به نام خدا

گزارشکار آزمایش پنج آزمایشگاه سیستم عامل

اعضای گروه:

علی زمانی 810101436

برنا فروهری 810101480

نیما خنجری 810101416

پرسش 1: راجع به مفهوم ناحیه مجازی در لینوکس به طور مختصر توضیح داده و آن را با xv6 مقایسه کنید.

در لینوکس, ناحیه مجازی Virtual Memory Area) یا (VMAبه بخشی از فضای آدرس مجازی اشاره دارد که شامل اطلاعاتی در مورد نحوه نگاشت فضای آدرس مجازی به فضای آدرس فیزیکی هستند. این ساختمان داده بازه ای از آدرس های مجازی را توصیف می کند که هر کرام از این آدرس ها شامل اطلاغاتی هستند که نشان می دهند اطلاعات مربوط به آن آدرس از حافظه چیست. پس به وسیله آن می توان چک کرد که آیا آن آدرس حاوی دیتایی است یا نه. در نتیجه ناحیه مجازی در لینوکس میتواند برای کنترل کردن حافظه مجازی پردازه ها به کار رود. به طور کلی هر ناحیه مجازی در لینوکس شامل تعدادی page table است که با استفاده از این page table ها می توان آدرس های مجازی را به آدرس فیزیکی متناظر تبدیل کرد. اما در سیستم عامل xv6, برای نگاشت فضای مجازی به فیزیکی مستقیم از جداول صفحه (Page Tables) استفاده می شود و پیچیدگی‌هایی مانند VMA یا مدیریت پویای حافظه را ندارد. به عنوان مثال، تمام فضای آدرس یک فرآیند هنگام اجرا در حافظه بارگذاری می‌شود.

پرسش 2: چرا ساختار سلسله مراتبی منجر به کاهش مصرف حافظه میگردد؟

در صورت استفاده از ساختار سلسله مراتبی , می توان داده هایی که در طول اجرای پردازه ها, به دفعات استفاده می شوند و بیشتر مورد نیاز هستند را در cache ذخیره سازی کرد تا دسترسی به آن ها سرعت بسیار بالاتری داشته باشد. همچنین, با استفاده از این ساختار, پردازه ها قابلیت مپ کردن دیتا های ذخیره شده در حافظه را خواهد داشت, همین امر باعث کی شود که از main memory تا حد قابل توجهی درخواست دسترسی کمتری داشته باشیم و در نتیجه مصرف حافظه کاهش می یابد.

پرسش 3: محتوای هر بیت یک مدخل 32 بیتی در هر سطح چیست؟ چه تفاوتی میان آنها وجود دارد؟

یک مدخل 32 بیتی معمولاً شامل دو بخش است:

1- آدرس های فیزیکی : معمولا 20 بیت بالایی از 32 بیت مدخل, برای این بخش استفاده می شود. این بخش شامل آدرس جدول صفحه بعدی (Page Table) در سلسله‌مراتب می باشد.

2- بیت‌های کنترلی: این بخش که معمولا 12 بیت پایینی به آن اختصاص می یابد, شامل بیت هایی است که رفتار و ویژگی‌های page را تعیین می کنند. در این بخش, آدرس قاب فیزیکی (Physical Frame) که به آدرس مجازی نگاشت می‌شود, مشخص خواهد شد .

پرسش 4: تابع kalloc چه نوع حافظه ای تخصیص میدهد؟ فیزیکی یا (مجازی)

تابع kalloc حافظه‌ای از نوع فیزیکی تخصیص می‌دهد. این تابع بخشی از مدیریت حافظه فیزیکی است و برای اختصاص دادن یک صفحه فیزیکی (Physical Page) استفاده می‌شود. با استفاده از kalloc, حافظه فیزیکی را تخصیص داده می شود و یک اشاره گر به آدرس فیزیکی آن حافظه را بازمی‌گرداند.

در نتیجه این تابع فقط حافظه فیزیکی را تخصیص می‌دهد و هیچ نگاشت (Mapping) به فضای آدرس مجازی انجام نمی‌دهد. نگاشت آدرس مجازی به آدرس فیزیکی توسط جداول صفحه انجام می‌شود.

پرسش 5: تابع mappages چه کاربردی دارد؟

به طور کلی, تابع mappages برای مدیریت حافظه مجازی کاربرد دارد. برای مثال, هنگامی که نیاز است یک فضای آدرس مجازی به یک صفحه فیزیکی جدید مپ شود, برای تخصیص حافظه برای فرآیند جدید و نگاشت حافظه کرنل به آدرس‌های فیزیکی, از این تابع استفاده می شود. هم چنین برای مپ کردن آدرس ها به هنگام تنظیم اولیه فضای آدرس کاربر و کرنل هم این تابع کاربرد دارد.

پرسش 7: راجع به تابع walkpgdir توضیح دهید. این تابع چه عمل سخت افزاری را شبیه سازی میکند؟

تابع walkpgdir در xv6 برای پیمایش جداول صفحه (Page Tables) استفاده می‌شود. این تابع به آدرس‌های مناسب در جداول صفحه دسترسی پیدا می‌کند و موقعیت نگاشت یک آدرس مجازی به آدرس فیزیکی را تعیین می‌کند.

```
static pte_t *
walkpgdir(pde_t *pgdir, const void *va, int alloc)
{
  pde_t *pde;
  pte_t *pgtab;

  pde = &pgdir[PDX(va)];
  if(*pde & PTE_P){
    pgtab = (pte_t*)P2V(PTE_ADDR(*pde));
  } else {
    if(!alloc || (pgtab = (pte_t*)kalloc()) == 0)
      return 0;
    // Make sure all those PTE_P bits are zero.
    memset(pgtab, 0, PGSIZE);
    // The permissions here are overly generous, but they can
    // be further restricted by the permissions in the page table
    // entries, if necessary.
    *pde = V2P(pgtab) | PTE_P | PTE_W | PTE_U;
  }
  return &pgtab[PTX(va)];
}
```

)تابع walkpgdir چک می کند که اگر page table entry ای که به آدرس مجازی با شروع از pr اشاره کند وجود داشت, آدرس آن را برمی گرداند. در غیر این صورت, جدول جدیدی ساخته شده و آدرس آن برگردانده میشود.)

پس عمل تقسیم آدرس مجازی و پیمایش جداول صفحه از عمل های سخت افزاری ای است که این تابع شبیه سازی می کند. چون در پردازنده‌های واقعی، این عملیات توسط MMU انجام می‌شود.

پرسش 8: توابع allocuvm و mappages که در ارتباط با حافظه ی مجازی هستند را توضیح دهید.

همانطور که پیشتر توضیح دادیم, تابع mappages برای مدیریت حافظه مجازی و مپ کردن یک فضای آدرس مجازی به یک صفحه فیزیکی استفاده می شود.

از طرفی تابع allocuvm که مخفف allocate user virtual memory است, وظیفه دارد که در یک دایرکتوری صفحه مشخص شده, حافظه مجازی را به یوزر اختصاص دهد و آن را افزایش دهد.

```
allocuvm(pde_t *pgdir, uint oldsz, uint newsz)
```

این تابع دو آرگومان oldsz و newsz را دریافت کرده و در صورتی که oldsz از newsz کمتر باشد (که به معنی افزایش فضای آدرس خواهد بود که مطلوب ماست), آدرس‌های بین oldsz تا newsz را به صفحات فیزیکی جدید نگاشت می کند. برای هر صفحه در این محدوده هم با استفاده از mappages, صفحه فیزیکی را به آدرس مجازی در دایرکتوری صفحه فرآیند نگاشت می کند.

پس mappages برای مپ کردن فضای آدرس استفاده می شود و allocuvm برای گسترش حافظه مجازی کاربر.

پرسش 9: شیوه ی بارگذاری برنامه در حافظه توسط فراخوانی سیستمی exec را شرح دهید.

در ابتدا, فایل اجرایی مشخص شده توسط path باز می‌شود و بررسی می‌شود تا مطمئن شویم یک فایل اجرایی معتبر است. بعد از آن, هدر ELF فایل را می‌خواند تا مشخص شود section ها و segment ها چگونه است. سپس فضای آدرس فرآیند Page Directory) و جداول صفحه (به طور کامل آزاد می‌شود. در مرحله بعدی , هر سگمنت مشخص‌شده در ELF به ترتیب در حافظه کپی می‌شود و آدرس فیزیکی حافظه برای سگمنت‌ها با استفاده از allocuvm تخصیص داده می‌شود. هم چنین داده‌های سگمنت از فایل اجرایی به حافظه منتقل می‌شوند. سپس پشته فرآیند جدید ایجاد می شود و آرگومان‌های برنامه (argv) روی آن قرار داده می‌شوند. بعد از آن رجیسترهای CPU برای شروع اجرای برنامه جدید تنظیم می‌شوند. در نهایت منابعی که مورد نیاز نیستند و اضافی اند آزاد شده و برنامه اجرا می شود.

توضیحات درباره struct و functionهای اضافه شده در فایل های sharedMemory.h و sharedMemory.c :

```c
#define NUM_OF_SHARED_PAGES 8


struct shared_page{
    int id;
    int num_of_refs;
    char* frame;
};
```

shared_page شامل سه بخش است:

- **id** : این متغیر یک شناسه یکتا (ID) برای هر صفحه مشترک تعریف می‌کند. با استفاده از این شناسه، پردازش‌ها می‌توانند صفحه‌های مختلف حافظه مشترک را شناسایی کنند.
- **num_of_refs:** این متغیر تعداد (References) به این صفحه را ذخیره می‌کند. این نشان‌دهنده تعداد پردازش‌هایی است که در حال حاضر به این صفحه دسترسی دارند. وقتی مقدار این متغیر به صفر برسد، سیستم می‌تواند این صفحه را آزاد کند.

- **frame:** این pointer به موقعیت فیزیکی یا مجازی حافظه‌ای اشاره می‌کند که داده‌های صفحه در آن قرار دارند. این حافظه می‌تواند داده‌های اشتراکی واقعی باشد که پردازش‌ها با هم استفاده می‌کنند.

```
11
12  struct{
13      struct shared_page table[NUM_OF_SHARED_PAGES];
14      struct spinlock lock;
15  } shared_memory;
16
```

shared_memory:

- این یک نمونه از ساختار تعریف‌شده است که شامل جدول صفحات و قفل مربوطه می‌باشد.
- از این متغیر برای مدیریت کل سیستم حافظه مشترک استفاده می‌شود.

```
16
17  void
18  init_sharedmem(){
19      acquire(&shared_memory.lock);
20      for (int i = 0; i < NUM_OF_SHARED_PAGES; i++){
21          shared_memory.table[i].num_of_refs = 0;
22      }
23      release(&shared_memory.lock);
24  }
25
```

تابع init_sharedmem برای **initialize اولیه (shared memory)** استفاده می‌شود. این تابع وظیفه دارد جدول حافظه مشترک را به حالت اولیه تنظیم کند، به طوری که آماده استفاده توسط پردازش‌ها باشد.

- مقداردهی صفر به num_of_refs برای هر صفحه، حافظه مشترک را به حالت خالی و آماده استفاده تنظیم می‌کند.
- با استفاده از قفل چرخشی (spinlock)، این تابع مطمئن می‌شود که هیچ پردازش دیگری در حین مقداردهی اولیه نمی‌تواند به جدول دسترسی پیدا کند.

```
25
26  int
27  find_shared_page(int id){
28      for (int i = 0; i < NUM_OF_SHARED_PAGES; i++){
29          if (shared_memory.table[i].id == id){
30              return i;
31          }
32      }
33      for (int i = 0; i < NUM_OF_SHARED_PAGES; i++){
34          if (shared_memory.table[i].num_of_refs == 0){
35              shared_memory.table[i].id = id;
36              return i;
37          }
38      }
39      return -1;
40  }
```

find_shared_page، وظیفه دارد تا یک صفحه مشترک را در جدول حافظه مشترک پیدا کند یا یک صفحه جدید را برای یک شناسه مشخص (ID) تخصیص دهد.

حلقه اول:

اگر صفحه‌ای پیدا شود که مقدارid آن برابر با id ورودی باشد:

- ایندکس (i) این صفحه برگردانده می‌شود.
- این به این معناست که صفحه با شناسه مورد نظر قبلاً وجود داشته و نیازی به تخصیص جدید نیست.

حلقه دوم:

این حلقه به دنبال صفحه‌ای است که تعداد ارجاعات (num_of_refs) آن برابر با صفر باشد.

- این نشان‌دهنده یک صفحه "خالی" یا "آزاد" است که می‌توان آن را به شناسه جدید تخصیص داد.

وقتی چنین صفحه‌ای پیدا شود:

- مقدار id آن به شناسه جدید تنظیم می‌شود.
- ایندکس این صفحه برگردانده می‌شود.

اگر هیچ صفحه‌ای با id مشخص وجود نداشته باشد و همچنین هیچ صفحه خالی در جدول موجود نباشد:

- **مقدار -1 برگردانده می‌شود.**
- **این نشان‌دهنده شکست در یافتن یا تخصیص صفحه است.**

```
42  char*
43  open_sharedmem(int id){
44      struct proc* proc = myproc();
45      pde_t *pgdir = proc->pgdir;
46      if (proc->shmem != 0){
47          return 0;
48      }
49      acquire(&shared_memory.lock);
50      int index = find_shared_page(id);
51      if (index == -1){
52          release(&shared_memory.lock);
53          return 0;
54      }
55      if (shared_memory.table[index].num_of_refs == 0){
56          shared_memory.table[index].frame = kalloc();
57          memset(shared_memory.table[index].frame, 0, PGSIZE);
58      }
59      char* start_mem = (char*)PGROUNDUP(proc->sz);
60      //cprintf("start_mem: %d\n", start_mem);
61
62      mappages(pgdir, start_mem, PGSIZE, V2P(shared_memory.table[index].frame), PTE_W|PTE_U);
63      shared_memory.table[index].num_of_refs++;
64      shared_memory.table[index].id = id;
65      proc->shmem = start_mem;
66      proc->shmem_id = id;
67
68      release(&shared_memory.lock);
69      return start_mem;
70  }
```

تابع open_sharedmem برای ایجاد یا دسترسی به یک (shared memory) طراحی شده است. این تابع بر اساس یک (ID)، بررسی می‌کند که آیا حافظه مشترک با این شناسه موجود است یا نه. در صورت موجود بودن، آدرس حافظه مشترک به پردازش بازگردانده می‌شود؛ در غیر این صورت، یک صفحه جدید تخصیص داده می‌شود.

با استفاده از myproc()، پردازش جاری که این تابع را فراخوانی کرده است، دریافت می‌شود.

سپس جدول صفحه (page directory) پردازش گرفته می‌شود تا برای نگاشت حافظه استفاده شود.

اگر پردازش قبلاً یک حافظه مشترک اختصاص داده داشته باشد (proc->shmem != 0)، تابع مقدار 0 را برمی‌گرداند و نشان می‌دهد که حافظه جدید نمی‌تواند تخصیص یابد.

تابع find_shared_page(id) فراخوانی می‌شود تا صفحه‌ای که با شناسه ورودی (id) مطابقت دارد، پیدا شود.

اگر تعداد ارجاعات به صفحه (num_of_refs) صفر باشد، این نشان می‌دهد که صفحه جدید است و باید به آن حافظه تخصیص داده شود.

با استفاده از kalloc()، یک صفحه حافظه جدید تخصیص داده می‌شود.سپس کل صفحه با مقدار صفر (memset) مقداردهی می‌شود.

آدرس شروع حافظه مشترک برای پردازش محاسبه می‌شود (PGROUNDUP(proc->sz)).

سپس با استفاده از mappages، صفحه به فضای آدرس مجازی پردازش نگاشت می‌شود.

اطلاعات مربوط به شناسه (id) و آدرس شروع (shmem) در پردازش به‌روزرسانی می‌شود.

در نهایت، تابع آدرس شروع حافظه مشترک (start_mem) را برمی‌گرداند.

```
73  void
74  close_sharedmem(int id){
75      struct proc* proc = myproc();
76      pde_t *pgdir = proc->pgdir;
77      if (proc->shmem_id != id || proc->shmem_id == 0){
78          return;
79      }
80      acquire(&shared_memory.lock);
81      int index = find_shared_page(id);
82      shared_memory.table[index].num_of_refs--;
83
84      //delete from proc's page table
85      uint a = PGROUNDUP((uint)proc->shmem);
86      pte_t *pte = walkpgdir(pgdir, (char*)a, 0);
87      if(!pte)
88          a = PGADDR(PDX(a) + 1, 0, 0) - PGSIZE;
89      else if((*pte & PTE_P) != 0){
90          *pte = 0;
91      }
92
93      proc->shmem = 0;
94      proc->shmem_id = 0;
95
96      if (shared_memory.table[index].num_of_refs == 0){
97          kfree(shared_memory.table[index].frame);
98
99      }
100
101     release(&shared_memory.lock);
102 }
```

تابع close_sharedmem برای بستن و آزاد کردن یک صفحه حافظه مشترک (shared memory) طراحی شده است. این تابع بررسی می‌کند که آیا پردازش جاری (current process) به صفحه مشخص‌شده با id متصل است یا خیر. اگر متصل باشد، ارجاع به آن صفحه را حذف می‌کند و در صورت لزوم حافظه صفحه را آزاد می‌کند.

اگر پردازش به صفحه‌ای که شناسه آن id است متصل نباشد (proc->shmem_id != id) یا هیچ حافظه مشترکی نداشته باشد (proc->shmem_id == 0)، تابع بدون انجام کاری خاتمه می‌یابد.

تابع find_shared_page(id) فراخوانی می‌شود تا صفحه مورد نظر در جدول حافظه مشترک پیدا شود.تعداد ارجاعات (num_of_refs) به این صفحه یک واحد کاهش می‌یابد.

آدرس شروع صفحه حافظه مشترک محاسبه می‌شود.از walkpgdir برای یافتن مدخل جدول صفحه (Page Table Entry) پردازش استفاده می‌شود.

اگر صفحه پیدا شد و معتبر بود ((*pte & PTE_P) != 0)، مقدار آن به صفر تنظیم می‌شود.

متغیرهای proc->shmem و proc->shmem_id که آدرس و شناسه حافظه مشترک پردازش را ذخیره می‌کنند، به صفر تنظیم می‌شوند.

اگر تعداد ارجاعات به صفحه برابر با صفر شود تابع kfree فراخوانی می‌شود تا حافظه صفحه آزاد شود.

Result: