به نام خدا

آزمایشگاه سیستم عامل

آشنایی، اجرا و اشکال‌زدایی هسته سیستم عامل xv6

(بخش اول: آشنایی با xv6)

مقدمه

سیستم عامل xv6 یک سیستم عامل آموزشی است که در سال ۲۰۰۶ توسط محققان دانشگاه MIT به وجود آمده است. این سیستم عامل به زبان C و با استفاده از هسته Unix Version 6 نوشته شده و برروی معماری Intel x86 قابل اجرا می‌باشد. سیستم عامل xv6 علی‌رغم سادگی و حجم کم، نکات اساسی و مهم در طراحی سیستم عامل را دارا است و برای مقاصد آموزشی بسیار مفید می‌باشد. تا پیش از این، در درس سیستم عامل دانشگاه تهران از هسته سیستم عامل لینوکس استفاده می‌شد که پیچیدگی‌های زیادی دارد. در ترم پیشرو، دانشجویان آزمایشگاه سیستم عامل بایستی پروژه‌های مربوطه را برروی سیستم عامل xv6 اجرا و پیاده‌سازی نمایند. در این پروژه، ضمن آشنایی به معماری و برخی نکات پیاده‌سازی سیستم عامل، آن را اجرا و اشکال‌زدایی خواهیم کرد و همچنین برنامه‌ای در سطح کاربر خواهیم نوشت که برروی این سیستم عامل قابل اجرا باشد.

آشنایی با سیستم‌عامل xv6

کدهای مربوط به سیستم عامل xv6 از لینک زیر قابل دسترسی است:

https://github.com/mit-pdos/xv6-public

همچنین مستندات این سیستم عامل و فایلی شامل کدهای آن در صفحه درس بارگزاری شده است. برای این پروژه، نیاز است که فصل‌های ۰ و ۱ از مستندات فوق را مطالعه کرده و به سوالات زیر پاسخ دهید. پاسخ این سوالات را در قالب یک گزارش آپلود خواهید کرد.

- سوال ۱ : معماری سیستم عامل xv6 چیست؟ چه دلایلی در دفاع از نظر خود دارید؟
- سوال ۲ : یک پردازه[1] در سیستم‌عامل xv6 از چه بخش‌هایی تشکیل شده است؟ این سیستم‌عامل به طور کلی چگونه cpu را به پردازه‌های مختلف اختصاص می‌دهد.
- سوال ۳ : فراخوانی‌های سیستمی fork و exec چه عملی انجام می‌دهند؟ از نظر طراحی، ادغام نکردن این دو چه مزیتی دارد؟
- سوال ۴: مفهوم file descriptor در سیستم عامل‌های مبتنی بر Unix چیست؟ عملکرد pipe در سیستم‌عامل xv6 چگونه است و به طور معمول برای چه هدفی استفاده می‌شود؟

اجرا و اشکال‌زدایی

در این بخش به اجرای سیستم عامل xv6 خواهیم پرداخت. علی‌رغم این که این سیستم عامل قابل boot شدن مستقیم بر روی سیستم است، به دلیل آسیب‌پذیری بالا و رعایت مسائل ایمنی، از این کار اجتناب می‌کنیم و سیستم عامل را به کمک امولاتور qemu روی سیستم عامل لینوکس اجرا می‌کنیم. برای این منظور لازم است که کدهای مربوط به سیستم عامل را از لینک ارائه شده clone و یا دانلود کنیم. در ادامه با اجرای دستور make در دایرکتوری دانلود، سیستم عامل کامپایل می‌شود. در نهایت با اجرای دستور make qemu سیستم عامل برروی ماشین‌مجازی[2] اجرا میشود (توجه شود که فرض شده qemu از قبل برروی سیستم شما نصب بوده است. در غیر این صورت ابتدا آن را نصب نمایید).

[1] Process

[2] Emulator

اضافه کردن یک متن به Boot Message

در این بخش، شما باید نام اعضای گروه را پس از بوت شدن سیستم‌عامل روی ماشین مجازی qemu، در انتهای پیام‌های نمایش داده شده در کنسول نشان دهید.
تصویر این اطلاعات را در گزارش خود قرار دهید.

اضافه کردن چند قابلیت به کنسول xv6

- پس از اجرای سیستم عامل برروی qemu، مشاهده می‌کنیم که درصورت استفاده از کلید Tab معادل ASCII این کاراکتر در کنسول نوشته می‌شوند. همان‌طور که از تجربه‌ی استفاده از ترمینال لینوکس می‌دانید،استفاده از این کلید باعث می‌شود تا دستور در صورت امکان کامل شود.
  در این قسمت از پروژه، باید یک ویژگی مشابه به این ویژگی به کنسول xv6 اضافه کنید. به این‌صورت که شما باید همواره ۱۰ دستور آخری که کاربر وارد کرده را نگه دارید و در صورت فشردن کلید Tab دستور را با بهترین گزینه از میان آن‌ها تکمیل کند. در صورتی که در بین این ۱۰ دستور هیچ‌کدام به عنوان تکمیل‌شده‌ی این دستور قابل قبول نبود هیچ‌کاری انجام داده نشود و اگر چند گزینه وجود داشت آن دستوری که زودتر استفاده شده بود جایگزین شود.

- همچنین در صورتی که کاربر دستور Ctrl + r را وارد کرد باید تمامی ترمینال پاک شده و بتوان یک دستور جدید در ترمینال وارد کرد. این دستور همانند دستور clear در ترمینال لینوکس می‌باشد.

اجرا و پیاده‌سازی یک برنامه‌ی سطح کاربر

در این قسمت شما باید یک برنامه‌ی سطح کاربر و به زبان C بنویسید به برنامه‌های سطح کاربر سیستم‌عامل اضافه کنید. اسم این برنامه‌ی سطح کاربر touch می‌باشد. دو حالت کلی برای اجرای این برنامه وجود دارد.
زمانی که تنها یک ورودی اولیه به برنامه‌ی سطح کاربر داده شود، برنامه یک فایل جدید با آن اسم ایجاد می‌کند.

```
$ touch file.txt
```

زمانی که پرچم w- در این دستور فعال باشد برنامه باید منتظر ورودی گرفتن از کاربر باشد و سپس یک با فایل آن محتوا ایجاد کند. در صورتی که این فایل موجود باشد باید محتوا بر روی فایل قبلی بازنویسی[3] شود.

```
$ touch -w file.txt
Hello World!
```

از فراخوانی‌های سیستمی open، read، write و close استفاده کنید که برای باز کردن، خواندن، نوشتن و بستن فایل‌ها استفاده می‌شود.

[3] overwrite

نکات مهم

- برای تحویل پروژه ابتدا یک مخزن خصوصی در سایت GitLab ایجاد نموده و سپس پروژه خود را در 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.
- در نهایت کدهای خود را در کنار گزارش با فرمت pdf در یک فایل zip آپلود نمایید.
- به تمامی سوالاتی که در صورت پروژه از شما پرسیده شده‌است پاسخ دهید و آن‌ها را در گزارش‌کار خود بیاورید.
- همه‌ی اعضای گروه باید به پروژه‌ی آپلود شده توسط گروه خود مسلط باشند و لزوماً نمره‌ی افراد یک گروه با یکدیگر برابر نیست.
- در صورت مشاهده‌ی هرگونه مشابهت بین کدها یا گزارش دو گروه، نمره‌ی ۰ به هر دو گروه تعلق می‌گیرد.
- تمامی سؤالات را در **کوتاه‌ترین اندازه ممکن** پاسخ دهید.

به نام خدا

آزمایشگاه سیستم عامل

آشنایی، اجرا و اشکال‌زدایی هسته سیستم عامل xv6

(بخش دوم: مراحل اجرا و بوت)

مقدمه‌ای درباره سیستم‌عامل و xv6

سیستم‌عامل جزو نخستین نرم‌افزارهایی است که پس از روشن شدن سیستم، اجرا می‌گردد. این نرم‌افزار، رابط نرم‌افزارهای کاربردی با سخت‌افزار رایانه است.

1. سه وظیفه اصلی سیستم‌عامل را بیان نمایید.
2. فایل‌های اصلی سیستم‌عامل xv6 در صفحه یک کتاب xv6 لیست شده‌اند. به طور مختصر هر گروه را توضیح دهید. نام پوشه اصلی فایل‌های هسته سیستم‌عامل، فایل‌های سرایند[1] و فایل‌سیستم در سیستم‌عامل لینوکس چیست. در مورد محتویات آن مختصراً توضیح دهید.

کامپایل سیستم‌عامل xv6

یکی از روش‌های متداول کامپایل و ایجاد نرم‌افزارهای بزرگ در سیستم‌عامل‌های مبتنی بر یونیکس استفاده از ابزار Make است. این ابزار با پردازش فایل‌های موجود در کد منبع برنامه، موسوم به Makefile، شیوه کامپایل و لینک فایل‌های دودویی به یکدیگر و در نهایت ساختن کد دودویی نهایی برنامه را تشخیص می‌دهد. ساختار Makefile قواعد خاص خود را داشته و می‌تواند بسیار پیچیده باشد. اما به طور کلی شامل قواعد[2] و متغیرها[3] می‌باشد. در xv6 تنها یک Makefile وجود داشته و تمامی فایل‌های سیستم‌عامل نیز در یک پوشه قرار دارند. بیلد سیستم‌عامل از طریق دستور make –j8 در پوشه سیستم‌عامل صورت می‌گیرد.

3. دستور make –n را اجرا نمایید. کدام دستور، فایل نهایی هسته را می‌سازد؟
4. در Makefile متغیرهایی به نام‌های UPROGS و ULIB تعریف شده است. کاربرد آن‌ها چیست؟

اجرا بر روی شبیه‌ساز QEMU

xv6 قابل اجرا بر روی سخت‌افزار واقعی نیز است. اما اجرا بر روی شبیه‌ساز قابلیت ردگیری و اشکال‌زدایی بیشتری ارایه می‌کند. جهت اجرای سیستم‌عامل بر روی شبیه‌ساز، کافی است دستور make qemu در پوشه سیستم‌عامل اجرا گردد.

5. دستور make qemu –n را اجرا نمایید. دو دیسک به عنوان ورودی به شبیه‌ساز داده شده است. محتوای آن‌ها چیست؟ (راهنمایی: این دیسک‌ها حاوی سه خروجی اصلی فرایند بیلد هستند.)

مراحل بوت سیستم‌عامل xv6

اجرای بوت‌لودر

هدف از بوت آماده‌سازی سیستم‌عامل برای سرویس‌دهی به برنامه‌های کاربر است. پس از بوت، سیستم‌عامل سازوکاری جهت ارائه سرویس به برنامه‌های کاربردی خواهد داشت که این برنامه‌ها بدون هیچ مزاحمتی بتوانند از آن استفاده نمایند. کوچکترین واحد دسترسی دیسک‌ها در رایانه‌های شخصی سکتور[4] است. در این‌جا هر سکتور ۵۱۲ بایت است. اگر دیسک قابل بوت باشد، نخستین

[1] Header Files
[2] Rules
[3] Variables
[4] Sector

سکتور آن سکتور بوت[5] نام داشته و شامل بوت‌لودر[6] خواهد بود. بوت‌لودر کدی است که سیستم‌عامل را در حافظه بارگذاری می‌کند. یکی از روش‌های راه‌اندازی اولیه رایانه، بوت مبتنی بر سیستم ورودی/خروجی مقدماتی[7] (BIOS) است. BIOS در صورت یافتن دیسک قابل بوت، سکتور نخست آن را در آدرس 0x7C00 از حافظه فیزیکی کپی نموده و شروع به اجرای آن می‌کند.

6. در xv6 در سکتور نخست دیسک قابل بوت، محتوای چه فایلی قرار دارد. (راهنمایی: خروجی دستور make –n را بررسی نمایید.)
7. برنامه‌های کامپایل شده در قالب فایل‌های دودویی نگه‌داری می‌شوند. فایل مربوط به بوت نیز دودویی است. نوع این فایل دودویی چیست؟ تفاوت این نوع فایل دودویی با دیگر فایل‌های دودویی کد xv6 چیست؟ چرا از این نوع فایل دودویی استفاده شده است؟ این فایل را به زبان قابل فهم انسان (اسمبلی) تبدیل نمایید. (راهنمایی: از ابزار objdump استفاده کنید. باید بخشی از آن مشابه فایل bootasm.S باشد.)
8. دستورهای objdump --dwarf=info و objdump --dwarf=decodedline را روی فایل هسته اجرا نمایید.[8] خروجی این دستورها چیست؟ در حد یک خط کاربرد هر یک را توضیح دهید.
9. بوت سیستم توسط فایل‌های bootasm.S و bootmain.c صورت می‌گیرد. چرا تنها از کد C استفاده نشده است؟

معماری سیستم شبیه‌سازی شده x86 است. حالت سیستم در حال اجرا در هر لحظه را به طور ساده می‌توان شامل حالت پردازنده و حافظه دانست. بخشی از حالت پردازنده در ثبات‌های آن نگه‌داری می‌شود.

10. یک ثبات عام‌منظوره[9]، یک ثبات قطعه[10]، یک ثبات وضعیت[11] و یک ثبات کنترلی[12] در معماری x86 را نام برده و وظیفه هر یک را به طور مختصر توضیح دهید.

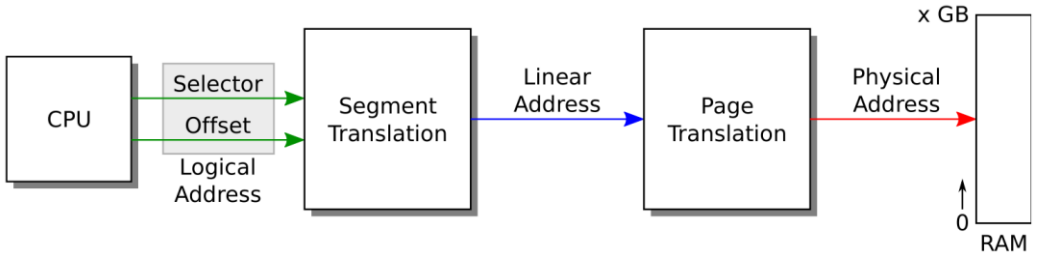
وضعیت ثبات‌ها را می‌توان به کمک gdb و دستور info registers مشاهده نمود. وضعیت برخی از ثبات‌های دیگر نیاز به دسترسی ممتاز[13] دارد. به این منظور می‌توان از qemu استفاده نمود. کافی است با زدن Ctrl + A و سپس C به ترمینال qemu رفته و دستور info registers را وارد نمود. با تکرار همان دکمه‌ها می‌توان به xv6 بازگشت.

11. پردازنده‌های x86 دارای مدهای مختلفی هستند. هنگام بوت، این پردازنده‌ها در مد حقیقی[14] قرار داده می‌شوند. مدی که سیستم‌عامل ام‌اس‌داس[15] (MS DOS) در آن اجرا می‌شد. چرا؟ یک نقص اصلی این مد را بیان نمایید؟
12. آدرس‌دهی به حافظه در این مد شامل دو بخش قطعه[16] و افست[17] بوده که اولی ضمنی و دومی به طور صریح تعیین می‌گردد. به طور مختصر توضیح دهید.

---

[5] Boot Sector
[6] Boot Loader
[7] Basic Input/Output System
[8] قالب DWARF، یک قالب متداول از فایل‌های دودویی است که به طور خاص اطلاعات اشکال‌زدایی را نگه‌داری می‌کند. در این‌جا در کنار قالب ELF استفاده شده است. دقت شود ELF، خود به تنهایی شامل اطلاعات اشکال‌زدایی نمی‌شود.
[9] General Purpose Register
[10] Segment Register
[11] Status Registers
[12] Control Registers
[13] Privileged Access
[14] Real Mode
[15] Microsoft Disk Operating System
[16] Segment
[17] Offset

در ابتدا qemu یک هسته را جهت اجرای کد بوت bootasm.S فعال می‌کند. فرایند بوت در بالاترین سطح دسترسی[۱۸] صورت می‌گیرد. به عبارت دیگر، بوت‌لودر امکان دسترسی به تمامی قابلیت‌های سیستم را دارد. در ادامه هسته به مد حفاظت‌شده[۱۹] تغییر مد می‌دهد (خط ۹۱۵۳). در مد حفاظت‌شده، آدرس مورد دسترسی در برنامه (آدرس منطقی) از طریق جداولی به آدرس فیزیکی حافظه[۲۰] نگاشت پیدا می‌کند. ساختار آدرس‌دهی در این مد در شکل زیر نشان داده شده است.



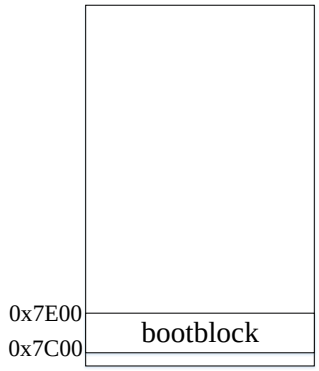
هر آدرس در کد برنامه یک آدرس منطقی[۲۱] است. این آدرس توسط سخت‌افزار مدیریت حافظه در نهایت به یک آدرس فیزیکی در حافظه نگاشت داده می‌شود. این نگاشت دو بخش دارد: ۱) ترجمه قطعه[۲۲] و ۲) ترجمه صفحه[۲۳]. مفهوم ثبات‌های قطعه در این مد تا حد زیادی با نقش آن‌ها در مد حقیقی متفاوت است. این ثبات‌ها با تعامل با جدولی تحت عنوان جدول توصیف‌گر سراسری[۲۴] (GDT) ترجمه قطعه را انجام می‌دهند. به این ترتیب ترجمه آدرس در مد محافظت‌شده بسیار متفاوت خواهد بود. در بسیاری از سیستم‌عامل‌ها از جمله xv6 و لینوکس ترجمه قطعه یک نگاشت همانی است. یعنی GDT به نحوی مقداردهی می‌گردد (خطوط ۹۱۸۲ تا ۹۱۸۵) که می‌توان از گزینش‌گر[۲۵] صرفنظر نموده و افست را به عنوان آدرس منطقی در نظر گرفت و این افست را دقیقاً به عنوان آدرس خطی[۲۶] نیز در نظر گرفت. به عبارت دیگر می‌توان فرض نمود که آدرس‌ها دوبخشی نبوده و صرفاً یک عدد هستند. یک آدرس برنامه (مثلاً آدرس یک اشاره‌گر یا آدرس قطعه‌ای از کد برنامه) یک آدرس منطقی (و همین‌طور در این‌جا یک آدرس خطی) است. به عنوان مثال در خط ۹۲۲۴ آدرس اشاره‌گر elf که به 0x10000 مقداردهی شده است یک آدرس منطقی است. به همین ترتیب آدرس تابع bootmain() که در زمان کامپایل تعیین می‌گردد نیز یک آدرس منطقی است. در ادامه بنابر دلایل تاریخی به آدرس‌هایی که در برنامه استفاده می‌شوند، آدرس مجازی[۲۷] اطلاق خواهد شد. نگاشت دوم یا ترجمه صفحه در کد بوت فعال نمی‌شود. لذا در این‌جا نیز نگاشت همانی وجود داشته و به این ترتیب آدرس مجازی برابر آدرس فیزیکی خواهد بود. نگاشت آدرس‌ها (و عدم استفاده مستقیم از آدرس فیزیکی) اهداف مهمی را دنبال می‌کند که در فصل مدیریت حافظه مطرح خواهد شد. از مهم‌ترین این اهداف، حفاظت محتوای حافظه برنامه‌های کاربردی مختلف از یکدیگر است. بدین ترتیب در لحظه تغییر مد، وضعیت حافظه (فیزیکی) سیستم به صورت شکل زیر است.

[18] سطوح دسترسی در ادامه پروژه توضیح داده خواهد شد.

[19] Protected Mode

[20] منظور از آدرس فیزیکی یک آدرس یکتا در سخت‌افزار حافظه است که پردازنده به آن دسترسی پیدا می‌کند.

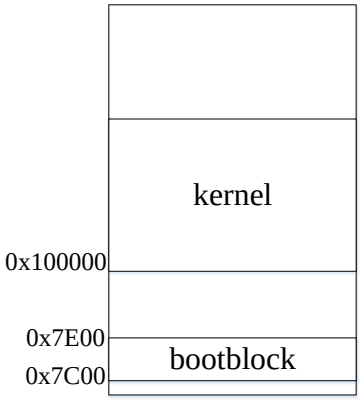
[21] Logical Address
[22] Segment Translation
[23] Page Translation
[24] Global Descriptor Table
[25] Selector
[26] Linear Address
[27] Virtual Address



Physical RAM

13. کد bootmain.c هسته را با شروع از سکتور بعد از سکتور بوت خوانده و در آدرس 0x100000 قرار می‌دهد.[۲۸] علت انتخاب این آدرس چیست؟

حالت حافظه پس از این فرایند به صورت شکل زیر است.



Physical RAM

به این ترتیب در انتهای بوت، کد هسته سیستم‌عامل به طور کامل در حافظه قرار گرفته است. در گام انتهایی، بوت‌لودر اجرا را به هسته واگذار می‌نماید. باید کد ورود به هسته اجرا گردد. این کد اسمبلی در فایل entry.S قرار داشته و نماد (بیانگر مکانی از کد) entry از آن فراخوانی می‌گردد. آدرس این نماد در هسته بوده و حدود 0x100000 است.

14. کد معادل entry.S در هسته لینوکس را بیابید.

اجرای هسته xv6

هدف از entry.S ورود به هسته و آماده‌سازی جهت اجرای کد C آن است. در شرایط کنونی نمی‌توان کد هسته را اجرا نمود. زیرا به گونه‌ای لینک شده است که آدرس‌های مجازی آن بزرگتر از 0x80100000 هستند. می‌توان این مسئله را با اجرای دستور cat kernel.sym بررسی نمود. در همین راستا نگاشت مربوط به صفحه‌بندی[۲۹] (ترجمه صفحه) از حالت همانی خارج خواهد شد. در صفحه‌بندی، هر کد در حال اجرا بر روی پردازنده، از جدولی برای نگاشت آدرس مورد استفاده‌اش به آدرس فیزیکی استفاده می‌کند.

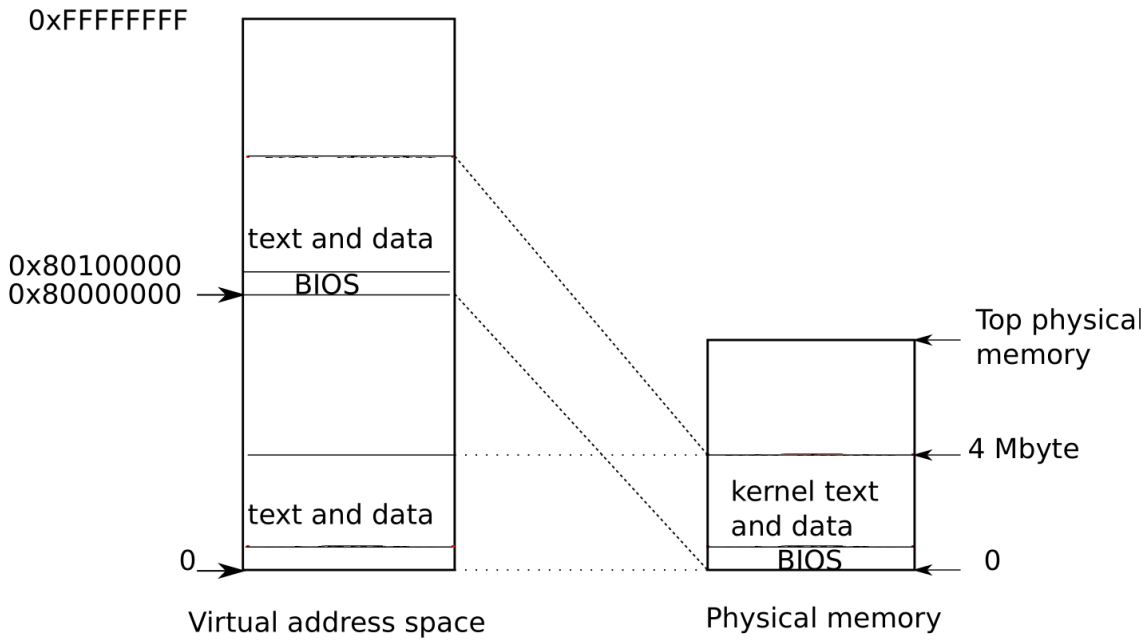
[28] دقت شود آدرس 0x10000 تنها برای خواندن هدر فایل elf استفاده شده است و محتوای فایل هسته در 0x100000 که توسط paddr (مخفف آدرس فیزیکی) تعیین شده است، کپی می‌شود. این آدرس در زمان لینک توسط kernel.ld تعیین شده و در فایل دودویی در قالب خاصی قرار داده شده است.

[29] Paging

این جدول خود در حافظه فیزیکی قرار داشته و یک آدرس فیزیکی مختص خود را دارد. در حین اجرا این آدرس در ثبات کنترلی cr3 بارگذاری شده[۳۰] و به این ترتیب پردازنده از محل جدول نگاشت‌های جاری اطلاع خواهد داشت.

15. چرا این آدرس فیزیکی است؟

جزئیات جدول نگاشت‌ها پیچیده است. به طور ساده این جدول دارای مدخل‌هایی است که تکه‌ای پیوسته از حافظه مجازی (یا خطی با توجه به خنثی شدن تأثیر آدرس منطقی) را به تکه‌ای پیوسته به همین اندازه از حافظه فیزیکی نگاشت می‌دهد. این اندازه‌ها در هر معماری، محدود هستند. به عنوان مثال در entry.S دو تکه پیوسته چهار مگابایتی از حافظه خطی به دو تکه پیوسته چهار مگابایتی از حافظه فیزیکی نگاشت داده شده است. هر تکه پیوسته یک صفحه[۳۱] نام دارد. یعنی حالت حافظه مطابق شکل زیر خواهد بود.



نیمه چپ شکل، فضای آدرس مجازی را نشان می‌دهد. جدول آدرس‌های نیمه چپ را به نیمه راست نگاشت می‌دهد. در این‌جا دو صفحه چهار مگابایتی به یک بخش چهار مگابایتی از حافظه فیزیکی نگاشت شده‌اند. یعنی برنامه می‌تواند با استفاده از دو آدرس به یک محتوا دسترسی یابد. این یکی دیگر از قابلیت‌های صفحه‌بندی است. در ادامه اجرا قرار است هسته تنها از بخش بالایی فضای آدرس مجازی استفاده نماید.[۳۲] به عبارت دیگر، نگاشت پایینی حذف خواهد شد. علت اصلی این است که باید حافظه مورد دسترسی توسط هسته از دسترسی برنامه‌های کاربردی یا به عبارت دقیق‌تر برنامه‌های سطح کاربر[۳۳] حفظ گردد. این یک شرط لازم برای ارائه سرویس امن به برنامه‌های سطح کاربر است. هر کد در حال اجرا دارای یک سطح دسترسی جاری[۳۴] (CPL) است. سطح دسترسی در پردازنده‌های x86 از صفر تا سه متغیر بوده که صفر و سه به ترتیب ممتازترین و پایین‌ترین سطح دسترسی هستند. در

[30] به طور دقیق‌تر این جداول سلسله‌مراتبی بوده و آدرس اولین لایه جدول در cr3 قرار داده می‌شود.

[31] Page

[32] در xv6 از آدرس 0x80000000 به بعد مربوط به سطح هسته و آدرس‌های 0x0 تا این آدرس مربوط به سطح کاربر هستند.

[33] User Level Programs

[34] Current Privilege Level

سیستم‌عامل xv6 اگر CPL=0 باشد در هسته و اگر CPL=3 باشد در سطح کاربر هستیم[۳۵]. تشخیص سطح دسترسی کد کنونی مستلزم خواندن مقدار ثبات cs است.[۳۶]

دسترسی به آدرس‌های هسته با CPL=3 نباید امکان‌پذیر باشد. به منظور حفاظت از حافظه هسته، در مدخل جدول نگاشت‌های صفحه‌بندی، بیت‌هایی وجود دارد که حافظه هسته را از حافظه برنامه سطح کاربر تفکیک می‌نماید (پرچم PTE_U (خط ۸۰۳) بیانگر حق دسترسی سطح کاربر به حافظه مجازی است). صفحه‌های بخش بالایی به هسته تخصیص داده شده و بیت مربوطه نیز این مسئله را تثبیت خواهد نمود. سپس توسط سازوکاری از دسترسی به مدخل‌هایی که مربوط به هسته هستند، زمانی که برنامه سطح کاربر این دسترسی را صورت می دهد، جلوگیری خواهد شد. در این‌جا اساس تفکر این است که هسته عنصر قابل اعتماد سیستم بوده و برنامه‌های سطح کاربر، پتانسیل مخرب بودن را دارند.

16. به این ترتیب، در انتهای entry.S، امکان اجرای کد C هسته فراهم می‌شود تا در انتها تابع main() صدا زده (خط ۱۰۶۵) شود. این تابع عملیات آماده‌سازی اجزای هسته را بر عهده دارد. در مورد هر تابع به طور مختصر توضیح دهید. تابع معادل در هسته لینوکس را بیابید.

در کد entry.S هدف این بود که حداقل امکانات لازم جهت اجرای کد اصلی هسته فراهم گردد. به همین علت، تنها بخشی از هسته نگاشت داده شد. لذا در تابع main() تابع kvmalloc() فراخوانی می‌گردد (خط ۱۲۲۰) تا آدرس‌های مجازی هسته به طور کامل نگاشت داده شوند. در این نگاشت جدید، اندازه هر تکه پیوسته، ۴ کیلوبایت است. آدرسی که باید در cr3 بارگذاری گردد، در متغیر kpgdir ذخیره شده است (خط ۱۸۴۲).

17. مختصری راجع به محتوای فضای آدرس مجازی هسته توضیح دهید.
18. علاوه بر صفحه‌بندی در حد ابتدایی از قطعه‌بندی به منظور حفاظت هسته استفاده خواهد شد. این عملیات توسط seginit() انجام می‌گردد. همان‌طور که ذکر شد، ترجمه قطعه تأثیری بر ترجمه آدرس منطقی نمی‌گذارد. زیرا تمامی قطعه‌ها اعم از کد و داده روی یکدیگر می‌افتند. با این حال برای کد و داده‌های سطح کاربر پرچم SEG_USER تنظیم شده است. چرا؟ (راهنمایی: علت مربوط به ماهیت دستورالعمل‌ها و نه آدرس است.)

اجرای نخستین برنامه سطح کاربر

تا به این لحظه از اجرا فضای آدرس حافظه هسته آماده شده است. بخش زیادی از مابقی تابع main()، زیرسیستم‌های مختلف هسته را فعال می‌نماید. مدیریت برنامه‌های سطح کاربر مستلزم ارائه انتزاعاتی برای ایجاد تمایز میان این برنامه‌ها و برنامه مدیریت آن‌ها است. کدی که تاکنون اجرا می‌شد را می‌توان برنامه مدیریت‌کننده سیستم و برنامه‌های سطح کاربر دانست.

19. جهت نگه‌داری اطلاعات مدیریتی برنامه‌های سطح کاربر ساختاری تحت عنوان struct proc (خط ۲۳۳۶) ارائه شده است. اجزای آن را توضیح داده و ساختار معادل آن در سیستم‌عامل لینوکس را بیابید.

از جمله اجزای ساختار proc متغیر pgdir است که آدرس جدول مربوط به هر برنامه سطح کاربر را نگه‌داری می‌کند. مشاهده می‌شود که این آدرس با آدرس مربوط به جدول کد مدیریت‌کننده سیستم که در kpgdir برای کل سیستم نگه‌داری شده بود، متفاوت است. تا پیش از فراخوانی userinit() (خط ۱۲۳۵) تقریباً تمامی زیرسیستم‌های هسته فعال شده‌اند. جهت ارائه واسطی با کاربر از طریق ترمینال و هم‌چنین آماده‌سازی بخش‌هایی از هسته که ممکن است توأم با به خواب رفتن کد باشد، تابع userinit()

[35] دو سطح دسترسی دیگر در اغلب سیستم‌عامل‌ها بلااستفاده است.

[36] در واقع در مد محافظت شده، دوبیت از این ثبات، سطح دسترسی کنونی را معین می‌کند. بیت‌های دیگر کاربردهای دیگری مانند تعیین افست مربوط به قطعه در gdt دارند.

فراخوانی می‌گردد. این تابع وظیفه ایجاد نخستین برنامه سطح کاربر را دارد. ابتدا توسط تابع allocproc() برای این برنامه یک ساختار proc تخصیص داده می‌شود (خط ۲۵۲۵). این تابع بخش‌هایی را که برنامه برای اجرا در سطح ممتاز (هسته) نیاز دارد، مقداردهی می‌کند. یکی از عملیات مهمی که در این تابع صورت می‌گیرد، مقداردهی p->context->eip به آدرس تابع forkret() است. این عمل منجر به این می‌شود که هنگام اجرای برنامه[۳۷] ابتدا forkret() اجرا گردد. آماده‌سازی بخش‌های باقی‌مانده سیستم در این تابع انجام می‌شود.

20. چرا به خواب رفتن در کد مدیریت‌کننده سیستم مشکل‌ساز است؟ (راهنمایی: به زمان‌بندی در ادامه توجه نمایید.)

در ادامه تابع userinit()، تابع setupkvm() فراخوانی شده و فضای آدرس مجازی هسته را برای برنامه سطح کاربر مقداردهی می‌کند.

21. تفاوت این فضای آدرس هسته با فضای آدرس هسته که توسط kvmalloc() در خط ۱۲۲۰ صورت گرفت چیست؟ چرا وضعیت به این شکل است؟

تابع inituvm() فضای آدرس مجازی سطح کاربر را برای این برنامه مقداردهی می‌نماید. به طوری که در آدرس صفر تا ۴ کیلوبایت، کد مربوط به initcode.S قرار گیرد.

22. تفاوت این فضای آدرس کاربر با فضای آدرس کاربر در کد مدیریت سیستم چیست؟

یک برنامه سطح کاربر می‌تواند برای دسترسی به سرویس‌های ممتاز سیستم به مد ممتاز (CPL=0) منتقل شود. به این ترتیب می‌تواند حتی به حافظه هسته نیز دسترسی داشته باشد. به منظور تغییر مد امن، سازوکارهایی مانند فراخوانی سیستمی[۳۸] وجود دارد. تفاوت در این سبک دسترسی این است که هسته آن را با یک سازوکار امن مدیریت می‌نماید. اجرای کد از فضای آدرس مجازی سطح کاربر به فضای آدرس مجازی هسته منتقل می‌شود. لذا باید وضعیت اجرای برنامه سطح کاربر در فضای آدرس مجازی سطح کاربر در مکانی ذخیره گردد. این مکان قاب تله[۳۹] نام داشته و در ساختار proc ذخیره می‌شود.[۴۰]

با توجه به این که اجرا در مد هسته است و جهت اجرای برنامه سطح کاربر باید به مد سطح کاربر منتقل شد، حالت سیستم به گونه‌ای شبیه‌سازی می‌شود که گویی برنامه سطح کاربر در حال اجرا بوده و تله‌ای رخ داده است. لذا فیلد مربوطه در proc باید مقداردهی شود. با توجه به این که قرار است کد به سطح کاربر بازگردد، بیت‌های مربوط به سطح دسترسی جاری ثبات‌های قطعه p->tf->cs و p->tf->ds به DPL_USER مقداردهی شده‌اند. p->tf->eip برابر صفر شده است (خط ۲۵۳۹). این بدان معنی است که زمانی که کد به سطح کاربر بازگشت، از آدرس مجازی صفر شروع به اجرا می‌کند. به عبارت دیگر اجرا از ابتدای کد initcode.S انجام خواهد شد. در انتها p->state به RUNNABLE مقداردهی می‌شود (خط ۲۵۵۰). این یعنی برنامه سطح کاربر قادر به اجرا است. حالت‌های ممکن دیگر یک برنامه در فصل زمان‌بندی بررسی خواهد شد.

در انتهای تابع main() تابع mpmain() فراخوانی شده (خط ۱۲۳۶) و به دنبال آن تابع scheduler() فراخوانی می‌شود (خط ۱۲۵۷). به طور ساده، وظیفه زمان‌بند تعیین شیوه اجرای برنامه‌ها بر روی پردازنده می‌باشد. زمان‌بند با بررسی لیست برنامه‌ها یک برنامه را که p->state آن RUNNABLE است بر اساس معیاری انتخاب نموده و آن را به عنوان کد جاری بر روی پردازنده اجرا می‌کند. این البته مستلزم تغییراتی در وضعیت جاری سیستم جهت قرارگیری حالت برنامه جدید (مثلاً تغییر cr3 برای اشاره به جدول نگاشت برنامه جدید) روی پردازنده است. این تغییرات در فصل زمان‌بندی تشریح می‌شود. با توجه به این که تنها برنامه قابل اجرا برنامه initcode.S است، پس از مهیا شدن حالت پردازنده و حافظه در اثر زمان‌بندی، این برنامه اجرا شده و به کمک یک

---

[37] دقت شود اجرا هنوز در کد مدیریت‌کننده سیستم است.

[38] System Call

[39] Trap Frame

[40] تله لزوماً هنگام انتقال از مد کاربر به هسته رخ نمی‌دهد.

فراخوانی سیستمی برنامه init.c را اجرا نموده که آن برنامه نیز در نهایت یک برنامه ترمینال (خط ۸۵۲۹) را ایجاد می‌کند. به این ترتیب امکان ارتباط با سیستم‌عامل را فراهم می‌آورد.

23. کدام بخش از آماده‌سازی سیستم، بین تمامی هسته‌های پردازنده مشترک و کدام بخش اختصاصی است؟ (از هر کدام یک مورد را با ذکر دلیل توضیح دهید.) زمان‌بند روی کدام هسته اجرا می‌شود؟
24. برنامه معادل initcode.S در هسته لینوکس چیست؟

نکات مهم

- برای تحویل پروژه ابتدا یک مخزن خصوصی در سایت GitLab ایجاد نموده و سپس پروژه خود را در 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 آپلود شده توسط گروه خود مسلط باشند و لزوماً نمره افراد یک گروه با یکدیگر برابر نیست.
- در صورت مشاهده هرگونه مشابهت بین کدها یا گزارش دو گروه، نمره ۰ به هر دو گروه تعلق می‌گیرد.
- تمامی سؤالات را در **کوتاه‌ترین اندازه ممکن** در گزارش خود پاسخ دهید.

به نام خدا

آزمایشگاه سیستم عامل

آشنایی، اجرا و اشکال‌زدایی هسته سیستم عامل xv6

(بخش سوم: اشکال‌زدایی)

مقدمه

کد در حین اجرا ممکن است دارای اشکال باشد. یکی از ابزارهای تسریع اشکال‌زدایی، اشکال‌زداها هستند. در همین راستا، اشکال‌زداهای متعددی برای سیستم‌عامل‌ها و معماری‌های مختلف ارائه شده است. ابزار اشکال‌زدای گنو[1] (GDB)، یک اشکال‌زدای متداول در سیستم‌های یونیکسی بوده که در این بخش از آزمایش روش کار با آن به طور اجمالی بررسی خواهد شد. به عنوان مثال ممکن است یک کد بزرگ، هنگام اجرا دچار خرابی شود. استفاده از GDB می‌تواند دستور منجر به ازکارافتادگی و زنجیره فراخوانی توابع منتهی به این دستور را نمایش دهد. جهت اشکال زدایی با GDB، در گام نخست باید سیستم‌عامل به صورتی بوت شود که قابلیت اتصال اشکال‌زدا به آن وجود داشته باشد. مراحل اتصال عبارت است از:

۱) در یک ترمینال دستور make qemu-gdb اجرا گردد.

۲) سپس در ترمینالی دیگر، فایل کد اجرایی به عنوان ورودی به GDB داده شود.

چنان‌چه در بخش دوم این آزمایش ذکر شد کد اجرایی شامل یک نیمه هسته و یک نیمه سطح کاربر بوده که نیمه هسته ثابت و نیمه سطح کاربر بسته به برنامه در حال اجرا بر روی پردازنده دائماً در حال تغییر است. به این ترتیب، به عنوان مثال، هنگام اجرای برنامه cat، کدهای اجرایی سیستم شامل کد هسته و کد برنامه cat خواهند بود. جهت اشکال‌زدایی بخش سطح کاربر، کافی است دستور gdb _cat و جهت اشکال‌زدایی بخش هسته دستور gdb kernel فراخوانی شود. دقت شود در هر دو حالت، هر دو کد سطح هسته و کاربر اجرا می‌شوند. اما اشکال‌زدا فقط روی یک کد اجرایی (سطح کاربر یا هسته) کنترل داشته و تنها قادر به انجام عملیات بر روی آن قسمت خواهد بود.

۳) نهایتاً با وارد کردن دستور target remote tcp::26000 در GDB، اتصال به سیستم‌عامل صورت خواهد گرفت.

با وارد کردن دستور c کد سیستم به اجرا ادامه خواهد داد. هم‌چنین با فشردن دکمه Ctrl + C می‌توان به اشکال‌زدا بازگشت.

اجرای اولیه اشکال‌زدا

برنامه cat، فایل ورودی را از دیسک خوانده و سپس محتوای آن را در ترمینال نمایش می‌دهد. خواندن از دیسک مستلزم انتقال از سطح کاربر به سطح هسته است. زیرا دسترسی به دستگاه‌های ورودی/خروجی، یک عملیات ممتاز است. بدین منظور یک تابع به نام read() در سطح کاربر وجود داشته که خود منجر به فراخوانی یک تابع در سطح هسته موسوم به sys_read() می‌گردد. چنان‌چه در پروژه بعدی آزمایشگاه خواهید دید، تابع دوم فراخوانی سیستمی نام داشته و یکی از فراخوانی‌های سیستمی پیاده‌سازی شده در هسته سیستم‌عامل است. در شکل زیر بخشی از برنامه cat که مربوط به خواندن از دیسک است نشان داده شده است.

[1] GNU Debugger

```
 7 void
 8 cat(int fd)
 9 {
10   int n;
11
12   while((n = read(fd, buf, sizeof(buf))) > 0) {
13     if (write(1, buf, n) != n) {
14       printf(1, "cat: write error\n");
15       exit();
16     }
17   }
18   if(n < 0){
19     printf(1, "cat: read error\n");
20     exit();
21   }
22 }
```

چنان‌چه مشاهده می‌شود در خط ۱۲، فایل محتوای فایل خوانده شده و در ترمینال چاپ می‌گردد. فراخوانی سیستمی sys_read() نیز که در فایل sysfile.c پیاده‌سازی شده است در شکل زیر نشان داده شده است.

```
69 int
70 sys_read(void)
71 {
72   struct file *f;
73   int n;
74   char *p;
75
76   if(argfd(0, 0, &f) < 0 || argint(2, &n) < 0 || argptr(1, &p, n) < 0)
77     return -1;
78   return fileread(f, p, n);
79 }
```

یک سازوکار مهم در اشکال‌زدایی نرم‌افزار نقطه توقف[2] است. این ابزار برای توقف اجرای برنامه در یک نقطه از پیش تعیین شده، طراحی شده است. این نقطه، آدرسی از کد برنامه می‌باشد. این آدرس را می‌توان بر اساس نام تابع، شماره خط در فایل کد منبع یا مقدار عددی آن به اشکال‌زدا اطلاع داد. به عنوان مثال اگر به دلایلی قصد اشکال زدایی در تابع read() یا بخش سطح هسته آن یعنی sys_read() وجود داشته باشد، می‌توان با استفاده از دستورهای مختلفی که در جدول زیر نشان داده شده است، عمل نمود.

	تابع سطح کاربر read()	فراخوانی سیستمی sys_read()
نام تابع	3b (cat.c:)cat	b (sysfile.c:)sys_read
مکان در کد منبع	b cat.c:12	b sysfile.c:71
مکان در حافظه	b *0x98	b *0x80104b10

۱) یک Breakpoint روی فراخوانی سیستمی sys_read() قرار داده[4] و دستور bt را اجرا نمایید. در مورد خروجی این دستور و رابطه اجزای آن توضیح دهید.

---

[2] Breakpoint

[3] دقت شود در این شیوه تعیین Breakpoint، تنها می‌توان ابتدای تابع دربرگیرنده نقطه اشکال‌زدایی را تعیین نمود. لذا Breakpoint روی خط ۹ (و نه خط ۱۲) قرار می‌گیرد. همچنین بخش داخل پرانتز، اختیاری است.

[4] دقت شود با توجه به این که sys_read() در هسته است، باید نقطه فراخوانی آن در سطح کاربر را متوقف نمایید. این نقطه در فایل usys.S قرار دارد. دقت نمایید read() فراخوانی سیستمی نیست. بلکه یک تابع کتابخانه‌ای است که فراخوانی سیستمی را فراخوانی می‌کند.

Breakpoint را می‌توان پیش از اجرای برنامه یا در زمان توقف اجرا (به سبب توقف روی یک Breakpoint قبلی یا توقف ناشی از Ctrl + C) قرار داد.

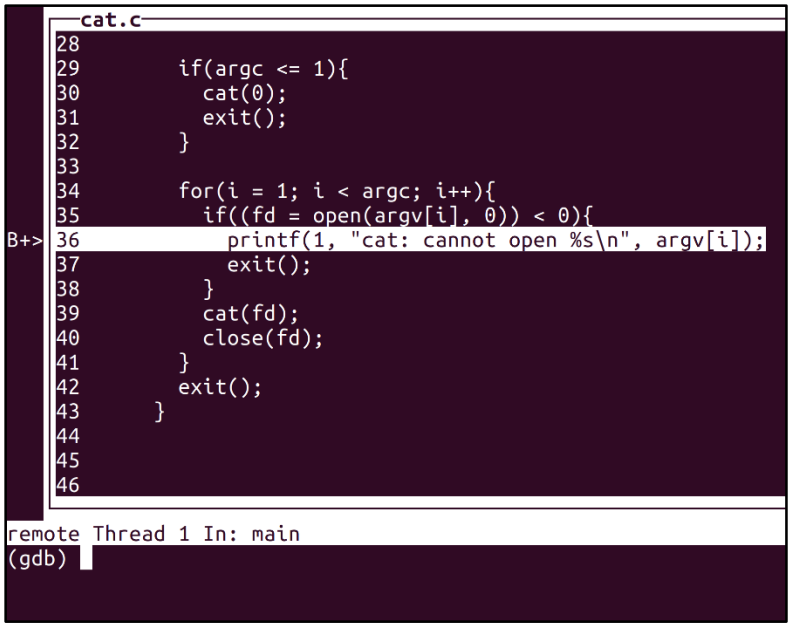
۲) یک حلقه بی‌نهایت در فراخوانی سیستمی sys_read() ایجاد نموده و نقطه اشکال در سطح هسته و کاربر را برای برنامه cat توسط GDB نشان دهید.

پس از توقف روی Breakpoint می‌توان با اجرای دستورهای n(ext)، s(tep) و fin(ish) به ترتیب به دستور بعدی، به درون دستور بعدی (اگر فراخوانی تابع باشد) و به خارج از تابع کنونی (یعنی بازگشت به تابع فراخواننده) منتقل شد. به عبارت دیگر، اجرا گام‌به‌گام قابل بررسی است. بدین معنی که پیش از اجرای خط جاری برنامه سطح کاربر یا هسته، امکان دستیابی به اطلاعات متغیرها و ثبات‌ها فراهم می‌باشد. به این ترتیب می‌توان برنامه را از جهت وجود حالات نادرست، بررسی نمود.

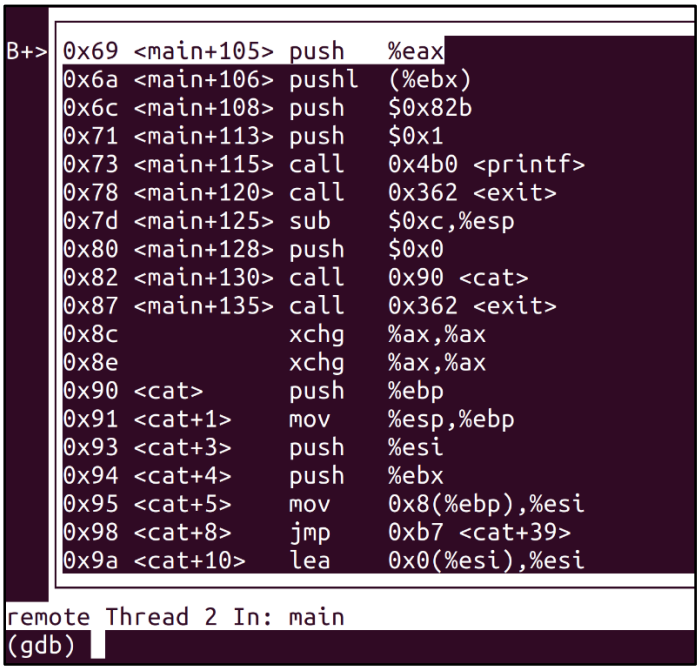
۳) با استفاده از دستور objdump آدرس توابع kernel و برنامه cat_ را استخراج نموده و بازه آن‌ها را بررسی کنید. این آدرس‌ها را گزارش نمایید (فقط توابع). چه نتیجه‌ای می‌گیرید؟

آشنایی با قابلیت‌های سطح پایین‌تر

اشکال‌زدایی برنامه در سطوح مختلفی قابل انجام است. با توجه به این که ممکن است بخشی از کد سطح بالا به دنبال بهینه‌سازی‌های کامپایلری اجرا نشده یا ترتیب اجرای آن تغییر کند، نیاز به اشکال‌زدایی در سطح کد اسمبلی خواهد بود. زیرا کدی که در واقع اجرا می‌گردد، همین کد است. ضمن این که ممکن است برخی از فایل‌های کد منبع، خود به دلایلی از جمله بهینه‌سازی یا عدم پشتیبانی زبان برنامه‌نویسی به صورت کد اسمبلی نوشته شده باشند. جهت آشنایی با این قابلیت، ابتدا یک Breakpoint در خط ۳۶ برنامه cat قرار دهید. سپس با وارد کردن ورودی مناسب، این خط را اجرا نمایید. جهت سهولت در مشاهده روند اجرا دستور layout src را در اشکال‌زدا اجرا نمایید. مشاهده خواهید نمود که مطابق شکل زیر، خط ۳۶ برنامه، علامت‌گذاری شده است.



در این حالت، اجرای دستور layout asm کد اسمبلی همین بخش از برنامه را نشان می‌دهد که در شکل زیر قابل مشاهده است.



مشاهده می‌کنید چهار دستور push، پیش از فراخوانی printf (توسط دستور call) وجود دارد که وظیفه قرار دادن پارامترها و مقادیر برخی ثبات‌ها در پشته را بر عهده دارند. جهت انتقال به دستور بعدی در سطح کد اسمبلی نمی‌توان از دستور s یا n استفاده کرد. زیرا تمامی این خطوط کد اسمبلی، مربوط به یک خط از کد سی (خط ۳۶) هستند.

۴) دستورهای معادل s و n در سطح کد اسمبلی را نام ببرید.

۵) برنامه cat محتوای یک فایل را در ترمینال می‌نویسد. برای نوشتن در ورودی/خروجی نیاز به دسترسی به سرویس‌های هسته سیستم‌عامل است. لذا فراخوانی سیستمی sys_write رخ خواهد داد. یک Breakpoint روی آن قرار دهید. سپس دستور bt را وارد نمایید. خروجی آن چیست؟ توضیح دهید.

۶) در مورد خروجی layout src و layout asm توضیح دهید. (راهنمایی: دستور نخست خروجی layout asm، شماره فراخوانی سیستمی sys_write را در ثبات eax قرار می‌دهد. در واقع طبق واسط باینری برنامه‌های کاربردی[5] (ABI)، این ثبات، باید بدین صورت مقداردهی شود. دو دستور بعدی چه نقشی دارند؟)

جهت انتقال به مد عادی اشکال‌زدایی ابتدا باید کلیدهای Ctrl + x و سپس کلید c فشار داد.

اشکال‌زدایی بر اساس داده

تا اینجا اشکال‌زدایی بر اساس قرار دادن نقاط توقف روی آدرس‌های برنامه بررسی شد. یک روش متفاوت اشکال‌زدایی، اشکال‌زدایی بر اساس داده‌ها بوده که به کمک قطعه کد زیر در ابتدای فایل فرضی foo.c قابل بیان است.

```
int i;
for (i = 0; i < 1000000; i++)
     x = rand();
```

[5] Application Binary Interface

اگر فرض شود مقدار ۱- برای x یک مقدار نامعتبر است، چگونه می‌توان وقوع این حالت را به کمک اشکال‌زدا تشخیص داد؟ البته یک روش، تغییر در کد و افزودن شرطی جهت تشخیص این حالت است. اما تشخیص بدون تغییر در کد توسط دستور زیر میسر می‌شود.

```
b foo.c:2 if (x = -1)
```

اما اگر متغیر x، یک متغیر سراسری بوده و در چندین فایل کد منبع مورد دسترسی قرار بگیرد، چه باید کرد؟ در این حالت تشخیص با تغییر در کد نیز ساده نخواهد بود. در شرایط پیچیده‌تر، ممکن است این دسترسی‌ها به طور غیرمستقیم و توسط اشاره‌گرها رخ دهد. در این حالت بهتر است به جای علامت‌گذاری خط کد (توسط Breakpoint) داده‌ها را علامت‌گذاری نمود. بدین‌منظور watchها طراحی شده‌اند که خود چندین نوع دارند. ساده‌ترین آن‌ها با استفاده از دستور زیر قابل تعریف است.

```
watch *0x12345678
```

در این‌جا نوشتن در این آدرس حافظه منجر به توقف اجرا در نقطه دسترسی می‌گردد. می‌توان watch را روی متغیرهای محلی هم گذاشت. البته باید به حوزه تعریف متغیرها نیز دقت نمود. دقت کنید موارد استفاده watch تنها به نکات مذکور در بالا ختم نمی‌شود. در کدهای همروند یا موازی اشکال‌زدایی بسیار دشوارتر از کدهای ترتیبی است. اگر دسترسی به متغیری به صورت همروند یا موازی صورت پذیرد و مثلاً متغیری مقدار نادرستی بگیرد، استفاده از watch جهت تشخیص حالت اشکال و عامل آن بسیار راهگشا خواهد بود. به‌خصوص که در این شرایط، تکرارپذیری آزمایش نیز به سادگی ممکن نبوده و بسیاری از اجراها منجر به خروجی صحیح می‌گردند.

نکات مهم

- برای تحویل پروژه ابتدا یک مخزن خصوصی در سایت GitLab ایجاد نموده و سپس پروژه خود را در 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 پروژه‌ی آپلود شده توسط گروه خود مسلط باشند و لزوماً نمره افراد یک گروه با یکدیگر برابر نیست.
- در صورت مشاهده هرگونه مشابهت بین کدها یا گزارش دو گروه، نمره ۰ به هر دو گروه تعلق می‌گیرد.
- تمامی سؤالات را در **کوتاه‌ترین اندازه ممکن** در گزارش خود پاسخ دهید.