

# Lab3 实验报告

## 一、实验思路、相关代码截图、结果截图

PartA: Speed up system calls:

在 xv6 中，如果用户态调用系统调用，就会切换到内核态，这中间一定是有开销的，至少 CPU 要保存用户态进程的上下文，然后 CPU 被内核占有，系统调用完成后再切换回来。

这个实验的目的就是要加速 getpid() 函数，根据文档提供的思路，首先为每一个进程多分配一个虚拟地址位于 USYSCALL 的页，然后这个页的开头保存一个 usyscall 结构体，结构体中存放这个进程的 pid。

```
75
76 struct usyscall {
77     int pid; // Process ID
78 };
79 #endif
80
```

在 proc.c 中分配一个页给 usyscall，并且对其中的参数 pid 进行初始化。

```
134
135 // Allocate a usyscall page.
136 if((p->usyscall = (struct usyscall *)kalloc()) == 0){
137     freeproc(p);
138     release(&p->lock);
139     return 0;
140 }
141
142 // An empty user page table.
143 p->pagetable = proc_pagetable(p);
144 if(p->pagetable == 0){
145     freeproc(p);
146     release(&p->lock);
147     return 0;
148 }
149
150 // Set up new context to start executing at forkret,
151 // which returns to user space.
152 memset(&p->context, 0, sizeof(p->context));
153 p->context.ra = (uint64)forkret;
154 p->context.sp = p->kstack + PGSIZE;
155
156 // store a struct usyscall
157 p->usyscall->pid = p->pid;
158
```

根据提示，在 proc\_pagetable() 中加入对 usyscall 的映射。(仿照 TRAMPOLINE 和 TRAPFRAME 的映射，首先取消其他页的映射，然后再设置对 usyscall 的映射)

```
217
218 // map the usyscall page just below the trapframe page
219 if(mappages(pagetable, USYSCALL, PGSIZE,
220     (uint64)(p->usyscall), PTE_R | PTE_U) < 0){
221     uvmunmap(pagetable, TRAMPOLINE, 1, 0);
222     uvmunmap(pagetable, TRAPFRAME, 1, 0);
223     uvmfree(pagetable, 0);
224     return 0;
225 }
```

根据提示，在 freeproc () 中应该释放页。

```
165 static void
166 freeproc(struct proc *p)
167 {
168     if(p->trapframe)
169         kfree((void*)p->trapframe);
170     p->trapframe = 0;
171     if(p->pagetable)
172         proc_freepagetable(p->pagetable, p->sz);
173     if(p->usyscall)
174         kfree((void*)p->usyscall);
```

同理，在 proc\_freepagetable()中应该解除映射，否则会报错 freewalk panic。

```
229
230 // Free a process's page table, and free the
231 // physical memory it refers to.
232 void
233 proc_freepagetable(pagetable_t pagetable, uint64 sz)
234 {
235     uvmunmap(pagetable, TRAMPOLINE, 1, 0);
236     uvmunmap(pagetable, TRAPFRAME, 1, 0);
237     uvmunmap(pagetable, USYSCALL, 1, 0);
238     uvmfree(pagetable, sz);
239 }
240
```

PartB: Print a page table

这个实验主要就是层次遍历页表，并输出每一层的页表。

首先，根据文档提示，在 exec.c 中加入：

```
if(p->pid==1) vmprint(p->pagetable);
```

保证可以在 pid 为 1 时就输出，即首先输出页表，然后再进行其他进程。

观察 freewalk 函数，发现其的遍历方式是对每一个页表进行深度遍历，可以借鉴其思路，然后实现了函数 freeprint ()：

其中 pte&PTE\_V 为 true 保证 pte 页永远是有效的，然后才进行输出。

增加限制条件：只输出两层，否则会一直遍历

输出为从第 0 层开始向下遍历，每一层会比上一层前的格式多输出“..”，第 0 层只有一个“..”。且输出内容包含页表序号、页表中的 pte 的地址、物理地址。（用 16 进制表示）

```
1 void
2 freeprint(pagetable_t pagetable, int level)
3 {
4     if(level<=2){
5         for(int i = 0; i < 512; i++){
6             pte_t pte = pagetable[i];
7             if(pte & PTE_V){ // pte not null
8                 for(int i=0;i<level;i++){
9                     printf(".. ");
10                }
11                uint64 child = PTE2PA(pte);
12                printf("..%d: pte %p pa %p\n", i, pte, child);
13                freeprint((pagetable_t)child, level+1);
14            }
15        }
16    }
17 }
```

函数 `vmprint()` 则是打印页表的起始地址，然后对页表进行输出。

```
9 void
9 vmprint(pagetable_t pagetable)
1 {
2     // there are 2^9 = 512 PTEs in a page table.
3     printf("page table %p\n", pagetable);
4     freeprint(pagetable, 0);
5 }
```

最后对 `vmprint` 进行声明，是 `exec.c` 可以调用。

```
176 | void vmprint(pagetable_t);
```

PartC: Detect which pages have been accessed

主要思路就是用户调用系统调用的时候，我们去查找页表，获得对应的 PTE，然后检查 PTE\_A(需要自己定义)，然后决定是否在答案设置对应有效位。

首先在 `sysproc.c` 中完善 `sys_pgaccess()`

```
73 #ifdef LAB_PGTBL
74 int
75 sys_pgaccess(void)
76 {
77     // lab pgtbl: your code here.
78     int number;
79     uint64 virt_addr;
80     uint64 bitmask;
81
82     argaddr(0, &virt_addr);
83     argint(1, &number);
84     argaddr(2, &bitmask);
85
86     if(number<0) return -1;
87     if(virt_addr<0) return -1;
88     if(bitmask<0) return -1;
89
90     return pgaccess((void*)virt_addr, number, (void*)bitmask);
91 }
92 #endif
```

设置输入的第三个参数为虚拟地址，第二个参数为页数，第三个参数为偏移量。

然后调用 `pgaccess()` 函数：

```
int
pgaccess(void* addr, int bit, void* offset)
{
    int ans = 0;
    struct proc* p = myproc();

    pagetable_t pagetable = p->pagetable;
    for(int i=0; i<bit; i++){
        pte_t* pte;
        pte = walk(pagetable, (uint64)(addr+i*PGSIZE), 0);
        if(pte != 0 && (*pte & (PTE_A))){
            ans=ans | 1<<i;
            *pte=*pte ^ PTE_A;
        }
    }

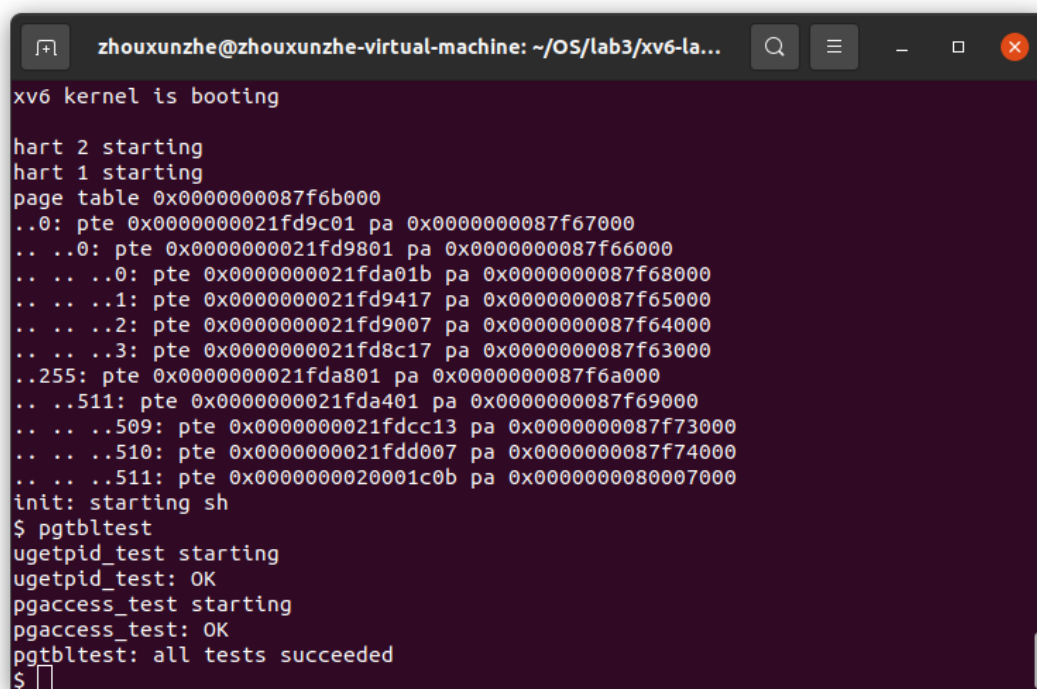
    return copyout(pagetable, (uint64)offset, (char*)&ans, sizeof(int));
}
```

利用 `walk` 函数对每一个页表进行遍历搜寻，然后对被 `access` 的页表进行记录，最后通过 `copyout` 函数进行输出。

查询页表，发现第 6 位就是 access 的标识符，所以判断是否被 access 即判断第 PTE\_A 是否为 1 即可。

```
#define PTE_A ((1L << 6))
```

最后的运行结果如下图：



```
zhouxunzhe@zhouxunzhe-virtual-machine: ~/OS/lab3/xv6-la...
xv6 kernel is booting
hart 2 starting
hart 1 starting
page table 0x0000000087f6b000
..0: pte 0x0000000021fd9c01 pa 0x0000000087f67000
.. ..0: pte 0x0000000021fd9801 pa 0x0000000087f66000
.. .. ..0: pte 0x0000000021fda01b pa 0x0000000087f68000
.. .. ..1: pte 0x0000000021fd9417 pa 0x0000000087f65000
.. .. ..2: pte 0x0000000021fd9007 pa 0x0000000087f64000
.. .. ..3: pte 0x0000000021fd8c17 pa 0x0000000087f63000
..255: pte 0x0000000021fda801 pa 0x0000000087f6a000
.. ..511: pte 0x0000000021fda401 pa 0x0000000087f69000
.. .. ..509: pte 0x0000000021fdcc13 pa 0x0000000087f73000
.. .. ..510: pte 0x0000000021fdd007 pa 0x0000000087f74000
.. .. ..511: pte 0x0000000020001c0b pa 0x0000000080007000
init: starting sh
$ pgtbltest
ugetpid_test starting
ugetpid_test: OK
pgaccess_test starting
pgaccess_test: OK
pgtbltest: all tests succeeded
$
```

## 二、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？

Fork () 函数也是返回 pid，似乎也可以利用页表结构存储 pid，使得系统调用加速。

(2) 虚拟内存有什么用处？

虚拟内存可以扩大内存空间，使得程序能够有效运行；除此以外，还可以利用虚拟内存进行数据的存储，然后调用时又无需花费访问主存那么多的时间，加速系统调用进程。

(3) 为什么现代操作系统采用多级页表？

多级页表可以有效节省物理内存空间，使页表可以在内存中离散存储，不需要连续存储消耗极大的空间。

(4) 简述 Part C 的 detect 流程。

首先设置起始查询地址，然后对每一个页表进行深度遍历查询，如果有页表被 access 并且是有效的，则将那一页所对应的标号进行记录，最后进行输出返回结果。

## 三、实验中碰到的问题。

1.一开始输出页表的格式不正确，并且会超出要求范围，将遍历顺序改为由上至下即可，并且设置范围为两层，就不会出现之前的问题。

2.一开始，在 `proc_freepagetable()` 中没有解除映射，报错 `freewalk panic`，解除映射之后问题就能解决。

## 四、实验感想

在本次实验中，我学会了页表的一些基本原理，以及如何对页表进行访问以及输出。我还了解了一种系统调用加速的方法——访问页表进行存储。本次的三个实验需要自己思考的地方很多，让我能够自主研究了解页表的基本构造，以及在程序中应该如何表达，收获颇丰。