



به نام خدا



آزمایشگاه سیستم عامل - بهار ۱۴۰۴

پروژه اول: آشنایی با هسته سیستم عامل xv6

طراحان: علی قنبری، هاتف رضائی

مقدمه

سیستم عامل xv6 یک سیستم عامل آموزشی است که در سال 2006 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 بر روی معماری Intel x86 قابل اجرا می باشد. سیستم عامل xv6 علی رغم سادگی و حجم کم، نکات اساسی و مهم در طراحی سیستم عامل را دارا است و برای مقاصد آموزشی بسیار مفید می باشد. تا پیش از این، در درس سیستم عامل دانشگاه تهران از هسته سیستم عامل لینوکس استفاده می شد که پیچیدگی های زیادی دارد. در ترم پیش رو، دانشجویان آزمایشگاه سیستم عامل بایستی پروژه های مربوطه را بر روی سیستم عامل xv6 اجرا و پیاده سازی نمایند. در این پروژه، ضمن آشنایی به معماری و برخی نکات پیاده سازی سیستم عامل، آن را اجرا و اشکال زدایی خواهیم کرد و همچنین برنامه ای در سطح کاربر خواهیم نوشت که بر روی این سیستم عامل قابل اجرا باشد.

آشنایی با سیستم عامل 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 فایل شامل کدهای آن نیز در صفحه درس بارگذاری شده است. برای این پروژه، نیاز است که فصل های 0 و 1 از مستندات فوق را مطالعه کرده و به برخی سؤالات منتخب پاسخ دهید. پاسخ این سؤالات را در قالب یک گزارش بارگذاری خواهید کرد.

1. معماری سیستم عامل xv6 چیست؟ چه دلایلی در دفاع از نظر خود دارید؟
2. یک پردازنده¹ در سیستم عامل xv6 از چه بخش هایی تشکیل شده است؟ این سیستم عامل به طور کلی چگونه پردازنده را به پردازنده های مختلف اختصاص میدهد؟
3. مفهوم file descriptor در سیستم عامل های مبتنی بر UNIX چیست؟ عملکرد pipe در سیستم عامل xv6 چگونه است و به طور معمول برای چه هدفی استفاده می شود؟

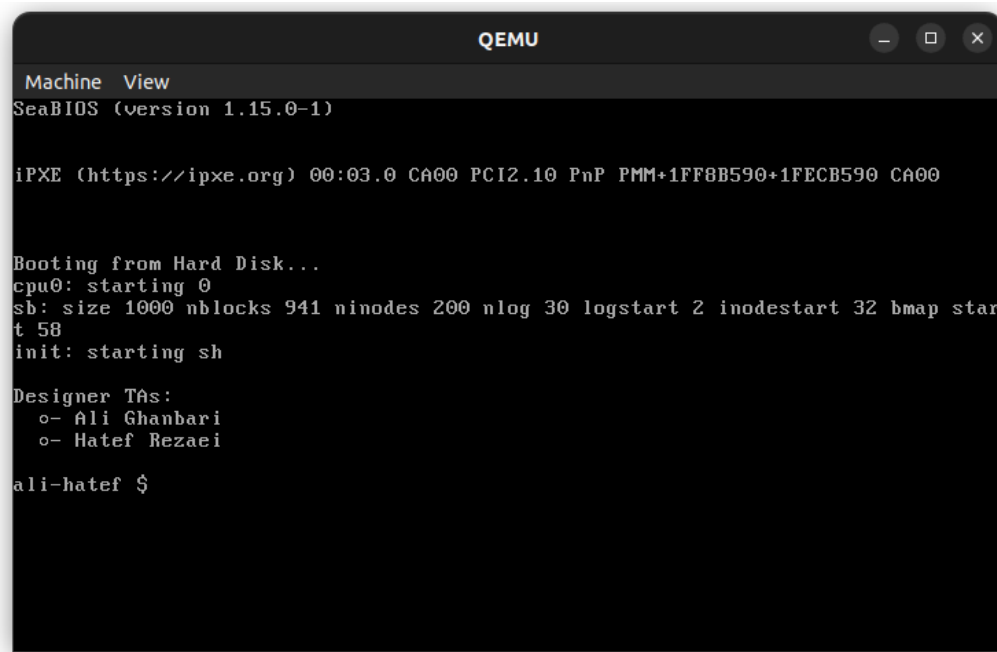
4. فراخوانی‌های سیستمی `exec` و `fork` چه عملی انجام میدهند؟ از نظر طراحی، ادغام نکردن این دو چه مزیتی دارد؟

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 برابر ساز² `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 برابر ساز 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).

شرح پروژه

1. کدهای خام را به نحوی تغییر دهید که پس از بوت شدن سیستم عامل روی ماشین مجازی `Qemu`، نام اعضای گروه نمایش داده شود. همچنین پیش از شروع شدن هر خط ورودی در ترمینال `xv6` نام اعضای گروه خود را نمایش دهید. بطور مثال:



```

QEMU
Machine View
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B590+1FECB590 CA00

Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh

Designer TAs:
  o- Ali Ghanbari
  o- Hatef Rezaei

ali-hatef $

```

2. اگر یک خط با ! شروع گردد و از کلمات کلیدی (شامل void و int, char, if, for, while, return) در آن استفاده شده باشد، پس از Enter در خروجی، همان جمله Paste شود ولی کلمات کلیدی به صورت highlight در بیایند و به رنگ آبی نوشته شوند. توجه کنید کاراکترهایی که بین دو # قرار داده می شوند را بایستی نادیده بگیرید و در خروجی نیاورید.

3. کاربر بتواند تکه ی مورد نظرش را از دستور روی کنسول انتخاب و کپی کند؛ به این صورت که با فشار دادن کلید < تا جای مورد نظر از انتها یک رشته کاراکتر ها را نادیده بگیرد و پس از آن با فشردن C + CTRL اولین کاراکتر مورد نظر را انتخاب و با فشار دادن مجدد < تا هر جایی که می خواهد را انتخاب کند. در نهایت با فشردن مجدد C + CTRL رشته ی انتخاب شده را در یک بافر ذخیره کند. ازین ب بعد هر زمان کاربر V + CTRL را فشار داد تکه ی موجود در بافر را روی ترمینال بنویسد. توجه کنید که به بافر در ابتدا خالیست. به عنوان مثال :

d : مقدار بافر ^C <- <- ^C makedir

4. با فشار دادن CTRL + H کاربر 5 دستور قبلی را که در ترمینال اجرا شده را ببیند.

5. در صورتی که کاربر دستور Tab را وارد کرد، متنی که در خط کنونی کنسول نوشته شده است، باید در صورت امکان کامل شود. همواره باید ده دستور آخری که کاربر وارد کرده است را ذخیره کنید تا در صورت وارد کردن دستور Tab متن با بهترین گزینه تکمیل شود. توجه کنید در صورتی که هیچ دستوری در بین ده تای ذخیره شده، متن را تکمیل نمی کرد، هیچ عملی انجام نشود و در صورتیکه چند گزینه وجود داشت، آخرین دستور جایگزین شود. توجه داشته باشید دستوراتی که با ! شروع می شوند نباید با Tab کامل شوند

توجه کنید که دستورات فوق باید قابلیت اجرای ترکیبی نیز داشته باشند و عملکرد درستی از خودشان بروز دهند.

برنامه سطح کاربر:

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

این برنامه باید رشته ای شامل دو کاراکتر '{ ' و '}' را بگیرد و در صورتی که رشته به فرم صحیح باشد خروجی : Right و در غیر این صورت خروجی Wrong را در فایلی به اسم Result.txt بنویسد.

ورودی و خروجی نمونه:

```
$ app_name {{}}
$ cat result.txt
Wrong
$
```

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 نام ببرید.
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 سرایند³ 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 قواعد⁴ و متغیرها⁵ 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 ارائه می‌کند. جهت اجرای سیستم‌عامل بر روی شبیه‌ساز، کافی است دستور make qemu در پوشه سیستم‌عامل اجرا گردد.

³ Header Files
⁴ Rules
⁵ Variables

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 سیستم عامل 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 سکتور⁶ است. در این‌جا هر سکتور ۵۱۲ بایت است. اگر دیسک قابل بوت باشد، نخستین سکتور آن سکتور بوت⁷ نام داشته و شامل بوت‌لودر⁸ خواهد بود. بوت‌لودر کدی است که سیستم‌عامل را در حافظه بارگذاری می‌کند. یکی از روش‌های راه‌اندازی اولیه رایانه، بوت مبتنی بر سیستم ورودی/خروجی مقدماتی⁹ (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. علت استفاده از دستور `objcopy` در حین اجرای عملیات `make` چیست؟

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.

⁶ Sector

⁷ Boot Sector

⁸ Boot Loader

⁹ Basic Input/Output System

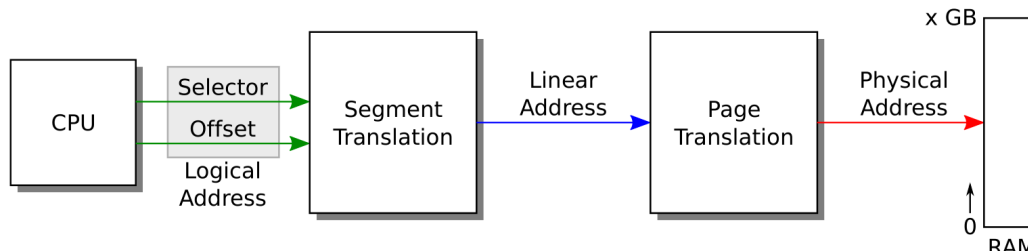
10. یک ثبات عام منظوره¹⁰، یک ثبات قطعه¹¹، یک ثبات وضعیت¹² و یک ثبات کنترلی¹³ در معماری x86 را نام برده و وظیفه هر یک را به طور مختصر توضیح دهید.

وضعیت ثبات‌ها را می‌توان به کمک gdb و دستور info registers مشاهده نمود. وضعیت برخی از ثبات‌های دیگر نیاز به دسترسی ممتاز¹⁴ 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 حقیقی¹⁵ قرار داده می‌شوند. مدی که سیستم عامل ام‌اس داس¹⁶ (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 قطعه¹⁷ و افسست¹⁸ 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 دسترسی¹⁹ صورت می‌گیرد. به عبارت دیگر، بوت‌لودر امکان دسترسی به تمامی قابلیت‌های سیستم را دارد. در ادامه هسته به مد حفاظت‌شده²⁰ تغییر مد می‌دهد (خط ۹۱۵۳). در مد حفاظت‌شده، آدرس مورد دسترسی در برنامه (آدرس منطقی) از طریق جداولی به آدرس فیزیکی حافظه²¹ نگاشت پیدا می‌کند. ساختار آدرس‌دهی در این مد در شکل زیر نشان داده شده است.



هر آدرس در کد برنامه یک آدرس منطقی²² 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

¹⁰ General Purpose Register

¹¹ Segment Register

¹² Status Registers

¹³ Control Registers

¹⁴ Privileged Access

¹⁵ Real Mode

¹⁶ Microsoft Disk Operating System

¹⁷ Segment

¹⁸ Offset

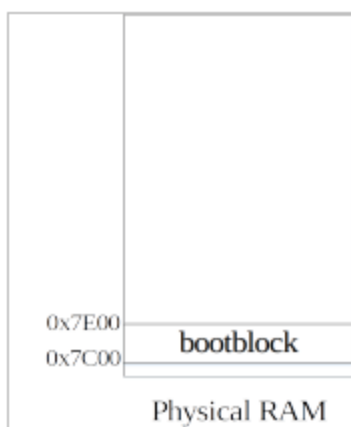
¹⁹ سطوح دسترسی در ادامه پروژه توضیح داده خواهد شد.

²⁰ Protected Mode

²¹ منظور از آدرس فیزیکی یک آدرس یکتا در سخت‌افزار حافظه است که پردازنده به آن دسترسی پیدا می‌کند.

²² Logical Address

قطعه²³ و ۲) ترجمه صفحه²⁴. 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 سراسری²⁵ (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 گزینش‌گر²⁶ صرف‌نظر نموده و افسست را به عنوان آدرس منطقی در نظر گرفت و این افسست را دقیقاً به عنوان آدرس خطی²⁷ 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 مجازی²⁸ 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.



²³ Segment Translation

²⁴ Page Translation

²⁵ Global Descriptor Table

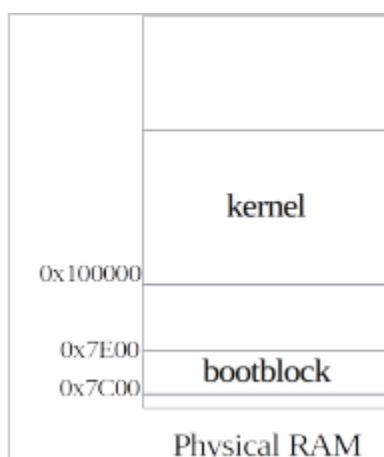
²⁶ Selector

²⁷ Linear Address

²⁸ Virtual Address

13. کد bootmain.c هسته را با شروع از سکتور بعد از سکتور بوت خوانده و در آدرس 0x100000 قرار می‌دهد.²⁹ علت انتخاب این آدرس چیست؟

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



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 صفحه‌بندی³⁰ (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 جدول خود در حافظه فیزیکی قرار داشته و یک آدرس فیزیکی مختص خود را دارد. در حین

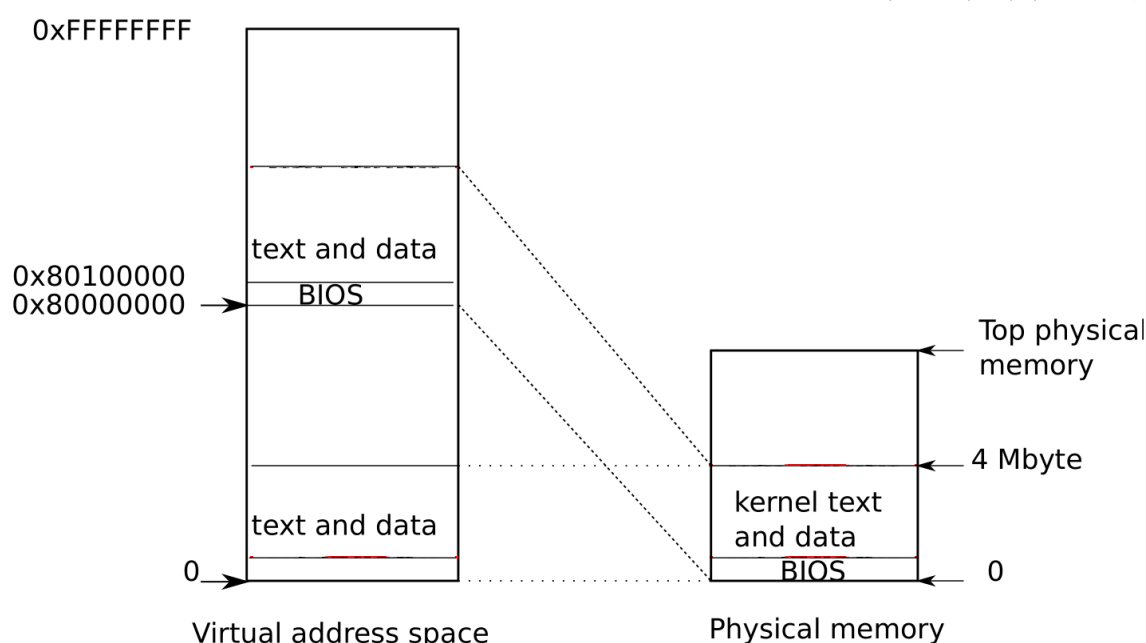
²⁹ دقت شود آدرس 0x100000 تنها برای خواندن هدر فایل elf استفاده شده است و محتوای فایل هسته در 0x100000 که توسط paddr (مخفف آدرس فیزیکی) تعیین شده است، کپی می‌شود. این آدرس در زمان لینک توسط kernel.ld تعیین شده و در فایل دودویی در قالب خاصی قرار داده شده است.

³⁰ Paging

اجرا این آدرس در ثبات کنترلی cr3 بارگذاری شده³¹ 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 صفحه³² 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 این جا 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.³³ 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 کاربر³⁴ حفظ گردد. این یک شرط لازم برای ارائه سرویس امن به برنامه‌های سطح کاربر است. هر

³¹ به طور دقیق‌تر این جداول سلسله‌مراتبی بوده و آدرس اولین لایه جدول در cr3 قرار داده می‌شود.

³² Page

³³ در xv6 از آدرس 0x80000000 به بعد مربوط به سطح هسته و آدرس‌های 0x0 تا این آدرس مربوط به سطح کاربر هستند.

³⁴ User Level Programs

کد در حال اجرا دارای یک سطح دسترسی جاری³⁵ (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 هستیم³⁶. تشخیص سطح دسترسی کد کنونی مستلزم خواندن مقدار ثبات CS است³⁷. 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 می‌دهد، 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.)

³⁵ Current Privilege Level

³⁶ دو سطح دسترسی دیگر در اغلب سیستم‌عامل‌ها بلااستفاده است.

³⁷ در واقع در مد محافظت شده، دو بیت از این ثبات، سطح دسترسی کنونی را معین می‌کند. بیت‌های دیگر کاربردهای دیگری مانند تعیین افسست مربوط به قطعه در gdt دارند.

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 سیستم عامل

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()` 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 برنامه³⁸ 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 ($CPL=3$) منتقل شود. به این ترتیب می‌تواند حتی به حافظه هسته نیز دسترسی داشته باشد. به منظور تغییر

³⁸ دقت شود اجرا هنوز در کد مدیریت‌کننده سیستم است.

مد امن، سازوکارهایی مانند فراخوانی سیستمی³⁹ 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 تله⁴⁰ نام داشته و در ساختار proc ذخیره می‌شود.⁴¹

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 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 سیستم عامل 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؟

اشکال زدایی

³⁹ System Call

⁴⁰ Trap Frame

⁴¹ تله لزوماً هنگام انتقال از مد کاربر به هسته رخ نمی‌دهد.

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 leak-check در ابزار Valgrind یک اشکال‌زدای پویا برای تشخیص نشتی حافظه⁴² است. از یک منظر می‌توان اشکال‌زدهای پویا را به دو دسته تقسیم نمود: (۱) اشکال‌زدهایی که بر یک نوع اشکال خاص مانند نشتی تمرکز دارند و (۲) اشکال‌زدهایی که مستقل از نوع اشکال بوده و تنها اجرا را ردگیری⁴³ 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 گنو⁴⁴ (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 تک‌ریسه‌ای⁴⁵، چندریسه‌ای⁴⁶ و حتی هسته‌های سیستم‌عامل توسط این ابزار ممکن است. جهت اشکال‌زدایی 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:

1. در یک ترمینال دستور `make qemu-gdb` اجرا گردد.
2. 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.
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.

⁴² Memory Leak

⁴³ Tracing

⁴⁴ GNU Debugger

⁴⁵ Single-Thread

⁴⁶ Multithread

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 دستورهای 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.

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 v 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 برنامه نویسی اطلاع داده می شود. اگر هم آدرسی از فضای پشته^{47,48} داده شود، ممکن است در حین اجرا متغیرها یا داده های نامرتب دیگري در آن آدرس نوشته شود. یعنی این آدرس در زمان های مختلف مربوط به داده های مختلف بوده و در عمل Watch کارایی مورد نظر را نداشته باشد.

یک مزیت مهم Watch، تشخیص وضعیت مسابقه⁴⁹ است که در فصول بعدی درس با آن آشنا خواهید شد. در این شرایط می توان تشخیص داد که کدام ریس⁵⁰ یا پردازنده مقدار نامناسب را در آدرس حافظه نوشته که منجر به خطا شده است.

همان طور که مشاهده می شود، 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 داده ساختارها رخ داده و یک لیست پیوندی، چهارمین عنصرش را حذف نماید.

(۱) برای مشاهده Breakpoint ها از چه دستوری استفاده می شود؟

(۲) برای حذف یک Breakpoint از چه دستوری و چگونه استفاده می شود؟

کنترل روند اجرا و دسترسی به حالت سیستم

پس از توقف می توان با استفاده از دستورهای به حالت سیستم دسترسی پیدا نمود. همچنین دستورهای 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 می توان 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 گام به گام قابل بررسی است. بدین معنی که پیش از اجرای خط جاری برنامه سطح کاربر یا هسته، امکان دستیابی به اطلاعات متغیرها و ثبات ها فراهم می باشد. به این ترتیب می توان برنامه را از جهت وجود حالات نادرست، بررسی نمود. همچنین دستور continue اجرا را تا رسیدن به نقطه توقف بعدی یا اتمام برنامه ادامه می دهد.

(۳) دستور زیر را اجرا کنید. خروجی آن چه چیزی را نشان می دهد؟

```
$ bt
```

(۴) دو تفاوت دستورهای x و print را توضیح دهید. چگونه می توان محتوای یک ثبات خاص را چاپ کرد؟ (راهنمایی: می توانید از دستور help استفاده نمایید: help x و help print با دستور list می توان کد نقطه توقف را مشاهده نمود.)

⁴⁷ Stack

⁴⁸ یعنی فضای آدرسی که داده هایی از جمله مقادیر متغیرهای محلی و آدرس های برگشت مربوط به توابع فراخوانی شده در آن قرار دارد.

⁴⁹ Race Condition

⁵⁰ Thread

۵) برای نمایش وضعیت ثبات‌ها از چه دستوری استفاده می‌شود؟ متغیرها 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.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 سی⁵¹) 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 متنی⁵² 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.

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

۸) برای جابجایی میان توابع زنجیره فراخوانی جاری (نقطه توقف) از چه دستوراتی استفاده می‌شود؟ 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 اشکال‌زدها 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 نوشتن نامرتب نوشته شده و مشکلی پیش نیاید.

C⁵¹Text user interface⁵²

3. در حافظه نوشتنی نامرتب 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 یافت⁵³. 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
- تشخیص اشکال‌های پیچیده مانند این که کدام ریس، 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.

⁵³ 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.

نکات مهم

- پروژه خود را در Github یا Gitlab پیش برده و در نهایت یک نفر از اعضای گروه کدها را به همراه پوشهٔ git و گزارش زیپ کرده و در سامانه با فرمت `OS-Lab1-<SID1>-<SID2>-<SID3>.zip` آپلود نمایید.
- تمامی مواردی که در جلسه توجیهی، گروه اسکایپ 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 اسکایپی درس در ارتباط باشید.
- به سوالات 1، 2، 3، 4، 5، 8، 13، 18، 19 و 23 پاسخ دهید و آن‌ها را در گزارش کار 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 شباهت بین کدها یا گزارش دو گروه، نمره 0 به هر دو گروه تعلق می‌گیرد.
- سوالات را در **کوتاه‌ترین اندازه ممکن** پاسخ دهید.

موفق باشید