Lab5

姓名: 陈锐林, 学号:21307130148

2023 年 11 月 21 日

一、实验思路

1. 这次实验出发点是对 xv6 的 fork 进行改进，达到高效、简洁、低耗的目标。

2. 大体思路如下：(1) 当 fork 时 (调用 uvmcopy 时)，让父子结点映射到同一目标；(2) 当遇到写操作,usertrap 的时候；对这些内容进行拷贝；(3) 完善设置：需要同时对 kfree/kalloc/copyout 进行修改；更新 PTE；并基于当前共享这一物理页的进程数作出判断；为 1 时要特殊处理。

3. 由 hint/之前的实验可以得到的借助: (1) 判断是否存在这一现象可以设置 RISC-V 的 RSW 位；(2)mappages 函数作映射；(3)walk/walkaddr 函数作地址转换；(4)r_scause 函数返回错误码；(5) 可以利用 spinlock 完成计数值的更改；(6)PGROUNDDOWN 等函数获取页表信息。

二、具体实现

1. 在 riscv.h 中先设置 PTE 位，其中取 1«8 为 PTE_COW，表示进行了 COW 操作。

2. 修改 uvmcopy，主要修改是删除了原先用于分配的 char *mem；并利用 mappages 将父子结点映射。其中根据计数的规则，fork 后要及时修改计数，add_ref 放在后面实现。

```
*pte = (*pte & ~PTE_W) | PTE_COW;  // hint change pte
flags = PTE_FLAGS(*pte);
//delete all about mem
if(mappages(new, i, PGSIZE, pa, flags) != 0){
  goto err;
}
if(add_ref((void*)pa) != 0){ //later sovle add_ref
  goto err;
}
```

3. 在 kalloc.c 中完成定义，具体来说是以下几步：

(1) 仿照 kmem 结构体，完成对计数结构体 ref 的定义；其中计数为 1 数组，大小应是 (地址空间大小/页大小) 最大页数。并在 kinit 里初始化锁；

```
struct{
  struct spinlock lock;
  uint64 count[PHYSTOP / PGSIZE];
}ref;
```

(a) struct ref

```
kinit()
{
  initlock(&kmem.lock, "kmem");
  initlock(&ref.lock, "ref"); //lock!
  freerange(end, (void*)PHYSTOP);
}
```

(b) kinit

(2) 在 kalloc 里初始化 count 数组为全 1。

```
if(r){
  acquire(&ref.lock);
  ref.count[(uint64)r / PGSIZE] = 1;
  release(&ref.lock);
}
```

(3) 在 kfree 里完成对当前页面 ref–，以及在 ref 为 0 时，真正 free。

```
acquire(&ref.lock);
ref.count[(uint64)pa / PGSIZE]--; //change ref
if(ref.count[(uint64)pa / PGSIZE] != 0){ //not release if ref != 0
  release(&ref.lock);
  return;
}
release(&ref.lock);
```

(4) 还需要注意 freearange 函数，对于从 pa_start 到 pa_end 的页面，需要都初始化计数器为 1，避免 0-1= 大正数。

```
freerange(void *pa_start, void *pa_end)
{
  char *p;
  p = (char*)PGROUNDUP((uint64)pa_start);
  for(; p + PGSIZE <= (char*)pa_end; p += PGSIZE){
    ref.count[(uint64)p / PGSIZE] = 1;
    kfree(p);
  }
}
```

(5) 完成 add_ref 函数的定义，其实逻辑很简单，就是 count[pa/pgsize]++；但是需要补一下判断，此时的 pa 是不是还在地址空间内，并且这个 pa 应该是整除 PGSIZE 的。还有 get_ref，简单的接口函数。

```
int
add_ref(void* pa){
  if(((uint64)pa % PGSIZE)){
    printf("Invalid address: not divided\n");
    return -1;
  }
  if((char*)pa < end || (uint64)pa >= PHYSTOP){
    printf("Invalid address: outof range.\n");
    return -1;
  }
  acquire(&ref.lock);
  ref.count[(uint64)pa / PGSIZE]++;
  release(&ref.lock);
  return 0;
}

int get_ref(void* pa){
  return ref.count[(uint64)pa / PGSIZE];
}
```

4. 在 trap.c 中完善定义：主要分为修改 usertrap，定义 cow_or_not 和定义 cow_lloc 函数。

(1) 修改 usertrap，顺着上面的 if(r_scause==8) 接着写一个 else if，当 r_scause 为 12/13/15 时，要进行 COW 的判断。在下面新起的 if 语句中主要完成：对 va 判断是否超过 PGSIZE；判断是不是 COW 页面，是会返回 1；为这个进程分配页面，不返回 0 即可。否则就是异常情况 (Tips-3)，直接 kill 进程。

```
else if(r_scause() == 13 || r_scause() == 15 || r_scause() == 12){
uint64 va = r_stval();
if(va < PGSIZE || va >= p->sz ||cow_or_not(p->pagetable,va) != 1 ||cow_lloc(p->pagetable,va) == 0)
  p->killed = 1;
```

(2) 完成对 cow_or_not 的定义。判断的依据就是 PTE 值，首先它应该和当前 va 映射的相同；并且要有 PTE_COW 位。

```c
int cow_or_not(pagetable_t pagetable, uint64 va) {
  if(va > MAXVA)
    return 0;
  pte_t* pte;
  if((pte = walk(pagetable, va, 0)) == 0){
    return 0;
  }
  if((*pte & PTE_V) == 0)
    return 0;
  if((*pte & PTE_U) == 0)
    return 0;
  if(*pte & PTE_COW)
    return 1;
  return 0;
}
```

(3) 完成 cow_lloc 函数的定义。首先要得到物理地址 pa 和 PTE，分别利用 walkaddr 函数和 walk 函数得到。接着我们对计数值进行判断；如果计数为 1，皆大欢喜，直接开写权限，返回；如果不为 1，就要分配一个新地址并且复制当前页面过去。

```c
va = PGROUNDDOWN(va);
if(va % PGSIZE != 0)
  return 0;
uint64 pa = walkaddr(pagetable, va); //
if(pa == 0)
  return 0;
pte_t* pte = walk(pagetable, va, 0);  /
int ref = get_ref((void*)pa);
if(ref == 1) {  // the only page, write
  *pte = (*pte & ~PTE_COW) | PTE_W;
  return (void*)pa;
}
```

(a) get pa/PTE and ref == 1

```c
else {  //get new page
  char* new;
  if((new_pa = kalloc()) == 0){
    return 0;
  }
  memmove(new, (char*)pa, PGSIZE);
  *pte = (*pte) & ~PTE_V; //remove PTE_V
  if(mappages(pagetable, va, PGSIZE, (uint64)new,
  (PTE_FLAGS(*pte) & ~PTE_COW) | PTE_W) != 0) {
    kfree(new);
    *pte = (*pte) | PTE_V;
    return 0;
  }
  kfree((void*)PGROUNDDOWN(pa));
  return (void*)new;
}
```

(b) ref > 1

三、测试结果 (make grade 后得到的结果)

```
== Test running cowtest ==
$ make qemu-gdb
(9.1s)
== Test   simple ==
  simple: OK
== Test   three ==
  three: OK
== Test   file ==
  file: OK
== Test usertests ==
$ make qemu-gdb
(70.0s)
    (Old xv6.out.usertests failure log removed)
== Test   usertests: copyin ==
  usertests: copyin: OK
== Test   usertests: copyout ==
  usertests: copyout: OK
== Test   usertests: all tests ==
  usertests: all tests: OK
```