Copy on Write 实验报告

专　业：软　件　工　程

指　导　老　师：王　冬　青

年　级：大　学　二　年　级

学　号：2153061

姓　名：谢　嘉　麒

1.The Problem

Xv6中的fork()系统调用会将父进程所有用户空间内存复制到子进程中。如果父进程很大，泽福只需要很长时间。比如，子进程中的fork()后跟exec()会导致紫禁城丢弃复制的内存，可能甚至没有使用过这些内存。另一方面，如果父进程和子进程使用同一个页面，且其中一个或他们同时编写它，则需要一个副本。

2.The Solution

写时复制fork()的目标是推迟子进程分配和复制物理内存页，直到实际需要副本（如果有的话）。

COW fork()只为子进程创建一个分页表，其中用于用户内存的PTE指向父级的物理页。COW fork()将父进程和子进程中的所有用户PTE标记为不可写。当任一进程尝试写入COW页之一时，CPU将强制出现页错误。内核页面错误处理程序检测到这种情况，为错误进程分配一页物理内存，将原始页面复制到新页面中，并在错误进程中修改相关PTE以引用新页面。这一PTE标记为可写。当页面错误处理程序返回时，用户进程将能够写入其页面副本。

COW fork()使释放实现用户内存的物理页面变得略显棘手。给定的物理页可能由多个进程的页表引用，并且只有在最后一个引用消失的时候才应该释放。

3.Implement copy-on write(hard)

3.1.实验目的

实验要求在Xv6内核中实现写时复制fork。

3.2.实验步骤

3.2.1.在kalloc中添加相关内容

首先在kernel/kalloc.c的kmem结构中添加物理内存引用计数器uint *ref_count，并为其定义互斥锁struct spinlock reflock，这些新添加的项在kinit()函数中进行初始化。

在kinit()函数中将kmem.ref_count大小初始化为(uint*)end。在freerange()函数里将引用计数器设置为1。

修改kalloc函数，使其在分配页面时将引用计数器设置为1，即如下语句：

```
kmem.ref_count[kgetrefindex((void*)r)]=1;
```

释放内存时查看引用计数是否为0，为0则释放内存，否则返回，由其他进程继续使用。

```
acquire(&kmem.lock);
 if(--kmem.ref_count[kgetrefindex(pa)])
 {
   release(&kmem.lock);
   return;
 }
 release(&kmem.lock);
```

增加辅助函数kgetref()，kaddref()，acquire_refcnt()和release_refcnt()分别担任获取当前ref，增加ref和管理ref互斥锁的任务。

实现这些辅助函数后在defs.h中声明这些函数即可。

3.2.2.增加COW标志位

系统在页表项中预留两位，设置第八位是COW标志位。即在riscv.h中添加语句#define PTE_COW (1L << 8))

3.2.3.修改fork

需要对fork()函数进行修改，使其不对地址空间进行拷贝。而又因为fork()函数中调用vm.c中的uvmcopy()进行拷贝，因此修改uvmcopy()即可。

删除uvmcopy()函数中的kalloc函数，将子进程页表映射到父进程的物理地址，并去掉写标志位。

添加COW标识位，设置父进程的PTE_W为不可写，且为COW页。

不为子进程分配内存，设置pa = PTE2PA(*pte)，指向pa，页表属性设置为flags。

3.2.4.usertrap()

在trap.c的usertrap()函数中增加页面错误处理内容，需要判断是否为COW。只需要处理scause==15的情况即可，因为13是页面读错误， 而COW不会引起读错误。

需要先判断COW标志位，当该页面是COW页面时，根据引用计数进行处理。

如果引用计数大于1，则通过kalloc申请一个新页面，拷贝内容，之后对这个页面进行映射，清除COW标志位，设置PTE_W标志位；

如果引用计数等于1，则不需要申请新页面，只需要对这个页面的标志位修改即可。

注意要在程序中声明extern pte_t* walk(pagetable_t, uint64, int);

3.2.5.copyout()

还需要在kernel/vm.c的copyout()函数中增加页面错误处理。copyout使内核状态下修改用户态的页面内存，缺页不会进入usertrap，因此进行特殊处理。

先声明外部函数walk。

使用于刚才相似的方法，来分配页面，复制数据释放页面，修改页面权限和标志位即可。

3.2.6.编译与检测

在完成以上步骤后，进入Xv6目录终端进行编译，编译完成后执行cowtest指令，得到预期结果：

```
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ cowtest
simple: ok
simple: ok
three: ok
three: ok
three: ok
file: ok
ALL COW TESTS PASSED
```

执行make grade指令，也通过全部测试：

```
make[1]: Leaving directory '/home/cinderlord/xv6-labs-2021'
== Test running cowtest ==
$ make qemu-gdb
(8.2s)
== Test   simple ==
  simple: OK
== Test   three ==
  three: OK
== Test   file ==
  file: OK
== Test usertests ==
$ make qemu-gdb
(261.1s)
== Test   usertests: copyin ==
  usertests: copyin: OK
== Test   usertests: copyout ==
  usertests: copyout: OK
== Test   usertests: all tests ==
  usertests: all tests: OK
== Test time ==
time: OK
Score: 110/110
```

3.3.实验中遇到的问题和解决方法

实验中问题在于分别对uvmcopy()和copyout()进行修改。这两个函数的修改总体思想相似，但要注意到内核态并不会引起usertrap。同时覆写uvmcopy时，起初并不理解scause寄存器设置为12或15的含义，也不清楚STVAL寄存器的意义。这些内容通过查阅实验给出的RISC-V手册可以确定：前者是写页面权限造成的错误引发中断，后者是存储违反权限的虚拟内存地址。了解了每一个代码段具体的含义

后，书写代码就变得简单明了。

3.4.实验心得

写时复制实验简单引入了一种内存管理机制，使笔者对于系统的内存拷贝等机制有了更进一步的了解。同时这种对于惰性机制的引入也引发了笔者的兴趣，因此笔者额外了解了操作系统中引入惰性机制的案例与意义。发现这种机制在操作系统的各个子模块，乃至数据库中都有一定的应用，避免了资源浪费，是一种良好的软件思想。