**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*\*