به نام خدا

گزارش آزمایشگاه آزمایش چهارم درس سیستم عامل

آیدین کاظمی: ۸۱۰۱۰۱۵۶۱ علی زیلوچی: ۸۱۰۱۰۱۵۶۰ بابک حسینی محتشم: ۸۱۰۱۰۱۴۰۸

مقدمه:

۱) راجع به مفهوم ناحیه مجازی[4] در لینوکس به طور مختصر توضیح داده و آن را با xv6 مقایسه کنید.

پاسخ سوال ۱:

در لینوکس Virtual Memory Areas یا همان VMA نشان‌دهنده بازه پیوسته‌ای از حافظه در فضای آدرس‌دهی پردازه‌ها هستند. هر VMA مجوزهای یکسانی از لحاظ قابلیت خواندن، نوشتن و اجرا دارد. در هر ساختار داده VMA آدرس ابتدا و انتها نیز مشخص است. هر پردازه برای بخش‌های مختلفش VMA مختلف دارد مثلا یک VMA برای بخش کد، یکی برای بخش داده و .... همچنین هر VMA از تعداد page تشکیل شده که هر کدام در page table آن پردازه قرار گرفته‌اند.

در سیستم‌عامل xv6 مدیریت حافظه ساده‌تر است و هر پردازه، page table خود را دارد. بدین ترتیب، برخی از قابلیت‌های پیشرفته برای مدیریت حافظه از جمله حافظه مشترک بین چند پردازه را ندارد.

۲) چرا ساختار سلسله‌مراتبی منجر به کاهش مصرف حافظه می‌گردد؟

پاسخ سوال ۲:

ساختار سلسله مراتبی ذکر شده بدین صورت عمل میکند که هر آدرس مجازی برای تبدیل شدن به آدرس فیزیکی، ابتدا ۱۰ بیت بالای آن نشان دهنده شماره سطر در page directory است. سپس از page directory، آدرس page table مربوطه پیدا میشود. سپس ۱۰ بیت بعدی برای پیدا کردن آدرس صفحه در page table استفاده میشوند و نهایتا، از ۱۲ بیت کم ارزش برای آدرس دهی در آن صفحه استفاده میشود. اگر page directory وجود نداشت، هر پردازه باید یک page table بزرگ که تمام حافظه فیزیکی را شامل میشود، را شامل میشد که این page table حجیم، حافظه بیشتری از حافظه اصلی نیاز داشت ولی با ساختار سلسه مراتبی، تنها page tableهایی که توسط page directory و آدرس مجازی به آن ها نیاز داشتیم، در حافظه اصلی میایند و بدین ترتیب مصرف حافظه کاهش میابد.

۳) محتوای هر بیت یک مدخل (۳۲ بیتی) در هر سطح چیست؟ چه تفاوتی میان آن‌ها وجود دارد؟

پاسخ سوال ۳:

اکثر بیت های این دو یکسان است با این تفاوت که در page directory بیت ها نشان دهنده وضعیت page table هستند درحالیکه در page table بیت ها نشان دهنده وضعیت خود صفحات هستند.

Bits	Name	Description for Page Directory	Description for Page Table
0	Present	The page table exists in memory.	The page exists in memory.
1	Writable	Writable or Read-only page table	Writable or Read-only page
2	User	Page table is accessible from user mode	Page is accessible from user mode
3	Write-through	Caching policy for page table	Caching policy for page
4	Cache Disabled	Disables Caching for this page table	Disables Caching for this page
5	Accessed	Set by hardware when page table is accessed	Set by hardware when page is accessed
6	Dirty	Unused	Set by hardware when page is modified
9-11	Available	Reserved for OS	Reserved for OS
12-31	Base Address	Physical address of the page table	Physical address of the page

کد مربوط به ایجاد فضاهای آدرس در xv6:

۴) تابع kalloc چه نوع حافظه‌ای تخصیص می‌دهد؟ (فیزیکی یا مجازی)

پاسخ سوال ۴:

در این تابع، حافظه فیزیکی به پردازه تخصیص داده میشود. برای تخصیص حافظه مجازی باید از تابع allocuvm استفاده کرد که خود این تابع نیز از kalloc برای افزایش حافظه فیزیکی تخصیص داده شده استفاده میکند.

۵) تابع mappages چه کاربردی دارد؟

پاسخ سوال ۵:

این تابع page directory یک پردازه، آدرس شروع مجازی و فیزیکی و سایز و مجوزها را دریافت میکند و از آدرس شروع فیزیکی، به اندازه سایز، تمام آدرس های مجازی در page table پردازه را به آدرس فیزیکی مرتبط میکند. همچنین مجوزهای خواندن و نوشتن برای این بازه از آدرس را نیز ثبت میکند. از این تابع هنگام ایجاد پردازه جدید برای مرتبط کردن آدرس مجازی آن به آدرس فیزیکی و همچنین هنگام افزایش حافظه یک پردازه استفاده میشود.

۷) راجع به تابع walkpgdir توضیح دهید. این تابع چه عمل سخت‌افزاری را شبیه‌سازی می‌کند؟

پاسخ سوال ۷:

این تابع، page directory و آدرس مجازی و متغیر alloc را دریافت میکند. ابتدا در page directory سعی میکند page table مرتبط با آدرس مجازی را پیدا کند. اگر page table وجود نداشت، در صورت صفر بودن alloc، صفر را به نشانه ارور برمی‌گرداند وگرنه page table جدیدی ایجاد میکند. در نهایت، سطر مربوط به آدرس مجاز از جدول صفحات را برمی‌گرداند.

این تابع عمل سخت افزاری page table walk را شبیه‌سازی میکند. در عمل، سخت افزار برای تبدیل آدرس مجازی به فیزیکی، مراحلی را طی میکند که به آن page table walk میگوییم. اگر سخت افزار دارای page directory و page table باشد مشابه عملیات صورت گرفته در xv6 اتفاق میفتد با این تفاوت که در صورت پیدا نشدن صفحه، page fault به سیستم‌عامل میدهد تا آن را برطرف کند.

۸) توابع allocuvm و mappages که در ارتباط با حافظه‌ی مجازی هستند را توضیح دهید.

پاسخ سوال ۸:

تابع mappages برای مرتبط کردن محدود ای از آدرس مجازی با آدرس فیزیکی استفاده میشود. تابع allocuvm، مقدار حافظه ی پردازه را به میزان دلخواه که به عنوان ورودی دریافت میکند افزایش میدهد و برای این کار، از kalloc برای تخصیص حافظه فیزیکی و mappages برای مرتبط کردن حافظه مجازی با حافظه فیزیکی ایجاد شده استفاده میکند.

۹) شیوه‌ی بارگذاری [19] برنامه در حافظه توسط فراخوانی سیستمی exec را شرح دهید.

پاسخ سوال ۹:

در این فراخوانی سیستمی ابتدا معتبر بودن کد بررسی و کد خوانده میشود. سپس قسمت هسته جدول صفحات پردازه با تابع setupkvm ایجاد میشود. سپس حافظه ای به اندازه نیاز برنامه با تابع allocuvm به پردازه تخصیص داده میشود. سپس با صدا زدن تابع loaduvm، قسمت های مختلف کد در حافظه تخصیص داده شده به پردازه کپی میشوند. سپس دو صفحه دیگر نیز به پردازه تخصیص داده میشود. از صفحه دوم به عنوان استک برای پردازه استفاده میشود و پارامترهای داده شده به exec در این صفحه ذخیره میشود. صفحه اول guard page است و دسترسی به آن غیر مجاز است. با گذاشتن این صفحه، اگر پردازه ای داشت از صفحه استک خارج میشد متوجه میشویم و جلوی آن را میگیریم. در نهایت page directory قبلی این پردازه را free میکنیم و به صفحه switch میکنیم.

شرح پروژه:

با توجه به اینکه توابع و داده‌ساختارهای مربوط به حافظه مجازی در فایل vm.c قرار گرفته‌اند، تصمیم گرفتیم جدول خواسته شده را به این فایل اضافه کنیم.

```
static struct
{
  struct spinlock lock;
  int id[_NSHAREDPAGES];
  char* pa[_NSHAREDPAGES];
  int ref_count[_NSHAREDPAGES];
  int va[NPROC][_NSHAREDPAGES];
} shm_table;
```

در داده‌ساختار shm_table قفلی قرار گرفته تا موقع ایجاد تغییرات در این داده‌ساختار توسط چند پردازنده به طور موازی، مشکلی پیش نیاید. متغیر _NSHAREDPAGES را در فایل param.h برابر ۱۰ در نظر گرفتیم که تعداد صفحات اشتراکی موجود را نشان میدهد. آرایه id، آیدی هر صفحه را نشان می‌دهد که می‌تواند هر عدد صحیحی باشد. آرایه‌ای از پوینترها به آدرس شروع فریم فیزیکی را در آرایه pa ذخیره کرده‌ایم. آرایه ref_count تعداد رفرنس‌های هر صفحه اشتراکی را نشان می‌دهد. در نهایت، آرایه va آدرس مجازی شروع هر صفحه اشتراکی را برای هر پردازه ذخیره می‌کند.

تابع بعدی _shared_mem_init است که حافظه مجازی اشتراکی را مقداردهی اولیه می‌کند و در تابع main در شروع سیستم‌عامل صدا زده می‌شود. در این تابع، آدرس فیزیکی صفحات ذخیره شده و تعداد رفرنس‌ها صفر می‌شود.

```
void _shared_mem_init(void)
{
  initlock(&shm_table.lock, "shared memory table");
  for (int i = 0; i < _NSHAREDPAGES; i++)
  {
    shm_table.pa[i] = kalloc();
    if (shm_table.pa[i] == 0)
      panic("_shared_mem_init: kalloc failed");
    shm_table.ref_count[i]=0;
  }
}
```

سیستم‌کال open_sharedmem را نیز در فایلم vm.c پیاده کردیم. در این سیستم‌کال، پس از گرفتن قفل جدول صفحات اشتراکی، ابتدا دنبال صفحه اشتراکی با آیدی داده شده می‌گردیم. اگر صفحه پیدا نشد ولی صفحه خالی موجود بود، آیدی آن را برابر آیدی داده شده قرار می‌دهیم. اگر هیچ صفحه اشتراکی خالی موجود نباشد - ۱ برمی‌گردانیم. سپس تعداد رفرنس‌های صفحه پیدا دشه را یکی افزایش می‌دهیم. حال از انتهای آدرس مجازی پردازه که همان سایز آن در pcbاش است، آدرس مجازی پردازه را به آدرس فیزیک صفحه تبدیل می‌کنیم. نهایتا متغیر sz پردازه را به اندازه سایز یک صفحه زیاده کرده و پس از ذخیره کردن آدرس شروع مجازی در متغیر va جدول، آین متغیر را به عنوان آدرس مجازی شروع صفحه برمی‌گردانیم.

```
int open_sharedmem(int id)
{
  int mem_idx = -1;
  struct proc *curproc = myproc();
  uint va = PGROUNDUP(curproc->sz);
  pde_t *pgdir = curproc->pgdir;
  acquire(&shm_table.lock);
  for (int i = 0; i < _NSHAREDPAGES; i++)
  {
    if(shm_table.ref_count[i]==0)
      mem_idx = i;
    else if (shm_table.id[i]==id)
    {
      mem_idx = i;
      break;
    }
  }
  if (mem_idx==-1)
  {
    release(&shm_table.lock);
    return -1;
  }
  shm_table.id[mem_idx]=id;
  shm_table.ref_count[mem_idx]++;
  release(&shm_table.lock);
  if (mappages(pgdir, (char *)va, PGSIZE, V2P(shm_table.pa[mem_idx]), PTE_W | PTE_U) < 0)
  {
    acquire(&shm_table.lock);
    shm_table.ref_count[mem_idx]--;
    release(&shm_table.lock);
    return -1;
  }
  switchuvm(curproc);
  shm_table.va[curproc->pid][mem_idx] = va;
  curproc->sz += PGSIZE;
  return (int)va;
}
```

پیش از نوشتن سیستم‌کال close_sharedmem تابع unmappages را می‌نویسیم که از آن استفاده کنیم. این تابع، عمل معکوس mappages را اجرا می‌کند تمام آدرس‌های مجازی تبدیل شده را غیرمعتبر می‌کند.

این تابع مشابه mappages ابتدا page table entry مربوط به پردازه را پیدا می‌کند و مقدار موجود در این خانه را صفر می‌کند تا غیر معتبر شود.

```
// Remove PTEs for virtual addresses starting at va that refer to
// physical addresses starting at pa. va and size might not
// be page-aligned.
static int
unmappages(pde_t *pgdir, void *va, uint size)
{
  char *a, *last;
  pte_t *pte;

  a = (char *)PGROUNDDOWN((uint)va);
  last = (char *)PGROUNDDOWN(((uint)va) + size - 1);
  for (;;)
  {
    if ((pte = walkpgdir(pgdir, a, 1)) == 0)
      return -1;
    if (!*pte || !PTE_P)
      panic("reunmap");
    *pte = 0;
    if (a == last)
      break;
    a += PGSIZE;
  }
  return 0;
}
```

سیستم‌کال close_sharedmem پس از گرفتن قفل جدول صفحات اشتراکی، صفحه اشتراکی با آیدی داده شده را پیدا می‌کند و تعداد رفرنس‌های آن را یکی کم می‌کند تابع unmappages را فراخوانی میکند. سپس متغیر sz پردازه فعلی را به اندازه سایز یک صفحه کاهش می‌دهد.

```
int close_sharedmem(int id)
{
  int mem_idx = -1;
  struct proc *curproc = myproc();
  pde_t *pgdir = curproc->pgdir;
  acquire(&shm_table.lock);
  for (int i = 0; i < _NSHAREDPAGES; i++)
  {
    if (shm_table.id[i] == id && shm_table.ref_count[i])
    {
      mem_idx = i;
      break;
    }
  }
  if (mem_idx == -1)
  {
    release(&shm_table.lock);
    return -1;
  }
  shm_table.ref_count[mem_idx]--;
  release(&shm_table.lock);
  uint va = shm_table.va[curproc->pid][mem_idx];
  if (unmappages(pgdir, (char *)va, PGSIZE) < 0)
  {
    acquire(&shm_table.lock);
    shm_table.ref_count[mem_idx]++;
    release(&shm_table.lock);
    return -1;
  }
  switchuvm(curproc);
  curproc->sz -= PGSIZE;
  return 0;
}
```

سیستم‌کال calculate_factorial را برای تست کردن درست کار کردن حافظه اشتراکی می‌نویسیم. این تابع پس از دریافت عدد n و آیدی یک صفحه اشتراکی، ابتدا سیستم‌کال open_sharedmem را با آیدی داده شده فراخوانی می‌کند تا آدرس ابتدای حافظه مجازی را دریافت کند. طبق قرارداد در خانه اول، شماره فاکتوریل فعلی و در خانه دوم مقدار آن نوشته شده است. در نتیجه پس از گرفتن قفل شماره فاکتوریل فعلی خوانده می‌شود و یکی افزایش میابد و سپس در مقدار فاکتوریل ضرب می‌شود و در حافظه نوشته می‌شود و بعد قفل رها می‌شود. رها کردن قفل مهم است تا بقیه پردازه‌ها نیز بتوانند حافظه اشتراکی را تغییر دهند. پس از اینکه شماره فاکتوریل نوشته شده برابر n شد، سیستم‌کال close_sharedmem صدا زده می‌شود و کار تمام می‌شود.

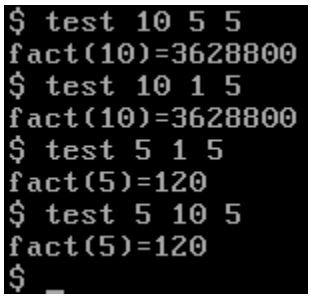
لازم به ذکر است قفل ایجاد شده برای این سیستم‌کال در تابع _factorial_init مقداردهی اولیه می‌شود و این تابع در تابع main صدا زده می‌شود.

```
static struct spinlock factorial_lock;
void _factorial_init()
{
  initlock(&factorial_lock, "factorial");
}
void calculate_factorial(int n, int id)
{
  int last = 0;
  int *mem = (int*)open_sharedmem(id);
  if ((int)mem == -1)
  {
    cprintf("ERROR: open_sharedmem failed for process %d\n",myproc()->pid);
    return;
  }
  while (last < n)
  {
    acquire(&factorial_lock);
    last = *mem;
    // cprintf("Process %d writing to shared memory: number = %d\n", myproc()->pid, last);
    if(last<n)
    {
      *(mem + 1) *= ++last;
      *mem = last;
    }
    release(&factorial_lock);
  }
  if (close_sharedmem(id))
  {
    cprintf("ERROR: close_sharedmem failed for process %d\n", myproc()->pid);
    return;
  }
  return;
}
```

در برنامه آزمون، ابتدا با استفاده از open_sharedmem یک صفحه اشتراکی را دریافت می‌کنیم و مقدار ۰ را برای شماره و ۱ را برای فاکتوریل در دو خانه ابتدای آن می‌نویسیم. سپس به تعداد داده شده پردازه ایجاد می‌کنیم و برای هر پردازه سیستم‌کال calculate_factorial را صدا می‌زنیم. در نهایت پس از اتمام کار پردازه‌های فرزند مقدار فاکتوریل را می‌خوانیم.

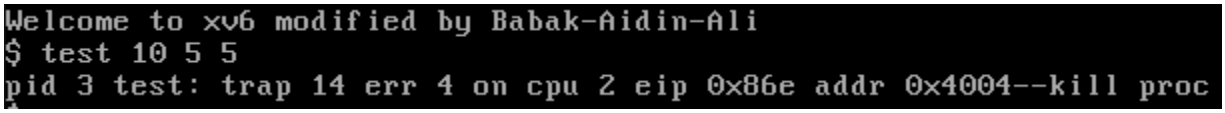
```
void ca5_test(int argc, char *argv[])
{
  if (argc < 3)
  {
    printf(2, "usage: test number n_children_processes...\n");
    exit();
  }
  int n=atoi(argv[1]),n_children=atoi(argv[2]),pid,mem_id=0;
  int *shared_mem=(int*)open_sharedmem(mem_id);
  if((int)shared_mem==-1)
  {
    printf(2, "ERROR: open_sharedmem\n");
    exit();
  }
  *shared_mem = 0;
  *(shared_mem+1) = 1;
  for (int i = 0; i < n_children; i++)
  {
    pid = fork();
    if (!pid){
      calculate_factorial(n, mem_id);
      exit();
    }
  }
  for (int i = 0; i < n_children; i++)
    wait();
  printf(1, "fact(%d)=%d\n", (*shared_mem), (*(shared_mem + 1)));
  if (close_sharedmem(mem_id) < 0)
  {
    printf(2, "ERROR: close_sharedmem\n");
    exit();
  }
  exit();
}
```

خروجی این سیستم‌کال:



می‌توان دید که خروجی که انتظار داشتیم به درستی چاپ می‌شود و برنامه به درستی کار می‌کند. همچنین اگر مقدار فاکتوریل عددی بزرگتر از ۱۲ را بخواهیم به دلیل overflow مقدار نادرستی نمایش داده می‌شود ولی باز هم مقدار ثابتی است که یعنی حافظه اشتراکی به درستی کار می‌کند.

برای آزمودن close_sharedmem و unmappages، اگر چاپ خروجی را پس از صدا زدن این سیستم‌کال انجام دهیم، به دلیل دسترسی به حافظه غیر مجاز ارور زیر را دریافت می‌کنیم:



اگر از قفل برای نوشتن و خواندن از حافظه اشتراکی استفاده نکنیم، انتظار داریم که با احتمالی، پاسخ متفاوتی پس از هر اجرا بگیریم ولی با برداشتن قفل، پاسخ همچنان یکسان است. دلیل این است که چون عملیات طولانی و زیادی داخل critical section رخ نمی‌دهد همچنان احتمال اشتباه شدن پاسخ است ولی این احتمال بسیار کم است. برای افزایش احتمال خطا، یک خطا چاپ اضافه می‌کنیم که به دلیل کندی، احتمال تغییر پردازه داخل critical section را افزایش دهیم و بدین ترتیب هر بار خروجی متفاوتی دریافت می‌کنیم.

```
void calculate_factorial(int n, int id)
{
  int last = 0;
  int *mem = (int*)open_sharedmem(id);
  if ((int)mem == -1)
  {
    cprintf("ERROR: open_sharedmem failed for process %d\n",myproc()->pid);
    return;
  }
  while (last < n)
  {
    // acquire(&factorial_lock);
    last = *mem;
    cprintf("Process %d writing to shared memory: number = %d\n", myproc()->pid, last);
    if(last<n)
    {
      *(mem + 1) *= ++last;
      *mem = last;
    }
    // release(&factorial_lock);
  }
  if (close_sharedmem(id))
  {
    cprintf("ERROR: close_sharedmem failed for process %d\n", myproc()->pid);
    return;
  }
  return;
}
```

خروجی‌های نادرست بدون قفل:

```
Process 12 writing to shared memory: number = 4
Process 11 writing to shared memory: number = 4
fact(5)=17280000
$ test 5 5 5
Process 16 writing to shared memory: number = 0
Process 17 writing to shared memory: number = 0
Process 16 writing to shared memory: number = 1
Process 18 writing to shared memory: number = 1
Process 17 writing to shared memory: number = 1
Process 16 writing to shared memory: number = 2
Process 19 writing to shared memory: number = 2
Process 16 writing to shared memory: number = 3
Process 17 writing to shared memory: number = 2
Process 18 writing to shared memory: number = 2
Process 17 writing to shared memory: number = 3
Process 16 writing to shared memory: number = 4
Process 18 writing to shared memory: number = 3
Process 17 writing to shared memory: number = 4
Process 20 writing to shared memory: number = 5
Process 18 writing to shared memory: number = 4
Process 19 writing to shared memory: number = 3
Process 19 writing to shared memory: number = 4
fact(5)=103680000
$ _
```

پس نیاز به قفل و همگام‌سازی برای استفاده از حافظه اشتراکی امری ضروری است.