به نام خدا

دانشکده مهندسی کامپیوتر

درس سیستم‌های عامل
نیم‌سال اول ۰۲-۰۳
استاد: دکتر حسین اسدی

پاسخ تمرین عملی سری هفتم

امیرحسین رحمتی

ابتدا به تغییرات فایل kalloc.c می‌پردازیم.

```
uint8 references_counts[128*1024*1024 >> PGSHIFT] = {0};
```

در این فایل ما یک آرایه برای نگهداری تعداد رفرنس های به یک آدرس فیزیکی ایجاد کردیم.

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

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

  if(references_counts[((uint64)pa - (uint64)end) >> PGSHIFT] > 0)
  {
  references_counts[((uint64)pa - (uint64)end) >> PGSHIFT] -= 1;
  if(references_counts[((uint64)pa - (uint64)end) >> PGSHIFT] > 0)
    {
    return;
    }
  }

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

```
// increase the reference count for the physical page
void
kcopy(void *pa){
  references_counts[((uint64)pa - (uint64)end) >> PGSHIFT] += 1;
}

// Allocate one 4096-byte page of physical memory
```

```
// Returns 0 if the memory cannot be allocated.
void *
kalloc(void)
{
  struct run *r;

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

  if(r){
    memset((char*)r, 5, PGSIZE); //  typedef unsigned long uint64
    references_counts[((uint64)r - (uint64)end) >> PGSHIFT] = 1;
  }
  return (void*)r;
}
```

سپس با استفاده از سه تابع kfree, kalloc ,kcopy تعداد رفرنس های به این آدرس ها را تغییر دادیم زمانی که تابع kcopy صدا زده شود. یک رفرنس به تعداد رفرنس های این آدرس اضافه می‌شود. زمانی که تابع kalloc صدا زده شود برای اولین بار به این آدرس رفرنس داده می‌شود و تعداد رفرنس های آن یک عدد زیاد می‌شود. و زمانی که تابع kfree صدا زده شود یکی از رفرنس های آن آدرس کم می‌شود و در صورتی که تعداد رفرنس های آن به صفر برسد آن آدرس از حافظه free می‌شود.

حال به سراغ تغییرات فایل vm.c می‌رویم.

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

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

در این فایل ابتدا تابع uvmcopy را که مربوط به copy کردن یک pagetable در pagetable دیگر است تغییر دادیم. پس از ایجاد تغییرات با فراخوانی تابع uvmcopy آدرس های فیزیکی موجود در pagetable اول به pagetable دوم نیز تخصیص پیدا می‌کنند و درواقع دو یا چند page از فضای مجازی به یک page از فضای فیزیکی اشاره می‌کند. که افزایش تعداد refrence های آن آدرس فیزیکی با فراخوانی تابع kcopy انجام شده است. همچنین باید در نظر داشته باشیم آدرسی فیزیکی که مربوط به ۲ یا چند آدرس مجازی است حق تغییر داده را ندارد. به همین علت دسترسی write بر روی این آدرس فیزیکی با خاموش کردن بیت مربوطه از آن گرفته می‌شود. همچنین یک بیت اختصاصی برای مشخص کردن COW روشن می‌شود. که نشان می‌دهد داده این آدرس قابلیت تغییر دارد اما اگر بخواهیم آن را تغییر دهیم باید از آن copy بگیریم.

```c
// the function will copy the data to new pa if the page is COW
int copyonwrite(pagetable_t pagetable, uint64 va){
    if (va >= MAXVA){
      // not a valid address
      return -1;
    }
    pte_t *pte = walk(pagetable, va, 0);
    if (pte == 0){
      // not a valid address
      return -1;
    }
    if ((*pte & PTE_U) == 0 || (*pte & PTE_V) == 0 ){
      // not a valid address
      return -1;
    }
    if ((*pte & PTE_COW) == 0){
      // not a COW page
      return -1;
    }
    uint64 pa = PTE2PA(*pte); // get the physical address
    uint64 new_pa = (uint64)kalloc(); // allocate a new page
    if (new_pa == 0){
      // out of memory
      return -1;
    }
    memmove((char*)new_pa, (char*)pa, PGSIZE); // copy the data
    kfree((void*)pa); // free the old page
    uint flags = PTE_FLAGS(*pte);
    *pte = (PA2PTE(new_pa) | flags | PTE_W) & ~(PTE_COW); // update flags
    return 0;
}
```

یک تابع جدید با نام copyonwrite بر اساس نیازمندی که در قسمت قبل مطرح کردیم طراحی شده است. این تابع وظیفه دارد تا برای page هایی از حافظه که نیاز داریم بر روی آنها بنویسیم و دسترسی write نداریم و علت آن این است که در حالت copyonwrite هستیم. یک آدرس فیزیکی جدید در نظر بگیرد و برای آن آدرس دسترسی write را به کاربر بدهد. همچنین در این تابع تمام بررسی خطاهای ممکن مانند out of memory شدن یا حتی به درستی فراخوانی نشدن تابع copyonwrite بررسی شده است.

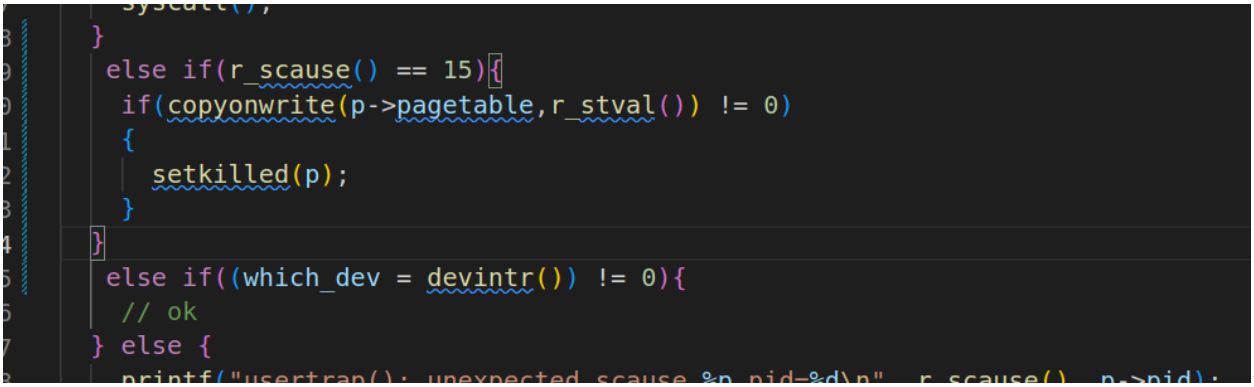
```
int
copyout(pagetable_t pagetable, uint64 dstva, char *src, uint64 len)
{
  uint64 n, va0, pa0;

  while(len > 0){
    va0 = PGROUNDDOWN(dstva);
    pa0 = walkaddr(pagetable, va0);
    if(pa0 == 0)
      return -1;
    if(copyonwrite(pagetable,va0) == 0){
      // copy on write
      pa0 = walkaddr(pagetable, va0);
      }
    n = PGSIZE - (dstva - va0);
    if(n > len)
      n = len;
    memmove((void *)(pa0 + (dstva - va0)), src, n);

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

تابع بعدی که آن را تغییر دادیم تابع copyout است. همانطور که در این تابع مشخص است در این تابع بر روی آدرس هایی از حافظه عملیات write صورت می‌گیرد. حال که ما مکانیزم مربوط به خواندن و نوشتن بر روی حافظه را دستخوش تغییر کردیم بایستی در این تابع نیز قبل از انجام عملیات نوشتن بررسی کنیم که آیا حافظه فیزیکی در مربوط به این آدرس مجازی در حالت cow قرار دارد یا نه. برای این کار تابع copyonwrite را صدا می‌زنیم اگر این تابع عدد صفر را برگرداند یعنی page در حالت cow قرار داشته و آدرس فیزیکی جدید به آن تخصیص پیدا کرده است. و نیاز است تا مجدد آدرس فیزیکی مربوط به این page را پیدا کنیم. اما اگر چنین نباشد تابع به روند قبلی خود ادامه می‌دهد.

و در نهایت تغییرات مربوط به فایل trap.c



تغییر نشان داده شده در تصویر بالا در تابع usertrap صورت گرفته است. همانطور که قبل تر اشاره کردیم ما دسترسی write را برای آدرس های فیزیکی با بیش از یک reference گرفتیم. به همین علت زمانی که بر روی این آدرس ها بخواهد چیزی نوشته شود دچار interrupt مربوط به ذخیره سازی خواهیم شد. در صورت دریافت این interrupt تابع copyonwrtie که قبل در درباره آن حرف زدیم را فراخوانی می کنیم. این تابع در صورتی که علت عدم دسترسی cow باشد، یک آدرس فیزیکی جدید به page تخصیص می‌دهد و در غیر این صورت عدد -1 را بر می‌گرداند. در صورت بازگشت عدد -1 ما متوجه می‌شویم که پردازه واقعا دسترسی مورد نیاز برای انجام این عملیات را ندارد یا مشکلاتی از جمله out of memory شدن پیش آمده و آن را kill می‌کنیم.

همچنین نتیجه فراخوانی تست هارا در تصویر مقابل مشاهده می‌کنید.