Lab3

姓名: 陈锐林, 学号:21307130148

2023 年 10 月 27 日

实验 1: 加速系统调用速度

0. 这里在 ppt 里已经给出了完整的实现过程，所以就跟着 ppt，记录每个步骤的做法和代码。

1. 添加指针保存页表地址。根据 hint，我们可以通过 struct usycall 实现映射；并利用已经实现的函数。查阅 memlayout.h，usycall 已经定义好了，包含一个成员 pid。所以在 proc 的定义中添上如下代码即可：

```
struct usyscall *usyscall;
```

2. 这步要求为上面定义的 usycall 分配空间，并初始化 pid。翻看 allocproc() 的其余部分可以很快学习到写法，先用 kalloc 尝试分配，如果出错，解锁并 return。初始化 pid，应该是将 p->pid 给 p->usycall，这里用 memmove 应该就可以了。代码如下：

```
if((p->usyscall = (struct usyscall *)kalloc()) == 0){
  freeproc(p);
  release(&p->lock);
  return 0;
}
memmove(p->usyscall,&p->pid,sizeof(int));
```

3. 这步要求将该 PTE 写入 pagetable，设置为用户可读。观察到，proc_pagetable 这个函数的书写逻辑，和 allocproc 是有点像的；都是 if 里尝试 mappages 一个东西，如果失败；就要 free。该函数中先 mappages TRAMPOLINE，失败则释放 (uvmfree) 页表; 再 mappages TRAPFRAME，如果失败要解开映射 (uvmunmap)，再释放 (uvmfree) 页表。所以我们可以把对于 PTE 的写入放在最后，如果失败，先 uvmunmap TRAMPOLINE 和 TRAPFRAME，再 uvmfree 页表。至于用户权限，riscv.h 有写，该用 PTE_R | PTE_U。最后代码如下：

```
if(mappages(pagetable,USYSCALL,PGSIZE,(uint64)(p->usyscall),PTE_R | PTE_U) < 0){
  uvmunmap(pagetable,TRAMPOLINE,1,0);
  uvmunmap(pagetable,TRAPFRAME,1,0);
  uvmfree(pagetable,0);
  return 0;
}
```

4. 这步要让 freeproc 能释放共享页，并插入空闲页链表。这里在函数开始就 kfree，并且将该 usycall 指向的 pid 置 0 即可。最后代码如下：

```
if(p->usyscall)
  kfree((void*)p->usyscall);
p->usyscall = 0;
```

5. 这步解除映射关系，观察 proc_freepagetable 内的其他部分。同样对 usycall 进行 uvmunmap 即可。最后代码如下：

```
uvmunmap(pagetable,USYSCALL,1,0);
```

6. 测试结果，运行 pgtbltest 可以看到通过测试。如下：

```
init: starting sh
$ pgtbltest
ugetpid_test starting
ugetpid_test: OK
pgaccess_test starting
pgaccess_test: OK
pgtbltest: all tests succeeded
```

实验 2: 打印页表

1. 在 def.h 中如下定义即可，其中第二个 int 表示递归层数，这是由后面的代码决定的；与 ppt 不同，但是无伤大雅，可以不用再定义新函数。

```
void        vmprint(pagetable_t, int);
```

2.vmprint 函数是这个实验的关键，根据 hint，参考 freewalk 函数；我们可以知道是要考虑递归的树状结构。递归的思路就是一级一级调用 vmprint(); 而因为是多级页表，所以要做出区分，到底是不是叶节点。如果是叶结点就结束递归。怎么区分成了一个问题，但是看懂 freewalk 函数后就发现，情况是一样的。所以对 freewalk 函数进行一个拙劣的模仿就好了，级数 level 用来控制.. 生成的数量。其他的就是 PTE2PA 生成孩子，按照样例打印。代码如下：

```
void vmprint(pagetable_t pagetable, int level){
  for(int i = 0; i < 512; i++){
    pte_t pte = pagetable[i];
    if((pte & PTE_V) && (pte & (PTE_R|PTE_W|PTE_X)) == 0){
      uint64 child = PTE2PA(pte);
      for(int t=0;t<level;t++){
        printf(" ..");
      }
      printf("%d: pte %p pa %p\n", i, pte, child);
      vmprint((pagetable_t)child, level + 1);
    }else if(pte & PTE_V){
      uint64 child = PTE2PA(pte);
      for(int t=0;t<level;t++){
        printf(" ..");
      }
      printf("%d: pte %p pa %p\n", i, pte, child);
    }
  }
}
```

3. 这里就是添加，没什么不一样的。但是我把题目要求的 page table 这一行挪到这了 (当然在 vmprint 中判断 level 来解决也可以 qwq)。

4. 测试结果如下：其中左侧 make grade 包含三个实验测试

```
== Test pgtbltest ==
$ make qemu-gdb
(2.1s)
== Test   pgtbltest: ugetpid ==
  pgtbltest: ugetpid: OK
== Test   pgtbltest: pgaccess ==
  pgtbltest: pgaccess: OK
== Test pte printout ==
$ make qemu-gdb
pte printout: OK (0.7s)
```

(a)

```
hart 1 starting
hart 2 starting
page table 0x0000000087f6b000
 ..0: pte 0x0000000021fd9c01 pa 0x0000000087f67000
 .. ..0: pte 0x0000000021fd9801 pa 0x0000000087f66000
 .. .. ..0: pte 0x0000000021fda05b pa 0x0000000087f68000
 .. .. ..1: pte 0x0000000021fd9457 pa 0x0000000087f65000
 .. .. ..2: pte 0x0000000021fd9007 pa 0x0000000087f64000
 .. .. ..3: pte 0x0000000021fd8c57 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
```

(b)

实验 3: 检测哪些页被访问

1. 这步实现 sys_pgaccess()，具体分为以下部分：

(1) 根据题意，这个函数接受三个参数，用户页起始虚拟地址 (定义为 vp)，被检查页数 (Numpages)，用户空间地址 (ua)。我们可以用 argaddr 和 argint 导入。

(2)Numpages 可设上限，取为 64，超过就取 64。

(3) 定义变量 mask 表示掩码;获取方式如下:对于范围在当前页开始的 Numpages 内的所有页，通过 walk 函数寻找页表项；只要正确找到，就考察是否被访问 (PTE_A)，如果访问过了要及时清除 (对 PTE_A 取反再与)。

(4) 最后用 copyout 把 mask 送到 ua。

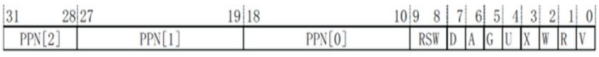
(5) 代码如下：

```
int sys_pgaccess(void)
{
  uint64 vp , ua;
  int Numpages , mask=0;
  argaddr( 0 , &vp );
  argint( 1 , &Numpages );
  argaddr( 2 , &ua );

  pagetable_t p;
  if(Numpages > 64) Numpages = 64;

  for( int i=0 ; i<Numpages ; i++ )
  {
    if( (p = walk(myproc()->pagetable , vp+i*4096 , 0 )) == 0 )
      continue;
    if( (*p & PTE_A) > 0 )
    {
      mask |= (1<<i);
      *p = (*p&(~PTE_A));
    }
  }
  copyout( myproc()->pagetable , ua , (char*)&mask , sizeof(uint64) );
  return 0;
}
```

2. 设置 PTE_A，这里要根据 RISCV 架构，经查询（如下）应该是 1 « 6。



3. 运行效果，在实验 1 已展示：

```
init: starting sh
$ pgtbltest
ugetpid_test starting
ugetpid_test: OK
pgaccess_test starting
pgaccess_test: OK
pgtbltest: all tests succeeded
```