به نام خدا

آزمایشگاه سیستم‌عامل

پروژه اول آزمایشگاه سیستم‌عامل

(آشنایی با هسته سیستم‌عامل xv6)

مقدمه

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

آشنایی با سیستم‌عامل xv6

کدهای مربوط به سیستم‌عامل xv6 از لینک زیر قابل دسترسی است:

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

همچنین مستندات این سیستم‌عامل و فایل شامل کدهای آن نیز در صفحه درس بارگذاری شده است. برای این پروژه، نیاز است که فصل‌های ۰ و ۱ از مستندات فوق را مطالعه کرده و به برخی سؤالات منتخب پاسخ دهید. پاسخ این سؤالات را در قالب یک گزارش بارگذاری خواهید کرد.

۱. معماری سیستم‌عامل xv6 چیست؟ چه دلایلی در دفاع از نظر خود دارید؟
۲. یک پردازه[۱،۲] در سیستم‌عامل xv6 از چه بخش‌هایی تشکیل شده است؟ این سیستم‌عامل به طور کلی چگونه پردازنده را به پردازه‌های مختلف اختصاص می‌دهد؟

اجرا و اشکال‌زدایی

در این بخش به اجرای سیستم عامل xv6 خواهیم پرداخت. علی‌رغم اینکه این سیستم‌عامل قابل اجرای مستقیم بر روی سخت افزار است، به دلیل آسیب‌پذیری بالا و رعایت مسائل ایمنی از این کار اجتناب نموده و سیستم عامل را به کمک برابرساز[۳] Qemu روی سیستم‌عامل لینوکس اجرا می‌کنیم. برای این منظور لازم است که کدهای مربوط به سیستم‌عامل را از لینک ارائه شده clone و یا دانلود کنیم. در ادامه با اجرای دستور make در پوشه دانلود، سیستم‌عامل کامپایل می‌شود. در نهایت با اجرای دستور make qemu سیستم‌عامل بر روی برابرساز اجرا می‌شود (توجه شود که فرض شده Qemu از قبل بر روی سیستم‌عامل شما نصب بوده است. در غیر این صورت ابتدا آن را نصب نمایید).

اضافه کردن یک متن به Message Boot

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

در این قسمت می‌خواهیم چند قابلیت کاربردی به کنسول xv6 اضافه کنیم.

پس از اجرای سیستم عامل بر روی Qemu، در صورت استفاده از کلیدهای Ctrl+O، Ctrl+T و Ctrl+A ، معادل کاراکتری آن‌ها، در کنسول چاپ می‌شود.

---

[۱] Process

[۲] به طور ساده، پردازه، برنامه سطح کاربر در حال اجرا است. از این به بعد در متن، هر یک ممکن است به جای دیگری استفاده شود.

[۳] Emulator

کد xv6 را به نحوی تغییر دهید تا قابلیت‌های زیر در آن پیاده‌سازی شده باشد:

۱. اگر کاربر دستور Ctrl+T را وارد کرد، دو حرف آخر پیش از نشانه‌گر باید جابه‌جا[۱] شوند.
۲. اگر کاربر دستور Ctrl+O را وارد کرد، باید تمام حروفی که پس از نشانه‌گر هستند (تا پیش از اولین Space یا انتهای خط)، به حالت Uppercase دربیایند (در صورتی که خود Uppercase باشند باید همان‌گونه باقی‌بمانند و در صورتی که کاراکتری جز حروف الفبا باشند نباید تغییر کنند).
۳. اگر کاربر دستور Ctrl+A را وارد کرد، نشانه‌گر باید به ابتدای خط تغییر مکان دهد و در صورتی که پس از این کاراکتری وارد شود، به ابتدای خط اضافه شود.

توجه شود که علاوه بر نمایش درست بر روی کنسول، باید دستورات نوشته شده با کلیدهای ترکیبی فوق، قابلیت اجرای درست را نیز داشته باشند.

اجرا و پیاده‌سازی یک برنامه سطح کاربر

در این قسمت شما باید یک برنامه سطح کاربر و به زبان C بنویسید و به برنامه‌های سطح کاربر سیستم‌عامل اضافه کنید. نام این برنامه factor می‌باشد. این برنامه یک عدد از ورودی دریافت نموده و تمام مقسوم علیه‌های آن محاسبه می‌کند. در نهایت، خروجی محاسبه را در یک فایل متنی با نام factor_result.txt ذخیره می‌کند. اگر فایل متنی از قبل موجود باشد، جواب بر روی آن بازنویسی می‌شود.

```
$ factor 20
$ cat factor_result.txt
1 2 4 5 10 20
```

از دستورات open، read، write و close استفاده کنید که برای باز کردن، خواندن، نوشتن و بستن فایل‌ها استفاده می‌شود. برای پیاده‌سازی این برنامه سطح کاربر، علاوه بر نوشتن کد، باید در فایل Makefile نیز تغییرات لازم را بوجود آورید تا این برنامه مثل دستورات دیگر از قبیل ls اجرا شود.

[۱] Swap

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.

۳. سه وظیفه اصلی سیستم‌عامل را نام ببرید.

۴. 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[۱] 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 Unix 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[۲] و متغیرها[۳] 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.

۵. دستور make -n را اجرا نمایید. کدام دستور، فایل نهایی هسته را می‌سازد؟

۶. 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.

۷. 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

ارائه سرویس به کاربر سیستم، مستلزم آمادگی اولیه سیستم‌عامل و پوسته[۴] است. به عبارت دیگر، باید زیرسیستم‌های سیستم‌عامل، فعال شده و پوسته نیز قادر به دریافت دستورها و اجرای برنامه‌های مختلف سطح کاربر باشد. در این راستا، در ابتدای اجرای سیستم، سیستم ورودی/خروجی مقدماتی[۵] (BIOS) کنترل را در دست می‌گیرد. در این لحظه هنوز سیستم‌عامل در حافظه بارگذاری نشده است. لذا BIOS، کدی تحت‌عنوان بوت‌لودر[۶] را اجرا نموده و این کد پس از قرار دادن پردازنده در مد مناسب، هسته را بارگذاری می‌نماید. هسته نیز زیرسیستم‌های خود از جمله زیرسیستم مدیریت حافظه را فعال نموده (موضوع آزمایش پنجم) و در نهایت، پوسته را در قالب یک برنامه سطح کاربر اجرا می‌کند. به طور کلی پس از این، اجرا در سطح کاربر و از طریق پوسته بوده و در صورت لزوم، مثلاً اجرای فراخوانی سیستمی

[۱] Header Files
[2] Rules
[3] Variables
[۴] Shell
[5] Basic Input/Output System
[6] Boot Loader

یا وقوع وقفه انتقال به هسته صورت می‌پذیرد (موضوع آزمایش دوم). سرویس‌های سیستم‌عامل، در راستای تحقق وظایف اصلی آن مانند اجرای ایزوله پردازه‌ها و حفاظت داده‌های سیستمی از پردازه‌های سطح کاربر ارائه می‌گردد.[۱] در ادامه، مراحل بوت تا اجرای پوسته تشریح می‌گردد.

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[۲] است. در این‌جا هر سکتور ۵۱۲ بایت است. اگر دیسک قابل بوت باشد، نخستین سکتور آن سکتور بوت[۳] نام داشته و شامل بوت‌لودر خواهد بود. بوت‌لودر کدی است که سیستم‌عامل را در حافظه بارگذاری می‌کند. یکی از روش‌های راه‌اندازی اولیه رایانه، بوت مبتنی بر 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.

۸. در xv6 در سکتور نخست دیسک قابل بوت، محتوای چه فایلی قرار دارد. (راهنمایی: خروجی دستور make -n را بررسی نمایید.)

۹. 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.)

۱۰. علت استفاده از دستور objcopy در حین اجرای عملیات make چیست؟

۱۱. 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.

۱۲. یک ثبات عام‌منظوره[۴]، یک ثبات قطعه[۵]، یک ثبات وضعیت[۶] و یک ثبات کنترلی[۷] 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[۸] 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.

۱۳. پردازنده‌های x86 دارای مدهای مختلفی هستند. هنگام بوت، این پردازنده‌ها در مد حقیقی[۹] قرار داده می‌شوند. مدی که سیستم‌عامل اماس‌داس[۱۰] (MS DOS) در آن اجرا می‌شد. چرا؟ یک نقص اصلی این مد را بیان نمایید؟

۱۴. مدهای مختلف در پردازنده‌های x86 را مختصراً توضیح دهید.

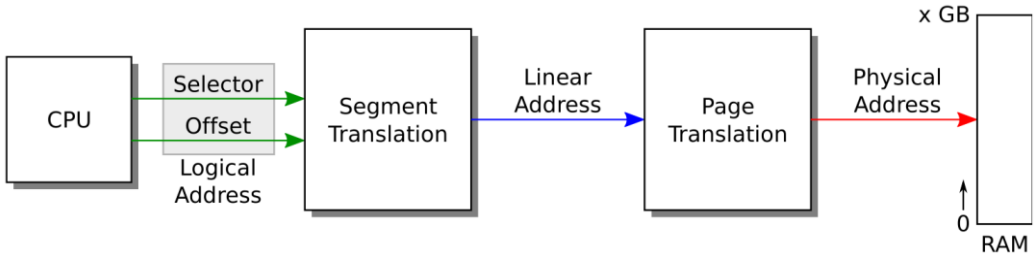
---

[۱] ممکن است در برخی سیستم‌عامل‌های خاص به کاربر آزادی عمل بیشتری داده شده و برخی سخت‌گیری‌ها لحاظ نشود. مثلاً ممکن است برنامه سطح کاربر نیز در مد ممتاز اجرا شده و بالاترین سطح دسترسی را دارا باشد. این سیستم‌های خاص‌منظوره، مورد بحث این درس نیستند.

[2] Sector
[3] Boot Sector
[4] General Purpose Register
[5] Segment Register
[6] Status Registers
[7] Control Registers
[8] Privileged Access
[9] Real Mode
[10] Microsoft Disk Operating System

۱۵. آدرس‌دهی به حافظه در این مد شامل دو بخش قطعه[۱] و افست[۲] بوده که اولی ضمنی و دومی به طور صریح تعیین می‌گردد. به طور مختصر توضیح دهید.

در ابتدا qemu یک هسته را جهت اجرای کد بوت bootasm.S فعال می‌کند. فرایند بوت در بالاترین سطح دسترسی[۳] صورت می‌گیرد. به عبارت دیگر، بوت‌لودر امکان دسترسی به تمامی قابلیت‌های سیستم را دارد. در ادامه هسته به مد حفاظت‌شده[۴] تغییر مد می‌دهد (خط ۹۱۵۳). در مد حفاظت‌شده، آدرس مورد دسترسی در برنامه (آدرس منطقی) از طریق جداولی به آدرس فیزیکی حافظه[۵] نگاشت پیدا می‌کند. ساختار آدرس‌دهی در این مد در شکل زیر نشان داده شده است.



هر آدرس در حین اجرا (شامل آدرس‌های داده‌ها و آدرس خطوط کد برنامه)، یک آدرس منطقی[۶] 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**[۷] **و ۲) ترجمه صفحه**[۸]. 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[۹] (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[۱۰] صرفنظر نموده و افست را به عنوان آدرس منطقی در نظر گرفت و این افست را دقیقاً به عنوان آدرس خطی[۱۱] 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[۱۲] اطلاق خواهد شد. نگاشت دوم یا ترجمه صفحه در کد بوت استفاده نشده است. یعنی این نگاشت نیز نگاشت همانی بوده و به این ترتیب آدرس مجازی برابر آدرس فیزیکی خواهد بود. البته نگاشت دوم (ترجمه صفحه) برخلاف نگاشت اول (ترجمه قطعه)، در مراحل بعدی بوت، توسط سیستم‌عامل از حالت همانی در خواهد آمد. 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.

---

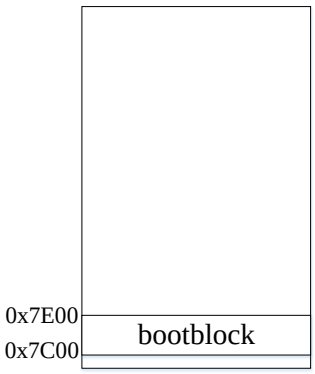
[1] Segment
[2] Offset

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

[4] Protected Mode

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

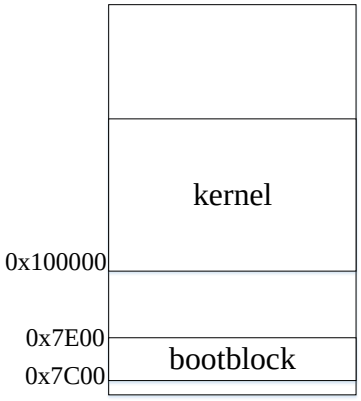
[6] Logical Addresss
[7] Segment Translation
[8] Page Translation
[9] Global Descriptor Table
[10] Selector
[11] Linear Address
[12] Virtual Address



Physical RAM

۱۶. کد bootmain.c هسته را با شروع از سکتور بعد از سکتور بوت خوانده و در آدرس 0x100000 قرار می‌دهد.[۱] 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.

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

۱۸. چرا از همان ابتدا هسته در حافظه توسط BIOS بارگذاری نمی‌شود؟ علت اصلی را بیان نمایید.

اجرای هسته xv6

هدف از اجرای entry.S ورود به هسته و آماده‌سازی شرایط، 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.[۲] می‌توان این مسئله را با اجرای دستور cat kernel.sym بررسی نمود. در همین راستا نگاشت مربوط به صفحه‌بندی[۳] (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. این

[۱] 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.

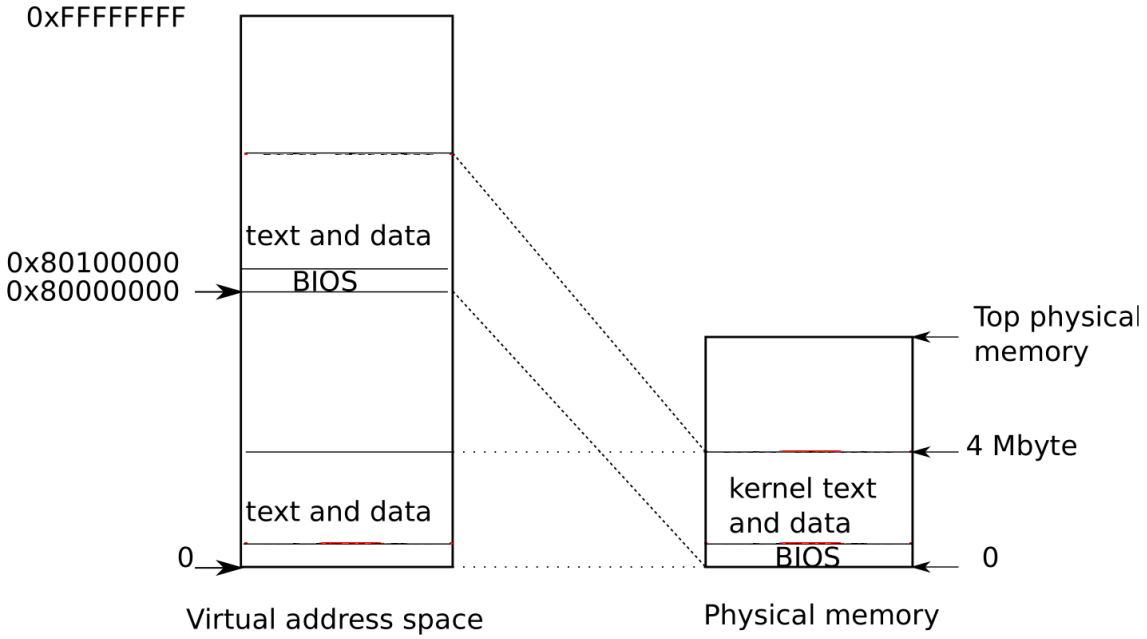
[۲] محتوای هسته در آدرس‌های فیزیکی متناظر (طبق نگاشت همانی)، یعنی بالاتر از 0x80100000 قرار ندارد.

[3] Paging

جدول، خود در حافظه فیزیکی قرار داشته و یک آدرس فیزیکی مختص خود را دارد. در حین اجرا این آدرس در ثبات کنترلی cr۳ بارگذاری شده[۱] و به این ترتیب، پردازنده از محل جدول نگاشت‌های جاری اطلاع خواهد داشت.

۱۹. 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[۲] نام دارد. یعنی حالت حافظه مجازی و فیزیکی 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 کد 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.[۳] به عبارت دیگر، نگاشت پایینی حذف خواهد شد. علت اصلی این است که نیمه پایین آدرس‌ها به برنامه‌های سطح کاربر اختصاص داده خواهد شد و نباید به داده‌های هسته دسترسی یابد. باید داده‌های هسته از دسترسی توسط برنامه‌های سطح کاربر[۴] 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[۵] (CPL) است. سطح دسترسی در پردازنده‌های xv6 از صفر تا سه متغیر بوده که صفر و سه به ترتیب ممتازترین و پایین‌ترین سطح دسترسی هستند. در سیستم‌عامل xv6 اگر CPL = 0 باشد در هسته و اگر CPL = 3 باشد در سطح کاربر هستیم.[۶] تشخیص سطح دسترسی کد کنونی، مستلزم خواندن مقدار ثبات cs است.[۷]

بدین ترتیب، دسترسی به آدرس‌های هسته نباید با CPL = 3 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

---

[۱] به طور دقیق‌تر این جداول سلسله‌مراتبی بوده و آدرس اولین لایه جدول در cr۳ قرار داده می‌شود.

[2] Page

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

[4] User Level Programs

[۵] Current Privilege Level

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

[۷] 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.

این مسئله را تثبیت خواهد نمود. سپس سازوکاری سخت‌افزاری، از دسترسی به مدخل‌هایی که مربوط به هسته هستند، زمانی که برنامه سطح کاربر این دسترسی را صورت می دهد، جلوگیری به عمل می‌آورد. یعنی اگر CPL = 3 باشد، دسترسی به مدخل‌های صفحه نیمه بالایی (PTE_U آن‌ها ست نشده است) جدول صفحه منجر به خطا خواهد شد. در این‌جا اساس تفکر این است که هسته، عنصر قابل اعتماد سیستم بوده و برنامه‌های سطح کاربر، پتانسیل مخرب بودن را دارند.

۲۰. 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 (یک اندازه مجاز دیگر برای x86) است. آدرسی که باید در cr3 بارگذاری گردد، در متغیر kpgdir ذخیره شده است (خط ۱۸۴۲).

۲۱. مختصری راجع به محتوای فضای آدرس مجازی هسته توضیح دهید.

۲۲. 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 (پیش از فراخوانی userinit() (خط ۱۲۳۵)) فضای آدرس حافظه هسته آماده شده است. بخش زیادی از مابقی تابع main()، زیرسیستم‌های مختلف هسته را فعال می‌نماید. گام‌های انتهایی، فراخوانی **کدهای آماده‌سازی که ممکن است به خواب بروند** و همچنین **اجرای پوسته** است. کدی که تاکنون اجرا می‌شد را می‌توان برنامه مدیریت‌کننده سیستم و برنامه‌های سطح کاربر دانست.[۱] برنامه‌های سطح کاربر مانند پوسته، به دلایلی از جمله افزایش بهره‌وری سخت‌افزار سیستم، زمان‌بندی می‌شوند (موضوع آزمایش سوم). یعنی زیرسیستمی از هسته، مسئول تعیین زمان اجرای کد آن‌ها روی پردازنده خواهد بود. کد مدیریت‌کننده سیستم برخلاف پردازه‌ها زمان‌بندی نمی‌گردد. به طور کلی اجرای پردازه‌ها در سطح کاربر و اجرای ریسه هسته[۲] (کد سرویس‌دهنده درخواست‌های سیستمی) متناظر با آن‌ها در سطح هسته صورت می‌گیرد. در زمان‌های خاصی، زمان‌بند، فعال شده، کنترل به هسته، منتقل شده و ریسه هسته و به دنبال آن خود پردازه‌ها را زمان‌بندی می‌نماید (مداخل نیمه پایینی جدول صفحه هر پردازه، مربوط به دسترسی‌های حافظه حین اجرای در سطح کاربر و مداخل نیمه بالایی، مربوط به اجرای ریسه هسته است).

برای اجرای کدهای آماده‌سازی به‌خواب‌رونده و اجرای پوسته باید پردازه‌ای با تمامی داده‌ساختارهای مورد نیازش ایجاد نموده و آن را به زمان‌بند داده تا بر اساس الگوریتم زمان‌بندی، انتخاب و اجرا شود. حافظه پردازه و ریسه هسته آن باید ایجاد و به درستی مقداردهی شود. زمان‌بند باید بداند ادامه اجرای سطح هسته پردازه (ریسه هسته مربوطه) از کجا رخ می‌دهد. ضمن این که هنگام ادامه اجرا در سطح کاربر باید نقطه ادامه اجرا از جایی به دست آید. کلیت اجرا تا اجرای پوسته بدین صورت خواهد بود که ابتدا یک پردازه، ایجاد شده و این طور **وانمود می‌شود** که در حال اجرا بوده و به دلیلی اجرای آن متوقف شده و به هسته رفته است. زمان‌بند، آن پردازه که تنها پردازه موجود برای اجرا است را برای اجرا انتخاب می‌کند. با توجه به این که زمان‌بندی در هسته رخ می‌دهد، ابتدا **ریسه هسته پردازه که شامل کدهای آماده‌سازی اولیه به‌خواب رونده نیز هست، اجرا شده** تا باقی آماده‌سازی سیستم انجام گیرد. سپس **ادامه اجرای پردازه در سطح کاربر از سر گرفته شده و پوسته اجرا می‌شود**. تخصیص داده‌ساختارهای پردازه نخست (و ریسه هسته آن)، مقداردهی آن‌ها و افزودن پردازه به صف پردازنده، با فراخوانی تابع userinit() صورت می‌گیرد (خط ۱۲۳۵) که در ادامه تشریح می‌شود.

---

[۱] این نام‌گذاری تنها برای سهولت درک مفاهیم است.

[۲] Kernel Thread

۲۳. مدیریت برنامه‌های سطح کاربر مستلزم ارائه انتزاعاتی برای ایجاد تمایز میان این برنامه‌ها و برنامه مدیریت آن‌ها است. 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.

در ابتدای userinit()، داده‌ساختار مربوط به اطلاعات پردازه با اجرای تابع allocproc() تخصیص و مقداردهی می‌شود (خط ۲۵۲۵).[۱] این تابع، اجزای مربوط به اجرای ریسه هسته پردازه در struct proc را مقداردهی نموده و آن را در صف پردازه‌ها قرار می‌دهد تا در نهایت، پردازه زمان‌بندی شود. 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 (به ظاهر) مجدد ریسه هسته برنامه ابتدا forkret() اجرا گردد. در واقع زمان‌بند، با مراجعه به context در داده‌ساختار proc به اطلاعات حالت پردازه در هسته، پیش از توقف، دسترسی یافته و مشاهده می‌کند که اشاره‌گر به دستورالعمل به forkret() اشاره می‌کند. با اجرای مجدد پردازه که ابتدایش در هسته خواهد بود، این تابع اجرا شده و آماده‌سازی کدهای به‌خواب‌رونده را انجام می‌دهد.

۲۴. چرا به خواب رفتن در کد مدیریت‌کننده سیستم مشکل‌ساز است؟ (راهنمایی: به زمان‌بندی مرتبط است.)

اجزای مربوط به اجرای پردازه در سطح کاربر و همچنین جدول صفحه پردازه نیز باید مقداردهی شوند تا اطلاعات، تکمیل شده و پردازه به صف پردازه‌های قابل اجرا توسط زمان‌بند افزوده شود. از جمله اجزای ساختار proc متغیر pgdir است که آدرس جدول صفحه مربوط به هر برنامه سطح کاربر را نگه‌داری می‌کند. مشاهده می‌شود که این آدرس با فراخوانی setupkvm() (خط ۲۵۲۸) مقدار گرفته و با آدرس مربوط به جدول کد مدیریت‌کننده سیستم که در kpgdir برای کل سیستم نگه‌داری شده بود، متفاوت است.

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

پس از این مرحله، مداخل جدول صفحه مربوط به نیمه بالایی فضای آدرس مجازی، مقداردهی شده‌اند. با اجرای تابع inituvm() (خط ۲۵۳۰) نیمه دیگر جدول صفحه نیز مقداردهی می‌شود. برنامه سطح کاربری که قرار است اجرا شده و نهایتاً پوسته را اجرا کند، برنامه initcode بوده که کد آن در فایل initcode.S پیاده‌سازی شده است. لذا نیمه سطح کاربر فضای آدرس مجازی به کد آن اشاره خواهد نمود. به طوری که در آدرس صفر تا ۴ کیلوبایت، کد مربوط به initcode.S قرار گیرد.

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

تا این مرحله کل جدول صفحه پردازه initcode مقداردهی شده است. پیش‌تر ذکر شد که وانمود می‌شود اجرای سطح کاربر، متوقف شده و ورود به هسته رخ داده است. این فرایند تله[۲] نام دارد (آشنایی بیشتر در آزمایش دوم). به طور کلی هنگام وقوع تله، اطلاعات اجرایی پردازه در لحظه وقوع تله در مکانی از حافظه تحت عنوان قاب تله[۳] ذخیره می‌شود (داده‌ساختار struct trapframe در خط ۶۰۲). هنگام بازگشت به سطح کاربر، این اطلاعات از فیلد مربوطه در struct proc (خط ۲۳۴۴) خوانده شده تا اجرا از نقطه توقف، ادامه یابد. برای اجرای پوسته، باید ادامه اجرا در سطح کاربر از ابتدای initcode.S انجام شود. زیرا این برنامه پوسته را فراخوانی نموده و در انتهای این کد اسمبلی، فراخوانی سیستمی exec رخ داده و پوسته اجرا خواهد شد (خط ۸۴۱۴). لذا قاب تله به گونه‌ای مقداردهی می‌شود که اجرای (به ظاهر) مجدد سطح کاربر، از آدرس صفر initcode به وقوع بپیوندد (خط ۲۵۳۹). با اتمام اجرای userinit() تمامی اطلاعات مربوط به اجرای مجدد در سطح هسته و کاربر و جدول صفحه برای برنامه سطح کاربر initcode در struct proc مقداردهی شده‌اند.

۲۷. فراخوانی سیستمی exec() در لینوکس چه وظیفه‌ای دارد؟

در انتهای تابع main() تابع mpmain() فراخوانی شده (خط ۱۲۳۶) و به دنبال آن تابع scheduler() فراخوانی می‌گردد (خط ۱۲۵۷). 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 (از جمله تغییر cr3 برای اشاره به جدول نگاشت برنامه جدید) روی پردازنده است. این تغییرات در فصل زمان‌بندی تشریح می‌شود. با توجه به این

[۱] این تابع برای ایجاد تمامی پردازه‌ها فراخوانی می‌شود. در خط ۲۵۸۷ در fork() نیز allocproc() فراخوانی شده است.

[۲] Trap

[۳] Trap Frame

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 پوسته را اجرا می‌نماید. به این ترتیب امکان ارتباط کاربر با سیستم‌عامل از طریق پوسته فراهم می‌شود.

۲۸. 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؟

۲۹. برنامه معادل initcode.S در هسته لینوکس چیست؟

اشکال زدایی

کد هر برنامه‌ای ممکن است دارای اشکال باشد. اشکال‌زدایی ممکن است ایستا، پویا و یا به صورت ترکیبی صورت پذیرد. کشف اشکال در روش‌های ایستا، بدون اجرا و تنها بر اساس اطلاعات کد برنامه صورت می‌گیرد. به عنوان مثال کامپایلر Clang دارای تحلیل‌گرهای ایستا برای اشکال‌زدایی اشکال‌های خاص است. اشکال‌زدایی پویا که معمولاً دقیق‌تر است، اقدام به کشف اشکال در حین اجرای برنامه می‌نماید. ابزار leakcheck- در ابزار Valgrind یک اشکال‌زدای پویا برای تشخیص نشتی حافظه[۱] است. از یک منظر می‌توان اشکال‌زداهای پویا را به دو دسته تقسیم نمود: ۱) اشکال‌زداهایی که بر یک نوع اشکال خاص مانند نشتی تمرکز دارند و ۲) اشکال‌زداهایی که مستقل از نوع اشکال بوده و تنها اجرا را ردگیری[۲] نموده و اطلاعاتی از حالت سیستم (شامل سخت‌افزار و نرم‌افزار) در حین اجرا یا پس از اجرا جهت درک بهتر رفتار برنامه برمی‌گردانند. در این بخش ابزار اشکال‌زدای گنو[۳] (GDB)، که یک اشکال‌زدای پویا از نوع دوم است معرفی خواهد شد.

GDB یک اشکال‌زدای متداول در سیستم‌های یونیکسی بوده که در بسیاری از شرایط، نقش قابل‌توجهی در تسریع روند اشکال‌زدایی ایفا می‌کند. اشکال‌زدایی برنامه‌های تک‌ریسه‌ای[۴]، چندریسه‌ای[۵] و حتی هسته‌های سیستم‌عامل توسط این ابزار ممکن است. جهت اشکال زدایی xv6 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 پیش‌تر 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.

روند اجرای GDB

GDB می‌تواند در هر گام از اجرا، با ارائه حالت سیستم، به برنامه‌نویس کمک کند تا حالت خطا را از حالت مورد انتظار تشخیص دهد. هنگام اجرای کد در GDB ممکن است چندین حالت رخ دهد:

۱. اجرا با موفقیت جریان داشته باشد یا خاتمه یابد.
۲. اجرا به علت اشکال، ناتمام مانده و برنامه متوقف شود.
۳. اجرا متوقف نشده ولی حالت سیستم در برخی نقاط درونی یا در خروجی‌های برنامه نادرست باشد.

هدف، یافتن حالات خطای سیستم در دو وضعیت ۲ و ۳ است. به عبارتی ابتدا باید در نقطه مورد نظر، توقف صورت گرفته و سپس به کمک دستورهایی حالت سیستم را استخراج نمود. برای توقف اجرا در نقاط مختلف اجرا در GDB سازوکارهای مختلفی وجود دارد:

۱. در اجرای ناتمام، اجرای برنامه به طور خودکار متوقف می‌شود.
۲. با فشردن کلید ترکیبی Ctrl + C به اشکال‌زدا بازگشت.

این عملیات در میان اجرا، آن را متوقف نموده و کنترل را به خط فرمان اشکال‌زدا منتقل می‌کند. مثلاً حلقه بی‌نهایت رخ داده باشد، می‌توان با این کلید ترکیبی، در نقطه‌ای از حلقه متوقف شد.

۳. روی نقطه‌ای از برنامه Breakpoint قرار داد. بدین ترتیب هر رسیدن اجرا به این نقطه منجر به توقف اجرا گردد.

روش‌های مختلفی برای تعیین نقطه استقرار Breakpoint وجود داشته که در این لینک قابل مشاهده است. از جمله:

انتخاب نام و شماره خط فایل

---

[۱] Memory Leak
[۲] Tracing
[۳] GNU Debugger
[۴] Single-Thread
[۵] Multithread

```
$ b(reak) cat.c:12
```

انتخاب نام تابع

```
$ b cat
```

انتخاب آدرس حافظه

```
$ b *0x98
```

این نقاط می‌توانند در سطح کاربر یا هسته سیستم‌عامل باشند. همچنین می‌توانند شرطی تعریف شوند.

۴. روی خانه خاصی از حافظه Watchpoint قرار داد تا دسترسی یا تغییر مقدار آن خانه، منجر به توقف اجرا گردد.

Watchpointها انواع مختلفی داشته و با دستورهای خاص خود مشخص می‌گردند.

دستور زیر:

```
$ watch *0x1234567
```

یک Watchpoint روی آدرس 0x1234567 در حافظه می‌گذارد. بدین ترتیب نوشتن در این آدرس، منجر به توقف اجرا خواهد شد.

می‌توان از نام متغیر هم استفاده نمود. مثلاً watch v، Watch را روی (آدرس) متغیر v قرار می‌دهد.

باید دقت نمود، اگر Watch روی متغیر محلی قرار داده شود، با خروج از حوزه دسترسی به آن متغیر، Watch حذف شده و به برنامه‌نویس اطلاع داده می‌شود. اگر هم آدرسی از فضای پشته[۱و۲] داده شود، ممکن است در حین اجرا متغیرها یا داده‌های نامرتبط دیگری در آن آدرس نوشته شود. یعنی این آدرس در زمان‌های مختلف مربوط به داده‌های مختلف بوده و در عمل Watch کارایی مورد نظر را نداشته باشد.

یک مزیت مهم Watch، تشخیص وضعیت مسابقه[۳] است که در فصول بعدی درس با آن آشنا خواهید شد. در این شرایط می‌توان تشخیص داد که کدام ریسه[۴] یا پردازه مقدار نامناسب را در آدرس حافظه نوشته که منجر به خطا شده است.

همان‌طور که مشاهده می‌شود، خیلی از حالات با استفاده از چهار سازوکار مذکور به سهولت قابل استخراج نیستند. مثلاً حالتی که یک زنجیره خاص فراخوانی توابع وجود داشته باشد یا این که مثلاً حالتی خاص در داده‌ساختارها رخ داده و یک لیست پیوندی، چهارمین عنصرش را حذف نماید.

۱) برای مشاهده Breakpointها از چه دستوری استفاده می‌شود؟

۲) برای حذف یک Breakpoint از چه دستوری و چگونه استفاده می‌شود؟

کنترل روند اجرا و دسترسی به حالت سیستم

پس از توقف می‌توان با استفاده از دستورهایی به حالت سیستم دسترسی پیدا نمود. همچنین دستورهایی برای تعیین شیوه ادامه اجرا وجود دارد. در ادامه، برخی از دستورهای کنترلی و دسترسی به حالت اجرا معرفی خواهد شد.

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. همچنین دستور c(ontinue) اجرا را تا رسیدن به نقطه توقف بعدی یا اتمام برنامه ادامه می‌دهد.

۳) دستور زیر را اجرا کنید. خروجی آن چه چیزی را نشان می‌دهد؟

```
$ bt
```

---

[۱] Stack

[۲] یعنی فضای آدرسی که داده‌هایی از جمله مقادیر متغیرهای محلی و آدرس‌های برگشت مربوط به توابع فراخوانی شده در آن قرار دارد.

[۳] Race Condition

[۴] Thread

۴) دو تفاوت دستورهای x و print را توضیح دهید. چگونه می‌توان محتوای یک ثبات خاص را چاپ کرد؟ (راهنمایی: می‌توانید از دستور help استفاده نمایید: help x و help print)

با دستور list می‌توان کد نقطه توقف را مشاهده نمود.

۵) برای نمایش وضعیت ثبات‌ها از چه دستوری استفاده می‌شود؟ متغیرهای محلی چطور؟ نتیجه این دستور را در گزارش‌کار خود بیاورید. همچنین در گزارش خود توضیح دهید که در معماری x86 رجیسترهای edi و esi نشانگر چه چیزی هستند؟

۶) به کمک استفاده از GDB، درباره ساختار struct input موارد زیر را توضیح دهید:

- توضیح کلی این struct و متغیرهای درونی آن و نقش آن‌ها
- نحوه و زمان تغییر مقدار متغیرهای درونی (برای مثال، input.c در چه حالتی تغییر می‌کند و چه مقداری می‌گیرد)

اشکال زدایی در سطح کد اسمبلی

اشکال زدایی برنامه در سطوح مختلفی قابل انجام است. با توجه به این که بسیاری از جزئیات اجرا در کد سطح بالا (زبان سی[۱]) قابل مشاهده نیست، نیاز به اشکال‌زدایی در سطح کد اسمبلی خواهد بود. به عنوان مثال بهینه‌سازی‌های ممکن است ترتیب اجرا در کد سطح بالا را تغییر داده یا بخشی از کد را حذف نماید. به عنوان مثال دیگر می‌توان از شیوه دسترسی به جداول لینکر نام برد. جزئیات دسترسی به یک تابع کتابخانه‌ای خاص یا یک متغیر سراسری آن کتابخانه دسترسی شده است، در سطح کد اسمبلی و با دسترسی به جداول لینک رخ داده و در سطح زبان سی قابل رؤیت نیست.

با فشردن همزمان سه دکمه Ctrl + X + A رابط کاربری متنی[۲] (TUI) GDB گشوده شده و کد اسمبلی مربوط به نقطه توقف، قابل رؤیت است. برای اطلاعات بیشتر در رابطه با این رابط کاربری می‌توانید به این صفحه مراجعه کنید.

۷) خروجی دستورهای layout src و layout asm در TUI چیست؟

۸) برای جابجایی میان توابع زنجیره فراخوانی جاری (نقطه توقف) از چه دستورهایی استفاده می‌شود؟

دستورهای s(tep)i و n(ext)i معادل‌های سطح اسمبلی s(tep) و n(ext) بوده و به جای یک دستور سی، در ریزدانگی یک دستورالعمل ماشین عمل می‌کنند. در شرایطی که کد مورد اشکال‌زدایی از ابتدا در زبان اسمبلی نوشته شده باشد، چاره‌ای جز استفاده از این دستورها وجود نخواهد داشت.

نکات پایانی

با توجه به کاستی‌هایی که در اشکال‌زداها وجود دارد، همچنان برخی از تکنیک‌ها در کدزنی می‌تواند بسیار راهگشا باشد. ساده‌ترین راه برای اشکال‌زدایی این است که تغییرها را اندک انجام داده و گام‌به‌گام از صحت اجرای کد، اطمینان حاصل شود. به عنوان مثال اگر آرایه‌ای ۱۰۰ عنصری تخصیص داده شده و در نقطه‌ای فراتر از مرز انتهایی آن نوشتن صورت گیرد، حافظه‌ای غیر از حافظه مربوط به آرایه دستکاری می‌گردد. چندین حالت ممکن است رخ دهد. از جمله اینکه:

۱. اقدام به نوشتن در حافظه‌ای فقط خواندنی مانند کد برنامه، صورت پذیرد. در چنین شرایطی خطا رخ داده و نقطه توقف به راحتی در GDB قابل رؤیت خواهد بود.
۲. در حافظه نوشتنی نامرتبط نوشته شده و مشکلی پیش نیاید.
۳. در حافظه نوشتنی نامرتبط نوشته شود و اجرای برنامه به طرز عجیبی متوقف گردد. به طوری که GDB نقطه نامربوطی را نشان دهد. یعنی تأثیر آن بلافاصله و به طور مستقیم رخ ندهد. در چنین شرایطی استفاده ابتدایی از اشکال‌زدا راحتی راهگشا نخواهد بود. چک کردن اندازه آرایه و احتمال دسترسی به خارج آن در سطح کد، می‌توانست راحت‌تر باشد. البته در برخی موارد به سادگی و یا با تکنیک‌هایی مانند استفاده از Watch، ضبط اجرا و حرکت رو به عقب از حالت نادرست، می‌توان اشکال را یافت[۳]. اما تکنیک قبلی بهتر بود.

بنابراین، استفاده از GDB در کنار دیگر ابزارها و تکنیک‌ها در پروژه‌های این درس توصیه می‌گردد. با توجه به آشنایی اولیه‌ای که با GDB فراهم شده است، می‌توان مزایای آن را برشمرد:

---

[۱] C

[۲] Text User Interface

[۳] GDB در برنامه‌های عادی قادر به ضبط و اجرای رو به عقب برنامه است. همچنین ابزار RR که توسط شرکت موزیلا برای اشکال‌زدایی فایرفاکس ارائه شده است امکان انجام همین عملیات را به صورت قطعی دارد. این قطعیت، در اشکال‌زدایی کدهای همروند و وضعیت مسابقه بسیار کمک‌کننده است.

- اشکال‌زدایی کدهای بزرگ و کدهایی که با پیاده‌سازی آن‌ها آشنایی وجود ندارد. ممکن است نیاز باشد یک کد بزرگ را به برنامه اضافه کنید. در این شرایط اشکال‌زدایی اجرای Crash کرده در GDB درک اولیه‌ای از نقطه خرابی ارائه می‌دهد.
- بررسی مقادیر حالت برنامه، بدون نیاز به قرار دادن دستورهای چاپ مقادیر در کد و کامپایل مجدد آن.
- بررسی مقادیر حالت سخت‌افزار و برنامه که در سطح کد قابل رؤیت نیستند. به عنوان مثال مقدار یک اشاره‌گر به تابع، مقصد یک تابع کتابخانه‌ای، اطمینان از قرارگیری آدرس متغیر محلی در بازه حافظه پشته، این که اجرا در کدام فایل کد منبع قرار دارد، اطلاع از وضعیت فضای آدرس حین اجرا، مثلاً این که هر کتابخانه در چه آدرسی بوده و در کدام کتابخانه در حال اجرا هستیم و ....
- تشخیص اشکال‌های پیچیده مانند این که کدام ریسه، یک متغیر را دستکاری نموده یا چرا یک متغیر مقدار نادرستی داشته یا مقداردهی اولیه نشده است. این اشکال‌های با کمک Watch و ضبط و اجرای مجدد رو به جلو/عقب به راحتی قابل تشخیص هستند.

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 نمایید. کافی است در محل بارگذاری در سایت درس، آدرس مخزن، شناسه آخرین Commit و گزارش پروژه را بارگذاری نمایید.
- به سؤالاتی که در صورت پروژه از شما خواسته شده است پاسخ دهید و آن‌ها را در گزارش‌کار خود بیاورید.
- 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.
- سؤالات را در **کوتاه‌ترین اندازه ممکن** پاسخ دهید.