Lab10：mmap

该实验由mmap组成。

1.mmap

实验目的

本次实验旨在向 xv6 操作系统添加 mmap 和 munmap 系统调用，实现对进程地址空间的详细控制。通过实现这两个系统调用，我们可以实现内存映射文件的功能，包括共享内存、将文件映射到进程地址空间等。这有助于理解虚拟内存管理和页面错误处理的机制。

实验步骤

1. 在makefile文件中添加mmaptest

    $U/\_mmaptest\

1. 仿照前面的实例，添加sys\_mmap和sys\_munmap系统调用

在syscall.h中添加：

extern uint64 sys\_mmap(void);

extern uint64 sys\_munmap(void);

[SYS\_mmap]    sys\_mmap,

[SYS\_munmap]  sys\_munmap,

在usys.pl中添加：

entry("mmap");

entry("munmap");

在user.h中添加：

void\* mmap(void \*, int, int, int, int, uint);

int munmap(void \*, int);

1. 在proc.c中定义一个`vm\_area`结构体来作为VMA，包含了内存信息、mmap映射的起始地址、mmap映射内存的大小、用户的权限、mmap标志位、文件的偏移量、文件结构体指针。

struct vm\_area

{

  struct file \*file;

  int fd;

  int used;

  uint64 addr;

  int length;

  int prot;

  int flags;

  int offset;

};

1. 在proc.h中添加

  struct vm\_area vma[VMASIZE]; // Virtual Memory Area

5.需要处理lazy allocation机制产生的page default，即修改`kernel/trap.c`中的`usertrap（）`函数。page fault的情况有三种，分别为r\_scause()为12,13或15，再根据缺页的地址找Vvm\_area结构体，进行lazy分配物理页，设置好对应的访问权限。

void

usertrap(void)

{

  int which\_dev = 0;

  if((r\_sstatus() & SSTATUS\_SPP) != 0)

    panic("usertrap: not from user mode");

  // send interrupts and exceptions to kerneltrap(),

  // since we're now in the kernel.

  w\_stvec((uint64)kernelvec);

  struct proc \*p = myproc();

  // save user program counter.

  p->trapframe->epc = r\_sepc();

  if(r\_scause() == 8){

    // system call

    if(p->killed)

      exit(-1);

    // sepc points to the ecall instruction,

    // but we want to return to the next instruction.

    p->trapframe->epc += 4;

    // an interrupt will change sstatus &c registers,

    // so don't enable until done with those registers.

    intr\_on();

    syscall();

  } else if((which\_dev = devintr()) != 0){

    // ok

  } else if (r\_scause() == 13 || r\_scause() == 15) {

    uint64 va = r\_stval();

    if(va >= p->sz || va > MAXVA || PGROUNDUP(va) == PGROUNDDOWN(p->trapframe->sp))

      p->killed = 1;

    else {

      struct vma \*vma = 0;

      for (int i = 0; i < VMASIZE; i++) {

        if (p->vma[i].used == 1 && va >= p->vma[i].addr &&

            va < p->vma[i].addr + p->vma[i].length) {

          vma = &p->vma[i];

          break;

        }

      }

      if(vma) {

        va = PGROUNDDOWN(va);

        uint64 offset = va - vma->addr;

        uint64 mem = (uint64)kalloc();

        if(mem == 0) {

          p->killed = 1;

        }

        else {

          memset((void\*)mem, 0, PGSIZE);

          ilock(vma->file->ip);

          readi(vma->file->ip, 0, mem, offset, PGSIZE);

          iunlock(vma->file->ip);

          int flag = PTE\_U;

          if(vma->prot & PROT\_READ) flag |= PTE\_R;

          if(vma->prot & PROT\_WRITE) flag |= PTE\_W;

          if(vma->prot & PROT\_EXEC) flag |= PTE\_X;

          if(mappages(p->pagetable, va, PGSIZE, mem, flag) != 0) {

            kfree((void\*)mem);

            p->killed = 1;

          }

        }

      }

    }

  } else {

    printf("usertrap(): unexpected scause %p pid=%d\n", r\_scause(), p->pid);

    printf("            sepc=%p stval=%p\n", r\_sepc(), r\_stval());

    p->killed = 1;

  }

  if(p->killed)

    exit(-1);

  // give up the CPU if this is a timer interrupt.

  if(which\_dev == 2)

    yield();

  usertrapret();

}

1. 在sysfile.c下实现两个系统调用。

根据实验指导，sys\_munmap会取消映射的内存，若取消的内存包括MAP\_SHARED那么还需要吧文件映射的内存修改回写到原文件。而实现sys\_munmap则需要根据变量addr和length在进程中寻找对应的vma结构体，根据不同的结果执行不同的操作，具体操作依照实验指导并参考了fiilewrite()函数。

void

syscall(void)

{

  int num;

  struct proc \*p = myproc();

  num = p->trapframe->a7;

  if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {

    p->trapframe->a0 = syscalls[num]();

  } else {

    printf("%d %s: unknown sys call %d\n",

            p->pid, p->name, num);

    p->trapframe->a0 = -1;

  }

}

uint64 sys\_mmap(void) {

  uint64 addr;

  int length, prot, flags, fd, offset;

  struct proc \*p = myproc();

  struct file \*file;

  if(argaddr(0, &addr) || argint(1, &length) || argint(2, &prot) ||

    argint(3, &flags) || argfd(4, &fd, &file) || argint(5, &offset))

    return -1;

  if(!file->writable && (prot & PROT\_WRITE) && flags == MAP\_SHARED)

    return -1;

  length = PGROUNDUP(length);

  if(p->sz > MAXVA - length)

    return -1;

  for(int i = 0; i < VMASIZE; i++) {

    if(p->vma[i].used == 0) {

      p->vma[i].used = 1;

      p->vma[i].addr = p->sz;

      p->vma[i].length = length;

      p->vma[i].prot = prot;

      p->vma[i].flags = flags;

      p->vma[i].fd = fd;

      p->vma[i].file = file;

      p->vma[i].offset = offset;

      filedup(file);

      p->sz += length;

      return p->vma[i].addr;

    }

  }

  return -1;

}

uint64

sys\_munmap(void)

{

  uint64 addr;

  int length;

  struct proc \*p = myproc();

  struct vma \*vma = 0;

  // 检查参数是否合法

  if(argaddr(0, &addr) || argint(1, &length))

    return -1;

  addr = PGROUNDDOWN(addr);

  length = PGROUNDUP(length);

  // 找对应vma结构体

  for(int i = 0; i < VMASIZE; i++) {

    if (addr >= p->vma[i].addr || addr < p->vma[i].addr + p->vma[i].length) {

      vma = &p->vma[i];

      break;

    }

  }

  if(vma == 0)

    return 0;

  if(vma->addr == addr) {

    vma->addr += length;

    vma->length -= length;

    if(vma->flags & MAP\_SHARED)

      filewrite(vma->file, addr, length);

    uvmunmap(p->pagetable, addr, length/PGSIZE, 1);

    if(vma->length == 0) {

      fileclose(vma->file);

      vma->used = 0;

    }

  }

  return 0;

}

6.在sysfile.c中修改uvmcopy和uvmunmap

为了防止非法panic，直接continue跳过即可。

    if((\*pte & PTE\_V) == 0)

      // panic("uvmcopy: page not present");

      continue;

7.exit和fork中的对应部分也要进行修改。

完成对进程文件映射内存和vma数组的处理：在exit()函数中需要遍历vma数组，取消掉所有文件映射内存的映射；在fork()函数中将父进程的vma数组复制到子进程即可。

exit():

  for(int i = 0; i < VMASIZE; i++) {

    if(p->vma[i].used) {

      if(p->vma[i].flags & MAP\_SHARED)

        filewrite(p->vma[i].file, p->vma[i].addr, p->vma[i].length);

      fileclose(p->vma[i].file);

      uvmunmap(p->pagetable, p->vma[i].addr, p->vma[i].length/PGSIZE, 1);

      p->vma[i].used = 0;

    }

  }

fork():

  for(int i = 0; i < VMASIZE; i++) {

    if(p->vma[i].used){

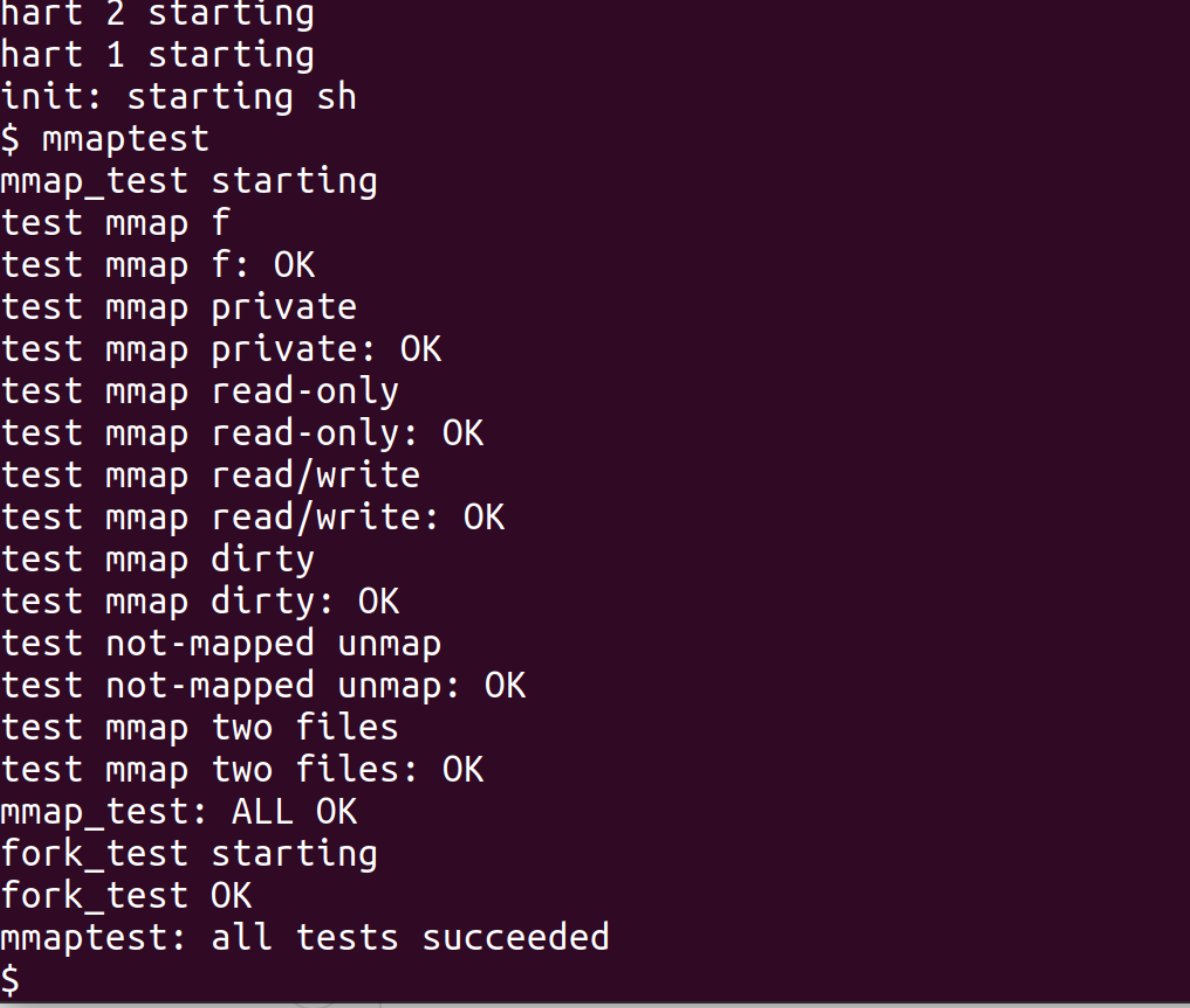
      memmove(&(np->vma[i]), &(p->vma[i]), sizeof(p->vma[i]));

      filedup(p->vma[i].file);

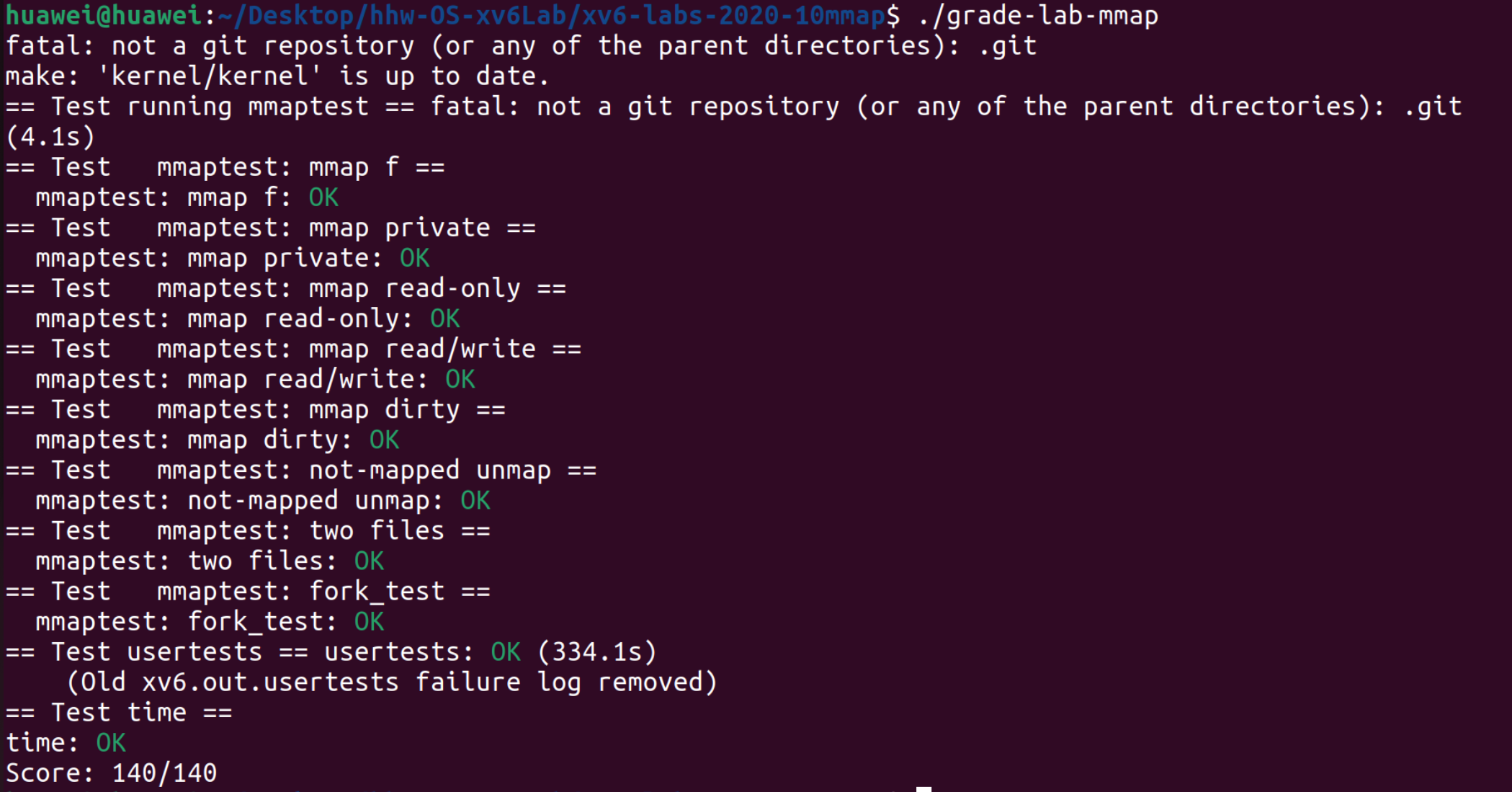
    }

  }

8.利用make qemu指令运行xv6, 在命令行中输入mmaptest:



9.退出xv6，输入./grade-lab-mmap:



遇到的问题与心得

遇到的问题：

在映射地址的确定过程中，需要考虑如何正确处理延迟申请的情况。

解决办法：选择从 trapframe 的底部向下生长，同时修改对 uvmunmap 的实现，使得只取消已经映射的页，以解决这个问题。

实验心得

本次实验也是关于系统与文件交互的内容，通过实验，实现了一个磁盘文件到虚拟内存地址的映射方式。这次实验让我对操作系统中的内存设计和组织理解更加深刻，让我了解了使用虚拟内存的原因和好处，理解创建和释放文件映射的方式，梳理了文件内存映射的生命周期。