

# Lab10: mmap

2351882 王小萌

Tongji University, 2025 Summer

代码仓库链接: <https://github.com/wmxmw/Tongji-University-xv6-labs-2021.git>

## 1.mmap

### 1.1 实验目的

### 1.2 实验步骤

### 1.3 实验中遇到的问题和解决办法

### 1.4 实验心得

## 2 实验检验得分

## 1 mmap

### 1.1 实验目的

掌握在操作系统中如何管理虚拟内存和内存映射。添加 mmap 与 munmap 两个系统调用，实现对进程地址空间的详细控制。前者将一个文件内存映射到进程的地址空间，后者取消已有地址空间的映射。

### 1.2 实验步骤

1. 在 kernel/syscall.h、syscall.c、defs.h、user/usys.pl、user.h 中添加有关 mmap 和 munmap 系统调用的定义声明“

```
21  #define SYS_mkdir  20
22  #define SYS_close  21
23  #define SYS_mmap   22
24  #define SYS_munmap 23
```

```
105  extern uint64 sys_write(void);
106  extern uint64 sys_uptime(void);
107  extern uint64 sys_mmap(void);
108  extern uint64 sys_munmap(void);
```

```
131  [SYS_close]    sys_close,
132  [SYS_mmap]     sys_mmap,
133  [SYS_munmap]   sys_munmap,
134  ];
135
```

```

37  entry("sleep");
38  entry("uptime");
39  entry("mmap");
40  entry("munmap");

```

```

25  int uptime(void);
26  int munmap(void *,int);
27  void *mmap(void *,int,int,int,int,int);
28

```

```

186  // sysfile.c
187  uint64 ..... munmap(uint64,int);
188

```

在 Makefile 中添加 \$U/\_mmaptest

```

189  $U/_wc\
190  $U/_zombie\
191  $U/_mmaptest\
192

```

在 kernel/proc.h 中定义 struct vma 结构体。并同时在 struct proc 结构体中定义相关字段。

```

85  struct vma{
86      int mapped;
87      uint64 addr;
88      int len;
89      int prot;
90      int flags;
91      int offset;
92      struct file *f;
93  }
94
95

```

```

96  // Per-process state
97  struct proc {
98      struct spinlock lock;
99      struct vma vma_table[MAXVMA]; //table of mapped regions
100     // p->lock must be held when using these:

```

2. 在 kernel/sysfile.c 中实现系统调用 sys\_mmap():

找到进程地址空间中未使用的区域，用于映射文件，并将 VMA 添加到进程的映射区域表中。VMA 应包含指向要映射文件的指针。确保在 mmap 中增加文件的引用计数，以防止文件在关闭时被删除：

```

488     uint64
489     sys_mmap(void)
490     {
491         // get argument and proc
492         struct proc *p = myproc();
493         uint64 addr;
494         int len;
495         int prot;
496         int flags;
497         int offset;
498         struct file *f;
499         if(argaddr(0, &addr) < 0 || argfd(4, 0, &f) < 0){
500             return -1;
501         }
502         if(argint(1, &len) < 0 || argint(2, &prot) < 0 || argint(3, &flags) <
503             return -1;
504         }
505         //check the mmaptest.c to solve ;Ensure permissions
506         if(!f->writable && (prot & PROT_WRITE) && flags == MAP_SHARED) return
507         // find a free vma and init
508         for(int i = 0; i < MAXVMA; i++){
509             if(p->vma_table[i].mapped == 0){
510                 p->vma_table[i].mapped = 1;
511                 //if addr = 0, vma_table[i].addr is the top of heap (i.e. bottom
512                 p->vma_table[i].addr = addr + p->sz;
513                 p->vma_table[i].len = PGROUNDUP(len);
514                 p->vma_table[i].prot = prot;
515                 p->vma_table[i].flags = flags;
516                 p->vma_table[i].offset = offset;
517                 p->vma_table[i].f = filedup(f);
518                 p->sz += PGROUNDUP(len);
519                 return p->vma_table[i].addr;
520             }
521         }
522         return -1;
523     }
524
525

```

### 3. kernel/sysfile.c 中实现系统调用 munmap():

找到要取消映射的地址范围的 VMA，并取消映射指定的页面。如果 file 的引用计数。如果一个页面被修改且文件是 munmap()，找到要取消映射的地址范围的 VMA，并取 munmap 移除了前一个 mmap 的所有页面，则应减少相应的 struct MAP\_SHARED 映射，则将修改的页面写回文件

```

525 uint64
526 munmap(uint64 addr, int len){// access arg and proc
527     struct proc *p = myproc();
528     struct vma *pvma;
529     int i = 0;
530     // find target VMA
531     for(; i < MAXVMA; i++){
532         pvma = &p->vma_table[i];
533         if(pvma->mapped == 1 && addr >= pvma->addr && ((addr + len) < (pvma->addr + pvma->len))){
534             break;
535         }
536     }
537     // not found
538     if(i > MAXVMA){
539         return -1;
540     }
541     int end = addr + len;
542     int _addr = addr;
543     //readonly file can be MAP_SHARED, so need another condition f->writable
544     // If the page has already been mapped, write back to modify and remap it
545     if((pvma->flags == MAP_SHARED) && pvma->f->writable){
546         //steal from filewritei()
547         while(addr < end){
548             int size = min(end-addr, PGSIZE);
549             begin_op();
550             ilock(pvma->f->ip);
551             if(writei(pvma->f->ip, 1, addr, addr - pvma->addr, size) != size){
552                 return -1;
553             }
554             iunlock(pvma->f->ip);
555             end_op();
556             uvmunmap(p->pagetable, addr, 1, 1);
557             addr += PGSIZE;
558         }
559     }
560     //Maintain file references and VMA validity
561     if(_addr == pvma->addr){
562         //head
563         pvma->addr += len;
564         pvma->len -= len;
565     }else if(_addr + len == pvma->addr + pvma->len){

```

file.c 及 defs.h 中补充声明及实现 fileundup 函数:

```

28 // file.c
29 struct file* filealloc(void);
30 void fileclose(struct file*);
31 struct file* filedup(struct file*);
32 struct file* fileundup(struct file*);

```

```

58 struct file*
59 fileundup(struct file *f)
60 {
61     acquire(&ftable.lock);
62     if(f->ref < 1)
63         panic("fileundup");
64     f->ref--;
65     release(&ftable.lock);
66     return f;
67 }
68

```

4. 在 kernel/sysfile.c 中补充实现系统调用 sys\_munmap():

```

577     uint64
578     sys_munmap(void)
579     {
580         uint64 addr;
581         int len;
582         if(argaddr(0, &addr) < 0 || argint(1, &len) < 0){
583             return -1;
584         }
585         return munmap(addr, len);
586         return 0;
587     }

```

5. 修改 kernel/trap.c 中的 usertrap() 函数，处理页错误（Page Fault）情况。由于在 sys\_mmap() 中对文件映射的内存采用的是 Lazy allocation，在访问未加载界面时，会产生一个缺页中断。因此需要对访问文件映射内存产生的 page fault 进行处理。当在 mmap 映射的区域发生页面错误时，分配一个物理内存页，从相关文件中读取 4096 字节到该页面，并将其映射到用户地址空间。

先添加头文件：

```

kernel > C trap.c
1  ✓ #include "types.h"
2    #include "param.h"
3    #include "memlayout.h"
4    #include "riscv.h"
5    #include "spinlock.h"
6    #include "proc.h"
7    #include "defs.h"
8    #include "sleeplock.h"
9    #include "fs.h"
10   #include "file.h"

```

```

kernel > C trap.c
37  usertrap(void)
53  if(r_scause() == 8){
67      syscall();
68  } else if(r_scause() == 13 || r_scause() == 15){ // interrupt
69      uint64 va = r_stval();
70      struct vma *pvma;
71      int i = 0;
72      //p->sz point to the top of heap (i.e. bottom of trapframe)
73      //p->trapframe->sp point to the top of stack
74      if((va >= p->sz) || (va < PGROUNDDOWN(p->trapframe->sp))){
75          p->killed = 1;
76      } else{
77          va = PGROUNDDOWN(va);
78          for(; i < MAXVMA; i++){
79              pvma = &p->vma_table[i];
80              if(pvma->mapped && (va >= pvma->addr) && (va < (pvma->addr + pvma->len)
81                  char *mem;
82                  mem = kalloc();
83                  if(mem == 0){
84                      p->killed = 1;
85                      break;
86                  }
87                  memset(mem, 0, PGSIZE);
88                  //PTE_R (1L << 1), so prot also needs to move left one bit
89                  if(mappages(p->pagetable, va, PGSIZE, (uint64)mem, (pvma->prot << 1)
90                      kfree(mem);
91                      p->killed = 1;
92                      break;
93                  }
94                  break;
95              }
96          }
97      }
98      if(p->killed != 1 && i <= MAXVMA){
99          ilock(pvma->f->ip);

```

6. 修改 kernel/proc.c 中的 exit() 函数和 fork() 函数，复制、删除当前进程的 VMA 字段。确保子进程与父进程具有相同的映射区域。不要忘记增加 VMA 的 struct file 的引用计数。在子进程的页面错误处理程序中，可以分配新的物理页面，而不是与父进程共享页面。

```

303  for(int i=0; i<MAXVMA; i++){
304      if(p->vma_table[i].mapped){
305          memmove(&np->vma_table[i], &p->vma_table[i], sizeof(struct vma));
306          filedup(np->vma_table[i], f);
307      }
308  }
309  safestrcpy(np->name, p->name, sizeof(p->name));

```

```

360     struct vma *pvma;
361     for(int i = 0; i < MAXVMA; i++){
362         pvma = &p->vma_table[i];
363         if(pvma->mapped){
364             //uvmunmap(p->pagetable, pvma->addr, pvma->len / PGSIZE, 0);
365             //memset(pvma, 0, sizeof(struct vma));
366             if(munmap(pvma->addr, pvma->len) < 0){
367                 panic("exit munmap");
368             }
369         }
370     }
371
372     begin_op();
373     iput(p->cwd);
374     end_op();
375     p->cwd = 0;

```

## 7. 修改 kernel/vm.c 中的 uvmcopy() 函数

```

300     int
301     uvmcopy(pagetable_t old, pagetable_t new, uint64 sz)
302     {
303         pte_t *pte;
304         uint64 pa, i;
305         uint flags;
306         char *mem;
307
308         for(i = 0; i < sz; i += PGSIZE){
309             if((pte = walk(old, i, 0)) == 0)
310                 panic("uvmcopy: pte should exist");
311             if((*pte & PTE_V) == 0)
312                 continue;
313             //panic("uvmcopy: page not present");

```

## 8. 运行，命令行中输入 mmaptest:

```

xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ mmaptest
mmap_test starting
test mmap f
test mmap f: OK
test mmap private
test mmap private: OK
test mmap read-only
test mmap read-only: OK
test mmap read/write
test mmap read/write: OK
test mmap dirty
test mmap dirty: OK
test not-mapped unmap
test not-mapped unmap: OK
test mmap two files
test mmap two files: OK
mmap_test: ALL OK
fork_test starting
fork_test OK
mmaptest: all tests succeeded

```

## 9. 单项得分

```
score: 140/140
vale@Puppyyoo: ~/xv6-labs-2021$ ./grade-lab-mmap mmap
make: 'kernel/kernel' is up to date.
== Test running mmaptest == (1.7s)
== Test mmaptest: mmap f ==
mmaptest: mmap f: OK
== Test mmaptest: mmap private ==
mmaptest: mmap private: OK
== Test mmaptest: mmap read-only ==
mmaptest: mmap read-only: OK
== Test mmaptest: mmap read/write ==
mmaptest: mmap read/write: OK
== Test mmaptest: mmap dirty ==
mmaptest: mmap dirty: OK
== Test mmaptest: not-mapped unmap ==
mmaptest: not-mapped unmap: OK
== Test mmaptest: two files ==
mmaptest: two files: OK
== Test mmaptest: fork_test ==
mmaptest: fork_test: OK
vale@Puppyyoo: ~/xv6-labs-2021$ _
```

## 1.3 实验中遇到的问题和解决办法

**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。

## 1.4 实验心得

在整个实验过程中，我加深了对操作系统内存管理的理解，掌握了操作系统内核中的数据结构和函数调用，提升了我的编程和调试能力。通过不断的实践和探索，我对操作系统的各个组成部分有了更深入的认识，为我今后在系统编程和操作系统领域的学习和研究打下了坚实的基础。mmap 系统调用的核心是将文件内容映射到虚拟内存中。这涉及到文件系统和虚拟内存的结合。操作系统需要维护文件的相关信息，如打开的文件描述符，然后通过文件描述符来获取文件内容，并将其映射到虚拟内存的合适位置。这种结合使得用户程序可以直接访问文件内容，而无需显式的读写操作。

## 2 实验检验得分

1. 在实验目录下创建 time.txt，填写完成实验时间数。

 time.txt

2025/8/26 19:47

文本文档

1 KB

2. 在终端中执行 make grade,得到 lab10 总分：



```

make[1]: Leaving directory '/home/vale/xv6-labs-2021'
== Test running mmaptest ==
$ make qemu-gdb
(2.2s)
== Test mmaptest: mmap f ==
mmaptest: mmap f: OK
== Test mmaptest: mmap private ==
mmaptest: mmap private: OK
== Test mmaptest: mmap read-only ==
mmaptest: mmap read-only: OK
== Test mmaptest: mmap read/write ==
mmaptest: mmap read/write: OK
== Test mmaptest: mmap dirty ==
mmaptest: mmap dirty: OK
== Test mmaptest: not-mapped unmap ==
mmaptest: not-mapped unmap: OK
== Test mmaptest: two files ==
mmaptest: two files: OK
== Test mmaptest: fork_test ==
mmaptest: fork_test: OK
== Test usertests ==
$ make qemu-gdb
usertests: OK (67.4s)
== Test time ==
time: OK
Score: 140/140
vale@Puppyyoo:~/xv6-labs-2021$

```

### 3. Lab10 代码提交

保存并提交到本地分支:

```

vale@Puppyyoo:~/xv6-labs-2021$ git add .
vale@Puppyyoo:~/xv6-labs-2021$ git commit -m"lab10 finished"
[mmap abff3a5] lab10 finished
25 files changed, 231 insertions(+), 9 deletions(-)
create mode 100644 Makefile:Zone.Identifier
create mode 100644 kernel/defs.h:Zone.Identifier
create mode 100644 kernel/file.c:Zone.Identifier
create mode 100644 kernel/proc.c:Zone.Identifier
create mode 100644 kernel/proc.h:Zone.Identifier
create mode 100644 kernel/syscall.c:Zone.Identifier
create mode 100644 kernel/syscall.h:Zone.Identifier
create mode 100644 kernel/sysfile.c:Zone.Identifier
create mode 100644 kernel/trap.c:Zone.Identifier
create mode 100644 kernel/vm.c:Zone.Identifier
create mode 100644 time.txt
create mode 100644 user/user.h:Zone.Identifier
create mode 100644 user/usys.pl:Zone.Identifier
vale@Puppyyoo:~/xv6-labs-2021$ git status
On branch mmap
Your branch is ahead of 'origin/mmap' by 1 commit.
(use "git push" to publish your local commits)

nothing to commit, working tree clean
vale@Puppyyoo:~/xv6-labs-2021$

```