工作：

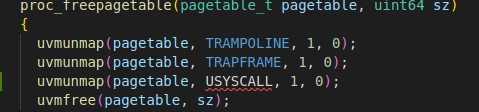
首先获取用户层传递的参数，然后通过传入的虚拟地址和walk函数找到其对应pte，然后查看pte所代表地址的PTE\_A标志位的值如何，最后将其传递给用户层。

实验中遇到的问题以及如何解决：

1、对xv6（或者说RISV）的页表结构不够熟悉，导致一开始很难下手，必须得查阅参考书籍。写代码时对于变量的类型也比较迷惑，不如uint64类型的值可能是地址什么的，需要很小心。

2、完成第三个实验时，对pgaccess\_test的行为一开始没用弄懂，见他传递buf为参数时，不知道究竟改用什么形式理解这个参数。在仔细思考后，确认传进的是一个地址。

3、在完成PartA时，vscode一直提示有错误，查阅资料后才知道makefile会添加一些东西，到时候是没有问题的。



实验感想：

这次实验第一个和最后一个相对来说有意思一点，第一个实验教了如何弄一个新page，并且是如何使用标志位来限制用户态只能读的。最后一个实验教了如何搞一个新的标志位，如何利用标志位完成想要的功能。总的来说，做完这次实验，对于xv6的三层页表相关机制的理解更上了一层楼。