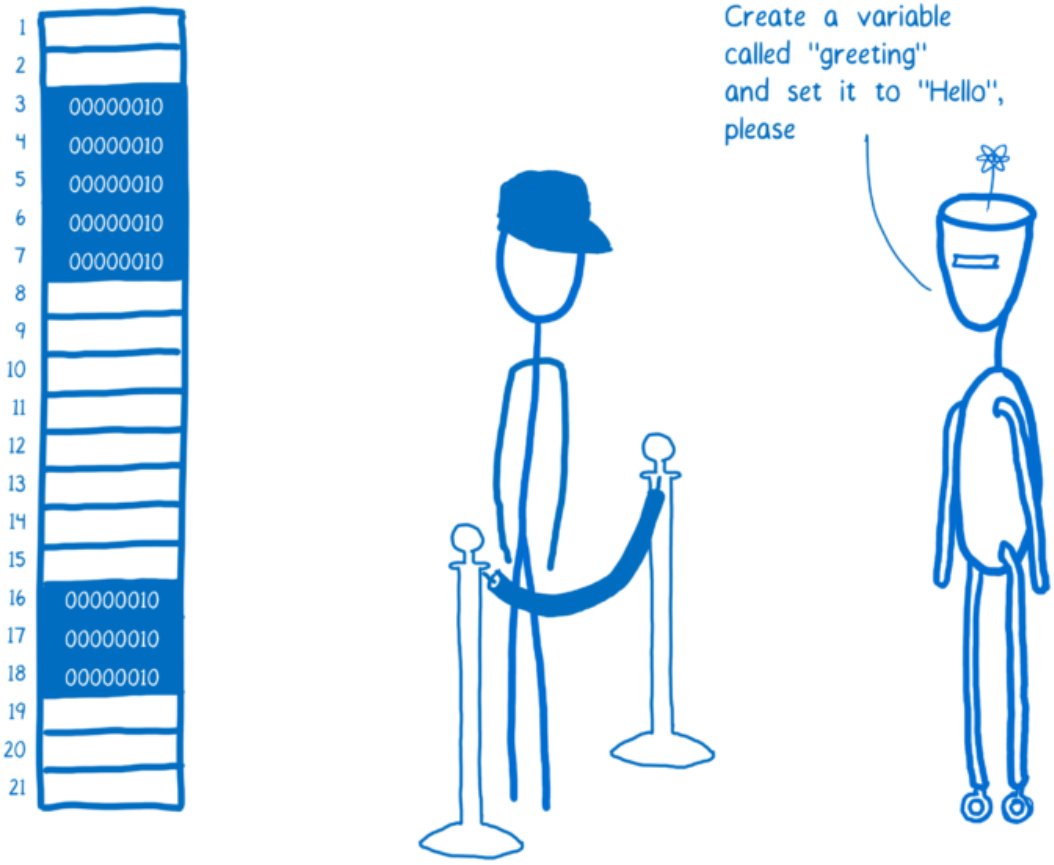
به نام خدا

آزمایشگاه سیستم‌عامل

پروژه پنجم: مدیریت حافظه

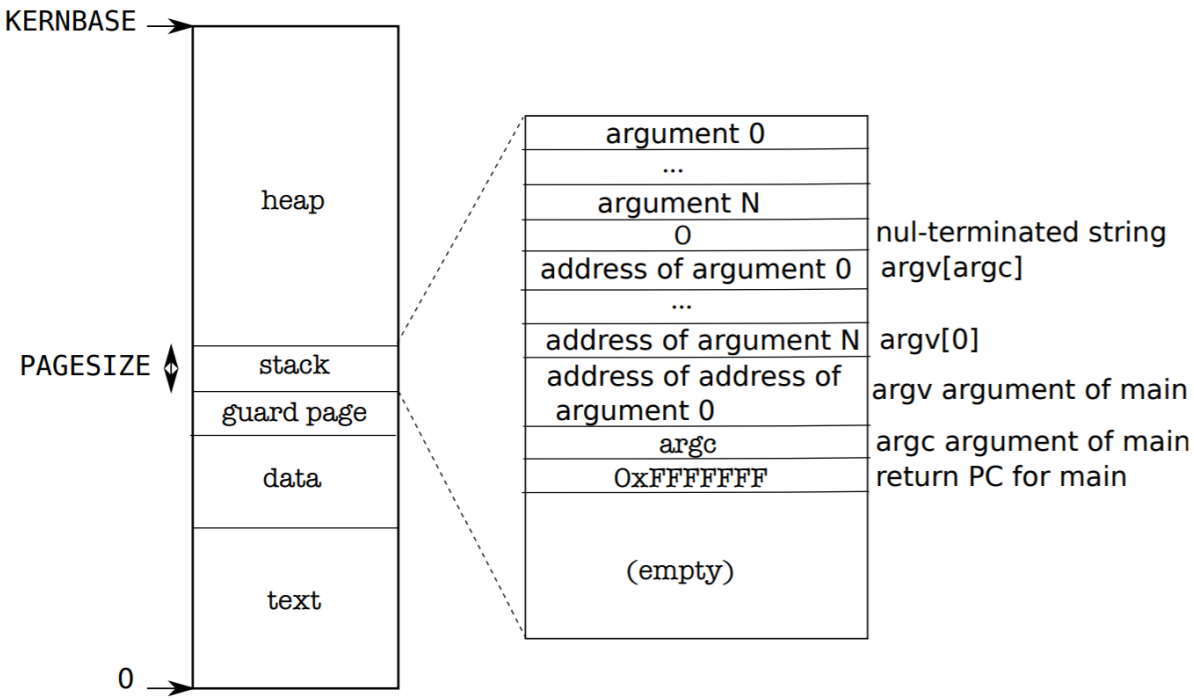
طراحان: سید علی اخوانی-صادق حائری



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

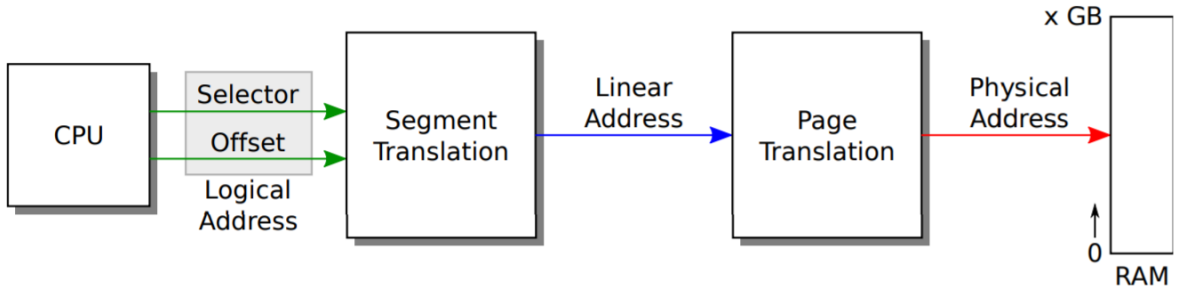
و از نمادها برای ارجاع به متغیرها و توابع استفاده می‌شود. این نمادها توسط کامپایلر و پیونددهنده[1] به آدرس تبدیل خواهد شد. حافظه یک برنامه سطح کاربر شامل بخش‌های مختلفی مانند کد، پشته[2] و هیپ[3] است. این ساختار برای یک برنامه در xv6 در شکل زیر نشان داده شده است.



همان‌طور که در آزمایش یک ذکر شد، در مد محافظت‌شده[4] در معماری x86 هیچ کدی (اعم از کد هسته یا کد برنامه سطح کاربر) دسترسی مستقیم به حافظه فیزیکی[5] نداشته و تمامی آدرس‌های برنامه

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

از خطی[۶] به مجازی[۷] 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[۸] داشته که در حین فرایند تعویض‌متن[۹] 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[۱۰] 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[۱۱] 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

---

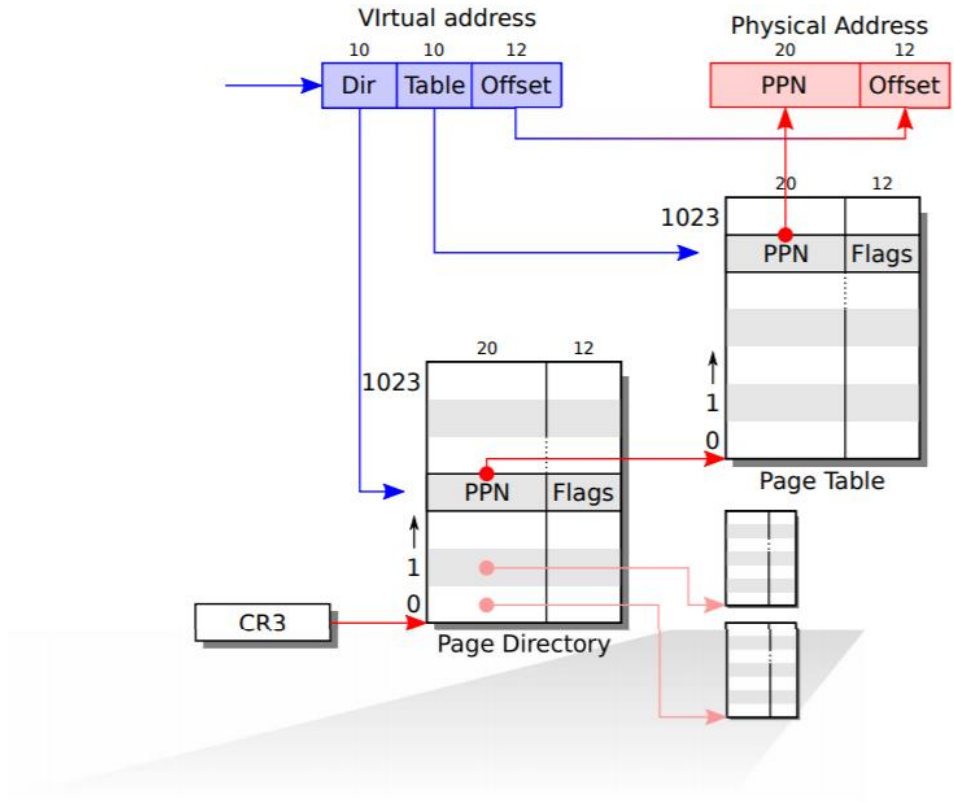
[6] Linear
[7] Virtual
[8] Page Table
[9] Context Switch
[10] Paging
[11] Address Spaces

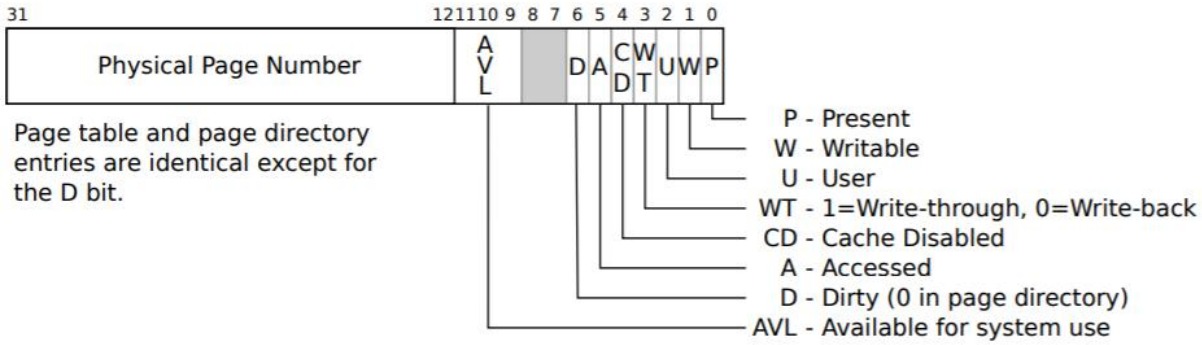
آدرس[۱۲] (ASLR)) از آدرس 0x08048000 آغاز شده و نیاز به تغییر در آدرس‌های برنامه‌ها متناسب با وضعیت جاری تخصیص حافظه فیزیکی نمی‌باشد.

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[۱۳] اطلاق می‌شود.

ساختار جدول صفحه در معماری x86 (در حالت بدون گسترش آدرس فیزیکی[۱۴] (PAE) و گسترش اندازه صفحه[۱۵] (PSE)) در شکل زیر نشان داده شده است.

12 Address Space Layout Randomization
13 Memory Swapping
14 Physical Address Extension
15 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.

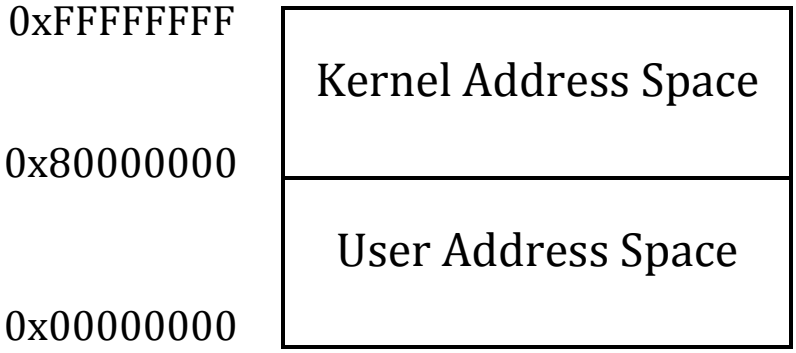
۱) چرا ساختار سلسله‌مراتبی منجر به کاهش مصرف حافظه می‌گردد؟

۲) 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.[۱۶] 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

[۱۶] 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.

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

۳) 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.

۴) 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[۱۷] (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).

۵) داده‌ساختار kmem در فایل kalloc.c چه کاربردی دارد؟ xv6 چگونه صفحات آزاد را ردیابی و نگهداری می‌کند؟

۶) در مورد دو فراخوانی سیستمی shmget() و shmat() در سیستم‌عامل‌های مبتنی بر یونیکس توضیح دهید.

[17] Process Control Block

شرح آزمایش

در این بخش می‌خواهیم تا برای این سیستم‌عامل قابلیت استفاده از حافظه مشترک[۱۸] را پیاده‌سازی کنید. به این معنی که دو یا چند پردازه بتوانند یک یا چند صفحه حافظه را به اشتراک بگذارند. این کار به این نحو پیاده‌سازی می‌شود که هر پردازه مدخلی[۱۹] در جدول صفحه خود داشته باشد که همه آن‌ها به یک صفحه فیزیکی اشاره کنند.

برای پیاده‌سازی این بخش از شما انتظار می‌رود تا سه فراخوانی سیستمی sys_shm_open()، sys_shm_attach() و sys_shm_close() را به این سیستم‌عامل اضافه کنید. در ادامه در مورد کاربرد آن‌ها توضیح داده خواهد شد.

۱) فراخوانی سیستمی int sys_shm_open(int id, int page_count, int flag)

این تابع وظیفه دارد که shm_table را بررسی نموده و در صورت عدم وجود سگمنتی با شناسه ورودی، یک صفحه تخصیص و نگاشت داده و در نهایت این اطلاعات را در shm_table ذخیره کند. در صورت وجود سگمنت در shm_table پیغام خطای مناسبی برگردانده خواهد شد. توجه شود که پردازه فراخواننده sys_shm_open() باید به عنوان مالک[۲۰] آن تعیین شود.[۲۱] این فراخوانی سیستمی سه ورودی از طرف برنامه سطح کاربر دریافت می‌کند: ۱) پارامتر id که شناسه مورد نظر است، ۲) پارامتر page_count که تعداد صفحات مورد نیاز بوده و ۳) flag که در ادامه مورد استفاده آن بیان خواهد شد. در نهایت این تابع باید موفقیت یا شکست تخصیص را به عنوان خروجی برگرداند (از اعداد منفی برای بیان خطاها استفاده شود).

---

18 Shared Memory
19 Entry
20 Owner
۲۱ در شرایط واقعی به عنوان مثال در پیاده‌سازی حافظه مشترک در لینوکس، مالک، موجودیت کاربر است که در این‌جا با توجه به عدم پشتیبانی سیستم‌عامل از این موجودیت، مالکیت به پردازه‌ها نسبت داده شده است.

۲) فراخوانی سیستمی void *sys_shm_attach(int id)

این تابع وظیفه دارد تا پردازه مورد نظر را به یک حافظه مشترک موجود الصاق نماید. برای این کار می‌بایست آن سگمنت، جستجو شده و مقدار refcnt متناظر با آن یک واحد افزایش یابد. سپس با استفاده از تابع mappages() در هسته xv6، یک نگاشت میان آدرس مجازی و آدرس فیزیکی ایجاد شود. این تابع id را به صورتی ورودی دریافت نموده و آدرس مجازی ابتدای حافظه مشترک را بر می‌گرداند.

۳) فراخوانی سیستمی int sys_shm_close(int id)

در هنگام فراخوانی این تابع، باید سگمنت متناظر با شناسه ورودی با جستجو در shm_table استخراج گردد. سپس از مقدار ref_cnt یک واحد کاسته شده و با رسیدن آن به صفر، باید تمام shm_table حذف شده و به عبارت دیگر، این حافظه مشترک از بین برود. نیازی به آزادسازی صفحه از فضای آدرس مجازی نیست.

راهنمای پیاده‌سازی

برای اضافه کردن حافظه اشتراکی یک فایل جدید با نام sharedm.c ایجاد کنید و کد خود و توابع مورد نیاز را در آن بنویسید. برای ردیابی صفحات اشتراکی می‌توانید از یک داده‌ساختار ساده استفاده کنید. با توجه به عدم پشتیبانی xv6 از لیست پیوندی برای راحتی کار می‌توانید از آرایه با اندازه مشخص استفاده کنید. در هنگام ایجاد پردازه فرزند، دقت کنید تعداد ارجاع‌ها به‌روزرسانی شود. هم‌چنین تخصیص را در محل مناسبی از فضای آدرس انجام دهید.

در طراحی خود، لازم است که برای هر سگمنت مشترک حداقل مقادیر زیر را نگه‌داری کنید:

- id: شناسه یکتا برای تشخیص حافظه مشترک
- owner: پردازه ایجادکننده حافظه مشترک
- flags: مشخص‌کننده خصوصیات حافظه مشترک

- ref_count: مشخص‌کننده تعداد کاربران جاری حافظه مشترک
- size: تعداد صفحات مشترک
- frames: آرایه‌ای برای ذخیره‌سازی آدرس‌های فیزیکی صفحات مشترک

پرچم‌ها

در این بخش چند مقدار برای پرچم[۲۲] تعریف شده که حافظه مشترک باید از آن‌ها پشتیبانی کند. این مقادیر و نحوه عملکرد آن‌ها عبارت است از:

۱) ONLY_OWNER_WRITE

با فعال‌سازی این پرچم، تنها پردازه ایجادکننده حافظه مشترک اجازه نوشتن در این فضا را داشته و دیگر پردازه‌ها تنها قادر به خواندن از حافظه خواهند بود.

۲) ONLY_CHILD_CAN_ATTACH

در صورتی که ایجادکننده حافظه مشترک، این پرچم را فعال کند، تنها فرزندان این پردازه قادر به اتصال به حافظه مشترک هستند.

دقت شود حافظه مشترک تنها تا زمان حیات مالک، دارای مالک بوده و پس از آن محدودیت‌های مربوط به مالکیت رفع می‌شود.

سایر نکات

- کدهای شما باید به زبان C بوده و کدهای مربوط به حافظه اشتراکی در فایل sharedm.c قرار گیرد.

[22] Flag

- نیازی به نوشتن گزارش برای بخش پیاده‌سازی نیست. اما سؤالاتی به صورت شفاهی در هنگام تحویل از شما پرسیده می‌شود.
- در نهایت کدهای خود را در کنار گزارش با فرمت PDF در یک فایل zip با نام‌گذاری مطابق الگوی stdNum1_stdNum2.zip 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.
- فصول ۱ و ۲ کتاب xv6 می‌تواند مفید باشد.
- هر گونه سؤال در مورد پروژه را فقط از طریق فروم درس مطرح نمایید.

موفق باشید