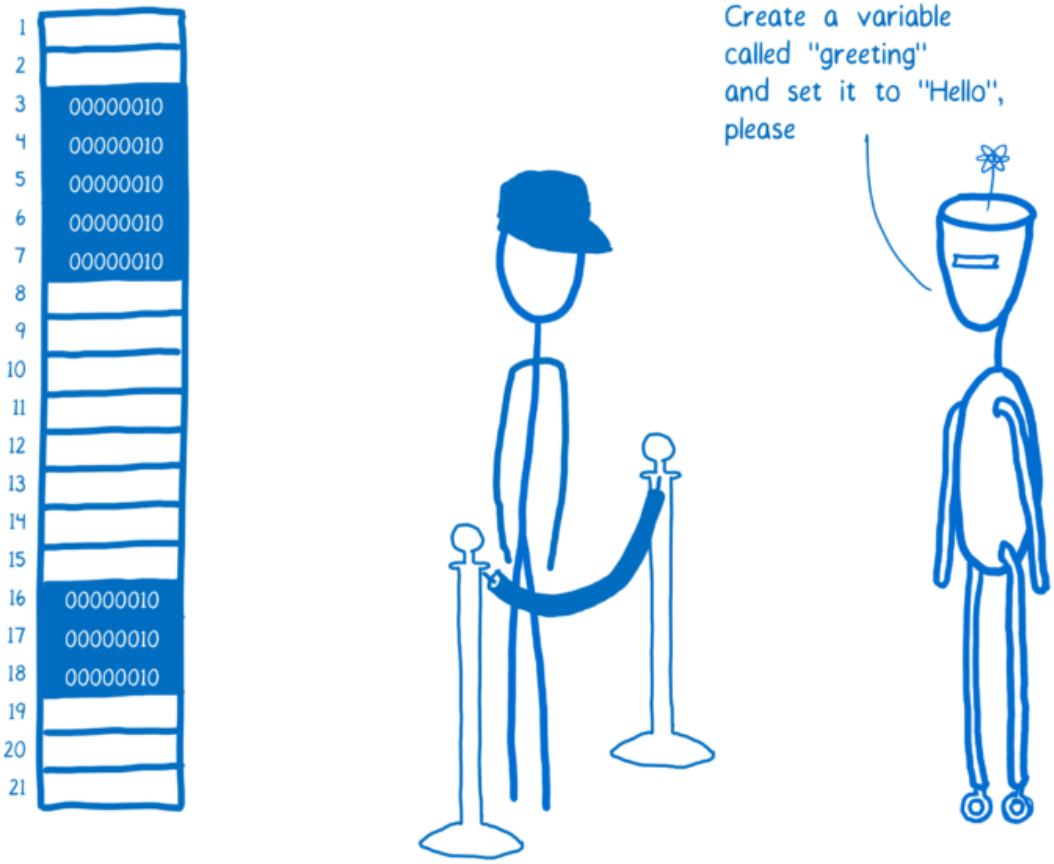
به نام خدا

آزمایشگاه سیستم‌عامل

پروژه پنجم: مدیریت حافظه

(آشنایی با حافظه مجازی در xv6)

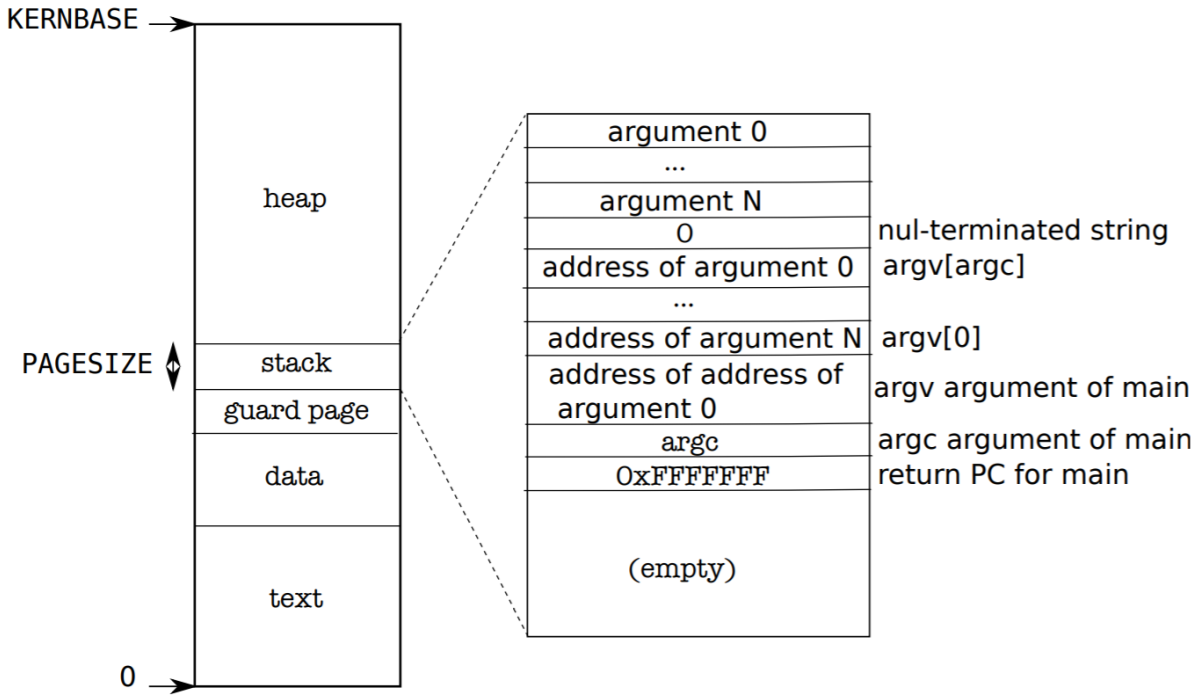
طراحان: نیما مدرس گرجی، غزل مینایی



در این پروژه شیوه مدیریت حافظه در سیستم‌عامل xv6 بررسی شده و قابلیت‌هایی به آن افزوده خواهد شد. در ادامه ابتدا مدیریت حافظه به طور کلی در xv6 معرفی شده و در نهایت صورت آزمایش شرح داده خواهد شد.

مقدمه

یک برنامه، حین اجرا تعامل‌های متعددی با حافظه دارد. دسترسی به متغیرهای ذخیره شده و فراخوانی توابع موجود در نقاط مختلف حافظه مواردی از این ارتباط‌ها می‌باشد. معمولاً کد منبع دارای آدرس نبوده و از نمادها برای ارجاع به متغیرها و توابع استفاده می‌شود. این نمادها توسط کامپایلر و پیونددهنده[۱] به آدرس تبدیل خواهد شد. حافظه یک برنامه سطح کاربر شامل بخش‌های مختلفی مانند کد، پشته[۲] و هیپ[۳] است. این ساختار برای یک برنامه در xv6 در شکل زیر نشان داده شده است.

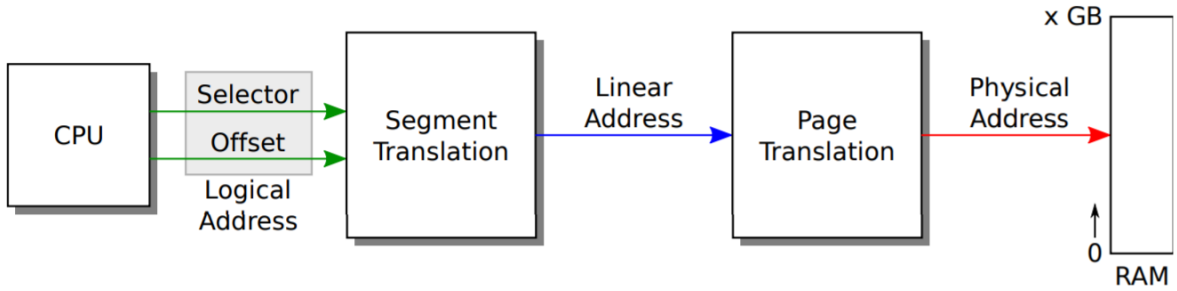


۱) ساختار حافظه مجازی (مشابه شکل بالا) یک برنامه در لینوکس در معماری x86 (۳۲ بیتی) را نشان دهید. (راهنمایی: می‌توانید به منبع [۱] رجوع کنید.)

همان‌طور که در آزمایش یک ذکر شد، در مد محافظت‌شده[۴] در معماری x86 هیچ کدی (اعم از کد هسته یا کد برنامه سطح کاربر) دسترسی مستقیم به حافظه فیزیکی[۵] نداشته و تمامی آدرس‌های برنامه

[1] Linker
[2] Stack
[3] Heap
[4] Protected Mode
[5] Physical Memory

از خطی[۱] به مجازی[۲] و سپس به فیزیکی تبدیل می‌شوند. این نگاشت در شکل زیر نشان داده شده است.



به همین منظور، هر برنامه یک جدول اختصاصی موسوم به جدول صفحه[۳] داشته که در حین فرایند تعویض‌متن[۴] بارگذاری شده و تمامی دسترسی‌های حافظه (اعم از دسترسی به هسته یا سطح کاربر) توسط آن برنامه توسط این جدول مدیریت می‌شود.

به علت عدم استفاده صریح از قطعه‌بندی در بسیاری از سیستم‌عامل‌های مبتنی بر این معماری، می‌توان فرض کرد برنامه‌ها از صفحه‌بندی[۵] و لذا آدرس مجازی استفاده می‌کنند. علت استفاده از این روش مدیریت حافظه در درس تشریح شده است. به طور مختصر می‌توان سه علت عمده را برشمرد:

۱) ایزوله‌سازی پردازه‌ها از یکدیگر و هسته از پردازه‌ها: با اجرای پردازه‌ها در فضاهای آدرس[۶] مجزا، امکان دسترسی یک برنامه مخرب به حافظه برنامه‌های دیگر وجود ندارد. ضمن این که با اختصاص بخش مجزا و ممتازی از هر فضای آدرس به هسته امکان دسترسی محافظت‌نشده پردازه‌ها به هسته سلب می‌گردد.

۲) ساده‌سازی ABI سیستم‌عامل: هر پردازه می‌تواند از یک فضای آدرس پیوسته (از آدرس مجازی صفر تا چهار گیگابیت در معماری x86) به طور اختصاصی استفاده نماید. به عنوان مثال کد یک برنامه در سیستم‌عامل لینوکس در معماری x86 همواره (در صورت عدم استفاده از تصادفی‌سازی چینش فضای

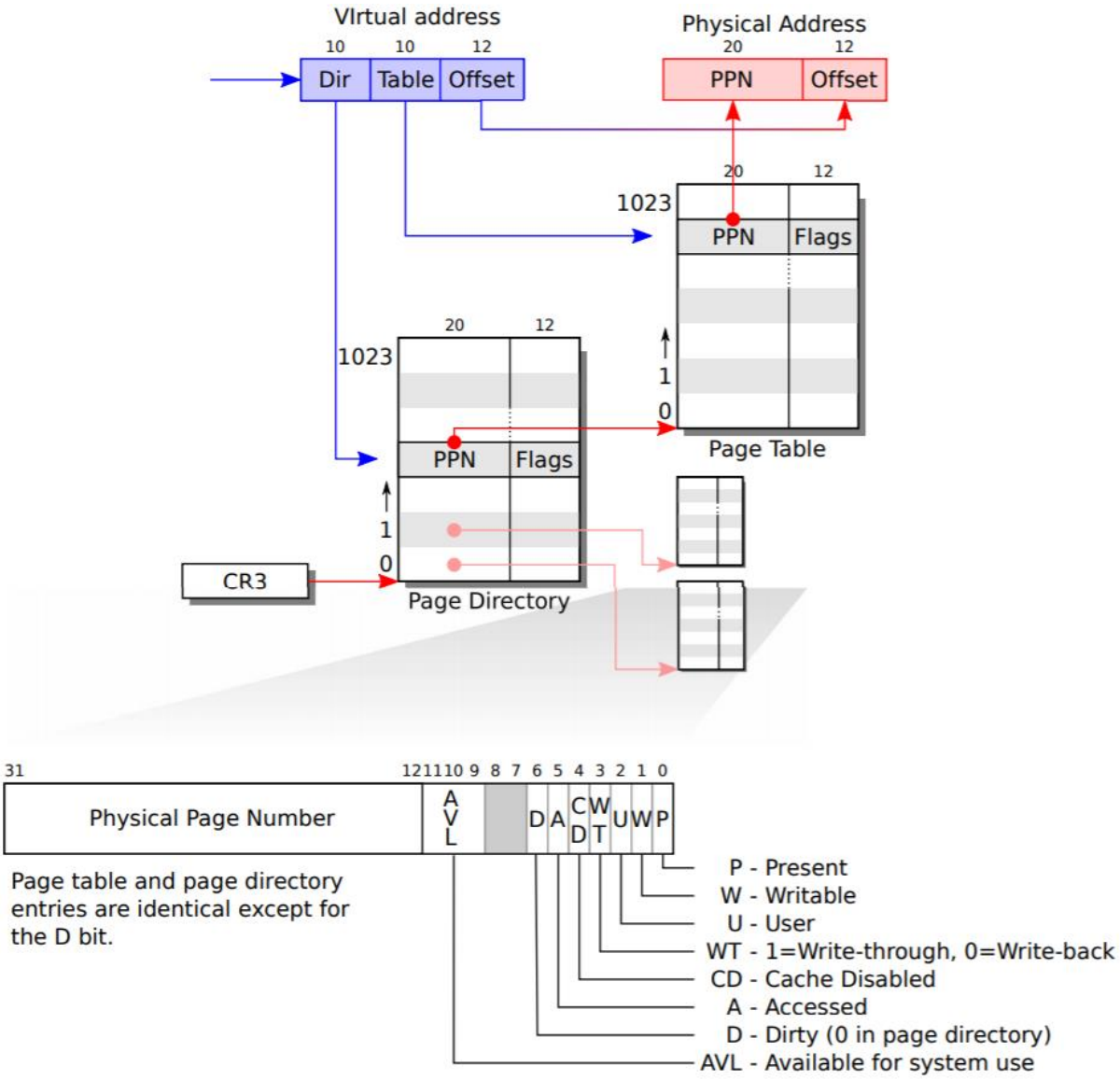
[1] Linear
[2] Virtual
[3] Page Table
[4] Context Switch
[5] Paging
[6] Address Spaces

آدرس[1] (ASLR)) از آدرس 0x08048000 آغاز شده و نیاز به تغییر در آدرس‌های برنامه‌ها متناسب با وضعیت جاری تخصیص حافظه فیزیکی نمی‌باشد.

۳) استفاده از جابه‌جایی حافظه: با علامت‌گذاری برخی از صفحه‌های کم‌استفاده (در جدول صفحه) و انتقال آن‌ها به دیسک، حافظه فیزیکی بیشتری در دسترس خواهد بود. به این عمل جابه‌جایی حافظه[2] اطلاق می‌شود.

ساختار جدول صفحه در معماری x86 (در حالت بدون گسترش آدرس فیزیکی[3] (PAE) و گسترش اندازه صفحه[4] (PSE)) در شکل زیر نشان داده شده است.

[1] Address Space Layout Randomization
[2] Memory Swapping
[3] Physical Address Extension
[4] Page Size Extension



هر آدرس مجازی توسط اطلاعات این جدول به آدرس فیزیکی تبدیل می‌شود. این فرایند، سخت‌افزاری بوده و سیستم‌عامل به طور غیرمستقیم با پر کردن جدول، نگاشت را صورت می‌دهد. جدول صفحه دارای سلسله‌مراتب دوسطحی بوده که به ترتیب Page Directory و Page Table نام دارند. هدف از ساختار سلسله‌مراتبی کاهش مصرف حافظه است.

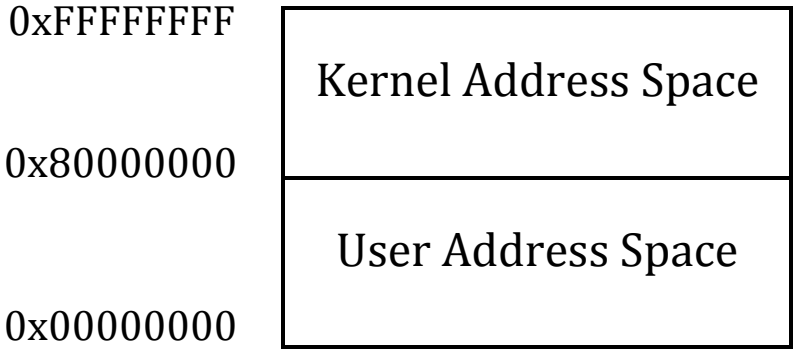
۲) چرا ساختار سلسله‌مراتبی منجر به کاهش مصرف حافظه می‌گردد؟

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

مدیریت حافظه در xv6

ساختار فضای آدرس در xv6

در xv6 نیز مد اصلی اجرای پردازنده، مد حفاظت‌شده و سازوکار اصلی مدیریت حافظه صفحه‌بندی است. به این ترتیب نیاز خواهد بود که پیش از اجرای هر کد، جدول صفحه آن در دسترس پردازنده قرار گیرد. کدهای اجرایی در xv6 شامل کد پردازه‌ها (کد سطح کاربر) و ریسه هسته متناظر با آن‌ها و کدی است که در آزمایش یک، کد مدیریت‌کننده نام‌گذاری شد.[۱] آدرس‌های کد پردازه‌ها و ریسه هسته آن‌ها توسط جدول صفحه‌ای که اشاره‌گر به ابتدای Page Directory آن در فیلد pgdir از ساختار proc (خط ۲۳۳۹) قرار دارد، نگاشت داده می‌شود. نمای کلی ساختار حافظه مجازی متناظر با جدول صفحه این دسته در شکل زیر نشان داده شده است.



دوگیگابایت پایین جدول صفحه مربوط به اجزای مختلف حافظه سطح کاربر پردازه است. دو گیگابایت بالای جدول صفحه مربوط به اجزای ریسه هسته پردازه بوده و در تمامی پردازه‌ها یکسان است. آدرس تمامی متغیرهایی که در هسته تخصیص داده می‌شوند در این بازه قرار می‌گیرد. جدول صفحه کد مدیریت‌کننده هسته، دو گیگابایت پایینی را نداشته (نگاشتی در این بازه ندارد) و دو گیگابایت بالای آن

[۱] بحث مربوط به پس از اتمام فرایند بوت است. به عنوان مثال، در بخشی از بوت، از صفحات چهار مگابایتی استفاده شد که از آن صرفنظر شده است.

دقیقاً شبیه به پردازه‌ها خواهد بود. زیرا این کد، همواره در هسته اجرا شده و پس از بوت غالباً در اوقات بی‌کاری سیستم اجرا می‌شود.

کد مربوط به ایجاد فضاهای آدرس در xv6

فضای آدرس کد مدیریت‌کننده هسته در حین بوت، در تابع main() ایجاد می‌شود. به این ترتیب که تابع kvmalloc() فراخوانی شده (خط ۱۲۲۰) و به دنبال آن تابع setupkvm() متغیر سراسری kpgdir را مقداردهی می‌نماید (خط ۱۸۴۲). به طور کلی هر زمان نیاز به مقداردهی ساختار فضای آدرس هسته باشد، از setupkvm() استفاده خواهد شد. با بررسی تابع setupkvm() (خط ۱۸۱۸) می‌توان دریافت که در این تابع، ساختار فضای آدرس هسته بر اساس محتوای آرایه kmap (خط ۱۸۰۹) چیده می‌شود.

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

۵) چگونه می‌توان روی یک آدرس فیزیکی خاص نوشت؟ دستور مورد نظر و شیوه دسترسی به آدرس مربوطه را ذکر کنید.

۶) تابع mappages() چه کاربردی دارد؟

فضای آدرس مجازی نخستین برنامه سطح کاربر (initcode) نیز در تابع main() ایجاد می‌گردد. به طور دقیق‌تر تابع userinit() (خط ۱۲۳۵) فراخوانی شده و توسط آن ابتدا نیمه هسته فضای آدرس با اجرای تابع setupkvm() (خط ۲۵۲۸) مقداردهی خواهد شد. نیمه سطح کاربر نیز توسط تابع inituvm() ایجاد شده تا کد برنامه نگاشت داده شود. فضای آدرس باقی پردازه‌ها در ادامه اجرای سیستم توسط توابع fork() یا exec() مقداردهی می‌شود. به این ترتیب که هنگام ایجاد پردازه فرزند توسط fork() با فراخوانی تابع copyuvm() (خط ۲۵۹۲) فضای آدرس نیمه هسته ایجاد شده (خط ۲۰۴۲)

و سپس فضای آدرس نیمه کاربر از والد کپی می‌شود. این کپی با کمک تابع walkpgdir() (خط ۲۰۴۵) صورت می‌پذیرد.

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

وظیفه تابع exec() اجرای یک برنامه جدید در ساختار بلوک کنترلی پردازه[1] (PCB) یک پردازه موجود است. معمولاً پس از ایجاد فرزند توسط fork() فراخوانده شده و کد، داده‌های ایستا، پشته و هیپ برنامه جدید را در فضای آدرس فرزند ایجاد می‌نماید. بدین ترتیب با اعمال تغییراتی در فضای آدرس موجود، امکان اجرای یک برنامه جدید فراهم می‌شود. روش متداول Shell در سیستم‌عامل‌های مبتنی بر یونیکس از جمله xv6 برای اجرای برنامه‌های جدید مبتنی بر exec() است. Shell پس از دریافت ورودی و فراخوانی fork1() تابع runcmd() را برای اجرای دستور ورودی، فراخوانی می‌کند (خط ۸۷۲۴). این تابع نیز در نهایت تابع exec() را فراخوانی می‌کند (خط ۸۶۲۶). چنان‌چه در آزمایش یک مشاهده شد، خود Shell نیز در حین بوت با فراخوانی فراخوانی سیستمی sys_exec() (خط ۸۴۱۴) و به دنبال آن exec() ایجاد شده و فضای آدرسش به جای فضای آدرس نخستین پردازه (initcode) چیده می‌شود. در پیاده‌سازی exec() مشابه قبل setupkvm() فراخوانی شده (خط ۶۶۳۷) تا فضای آدرس هسته تعیین گردد. سپس با فراخوانی allocuvm() فضای مورد نیاز برای کد و داده‌های برنامه جدید (خط ۶۶۵۱) و صفحه محافظ و پشته (خط ۶۶۶۵) تخصیص داده می‌شود. دقت شود تا این مرحله تنها تخصیص صفحه صورت گرفته و باید این فضاها در ادامه توسط توابع مناسب با داده‌های مورد نظر پر شود (به ترتیب خطوط ۶۶۵۵ و ۶۶۸۶).

[1] Process Control Block

شرح آزمایش

در این آزمایش می‌خواهیم به xv6 قابلیت استفاده از حافظه مشترک را اضافه کنیم. در این حالت دو یا بیشتر پردازه می‌توانند یک یا چند صفحه حافظه را به اشتراک بگذارند. برای انجام این کار یک جدول حافظه مشترک (shm_table) در نظر می‌گیریم که اطلاعات قطعه‌های مشترک (به ازای هر قطعه، یک عنصر) در آن ذخیره می شود. همه پردازه‌هایی که به یکی از این قطعات متصل هستند، مدخل‌هایی در جدول صفحه خود دارند که همه آن‌ها به صفحه‌های فیزیکی مربوط به این قطعه مشترک اشاره می‌کنند. در این آزمایش، اشتراک یک صفحه کافی است. به عبارت دیگر، قطعه مشترک حافظه[1]، تک‌صفحه‌ای است.

برای هر قطعه مشترک، لازم است که داده‌ساختاری موسوم به struct shmid_ds مشابه زیر تعریف شود:

- **perm_info**: شامل id و mode که از نوع struct ipc_perm است
- **ref_count**: مشخص‌کننده تعداد پردازه‌هایی که به این حافظه مشترک دسترسی دارند
- **attached_processes**: شناسه پردازه‌هایی که به این بخش از حافظه دسترسی دارند
- **frame**: آدرس فیزیکی صفحه مشترک

جزئیات داده‌ساختار مربوط به اطلاعات ساختاری و شیوه دسترسی قطعه مشترک (struct ipc_perm) بدین ترتیب است:

- **id**: شناسه یکتا برای تشخیص حافظه مشترک
- **mode**: مشخص‌کننده خصوصیات حافظه مشترک (در ادامه، حالات[2] مختلف، توضیح داده شده است)، تعیین بیت‌های هر حالت به صورت دلخواه

[1] Shared Memory Segment
[2] Modes

حالت‌های دسترسی عبارتند از:

۱) حالت USR_R:

در این حالت پردازه‌ها فقط دسترسی خواندن از قطعه را دارند.

۲) حالت USR_RW:

در این حالت پردازه‌ها هم دسترسی خواندن و هم نوشتن در قطعه را دارند.

۳) علامت‌گذاری شده برای حذف

همانطور که ذکر شد، جدول حافظه مشترک، شامل آرایه‌ای از داده‌های مربوط به قطعه‌های مشترک (struct shmid_ds) است و هم‌چنین دقت کنید که باید مکانیزمی برای عدم دسترسی همزمان به عناصر این آرایه در نظر بگیرید. برای این کار همانطور که قبلا یاد گرفته‌اید می‌توانید از قفل استفاده کنید. دقت کنید که ممکن است برای پیاده‌سازی فراخوانی‌های سیستمی به اطلاعات دیگری نیز نیاز داشته باشید.

برای پیاده‌سازی حافظه مشترک، سه فراخوانی سیستمی sys_shm_getat()، sys_shm_detach() و sys_shm_ctl() را مطابق توضیحاتی که در ادامه آمده به این سیستم‌عامل اضافه کنید.

۱) فراخوانی سیستمی int sys_shm_getat(int id):

در این فراخوانی سیستمی، پارامتر id شناسه‌ای است که لازم است به قطعه مشترک اختصاص داده شود. در این تابع ابتدا جدول قطعات مشترک، بررسی شده و در صورتی که قطعه‌ای با شناسه داده شده از قبل وجود داشته باشد، مقدار ref_count آن را یک واحد افزایش یافته و شناسه پردازه در لیست پردازه‌های متصل به آن قرار می‌گیرد، سپس با استفاده از تابع mappages() که در هسته xv6 موجود است، یک

نگاشت میان آدرس مجازی و فیزیکی ایجاد می‌شود. برای ایجاد نگاشت به حالت دسترسی قطعه، توجه کنید. در صورت موجود نبودن قطعه با شناسه داده شده، یک قطعه به اندازه یک صفحه تخصیص داده شده و اطلاعات لازم در struct shm_table ذخیره می‌شود. در انتها عمل الصاق مشابه حالت قبل، صورت می‌گیرد.

۲) فراخوانی سیستمی int sys_shm_detach(int id):

در این فراخوانی سیستمی، باید جدول قطعات مشترک بررسی گردد و در صورت وجود قطعه‌ای با شناسه داده شده، مقدار ref_count آن یک واحد کاهش یابد و پردازه صدا زننده نیز از لیست پردازه‌های متصل به آن قطعه حذف شود. در صورتی که قطعه برای آزادسازی علامت خورده باشد (در ادامه بیشتر درباره علامت خوردن قطعات توضیح داده شده است) و ref_count هم صفر شود، باید این قطعه از shm_table حذف شود. در غیر این صورت همچنان با دادن شناسه قابل اتصال است. حتی اگر شمارنده به صفر برسد. نیازی به آزادسازی صفحه از فضای آدرس مجازی نیست. در صورت موجود نبودن قطعه با شناسه داده شده باید خطای مناسب برگردانده شود.

۳) فراخوانی سیستمی int sys_shm_ctl(int shmid, int cmd, struct shmid_ds *buf):

در این فراخوانی سیستمی یک عملیات کنترلی بر روی حافظه مشترک با شناسه shmid صورت می‌گیرد. این عملیات با cmd مشخص می‌شود که می‌تواند یکی از سه مقدار IPC_STAT، IPC_SET و یا IPC_RMID باشد:

- **دستور IPC_SET**

در این دستور mode مشخص شده با shmid به mode مشخص شده در buf تغییر می‌یابد. با تغییر mode باید نوع دسترسی پردازه‌ها به قطعه نیز تغییر یابد.

- **دستور IPC_STAT**

اطلاعات (شامل id و mode) مربوط به ساختار shmid_ds مربوط به قطعه مشترک متناظر با شناسه مشخص شده با shmid در buf قرار می‌گیرد.

- **دستور IPC_RMID**

با اجرای این دستور، قطعه مشترک متناظر با shmid برای آزاد شدن علامت زده می‌شود. پس از این که سگمنتی علامت زده می‌شود دیگر پردازه‌ای نمی‌تواند به آن وصل شود. همچنین قطعه علامت زده شده بعد از اینکه آخرین پردازه متصل به آن جدا شود، آزاد می‌گردد. اگر اتصالی وجود نداشت بلافاصله آزاد می‌شود.

فرض کنید تمام پردازه‌ها قبل از این‌که از بین بروند sys_shm_detach() را صدا می‌زنند و نیازی به بررسی حالتی که پردازه‌ها قبل از صدا زدن این تابع خارج (یا کشته) شوند نیست.

سایر نکات

- کدهای خود را به زبان C بنویسید و کدهای مربوط به حافظه اشتراکی را در فایل sharedm.c قرار دهید.
- نیازی به نوشتن گزارش برای بخش پیاده سازی نیست ولی سوالاتی که در صورت پروژه پرسیده شده است را باید در گزارش بیاورید.
- جهت آزمون صحت عملکرد پیاده‌سازی، برای هر بخش یک برنامه سمت کاربر بنویسید که عملکرد

تغییرات اعمال‌شده را در شرایط گوناگون مورد بررسی قرار دهد. هنگام تحویل پروژه، صحت پیاده‌سازی شما مقابل برنامه‌های سمت کاربر دیگری نیز سنجیده خواهد شد.

- برای تحویل پروژه، یک مخزن خصوصی در سایت GitLab ایجاد نموده و کد هر دو بخش را در یک شاخه[1] آن Push کنید. سپس اکانت UT_OS_TA را با دسترسی Maintainer به مخزن خود اضافه کنید. کافی است در محل بارگذاری در سایت درس، آدرس مخزن، شناسه آخرین Commit و گزارش پروژه را بارگذاری نمایید.
- همه اعضای گروه باید به پروژه بارگذاری شده توسط گروه خود مسلط بوده و لزوماً نمره افراد یک گروه با یکدیگر برابر نخواهد بود.
- در صورت مشاهده هرگونه مشابهت بین کدها یا گزارش دو یا چند گروه، نمره صفر به همه آن‌ها تعلق می‌گیرد.
- پاسخ تمامی سؤالات را در کوتاه‌ترین اندازه ممکن در گزارش خود بیاورید.
- هر گونه سؤال در مورد پروژه را از طریق ایمیل با دستیاران آموزشی درس مطرح نمایید.

موفق باشید

[1] Branch