به نام خدا

آزمایشگاه سیستم‌عامل - پاییز ۱۴۰۴

پروژه اول: آشنایی با هسته سیستم‌عامل xv6

طراحان: بهراد علمی، کسری کاشانی، البرز محمودیان

مقدمه

سیستم‌عامل xv6 یک سیستم‌عامل آموزشی است که در سال 2006 توسط محققان دانشگاه MIT طرح و توسعه داده شده است. این سیستم‌عامل عموما به زبان C و با استفاده از هسته Unix Version 6 نوشته شده و روی معماری x86 قابل اجرا می‌باشد. سیستم‌عامل xv6 علی‌رغم سادگی و حجم کم، دربردارنده نکات حائز اهمیتی در طراحی بوده و برای مقاصد آموزشی بسیار پرکاربرد است. سابقا، در درس سیستم‌عامل دانشگاه تهران از هسته سیستم‌عامل لینوکس استفاده شده که به دلیل پیچیدگی بالا این تغییر (به xv6) اتفاق افتاده است. در این پروژه، ضمن آشنایی با معماری و برخی نکات پیاده‌سازی این سیستم‌عامل، آن را اجرا و اشکال‌زدایی خواهیم کرد و همچنین برنامه‌ای در سطح کاربر نوشته خواهد شد که روی آن قابل اجرا باشد.

آشنایی با سیستم‌عامل و xv6

کدهای مربوط به سیستم‌عامل xv6 از لینک زیر قابل دسترسی است:
https://github.com/mit-pdos/xv6-public
همچنین مستندات این سیستم‌عامل در صفحه درس بارگذاری شده است. برای این پروژه، نیاز است که فصل صفر و پیوست ب از مستندات فوق را مطالعه کرده و به سؤالات داده شده پاسخ دهید. پاسخ این سوالات را در قالب یک گزارش بارگذاری خواهید کرد. همچنین توصیه می‌شود برای باب آشنایی پیوست الف را نیز مطالعه کنید.

سیستم‌عامل جزو نخستین نرم‌افزارهایی است که پس از روشن شدن سیستم، اجرا می‌گردد. این نرم‌افزار، رابط نرم‌افزارهای کاربردی[1] با سخت‌افزار رایانه است.

1) سه وظیفه اصلی سیستم عامل را نام ببرید.

[1] Application programming interface (API)

2) آیا وجود سیستم‌عامل در تمام دستگاه ها الزامی است؟ چرا؟ در چه شرایطی استفاده از سیستم‌عامل لازم است؟
3) معماری سیستم‌عامل xv6 چیست؟ چه دلایلی در دفاع از نظر خود دارید؟
4) سیستم‌عامل xv6 یک سیستم تک‌وظیفه‌ای[2] است یا چندوظیفه‌ای[3]؟
5) همانطور که می‌دانید به طور کلی چندوظیفگی تعمیمی است از حالت چندبرنامگی[4]. چه تفاوتی میان یک برنامه[5] و یک پردازه[6] وجود دارد؟
6) ساختار یک پردازه در سیستم‌عامل xv6 از چه بخش‌هایی تشکیل شده است؟ این سیستم‌عامل به طور کلی چگونه پردازنده را به پردازه‌های مختلف اختصاص می دهد؟
7) مفهوم file descriptor در سیستم‌عامل‌های مبتنی بر UNIX چیست؟ عملکرد pipe در سیستم‌عامل xv6 چگونه است و به طور معمول برای چه هدفی استفاده می شود؟
8) فراخوانی‌های سیستمی exec و fork در سیستم‌عامل xv6 چه عملیاتی را انجام می‌دهند؟ از نظر طراحی، ادغام نکردن این دو چه مزیتی دارد؟

اجرا و برپایی[7] سیستم‌عامل xv6 روی شبیه ساز Qemu

در این بخش به اجرای سیستم‌عامل xv6 خواهیم پرداخت. علی‌رغم اینکه این سیستم‌عامل قابلیت اجرای مستقیم روی سخت‌افزار را دارد، به دلیل آسیب‌پذیری بالا و رعایت مسائل ایمنی از این کار اجتناب نموده و سیستم‌عامل را به کمک برابرساز[8] Qemu روی سیستم‌عامل لینوکس اجرا خواهیم کرد. برای این منظور لازم است که کدهای مربوط به سیستم‌عامل xv6 را از لینک ارائه شده clone و یا دانلود کنیم.

یکی از روش‌های متداول کامپایل و ایجاد نرم‌افزارهای بزرگ استفاده از ابزار یا سامانه ساخت[9] Makefile است. این ابزار با فراهم کردن بستری برای پردازش فایل‌های موجود در کد منبع برنامه، در قالب قوانین[10]، دستورالعمل‌ها[11] و متغیرها[12]، به توسعه دهنده امکان مشخص کردن نحوه کامپایل و لینک فایل‌های دودویی به یکدیگر را می‌دهد. در xv6 تنها یک Makefile وجود داشته و تمامی فایل‌های سیستم‌عامل نیز در یک پوشه قرار دارند.

---

[2] Single-tasking system
[3] Multitasking system
[4] Multiprogramming
[5] Program
[6] Process
[7] Setup
[8] Emulator
[9] Build system
[10] Rules
[11] Recipes
[12] Variables

در ادامه با اجرای دستور make در پوشه دانلود، سیستم‌عامل کامپایل می‌شود و در نهایت با اجرای دستور make qemu سیستم‌عامل روی برابرساز اجرا خواهد شد (توجه شود که فرض شده Qemu از قبل بر روی سیستم‌عامل شما نصب بوده است. در غیر این صورت ابتدا آن را نصب نمایید).

9) دستور make -n را اجرا نمایید. کدام دستور، فایل نهایی هسته را می سازد؟

اضافه کردن یک متن به Boot Message

کدهای خام را به نحوی تغییر دهید که پس از بوت شدن سیستم‌عامل روی ماشین مجازی Qemu، نام اعضای گروه نمایش داده شود. تصویر نتیجه را نیز در گزارش کار خود قرار دهید.

اضافه کردن قابلیت های جدید به کنسول

در این بخش، چند قابلیت جدید و پرکاربرد را به کنسول خود اضافه خواهید کرد:

1. **قابلیت جابه‌جایی مکان‌نما**[13]**:** به کمک این قابلیت کاربر می‌تواند:

- با فشردن کلیدهای → و ← مکان‌نما را بین کاراکترهای موجود در رشته ورودی کنونی کنسول، به چپ یا راست ببرد.
- با فشردن Ctrl+D، مکان‌نما در ابتدای (اولین کاراکتر از) کلمه بعد (در صورت وجود) قرار بگیرد.
- با فشردن Ctrl+A، مکان‌نما در ابتدای کلمه فعلی قرار بگیرد. در صورتی که مکان‌نما قبل از فشردن Ctrl+A، در ابتدای کلمه فعلی یا روی یک space بود، به ابتدای کلمه قبل (در صورت وجود) منتقل شود. دقت کنید که هر کلمه با یک یا چند space از هم جدا می‌شوند.

2. **قابلیت حذف آخرین کاراکتر وارد شده:** کاربر با فشردن دستور Ctrl+Z می‌تواند آخرین کاراکتری که روی رشته ورودی کنونی کنسول چاپ شده است (از نظر زمانی) را پاک کند. دقت کنید که مکان‌نما هرجا که باشد، آخرین کاراکتر وارد شده پاک می‌شود و همچنین با چند مرتبه فشرده شدن این دستور نیز به همان تعداد کاراکتر، برعکس ترتیب زمان وارد شدن، پاک خواهند شد.

[13] Cursor

3. **قابلیت select و copy - paste:** کاربر به کمک این قابلیت می‌تواند یک زیر رشته از رشته ورودی کنونی کنسول را انتخاب[14] کند، بدین صورت که روی یک کاراکتر دستور Ctrl+S یکبار زده شود و روی یک کاراکتر دیگر نیز دستور Ctrl+S را مجدد بزند تا رشته موجود در آن بازه انتخاب شود. توجه داشته باشید که رشته انتخاب شده با رنگ سفید هایلایت شده باشد. پس از فشرده شدن دستور Ctrl+S برای بار اول، کاربر همچنان می تواند در هر جایی از رشته، کاراکتر وارد یا حذف کند. در ادامه ۴ عمل زیر **روی رشته انتخاب شده**، قابل انجام می‌باشند:
    - با یک بار فشردن کلید Backspace کل رشته پاک شود.
    - با فشردن دستور Ctrl+C کل رشته صرفا کپی شود اما رشته همچنان انتخاب شده باقی بماند.
    - با فشردن دستور Ctrl+V، کل رشته با رشته از قبل کپی شده (در صورت وجود) جایگزین شود. دقت کنید که رشته از قبل کپی شده، در هر جای دیگری از ورودی (حتی بدون نیاز به انتخاب شدن) و در جایگاه مکان‌نما، با این دستور paste خواهد شد.
    - با فشردن هر کدام از کلیدهای کاراکترها کل رشته پاک شود و با همان کاراکتر جدید جایگزین شود.

در پایان هر یک از این ۴ عمل به جز عمل دوم (دستور Ctrl+C)، کنسول باید به حالت عادی خود برگردد. دقت کنید که در صورتی که هر کلید یا دستور دیگری، مثل کلید های → و ← یا دستورات Ctrl+Z و Ctrl+S و Ctrl+A و Ctrl+D، فشرده شود، رشته‌ی select شده فقط باید deselect شود و هیچ عمل دیگری (حتی کلید یا دستور وارد شده) انجام نشود و فقط کنسول به حالت عادی خود برگردد.

4. **قابلیت شما:** به نظر شما چه قابلیت جدید و کاربردی دیگری می تواند به کنسول اضافه شود، به طوری که قابل پیاده‌سازی باشد؟ حداقل 2 مورد را در گزارش کار خود نام ببرید. (دقت کنید که نیازی به پیاده‌سازی آن وجود ندارد). ممکن است در ترم های آینده، از ایده های شما (به همراه اسمتان) استفاده شود!

[14] Select

توجه کنید: دستورات فوق باید قابلیت اجرای ترکیبی نیز داشته باشند و عملکرد درستی از خودشان بروز دهند. هنگام تحویل، چند تست کیس ترکیبی نیز بررسی خواهند شد. علاوه بر آن، می‌بایست تست کیس‌های خود را که برای آزمون پروژه از آن استفاده می‌کنید با ذکر دلیل استفاده، در گزارش قرار دهید.

برنامه سطح کاربر

در این بخش به پیاده سازی یک برنامه سطح کاربر به زبان C به نام find_sum می پردازیم و آن را به برنامه های سطح کاربر سیستم عامل اضافه می کنیم.

این برنامه باید رشته ای شامل تعدادی کاراکتر و عدد را به عنوان ورودی بگیرد و مجموع تمام اعداد داخل آن فایل را در فایل خروجی به نام result.txt بنویسد.

شما باید از دستورات open, close, read, write داخل برنامه به درستی استفاده کنید. دقت کنید که برای پیاده سازی این برنامه سطح کاربر، شما باید در فایل Makefile نیز تغییرات لازم را بوجود آورید تا این برنامه نیز مانند دستورات دیگر از قبیل ls در سطح کاربر اجرا شوند.

ورودی و خروجی نمونه:

```
$ find_sum op1e2rati34ng5 6Sy7ste8m!
$ cat result.txt
63
$
```

10) در Makefile متغیرهایی به نام‌های UPROGS و ULIB تعریف شده است. کاربرد آن‌ها چیست؟
11) اگر به فایل‌های موجود در xv6 دقت کنید، می‌بینید که فایلی مربوط به دستور cd، برخلاف دستوراتی مانند ls و cat، وجود ندارد و این دستور در سطح کاربر اجرا نمی‌شود. توضیح دهید که این دستور cd در کجا اجرا می‌شود. به نظر شما دلیل این تفاوت میان دستور cd و دستورات دیگر مثل ls و cat چیست؟

قابلیت تکمیل خودکار[15]

در این بخش به پیاده‌سازی قابلیتی خواهید پرداخت که با وارد کردن یک یا چند حرف از یک برنامه سطح کاربر یا دستور پیش‌فرض و سپس با زدن کلید tab، باقیمانده نام برنامه سطح کاربر یا دستور را کامل کند. این قابلیت را در سه حالت زیر بررسی خواهیم کرد:

1. برنامه سطح کاربر یا دستور وجود داشته و برای ادامه آن تنها یک حالت وجود داشته باشد: در این صورت با فشردن کلید tab، ادامه دستور کامل شده و با زدن enter، اجرایی خواهد شد.
2. برنامه سطح کاربر یا دستور وجود داشته و برای ادامه آن چند حالت وجود داشته باشد: در این صورت با فشردن کلید tab برای بار اول، تغییری روی صفحه نمایش ظاهر نخواهد شد، اما با فشردن کلید tab برای بارهای بعد (برای همان رشته)، حالت‌های ممکن چاپ خواهند شد.
3. در صورتی که برنامه یا دستور وجود نداشته باشند نیز با فشردن کلید tab تغییری در صفحه نمایش ظاهر نخواهد شد.

دقت داشته باشد این قابلیت شامل تمامی برنامه‌ها و دستورات پیش‌فرض xv6 (مانند ls, ln, wc, cat و ...) و برنامه‌های سطح کاربری که به xv6 اضافه می‌کنید خواهد شد. همچنین به دستور cd توجه داشته و آن را نیز در پیاده‌سازی‌تان لحاظ کنید. با اجرای ls در xv6 می‌توانید تمامی فایل‌ها و مسیرهای موجود را مشاهده کنید. (**راهنمایی:** برای پیاده‌سازی این قابلیت می‌توانید از پیاده‌سازی ls کمک بگیرید.)

مراحل بوت سیستم عامل xv6

اجرای بوت‌لودر[16]

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[17] است. در این‌جا هر سکتور ۵۱۲ بایت است. اگر دیسک قابل بوت باشد، نخستین سکتورِ آن، سکتور بوت[18] نام داشته و شامل بوت‌لودر خواهد بود. بوت‌لودر کدی است که سیستم‌عامل را در حافظه بارگذاری می‌کند. یکی از روش‌های راه‌اندازی اولیه رایانه، بوتِ مبتنی بر سیستم ورودی/خروجی پایه[19] (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.

---

[15] Autocompletion
[16] Bootloader
[17] Sector
[18] Boot Sector
[19] Basic Input/Output System

12) در xv6 در سکتور نخست دیسک قابل بوت، محتوای چه فایلی قرار دارد؟ (راهنمایی: خروجی دستور make -n را بررسی نمایید.)
13) برنامه‌های کامپایل شده در قالب فایل‌های دودویی[20] نگه‌داری می‌شوند. فایل مربوط به بوت نیز دودویی است. نوع این فایل دودویی چیست؟ چرا از این نوع فایل دودویی استفاده شده است؟ تفاوت این نوع فایل دودویی با دیگر فایل های دودویی کد xv6 چیست؟ این فایل را به زبان قابل فهم انسان (اسمبلی[21]) تبدیل نمایید. (راهنمایی: از ابزار objdump استفاده کنید. باید بخشی از آن مشابه فایل bootasm.S باشد.)
14) علت استفاده از دستور objcopy در حین اجرای عملیات make چیست؟
15) در فایل‌های موجود در xv6، مشاهده می‌شود که بوت سیستم توسط فایل های 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.

16) یک ثبات عام منظوره[22]، یک ثبات قطعه[23]، یک ثبات وضعیت[24] و یک ثبات کنترلی[25] 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[26] 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.

17) پردازنده‌های x86 دارای مدهای مختلفی هستند. هنگام بوت، این پردازنده‌ها در مد حقیقی[27] قرار داده می‌شوند؛ مدی که سیستم عامل ام‌اس‌داس[28] (MS DOS) در آن اجرا می شد. چرا؟

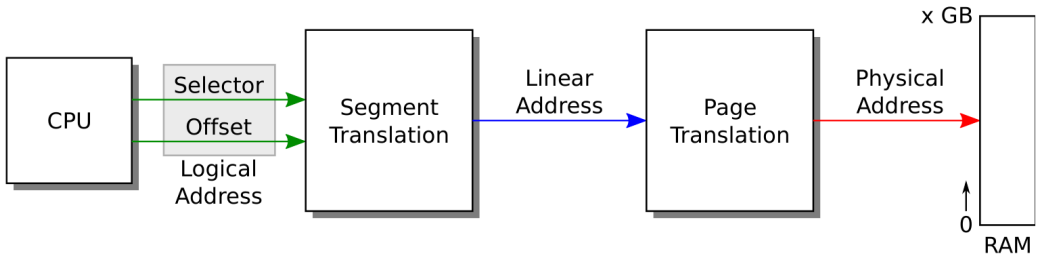
[20] Binary
[21] Assembly
[22] General Purpose Register
[23] Segment Register
[24] Status Registers
[25] Control Registers
[26] Privileged Access
[27] Real Mode
[28] Microsoft Disk Operating System

یک نقص اصلی این مد را بیان نمایید. آیا در پردازنده‌های دیگر مانند ARM یا RISC-V نیز مدها به همین شکل هستند یا خیر؟ توضیح دهید.

18) یکی دیگر در از مدهای مهم، مد حفاظت‌شده[29] می‌باشد. وظیفه‌ی اصلی این مود چیست؟ پردازنده‌ها در چه زمانی در این مود قرار می‌گیرند؟

در ابتدا qemu یک هسته را جهت اجرای کد بوت bootasm.S فعال می‌کند. فرایند بوت در بالاترین سطح دسترسی[30] صورت می‌گیرد. به عبارت دیگر، بوت‌لودر امکان دسترسی به تمامی قابلیت‌های سیستم را دارد. در ادامه هسته به مد حفاظت‌شده تغییر مد می‌دهد (خط ۹۱۵۳). در مد حفاظت‌شده، آدرس مورد دسترسی در برنامه (آدرس منطقی) از طریق جداولی به آدرس فیزیکی حافظه[31] نگاشت پیدا می‌کند.

ساختار آدرس‌دهی در این مد در شکل زیر نشان داده شده است.



19) کد bootmain.c، هسته را با شروع از یک سکتور پس از سکتورِ بوت، خوانده و در آدرس 0x100000 قرار می‌دهد.[32] علت انتخاب این آدرس چیست؟ چرا این آدرس از 0 شروع نشده است؟

در ادامه و در انتهای بوت، کد هسته سیستم‌عامل به طور کامل در حافظه قرار خواهد گرفت. در گام انتهایی نیز، 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.

[29] Protected Mode

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

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

[32] دقت شود آدرس 0x100000 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.

اشکال زدایی

کد هر برنامه‌ای ممکن است دارای اشکال باشد. اشکال‌زدایی ممکن است ایستا[33]، پویا[34] و یا به صورت ترکیبی[35] 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[36] برای اشکال‌زدایی اشکال‌های خاص است. اشکال‌زدایی پویا که معمولاً دقیق‌تر است، اقدام به کشف اشکال در حین اجرای برنامه می‌نماید. ابزار leak-check در ابزار Valgrind یک اشکال‌زدای پویا برای تشخیص نشتی حافظه[37] 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[38] 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[39] (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[40]، چندریسه‌ای[41] 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 به شرح زیر است:

1. در یک ترمینال دستور make qemu-gdb را اجرا کنید.
2. سپس در ترمینالی دیگر، فایل کد اجرایی را به عنوان ورودی به GDB بدهید.

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.

---

[33] Static
[34] Dynamic
[35] Hybrid
[36] Static analyzer
[37] Memory Leak
[38] Tracing
[39] GNU Debugger
[40] Single-Thread
[41] Multithread

3. 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:

1. اجرا با موفقیت جریان داشته باشد یا خاتمه یابد.
2. اجرا به علت اشکال، ناتمام مانده و برنامه متوقف شود.
3. 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:

1. در اجرای ناتمام، اجرای برنامه به طور خودکار متوقف می‌شود.
2. 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.

3. 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

```
$ break 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.

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

Watchpoint ها 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، Watchرا 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[42و43] 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[44] است که در فصول بعدی درس با آن آشنا خواهید شد. در این شرایط می‌توان تشخیص داد که کدام ریسه[45] 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.

20) برای مشاهده Breakpoint ها از چه دستوری استفاده می‌شود؟
21) 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 step و next و finish 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.

[42] Stack
[43] یعنی فضای آدرسی که داده‌هایی از جمله مقادیر متغیرهای محلی و آدرس‌های برگشت مربوط به توابع فراخوانی شده در آن قرار دارد.
[44] Race Condition
[45] Thread

به این ترتیب می‌توان برنامه را از جهت وجود حالات نادرست، بررسی نمود. همچنین دستور continue اجرا را تا رسیدن به نقطه توقف بعدی یا اتمام برنامه ادامه می‌دهد. با دستور list نیز می‌توان کد نقطه توقف را مشاهده نمود.

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

```
$ bt
```

23) 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)

24) برای نمایش وضعیت ثبات ها از چه دستوری استفاده می شود؟ برای متغیرهای محلی چطور؟ نتیجه این دستور را در گزارش کار خود بیاورید. همچنین در گزارش خود توضیح دهید که در معماری x86 رجیستر های edi و esi نشانگر چه چیزی هستند؟

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

- توضیح کلی این struct و متغیر های درونی آن و نقش آن‌ها.
- نحوه و زمان تغییر مقدار متغیر های درونی (برای مثال، input.e 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 C) 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[46] GDB یا همان TUI 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.

26) خروجی دستورات layout src و layout asm در TUI چیست؟

27) برای جابه‌جایی میان توابع زنجیره فراخوانی جاری (نقطه توقف) از چه دستوراتی استفاده می شود؟

[46] Text user interface

دستورهای stepi و nexti معادل‌های سطح اسمبلی step و next 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 رخ‌دهد، از جمله اینکه:

1. 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.
2. در حافظه نوشتنی نامرتبط نوشته شده و مشکلی پیش نیاید.
3. 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[47]. 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:

- 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 ....

[47] 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.

- 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

- پروژه خود را در یک مخزن خصوص در Github یا Gitlab پیش برده و در نهایت **یک نفر** از اعضای گروه کدها را به همراه پوشهٔ git.[48] و فایل pdf گزارش کار زیپ کرده و در سامانه با فرمت OS-Lab3-<SID1>-<SID2>-<SID3>.zip آپلود نمایید.
- رعایت نکردن مورد فوق کسر نمره را به همراه خواهد داشت.
- به تمامی سوالات پاسخ داده و پاسخ‌ها و نتایج را در گزارش کار خود قید کنید.
- بخش خوبی از نمرهٔ شما را پاسخ دهی به سوالات مطرح شده تشکیل می‌دهد که به شما در درک نحوهٔ کارکرد xv6 کمک می‌کند.
- سوالات را **به طور خلاصه** پاسخ دهید.
- در پیاده‌سازی خود در خصوص مواردی که ذکر نشده‌اند می‌توانید فرضی منطقی و بر اساس کارکرد پوسته[49] لینوکس خود در نظر بگیرید.
- مطالعه فصل صفر و پیوست ب از کتاب xv6 الزامی بوده مطالعه پیوست الف نیز توصیه می‌شود.
- تمامی مواردی که در جلسه توجیهی، گروه تیمز و فروم درس مطرح می شوند، جزئی از پروژه خواهند بود. در صورت وجود هرگونه سوال یا ابهام می‌توانید با ایمیل دستیاران مربوطه یا گروه تیمز درس در ارتباط باشید.
- همه اعضای گروه باید به تمام بخش های پروژه آپلود شده توسط گروه، چه کد و چه سوالات مسلط بوده و هنگام تحویل از تمامی اعضا و از تمامی قسمت‌ها به صورت تصادفی سوال پرسیده خواهد شد.
- در صورت مشاهده هرگونه شباهت بین کدها یا گزارش‌ها میان گروه‌ها در این ترم یا ترم‌های گذشته، مطابق با سیاست‌های درس برخورد خواهد شد.

موفق باشید

[48] برای اطمینان حاصل کردن از وجود پوشه git. می‌توانید گزینهٔ نمایش فایل‌های مخفی (Show Hidden Files) را فعال و یا از دستور ls -a استفاده کنید.

[49] Shell