گزارش فاز سوم پروژه سیستم‌های عامل

اشکان شکیبا (۹۹۳۱۰۳۰)، علی هاشم‌پور (۹۹۳۱۰۸۲)

مرحله اول: تعریف ساختار kref و توابع مربوط به آن در فایل kalloc.c

```
#define REF_CNT_IDX(pa) (((pa) - KERNBASE) / PGSIZE)

static struct {
    struct spinlock lock;
    int cnt[REF_CNT_IDX(PHYSTOP)];
} kref;

void kref_lock() {
  acquire(&kref.lock);
}

void kref_unlock() {
  release(&kref.lock);
}

static void set_refcnt(uint64 pa, int cnt) {
  kref.cnt[REF_CNT_IDX(pa)] = cnt;
}

uint64 inc_refcnt(uint64 pa) {
  return ++kref.cnt[REF_CNT_IDX(pa)];
}

uint64 dec_refcnt(uint64 pa) {
  return --kref.cnt[REF_CNT_IDX(pa)];
}

static void init_ref_cnt() {
  for (int i = 0; i < REF_CNT_IDX(PHYSTOP); ++i)
    kref.cnt[i] = 1;
}
```

مرحله دوم: فراخوانی lock مربوط به kref در تابع kinit در فایل kalloc.c

```
void
kinit()
{
```

```
  initlock(&kmem.lock, "kmem");
  initlock(&kref.lock, "kref");
  init_ref_cnt();
  freerange(end, (void*)PHYSTOP);
}
```

مرحله سوم: تکمیل تابع kfree در فایل kalloc.c، با افزودن شرطی که تنها در صورت صفر شدن تعداد ارجاع‌های یک صفحه آن را آزاد کند.

```
void
kfree(void *pa)
{
  struct run *r;

  if(((uint64)pa % PGSIZE) != 0 || (char*)pa < end || (uint64)pa >=
PHYSTOP)
    panic("kfree");

  acquire(&kref.lock);
  if (dec_refcnt((uint64)pa) > 0) {
    release(&kref.lock);
    return;
  }
  release(&kref.lock);

  // Fill with junk to catch dangling refs.
  memset(pa, 1, PGSIZE);

  r = (struct run*)pa;

  acquire(&kmem.lock);
  r->next = kmem.freelist;
  kmem.freelist = r;
  release(&kmem.lock);
}
```

مرحله چهارم: تکمیل تابع kalloc در فایل kalloc.c

```
void *
kalloc(void)
{
  struct run *r;

  acquire(&kmem.lock);
  r = kmem.freelist;
  if(r)
    kmem.freelist = r->next;
  release(&kmem.lock);
```

```
  if(r) {
    set_refcnt((uint64) r, 1);
    memset((char *) r, 5, PGSIZE); // fill with junk
  }
  return (void*)r;
}
```

مرحله پنجم: افزودن بیت COW در فایل riscv.h

```
#define PTE_V (1L << 0) // valid
#define PTE_R (1L << 1)
#define PTE_W (1L << 2)
#define PTE_X (1L << 3)
#define PTE_U (1L << 4) // user can access
#define PTE_COW (1L << 8)
```

مرحله ششم: تکمیل تابع uvmcopy در فایل vm.c، به طوری که اگر صفحه‌ای قابل نوشتن باشد آن را غیرقابل نوشتن کرده و سپس PTE_COW مربوط به آن را فعال و یکی به تعداد ارجاع‌هایش اضافه می‌کند.

```
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");
    if (*pte & PTE_W) {
      *pte &= ~PTE_W;
      *pte |= PTE_COW;
    }
    pa = PTE2PA(*pte);
    flags = PTE_FLAGS(*pte);
    if(mappages(new, i, PGSIZE, pa, flags) != 0){
      goto err;
    }
    kref_lock();
    inc_refcnt(pa);
    kref_unlock();
  }
  return 0;
```

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

مرحله هفتم: تکمیل تابع copyout در فایل vm.c

```
int
copyout(pagetable_t pagetable, uint64 dstva, char *src, uint64 len)
{
  uint64 n, va0, pa0;

  while(len > 0){
    va0 = PGROUNDDOWN(dstva);
    pa0 = walkaddr(pagetable, va0);
    if(pa0 == 0)
      return -1;
    n = PGSIZE - (dstva - va0);
    if(n > len)
      n = len;

    pte_t* pte = walk(pagetable, va0, 0);
    if (pte == 0)
      return -1;
    if (*pte & PTE_COW) {
      if (cowtrap(va0) < 0)
        return -1;
      pa0 = walkaddr(pagetable, va0);
    }

    memmove((void *)(pa0 + (dstva - va0)), src, n);

    len -= n;
    src += n;
    dstva = va0 + PGSIZE;
  }
  return 0;
}
```

مرحله هشتم: تعریف تابع cowtrap در فایل trap.c

```
int cowtrap(uint64 va) {
  if (va >= MAXVA)
    return -1;
  struct proc* p = myproc();
  pte_t* pte = walk(p->pagetable, va, 0);
  if (pte == 0)
    return -1;
```

```
  uint pte_flags = PTE_FLAGS(*pte);
  if ((pte_flags & PTE_COW) == 0)
    return -1;
  void* new = kalloc();
  if (new == 0)
    return -1;
  pte_flags |= PTE_W;
  pte_flags &= ~PTE_COW;
  void* old = (void*)PTE2PA(*pte);
  memmove(new, old, PGSIZE);
  kfree(old);
  *pte = PA2PTE(new) | pte_flags;
  return 0;
}
```

مرحله نهم: تکمیل تابع usertrap در فایل trap.c

```
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(killed(p))
      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 sepc, scause, and sstatus,
    // so enable only now that we're done with those registers.
    intr_on();

    syscall();
  } else if(r_scause() == 15) {  // Store/AMO Page Fault
    if (killed(p))
```

```
      exit(-1);
    if (cowtrap(r_stval()) < 0)
      setkilled(p);
  } else if((which_dev = devintr()) != 0){
    // ok
  } else {
    printf("usertrap(): unexpected scause %p pid=%d\n", r_scause(), p-
>pid);
    printf("            sepc=%p stval=%p\n", r_sepc(), r_stval());
    setkilled(p);
  }

  if(killed(p))
    exit(-1);

  // give up the CPU if this is a timer interrupt.
  if(which_dev == 2 && (myproc()->pid == 1 || myproc()->pid == 2))
    yield();

  usertrapret();
}
```

مرحله دهم: تکمیل تابع kerneltrap در فایل trap.c

```
void
kerneltrap()
{
  int which_dev = 0;
  uint64 sepc = r_sepc();
  uint64 sstatus = r_sstatus();
  uint64 scause = r_scause();

  if((sstatus & SSTATUS_SPP) == 0)
    panic("kerneltrap: not from supervisor mode");
  if(intr_get() != 0)
    panic("kerneltrap: interrupts enabled");

  if((which_dev = devintr()) == 0){
    printf("scause %p\n", scause);
    printf("sepc=%p stval=%p\n", r_sepc(), r_stval());
    panic("kerneltrap");
  }

  // give up the CPU if this is a timer interrupt.
  if(which_dev == 2 && myproc() != 0 && myproc()->state == RUNNING &&
(myproc()->pid == 1 || myproc()->pid == 2))
    yield();

  // the yield() may have caused some traps to occur,
  // so restore trap registers for use by kernelvec.S's sepc instruction.
  w_sepc(sepc);
```

```
  w_sstatus(sstatus);
}
```

مرحله یازدهم: تکمیل فایل defs.h

```
// kalloc.c
void*           kalloc(void);
void            kfree(void *);
void            kinit(void);
long            freeram(void);
long            totalram(void);
void            kref_lock();
void            kref_unlock();
uint64          inc_refcnt(uint64 pa);
uint64          dec_refcnt(uint64 pa);
```

```
// trap.c
extern uint     ticks;
void            trapinit(void);
void            trapinithart(void);
extern struct spinlock tickslock;
void            usertrapret(void);
int             cowtrap(uint64 va);
```

مرحله دوازدهم: ساخت فایل تست با نام cowTest.c در دایرکتوری user و پیاده‌سازی مراحل تست در آن

```
#include "../kernel/types.h"
#include "../kernel/memlayout.h"
#include "user.h"

void
test(int test_num) {
  uint64 phys_size = PHYSTOP - KERNBASE;
  int sz = (phys_size / 3) * 2;
  char *p = sbrk(sz);
  if (p == (char *) 0xffffffffffffffffL) {
    printf("Error: sbrk(%d) failed\n", sz);
    exit(-1);
  }
  for (char *q = p; q < p + sz; q += 4096) {
    *(int *) q = getpid();
  }
  int pid = fork();
  if (pid < 0) {
    printf("Error: fork() failed\n");
    exit(-1);
```

```
  }
  if (pid == 0) {
    exit(0);
  }
  wait(0);
  if (sbrk(-sz) == (char *) 0xffffffffffffffffL) {
    printf("Error: sbrk(-%d) failed\n", sz);
    exit(-1);
  }
  printf("Test #%d Result: OK\n", test_num);
}

int
main(int argc, char *argv[]) {
  test(1);
  test(2);
  printf("All Tests OK\n");
  return 0;
}
```

مرحله سیزدهم: افزودن فایل تست به Makefile

```
UPROGS=\
    $U/_cat\
    $U/_echo\
    $U/_forktest\
    $U/_grep\
    $U/_init\
    $U/_kill\
    $U/_ln\
    $U/_ls\
    $U/_mkdir\
    $U/_rm\
    $U/_sh\
    $U/_stressfs\
    $U/_usertests\
    $U/_grind\
    $U/_wc\
    $U/_zombie\
    $U/_getProcTickTest\
    $U/_sysinfoTest\
    $U/_schedulerTest\
    $U/_cowTest\
```