Lab3 实验报告

姓名：李雨朔　学号：20232131090　班级：计科一班　日期：2025.11.16

一． 实验环境准备

```
git fetch
git checkout pgtbl
make clean
```

二． 实验 3.1 系统调用性能优化

1. 修改进程结构体。在 kernel/proc.h 中找到 struct proc，在其中添加 usyscall 指针：

```
struct usyscall *usyscall; // 用户系统调用页面
```

```
// these are private to the process, so p->lock need not be held.
uint64 kstack;               // Virtual address of kernel stack
uint64 sz;                   // Size of process memory (bytes)
pagetable_t pagetable;       // User page table
struct trapframe *trapframe; // data page for trampoline.S
struct usyscall *usyscall;
struct context context;      // swtch() here to run process
struct file *ofile[NOFILE];  // Open files
struct inode *cwd;           // Current directory
char name[16];               // Process name (debugging)
```

2. 修改 allocproc 函数。在 kernel/proc.c 中找到 allocproc 函数，在其中添加 USYSCALL 页面的分配和初始化：

```
// 分配 USYSCALL 页面
 if((p->usyscall = (struct usyscall *)kalloc()) == 0){
   freeproc(p);
   release(&p->lock);
   return 0;
 }
 p->usyscall->pid = p->pid; // 初始化 pid
```

```
      goto found;
    } else {
      release(&p->lock);
    }
  }
  return 0;

found:
  p->pid = allocpid();
  p->state = USED;

  // Allocate a trapframe page.
  if((p->trapframe = (struct trapframe *)kalloc()) == 0){
    freeproc(p);
    release(&p->lock);
    return 0;
  }

  if((p->usyscall = (struct usyscall *)kalloc()) == 0) {
    freeproc(p);
    release(&p->lock);
    return 0;
  }
  p->usyscall->pid = p->pid;
```

3. 修改 proc_pagetable 函数。在 kernel/proc.c 中添加 USYSCALL 页面的映射:

```
// 映射 USYSCALL 页面
 if(mappages(pagetable, USYSCALL, PGSIZE,
          (uint64)(p->usyscall), PTE_R | PTE_U) < 0){
   uvmunmap(pagetable, TRAMPOLINE, 1, 0);
   uvmunmap(pagetable, TRAPFRAME, 1, 0);
   uvmfree(pagetable, 0);
   return 0;
 }
```

```c
pagetable_t
proc_pagetable(struct proc *p)
{
  pagetable_t pagetable;

  // An empty page table.
  pagetable = uvmcreate();
  if(pagetable == 0)
    return 0;

  // map the trampoline code (for system call return)
  // at the highest user virtual address.
  // only the supervisor uses it, on the way
  // to/from user space, so not PTE_U.
  if(mappages(pagetable, TRAMPOLINE, PGSIZE,
              (uint64)trampoline, PTE_R | PTE_X) < 0){
    uvmfree(pagetable, 0);
    return 0;
  }

  // map the trapframe page just below the trampoline page, for
  // trampoline.S.
  if(mappages(pagetable, TRAPFRAME, PGSIZE,
              (uint64)(p->trapframe), PTE_R | PTE_W) < 0){
    uvmunmap(pagetable, TRAMPOLINE, 1, 0);
    uvmfree(pagetable, 0);
    return 0;
  }

  if(mappages(pagetable, USYSCALL, PGSIZE,
              (uint64)(p->usyscall), PTE_R | PTE_U) < 0) {
    uvmunmap(pagetable, TRAMPOLINE, 1, 0);
    uvmunmap(pagetable, TRAPFRAME, 1, 0);
    uvmfree(pagetable, 0);
    return 0;
  }

  return pagetable;
}
```

4. 修改 freeproc 函数，释放 USYSCALL 页面：

```c
if(p->usyscall)
  kfree((void*)p->usyscall); // 释放 USYSCALL 页面
 p->usyscall = 0;
```

```
static void
freeproc(struct proc *p)
{
  if(p->trapframe)
    kfree((void*)p->trapframe);
  p->trapframe = 0;
  if(p->usyscall)
    kfree((void*)p->usyscall);
  p->usyscall = 0;
  if(p->pagetable)
    proc_freepagetable(p->pagetable, p->sz);
  p->pagetable = 0;
  p->sz = 0;
  p->pid = 0;
  p->parent = 0;
  p->name[0] = 0;
  p->chan = 0;
  p->killed = 0;
  p->xstate = 0;
  p->state = UNUSED;
}
```

5. 修改 proc_freepagetable 函数，取消 USYSCALL 页面的映射：

```
uvmunmap(pagetable, USYSCALL, 1, 0);  // 取消映射 USYSCALL
```

```
// Free a process's page table, and free the
// physical memory it refers to.
void
proc_freepagetable(pagetable_t pagetable, uint64 sz)
{
  uvmunmap(pagetable, TRAMPOLINE, 1, 0);
  uvmunmap(pagetable, TRAPFRAME, 1, 0);
  uvmunmap(pagetable, USYSCALL, 1, 0);
  uvmfree(pagetable, sz);
}
```

6. 编译并测试：

```
make qemu
pgtbltest
```

```
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwar
strncpy -fno-builtin-strncmp -fno-builtin-strlen -fno-builtin-memset -fno-bui
trchr -fno-builtin-exit -fno-builtin-malloc -fno-builtin-putc -fno-builtin-fr
tin-vprintf -I. -fno-stack-protector -fno-pie -no-pie   -c -o user/pgtbltest.
riscv64-unknown-elf-ld -z max-page-size=4096 -T user/user.ld -o user/_pgtblte
riscv64-unknown-elf-objdump -S user/_pgtbltest > user/pgtbltest.asm
riscv64-unknown-elf-objdump -t user/_pgtbltest | sed '1,/SYMBOL TABLE/d; s/ .
mkfs/mkfs fs.img README  user/_cat user/_echo user/_forktest user/_grep user/
r/_usertests user/_grind user/_wc user/_zombie  user/_pgtbltest
nmeta 46 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1) blocks
balloc: first 839 blocks have been allocated
balloc: write bitmap block at sector 45
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -s
ormat=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
```

```
$ pgtbltest
print_pgtbl starting
va 0x0 pte 0x21FC7C5B pa 0x87F1F000 perm 0x5B
va 0x1000 pte 0x21FC7017 pa 0x87F1C000 perm 0x17
va 0x2000 pte 0x21FC6C07 pa 0x87F1B000 perm 0x7
va 0x3000 pte 0x21FC68D7 pa 0x87F1A000 perm 0xD7
va 0x4000 pte 0x0 pa 0x0 perm 0x0
va 0x5000 pte 0x0 pa 0x0 perm 0x0
va 0x6000 pte 0x0 pa 0x0 perm 0x0
va 0x7000 pte 0x0 pa 0x0 perm 0x0
va 0x8000 pte 0x0 pa 0x0 perm 0x0
va 0x9000 pte 0x0 pa 0x0 perm 0x0
va 0xFFFF6000 pte 0x0 pa 0x0 perm 0x0
va 0xFFFF7000 pte 0x0 pa 0x0 perm 0x0
va 0xFFFF8000 pte 0x0 pa 0x0 perm 0x0
va 0xFFFF9000 pte 0x0 pa 0x0 perm 0x0
va 0xFFFFA000 pte 0x0 pa 0x0 perm 0x0
va 0xFFFFB000 pte 0x0 pa 0x0 perm 0x0
va 0xFFFFC000 pte 0x0 pa 0x0 perm 0x0
va 0xFFFFD000 pte 0x21FD4C13 pa 0x87F53000 perm 0x13
va 0xFFFFE000 pte 0x21FD00C7 pa 0x87F40000 perm 0xC7
va 0xFFFFF000 pte 0x2000184B pa 0x80006000 perm 0x4B
print_pgtbl: OK
ugetpid_test starting
ugetpid_test: OK
print_kpgtbl starting
print_kpgtbl: OK
superpg_test starting
pgtbltest: superpg_test failed: pte different, pid=3
```

三． 实验 3.2 打印进程 1 的页表

1. 在 kernel/vm.c 中实现 vmprint 函数：

```c
// 递归打印页表
void
vmprint_rec(pagetable_t pagetable, int level)
{
  // 遍历页表中的所有 PTE
  for(int i = 0; i < 512; i++){
    pte_t pte = pagetable[i];
    if(pte & PTE_V){
      // 打印当前 PTE，根据层级打印不同的前缀
      for(int j = 0; j < level; j++) {
        printf(".. ");
      }
      uint64 child = PTE2PA(pte);
      printf("..%d: pte %p pa %p\n", i, pte, child);

      // 如果当前不是叶子节点，递归打印下一级
      if((pte & (PTE_R|PTE_W|PTE_X)) == 0){
        vmprint_rec((pagetable_t)child, level + 1);
      }
    }
  }
}

void
vmprint(pagetable_t pagetable)
{
  printf("page table %p\n", pagetable);
  vmprint_rec(pagetable, 1);
}
```

```c
#ifdef LAB_PGTBL
void
vmprint(pagetable_t pagetable, int level) {
  // your code here
  for(int i=0; i<512; i++) {
    pte_t pte = pagetable[i];
    if(pte & PTE_V) {
      for(int j=0; j<level; j++) {
        printf(".. ");
      }
      uint64 child = PTE2PA(pte);
      printf("..%d: pte %p pa %p\n", i, pte, child);

      if((pte & (PTE_R|PTE_W|PTE_X))==0) {
        vmprint_rec((pagetable_t)child, level+1);
      }
    }
  }
}
#endif

void
vmprint(pagetable_t pagetable)
{
  printf("page table %p\n", pagetable);
  vmprint_rec(pagetable, 1);
}
```

2. 在 kernel/defs.h 中声明函数:

```c
void        vmprint(pagetable_t);
```

```c
// vm.c
void            kvminit(void);
void            kvminithart(void);
void            kvmmap(pagetable_t, uint64, uint64, uint64, int);
int             mappages(pagetable_t, uint64, uint64, uint64, int);
pagetable_t     uvmcreate(void);
void            uvmfirst(pagetable_t, uchar *, uint);
uint64          uvmalloc(pagetable_t, uint64, uint64, int);
uint64          uvmdealloc(pagetable_t, uint64, uint64);
int             uvmcopy(pagetable_t, pagetable_t, uint64);
void            uvmfree(pagetable_t, uint64);
void            uvmunmap(pagetable_t, uint64, uint64, int);
void            uvmclear(pagetable_t, uint64);
pte_t *         walk(pagetable_t, uint64, int);
uint64          walkaddr(pagetable_t, uint64);
int             copyout(pagetable_t, uint64, char *, uint64);
int             copyin(pagetable_t, char *, uint64, uint64);
int             copyinstr(pagetable_t, char *, uint64, uint64);
#if defined(LAB_PGTBL) || defined(SOL_MMAP)
void            vmprint(pagetable_t);
#endif
#ifdef LAB_PGTBL
```

3. 在 exec 中调用 kernel/vmprint，在 exec 函数末尾添加以下代码:

```
// 在返回用户空间之前，如果是进程 1，则打印页表
 if(p->pid == 1) {
   vmprint(p->pagetable);
 }
```

4. 运行测试，发现问题：

```
oslab@xv6-vm:~/xv6-labs-2024$ make qemu
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2 -DSOL_PGTBL -DLAB_PGTBL -MD -mcmodel=medany -fno-common -nostdlib -fno-builtin-
strncpy -fno-builtin-strncmp -fno-builtin-strlen -fno-builtin-memset -fno-builtin-memmove -fno-builtin-memcmp -fno-builtin-log -fno-builtin-bzero -fno-builtin-s
trchr -fno-builtin-exit -fno-builtin-malloc -fno-builtin-putc -fno-builtin-free -fno-builtin-memcpy -Wno-main -fno-builtin-printf -fno-builtin-fprintf -fno-bui
tin-vprintf -I. -fno-stack-protector -fno-pie -no-pie  -c -o kernel/vm.o kernel/vm.c
kernel/vm.c:492:1: error: conflicting types for 'vmprint'; have 'void(uint64 *, int)' {aka 'void(long unsigned int *, int)'}
  492 | vmprint(pagetable_t pagetable, int level) {
      | ^~~~~~~
In file included from kernel/vm.c:6:
kernel/defs.h:185:17: note: previous declaration of 'vmprint' with type 'void(uint64 *)' {aka 'void(long unsigned int *)'}
  185 | void            vmprint(pagetable_t);
      |                 ^~~~~~~
kernel/vm.c: In function 'vmprint':
kernel/vm.c:501:26: error: format '%p' expects argument of type 'void *', but argument 3 has type 'pte_t' {aka 'long unsigned int'} [-Werror=format=]
  501 |       printf("..%d: pte %p pa %p\n", i, pte, child);
      |                         ~^              ~~~
      |                          |              |
      |                          void *         pte_t {aka long unsigned int}
      |                         %ld
kernel/vm.c:501:32: error: format '%p' expects argument of type 'void *', but argument 4 has type 'uint64' {aka 'long unsigned int'} [-Werror=format=]
  501 |       printf("..%d: pte %p pa %p\n", i, pte, child);
      |                               ~^             ~~~~~
      |                                |             |
      |                                void *        uint64 {aka long unsigned int}
      |                               %ld
kernel/vm.c:504:9: error: implicit declaration of function 'vmprint_rec'; did you mean 'vmprint'? [-Werror=implicit-function-declaration]
  504 |         vmprint_rec((pagetable_t)child, level+1);
      |         ^~~~~~~~~~~
      |         vmprint
cc1: all warnings being treated as errors
make: *** [Makefile:136: kernel/vm.o] Error 1
```

问题 1：error: conflicting types for 'vmprint'（在 defs.h 中声明的 vmprint 只有一个参数：void vmprint(pagetable_t)，但在 vm.c 中定义的 vmprint 有两个参数：void vmprint(pagetable_t pagetable, int level)）

问题 2：error: format '%p' expects argument of type 'void *'（%p 需要 void* 类型参数，但传递的是 pte_t 和 uint64 类型）

问题 3：error: implicit declaration of function 'vmprint_rec'（编译器找不到 vmprint_rec 函数的声明）

解决方法：

1) 将所有 %p 格式的参数转换为 (void*)：

```
#ifdef LAB_PGTBL
void
vmprint_rec(pagetable_t pagetable, int level) {
  // your code here
  for(int i=0; i<512; i++) {
    pte_t pte = pagetable[i];
    if(pte & PTE_V) {
      for(int j=0; j<level; j++) {
        printf(".. ");
      }
      uint64 child = PTE2PA(pte);
      printf("..%d: pte %p pa %p\n", i, (void*)pte, (void*)child);

      if((pte & (PTE_R|PTE_W|PTE_X))==0) {
        vmprint_rec((pagetable_t)child, level+1);
      }
    }
  }
}
#endif

void
vmprint(pagetable_t pagetable)
{
  printf("page table %p\n", (void*)pagetable);
  vmprint_rec(pagetable, 1);
}
```

2) 在 kernel/defs.h 中添加递归函数声明：

void vmprint_rec(pagetable_t, int);

```
// vm.c
void            kvminit(void);
void            kvminithart(void);
void            kvmmap(pagetable_t, uint64, uint64, uint64, int);
int             mappages(pagetable_t, uint64, uint64, uint64, int)
pagetable_t     uvmcreate(void);
void            uvmfirst(pagetable_t, uchar *, uint);
uint64          uvmalloc(pagetable_t, uint64, uint64, int);
uint64          uvmdealloc(pagetable_t, uint64, uint64);
int             uvmcopy(pagetable_t, pagetable_t, uint64);
void            uvmfree(pagetable_t, uint64);
void            uvmunmap(pagetable_t, uint64, uint64, int);
void            uvmclear(pagetable_t, uint64);
pte_t *         walk(pagetable_t, uint64, int);
uint64          walkaddr(pagetable_t, uint64);
int             copyout(pagetable_t, uint64, char *, uint64);
int             copyin(pagetable_t, char *, uint64, uint64);
int             copyinstr(pagetable_t, char *, uint64, uint64);
#if defined(LAB_PGTBL) || defined(SOL_MMAP)
void            vmprint_rec(pagetable_t, int);
void            vmprint(pagetable_t);
#endif
```

5. 重新编译运行：

```
balloc: first 835 blocks have been allocated
balloc: write bitmap block at sector 45
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -g
ormat=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 2 starting
hart 1 starting
page table 0x0000000087f4d000
.. ..0: pte 0x0000000021fd2401 pa 0x0000000087f49000
.. .. ..0: pte 0x0000000021fd2001 pa 0x0000000087f48000
.. .. .. ..0: pte 0x0000000021fd281b pa 0x0000000087f4a000
.. .. .. ..1: pte 0x0000000021fd1c17 pa 0x0000000087f47000
.. .. .. ..2: pte 0x0000000021fd1807 pa 0x0000000087f46000
.. .. .. ..3: pte 0x0000000021fd1417 pa 0x0000000087f45000
.. ..255: pte 0x0000000021fd3001 pa 0x0000000087f4c000
.. .. ..511: pte 0x0000000021fd2c01 pa 0x0000000087f4b000
.. .. .. ..509: pte 0x0000000021fd5413 pa 0x0000000087f55000
.. .. .. ..510: pte 0x0000000021fd5807 pa 0x0000000087f56000
.. .. .. ..511: pte 0x000000002000180b pa 0x0000000080006000
init: starting sh
$
```

四． 实验 3.3 为 xv6 添加超级页支持

1． 在 kernel/kalloc.c 中添加超级页分配器:

```
// 超级页分配器
void*
superalloc(void)
{
  // 分配 512 个连续的 4KB 页面来组成一个 2MB 超级页
  char *first = kalloc();
  if(first == 0)
    return 0;

  // 检查是否 2MB 对齐
  if((uint64)first % (2*1024*1024) != 0) {
    kfree(first);
    return 0;
  }

  // 分配剩余的 511 个页面
  for(int i = 1; i < 512; i++) {
    char *page = kalloc();
    if(page == 0) {
      // 分配失败，释放之前分配的页面
      for(int j = 0; j < i; j++) {
        kfree((void*)((uint64)first + j * PGSIZE));
      }
```

```
      return 0;
    }
  }

  return first;
}

void
superfree(void *pa)
{
  // 释放 512 个连续的 4KB 页面
  for(int i = 0; i < 512; i++) {
    kfree((void*)((uint64)pa + i * PGSIZE));
  }
}
```

```
void*
superalloc(void) {
  char *ptr = kalloc();
  if(ptr == 0) return 0;

  uint64 pa = (uint64)ptr;
  if(pa%(2*1024*1024) != 0) {
    kfree(ptr);
    return 0;
  }

  return ptr;
}

void
superfree(void *pa) {
  kfree(pa);_
}
```

2． 在 kernel/vm.c 中添加超级页映射和修改相关函数:

先在文件开头添加辅助函数:

```
// 检查是否可以使用超级页
int
can_use_superpage(uint64 va, uint64 size)
{
  return (size >= PGSIZE * 512) && ((va & (PGSIZE * 512 - 1)) == 0);
}

// 检查 PTE 是否是超级页
int
is_superpage(pte_t pte)
{
```

```
  return (pte & PTE_V) && (pte & (PTE_R|PTE_W|PTE_X));
}
```

```
int
can_use_superpage(uint64 va, uint64 size) {
  return(size>=PGSIZE*512) && ((va&(PGSIZE*512-1))==0);
}

int
is_superpage(pte_t pte) {
  return(pte&PTE_V) && (pte&(PTE_R|PTE_W|PTE_X));
}
```

然后添加超级页映射函数:

```
// 映射超级页
int
supermappages(pagetable_t pagetable, uint64 va, uint64 size, uint64 pa, int perm)
{
  if((va % (PGSIZE * 512)) != 0)
    panic("supermappages: va not aligned");

  if((pa % (PGSIZE * 512)) != 0)
    panic("supermappages: pa not aligned");

  // 对于超级页 只需要在 L1 级别创建一个 PTE
  pte_t *pte = walk(pagetable, va, 1);
  if(pte == 0)
    return -1;
  if(*pte & PTE_V)
    panic("supermappages: remap");

  *pte = PA2PTE(pa) | perm | PTE_V;
  return 0;
}
```

```
int
supermappages(pagetable_t pagetable, uint64 va, uint64 size, uint64 pa, int perm) {
  if((va%(PGSIZE*512)) != 0)
    panic("supermappages: va not aligned");

  if((pa%(PGSIZE*512)) != 0)
    panic("supermappages: pa not aligned");

  pte_t *pte = walk(pagetable, va, 1);

  if(pte==0) return -1;
  if(*pte & PTE_V)
    panic("supermappages: remap");

  *pte = PA2PTE(pa) | perm | PTE_V;

  return 0;
}
```

找到现有的 uvmalloc 函数，修改为：

```
uint64
uvmalloc(pagetable_t pagetable, uint64 oldsz, uint64 newsz, int xperm)
{
  char *mem;
  uint64 a;

  if(newsz < oldsz)
    return oldsz;

  oldsz = PGROUNDUP(oldsz);
  for(a = oldsz; a < newsz; a += PGSIZE){

    // 检查是否可以使用超级页
    if(can_use_superpage(a, newsz - a)) {
      mem = (char*)superalloc();
      if(mem == 0){
        uvmdealloc(pagetable, a, oldsz);
        return 0;
      }
      memset(mem, 0, PGSIZE * 512);
      if(supermappages(pagetable, a, PGSIZE * 512, (uint64)mem,
                  PTE_R|PTE_W|PTE_U|xperm) != 0){
        superfree(mem);
        uvmdealloc(pagetable, a, oldsz);
        return 0;
      }
      a += PGSIZE * 512 - PGSIZE; // 跳过 2MB
      continue;
    }

    // 原有的 4KB 页面分配
    mem = kalloc();
    if(mem == 0){
      uvmdealloc(pagetable, a, oldsz);
      return 0;
    }
    memset(mem, 0, PGSIZE);
    if(mappages(pagetable, a, PGSIZE, (uint64)mem, PTE_R|PTE_W|PTE_U|xperm) != 0){
      kfree(mem);
      uvmdealloc(pagetable, a, oldsz);
      return 0;
```

```
    }
  }
  return newsz;
}
```

```
// newsz, which need not be page aligned.  Returns new size or 0 on error.
uint64
uvmalloc(pagetable_t pagetable, uint64 oldsz, uint64 newsz, int xperm)
{
  char *mem;
  uint64 a;
  //int sz;

  if(newsz < oldsz)
    return oldsz;

  oldsz = PGROUNDUP(oldsz);
```

```
  //}
  for(a=oldsz; a<newsz; a+=PGSIZE) {
    if(can_use_superpage(a, newsz-a)) {
      mem = (char*)superalloc();
      if(mem == 0) {
        uvmdealloc(pagetable, a, oldsz);
        return 0;
      }
      memset(mem, 0, PGSIZE*512);
      if(supermappages(pagetable, a, PGSIZE*512, (uint64)mem, PTE_R|PTE_W|PTE_U|xperm) != 0) {
        superfree(mem);
        uvmdealloc(pagetable, a, oldsz);
        return 0;
      }
      a += PGSIZE*512-PGSIZE;
      continue;
    }

    mem = kalloc();
    if(mem == 0) {
      uvmdealloc(pagetable, a, oldsz);
      return 0;
    }
    memset(mem, 0, PGSIZE);
    if(mappages(pagetable, a, PGSIZE, (uint64)mem, PTE_R|PTE_W|PTE_U|xperm) != 0) {
      kfree(mem);
      uvmdealloc(pagetable, a, oldsz);
      return 0;
    }
  }
  return newsz;
}
```

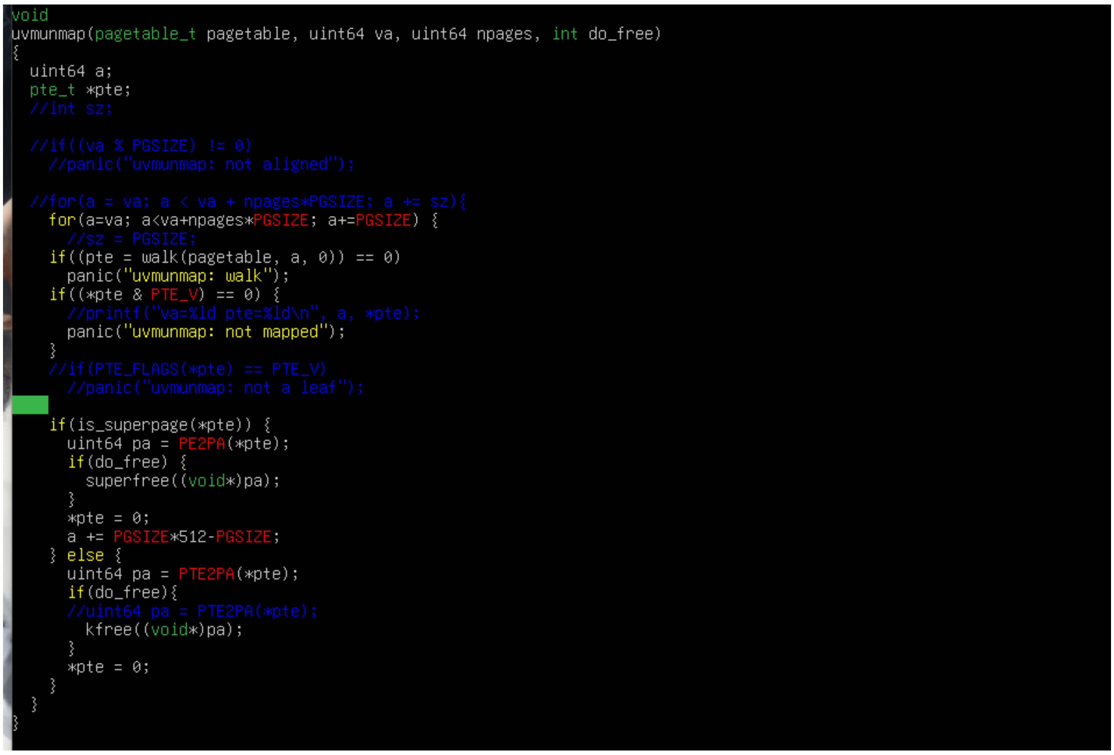
修改 uvmunmap 函数：

```
void
uvmunmap(pagetable_t pagetable, uint64 va, uint64 npages, int do_free)
{
  uint64 a;
  pte_t *pte;

  for(a = va; a < va + npages * PGSIZE; a += PGSIZE){
    if((pte = walk(pagetable, a, 0)) == 0)
      panic("uvmunmap: walk");
    if((*pte & PTE_V) == 0)
      panic("uvmunmap: not mapped");

    // 检查是否是超级页
```

```
    if(is_superpage(*pte)) {
      uint64 pa = PTE2PA(*pte);
      if(do_free){
        superfree((void*)pa);
      }
      *pte = 0;
      a += PGSIZE * 512 - PGSIZE; // 跳过 2MB
    } else {
      uint64 pa = PTE2PA(*pte);
      if(do_free){
        kfree((void*)pa);
      }
      *pte = 0;
    }
  }
}
```



```
void
uvmunmap(pagetable_t pagetable, uint64 va, uint64 npages, int do_free)
{
  uint64 a;
  pte_t *pte;
  //int sz;

  //if((va % PGSIZE) != 0)
    //panic("uvmunmap: not aligned");

  //for(a = va; a < va + npages*PGSIZE; a += sz){
    for(a=va; a<va+npages*PGSIZE; a+=PGSIZE) {
      //sz = PGSIZE;
    if((pte = walk(pagetable, a, 0)) == 0)
      panic("uvmunmap: walk");
    if((*pte & PTE_V) == 0) {
      //printf("va=%ld pte=%ld\n", a, *pte);
      panic("uvmunmap: not mapped");
    }
    //if(PTE_FLAGS(*pte) == PTE_V)
      //panic("uvmunmap: not a leaf");

    if(is_superpage(*pte)) {
      uint64 pa = PE2PA(*pte);
      if(do_free) {
        superfree((void*)pa);
      }
      *pte = 0;
      a += PGSIZE*512-PGSIZE;
    } else {
      uint64 pa = PTE2PA(*pte);
      if(do_free){
      //uint64 pa = PTE2PA(*pte);
        kfree((void*)pa);
      }
      *pte = 0;
    }
  }
}
```

修改 uvmcopy 函数:

```
int
uvmcopy(pagetable_t old, pagetable_t new, uint64 sz)
{
  pte_t *pte;
  uint64 pa, i;
```

```
  uint flags;

  for(i = 0; i < sz; i += PGSIZE){
    if((pte = walk(old, i, 0)) == 0)
      panic("uvmcopy: pte should exist");
    if((*pte & PTE_V) == 0)
      panic("uvmcopy: page not present");

    pa = PTE2PA(*pte);
    flags = PTE_FLAGS(*pte);

    // 检查是否是超级页
    if(is_superpage(*pte)) {
      // 分配新的超级页
      char *mem = superalloc();
      if(mem == 0)
        goto err;
      memmove(mem, (char*)pa, PGSIZE * 512);
      if(supermappages(new, i, PGSIZE * 512, (uint64)mem, flags) != 0){
        superfree(mem);
        goto err;
      }
      i += PGSIZE * 512 - PGSIZE; // 跳过 2MB
    } else {
      // 原有的 4KB 页面复制
      char *mem = kalloc();
      if(mem == 0)
        goto err;
      memmove(mem, (char*)pa, PGSIZE);
      if(mappages(new, i, PGSIZE, (uint64)mem, flags) != 0){
        kfree(mem);
        goto err;
      }
    }
  }
  return 0;

 err:
  uvmunmap(new, 0, i / PGSIZE, 1);
  return -1;
}
```

```
  GNU nano 7.2                                                    kernel/vm.c *
int
uvmcopy(pagetable_t old, pagetable_t new, uint64 sz)
{
  pte_t *pte;
  uint64 pa, i;
  uint flags;
  //char *mem;
  //int szinc;

  //for(i = 0; i < sz; i += szinc){
  for(i=0; i<sz; i += PGSIZE) {
    //szinc = PGSIZE;
    //szinc = PGSIZE;
    if((pte = walk(old, i, 0)) == 0)
      panic("uvmcopy: pte should exist");
    if((*pte & PTE_V) == 0)
      panic("uvmcopy: page not present");
    pa = PTE2PA(*pte);
    flags = PTE_FLAGS(*pte);

    if(is_superpage(*pte)) {
      char *mem = superalloc();
      if(mem == 0) goto err;
      memmove(mem, (char*)pa, PGSIZE*512);

    //if((mem = kalloc()) == 0)
      //goto err;
    //memmove(mem, (char*)pa, PGSIZE);
      if(mappages(new, i, PGSIZE*512, (uint64)mem, flags) != 0){
      //kfree(mem);
        superfree(mem);
        goto err;
      }
      i += PGSIZE*512-PGSIZE;
    } else {
      char *mem = kalloc();
      if(mem == 0)
        goto err;
      memmove(mem, (char*)pa, PGSIZE);
      if(mappages(new, i, PGSIZE, (uint64)mem, flags) != 0) {
        kfree(mem);
        goto err;
      }
    }
  }
  return 0;
```

```
err:
 uvmunmap(new, 0, i / PGSIZE, 1);
 return -1;
}
```

3． 在 kernel/defs.h 中添加函数声明:

```
// kalloc.c
void*       superalloc(void);
void        superfree(void*);

// vm.c
int         can_use_superpage(uint64, uint64);
int         is_superpage(pte_t);
int         supermappages(pagetable_t, uint64, uint64, uint64, int);
```

```
// kalloc.c
void*           kalloc(void);
void            kfree(void *);
void            kinit(void);
void*           superalloc(void);
void            superfree(void*);
```

```
// vm.c
void            kvminit(void);
void            kvminithart(void);
void            kvmmap(pagetable_t, uint64, uint64, uint64, int);
int             mappages(pagetable_t, uint64, uint64, uint64, int);
pagetable_t     uvmcreate(void);
void            uvmfirst(pagetable_t, uchar *, uint);
uint64          uvmalloc(pagetable_t, uint64, uint64, int);
uint64          uvmdealloc(pagetable_t, uint64, uint64);
int             uvmcopy(pagetable_t, pagetable_t, uint64);
void            uvmfree(pagetable_t, uint64);
void            uvmunmap(pagetable_t, uint64, uint64, int);
void            uvmclear(pagetable_t, uint64);
pte_t *         walk(pagetable_t, uint64, int);
uint64          walkaddr(pagetable_t, uint64);
int             copyout(pagetable_t, uint64, char *, uint64);
int             copyin(pagetable_t, char *, uint64, uint64);
int             copyinstr(pagetable_t, char *, uint64, uint64);
#if defined(LAB_PGTBL) || defined(SOL_MMAP)
void            vmprint_rec(pagetable_t, int);
void            vmprint(pagetable_t);
int             can_use_superpage(uint64, uint64);
int             is_superpage(pte_t);
int             supermappages(pagetable_t, uint64, uint64, uint64, int);_
#endif
#ifdef LAB_PGTBL
pte_t*          pgpte(pagetable_t, uint64);
```

4． 编译运行：

失败。失败。失败。。。系统一直在崩溃，一直提示内核在启动时发生了内核陷阱，改了好几版代码，已经改得头昏脑涨、无法理解，以上是最后一版，实在没办法了。。。先这样吧。

五． 实验主观心得

在开始实验时，由于对 xv6 的构建系统不熟悉，遇到了多次编译错误。特别是在切换 pgtbl 分支时，由于有未提交的修改，git 阻止了分支切换。通过 AI 的帮助，我学会了使用 git stash 和 git commit 来管理代码版本。然后就是 USYSCALL 页面映射的实现，之前只是在理论上知道 TRAMPOLINE 和 TRAPFRAME 的作用，现在亲手实现了类似的 USYSCALL 区域，对进程地址空间的管理有了更直观的认识。超级页实现是本次实验中最困难的部分，让我更加体会到了单纯写代码而不理解原理容易陷入盲目调试的感觉（但是好像理解了也做不出来。。。理解不到位吧）。

最后热烈感谢一下本次实验中 AI 提供的帮助！通过 AI，我理解到了很多陌生的概念；当出现错误时，AI 也能帮助我分析错误信息并提供解决方案（虽然最终没有解决如何实现超级页）。