@[TOC](Mit6.S081-实验6-Copy-on-Write Fork for xv6)

# 一、前言

虚拟内存提供一定程度的重定向：kernel可以通过标记PTEs无效、只读（导致page faults）来中断内存引用；

kernel也可以通过改变地址含义（通过更改PTES）。

在电脑系统中有个说法：系统问题可以通过一定程度的重定向解决。

lazy allocation lab提供了一个例子。这个lab探索另外的例子：copy-on write fork。

开始本lab前，先切换到cow分支

git fetch、git checkout cow、make clean

# 二、The problem

xv6中的fork() system call，复制parent进程所有的用户空间内存到child。

如果parent是非常大的，copying将花费很长时间。糟糕的是，这个工作通常是大量浪费的；

例如，fork()后紧接着是exec()在child进程中，这将导致child会丢弃拷贝的内存，可能绝大多数都不使用。

另一方面，如果parent和child使用一个page，并且其中一个或两个写，那么确实需要一个副本。

# 三、The solution

copy-on-write（COW）fork()的目的是：推迟对child的分配和拷贝物理内存页，直到拷贝确实需要。

COW fork()仅仅给child创建一个pagetable，其用户内存的PTES指向parent的物理页。

COW fork()标记parent和child的所有用户内存PTES是不可写的。

当某个进程尝试写其中一个COW页时，cpu将强制一个page fault。

kernel page-fault handler检测这种情形，为faulting进程分配一页物理内存，复制原始页到新页，

更改faulting进程相关PTE来指向新页，这次让PTE标记为可写。当page fault handler返回时，用户进程将能够向拷贝页写入。

COW fork()让物理页（实现用户内存）的释放更有技巧性。

一个给定物理页可能被多个进程的page table指向，仅应该在最后的指向消失时，才释放物理页。

# 四、Implement copy-on write

## 1，实验要求

你的任务是在xv6 kernel实现copy-on fork。如果更改的kernel code通过cowtests和usertests，则成功了。

为了帮你测试你的实现，我们已经提供了一个xv6程序（cowtest，源码在user/cowtest.c）。

cowtest运行多个tests，未改xv6时第一个就会失败。因此初始化，你将看到：

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201230131641754.png)

“simple” test分配可用内存的一半多，然后fork()s。因为没有足够的空闲物理内存，来给child一个完整parent内存拷贝，所以fork失败。

当你完成时，你的kernel应该通过cowtest和usertests中的所有测试：

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201230131807383.png)

## 2，进攻计划

1. 更改uvmcopy()来映射parent物理页到child，而不是分配新页。清除parent和child PTES的PTE\_W。

2. 更改usertrap()来识别page faults。

当一个page-fault发生在一个COW page，通过kalloc()分配一个新页，复制旧页到新页，并且安装新页到PTE（设置PTE\_W）。

3. 确保每个物理页被释放，当最后的PTE指向移除时。

这么做的一个好方式是：对每个物理页保存一个“reference count”，表明指向此物理页的page table数量。

设置page的reference数目为1，当kalloc()分配它时。

增加page的reference数目，当fork导致child分享此页时；减少page的reference数目，每次任意进程从page table中删除此page时。

kfree()应该仅仅放置一个page在free list最后，如果它的reference数目为0。

将这些计数放到一个固定长度的整数数组中。你将不得不找出一个计划：如何索引数组，如何选择它的尺寸。

例如：你可以用页物理地址除以4096对数组进行索引，并给数组一些元素，这些元素，通过kalloc.c中的kinit()放在free list中的页

4. 当遇到一个COW page时，更改copyout()，使用与page fault相同的方法。

## 3，一些提示

1. lazy page allocation lab可能已经让你熟悉了一些xv6 kernel代码（与copy-on-write相关的）。

然而，你不应该让本实验基于lazy allocation的方案。而是根据上面引导的，开始一个新的xv6拷贝。

2. 使用RISC-V PTE的预留标志位，来记录每个PTE是不是一个COW映射，这可能是有用的。

3. usertests探索一些cowtest没有测试到的地方，不要忘记核对两个测试都通过。

4. 一些对页表标志位有帮助的宏指令和定义在kernel/riscv.h下面。

5. 如果一个COW page fault发生，但没有空闲内存，此进程应该被杀掉。

## 4，具体实现

1）修改kernel/vm.c，新增int refNum[32768];来记录关联物理页的页表数量。32768是根据(PHYSTOP-KERNBASE)/PGSIZE得出。

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201230132910532.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)

2）修改kernel/riscv.h，新增PTE\_COW标志位。

![在这里插入图片描述](https://img-blog.csdnimg.cn/202012301336342.png)

参考riscv对PTE标志位定义，第9-10位为预留标志位。

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201230133713526.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)

3）修改kernel/vm.c的uvmcopy()，让进程fork时，不赋值物理页，而是child进程页表指向parent进程的物理页，但标记要将parent和child的PTE都清除PTE\_W标志位，并添加COW标志位，表明两个PTE指向一个物理页。

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201230133453663.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)

4）修改kernel/vm.c的mappages()，在页表与物理页绑定时，增加refNum对应元素计数。

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201230133947858.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)

5）修改kernel/vm.c的uvmunmap()，在页表与物理页解绑时，减少refNum对应元素计数，当refNum==1即仅kernel pagetable持有时，释放内存。

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201230191020956.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)

6）将kernel/vm.c中walk()定义在defs.h中。

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201230134658987.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)

7）修改kernel/trap.c的usertrap()，引入refNum，在发生page fault时，若该虚拟地址关联的PTE，表明关联的物理页是一个COW页，则新申请一个物理页，让此虚拟地址指向新物理页，并修改refNum计数。

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201230135054203.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201230193546709.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)

8）修改kernel/vm.c的copyout()，同kernel/trap.c的usertrap。

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201230193746730.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)

## 5，执行效果

运行xv6后，执行cowtest测试本实验

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201230151540868.png)

执行usertests回归测试

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201231102728790.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70)