



به نام خدا

# پروژه اول آزمایشگاه سیستم عامل

آشنایی با هسته سیستم عامل xv6

مسئولان تمرین: علی قنبری، نسا عباسی



## 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 پردازنده<sup>۱</sup> در سیستم‌عامل xv6 از چه بخش‌هایی تشکیل شده است؟ این سیستم‌عامل به طور کلی چگونه پردازنده را به پردازنده‌های مختلف اختصاص می‌دهد؟
۳. مفهوم file descriptor در سیستم‌عامل‌های مبتنی بر UNIX چیست؟ عملکرد pipe در سیستم‌عامل xv6 چگونه است و به طور معمول برای چه هدفی استفاده می‌شود؟
۴. فراخوانی‌های سیستمی fork و exec چه عملی انجام می‌دهند؟ از نظر طراحی، ادغام نکردن این دو چه مزیتی دارد؟

## اجرا و اشکال‌زدایی

در این بخش به اجرای سیستم‌عامل 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 سیستم‌عامل را به کمک برابرساز<sup>۲</sup> 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

<sup>۱</sup> Process  
<sup>۲</sup> Emulator

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

## اضافه کردن یک متن به Boot Message

در این بخش، شما باید نام اعضای گروه را پس از بوت شدن سیستم عامل روی ماشین مجازی Qemu، در انتهای پیام های 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 می خواهیم چند قابلیت کاربردی به کنسول xv6 اضافه کنیم.
- پس از اجرای سیستم عامل بر روی Qemu، در صورت استفاده از کلیدهای Shift+x، Shift+c، Shift+v، Ctrl+e و Tab معادل کاراکتری آنها، در کنسول چاپ می شود.
- کد xv6 را به نحوی تغییر دهید تا قابلیت های زیر در آن پیاده سازی شده باشد (نحوه محاسبه عدد k که در دستورات ذکر شده، در انتهای این بخش توضیح داده شده است):
1. در صورتی که کاربر دستور Shift+c را وارد کرد، k کاراکتر قبل از اشاره گر که در خط کنونی کنسول نوشته شده است، کپی (copy) شود.
  2. در صورتی که کاربر دستور Shift+x را وارد کرد، k کاراکتر قبل از اشاره گر که در خط کنونی کنسول نوشته شده است، کات (cut) شود.
  3. در صورتیکه کاربر دستور Shift+v را وارد کرد، متن انتخاب شده (کپی یا کات) الصاق شود. توجه شود نیازمندی عملکرد این دستور، حداقل وجود یک دستور قبل تر می باشد و لذا با شروع بوت سیستم عامل در صورت استفاده، عملی انجام نخواهد شد.
  4. در صورتی که کاربر دستور Ctrl+e را وارد کرد، اعمال زیر رخ دهد:
    - تمام اعدادی که در رشته ورودی کنسول نوشته شده اند، k تا افزایش یابند و جایگزین اعداد قبلی گردند. (دقت کنید که اعداد به صورت تک کاراکتری در نظر گرفته می شوند مثلاً اگر رشته ورودی k13b باشد، با فشردن کلید Ctrl+e دو عدد 1 و 3 تحت تأثیر الگوریتم این دستور قرار می گیرند. همچنین توجه کنید اگر حاصل جمع اعداد با k از 9 بیشتر شد بایستی با کاراکترهای A، B، C، ... جایگزین شوند.)
    - حروف کوچک به حروف بزرگ و حروف بزرگ به کوچک تبدیل شوند.
    - کاراکترهای غیر از اعداد و حروف نظیر '!'، '#' و ... حذف گردند.
  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 10 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 صورتیکه هیچ دستوری در بین 10 تای ذخیره شده، متن را تکمیل نمی‌کرد، هیچ عملی انجام نشود و در صورتیکه چند گزینه وجود داشت، آخرین دستور جایگزین شود.

❖ نحوه محاسبه عدد  $k$  بصورت زیر است:

$$k = (\text{جمع یکان شماره دانشجویی اعضای گروه}) \bmod 3 + 2$$

❖ دقت کنید در صورتی که کمتر از  $k$  کاراکتر در خط ورودی کنسول موجود بود، همان تعداد کاراکتر توسط دستورهای Shift+c و Shift+x باید لحاظ شوند.

❖ توجه کنید که اگر ترکیبی از دستورهای Shift+c و Shift+x توسط کاربر وارد شوند، تأثیر آخرین دستور وارد شده لحاظ شود، اما برای الصاق شدن چندین بار رشته ذخیره شده با اجرای دستور Shift+v در کنسول مانعی وجود ندارد.

❖ 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 sdvar می‌باشد. این برنامه تا سقف 7 عدد از ورودی دریافت کرده و میانگین آنها را محاسبه می‌کند. سپس برای اعداد کمتر یا مساوی میانگین در دنباله اعداد ورودی، انحراف معیار و برای اعداد بیشتر از میانگین در دنباله اعداد ورودی، واریانس را محاسبه می‌کند. در نهایت، خروجی‌های محاسبه شده را به ترتیب میانگین، انحراف معیار و سپس واریانس در یک فایل تکست با نام sdvar\_result.txt ذخیره می‌کند. (توجه کنید که در صورت اعشاری بودن جواب، تنها قسمت صحیح در نظر گرفته می‌شود. اگر فایل تکست از قبل موجود بود، جواب بر روی آن بازنویسی شود. همچنین توجه داشته باشید که اگر تعداد ورودی از تعداد مد نظر بیشتر باشد یا کاراکتر غیر عددی وارد شود، باید خطای مناسب به کاربر نمایش داده شود.)

```
$ sdvar 7 19 23 47 69 13 32
$ cat sdvar_result.txt
30 6 230
```

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.

## مقدمه‌ای درباره سیستم‌عامل xv6

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

5. سه وظیفه اصلی سیستم‌عامل را نام ببرید.
6. 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<sup>3</sup> 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<sup>4</sup> و متغیرها<sup>5</sup> 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.

7. دستور make -n را اجرا نمایید. کدام دستور، فایل نهایی هسته را می‌سازد؟
8. 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.

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

## مراحل بوت سیستم‌عامل xv6

### اجرای بوت‌لودر

هدف از بوت آماده‌سازی سیستم‌عامل برای سرویس‌دهی به برنامه‌های کاربر است. پس از بوت، سیستم‌عامل سازوکاری جهت ارائه سرویس به برنامه‌های کاربردی خواهد داشت که این برنامه‌ها بدون هیچ مزاحمتی بتوانند از آن استفاده نمایند. کوچکترین واحد دسترسی دیسک‌ها در رایانه‌های

<sup>3</sup> Header Files

<sup>4</sup> Rules

<sup>5</sup> Variables

شخصی سکتور<sup>۶</sup> است. در این‌جا هر سکتور ۵۱۲ بایت است. اگر دیسک قابل بوت باشد، نخستین سکتور آن سکتور بوت<sup>۷</sup> نام داشته و شامل بوت‌لودر<sup>۸</sup> خواهد بود. بوت‌لودر کدی است که سیستم‌عامل را در حافظه بارگذاری می‌کند. یکی از روش‌های راه‌اندازی اولیه رایانه، بوت مبتنی بر سیستم ورودی/خروجی مقدماتی<sup>۹</sup> (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.

10. در xv6 در سکتور نخست دیسک قابل بوت، محتوای چه فایلی قرار دارد. (راهنمایی: خروجی

دستور `make -n` را بررسی نمایید.)

11. 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.)

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

13. بوت سیستم توسط فایل‌های `bootmain.c` و `bootasm.S` 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.

14. یک ثبات عام‌منظوره<sup>۱۰</sup>، یک ثبات قطعه<sup>۱۱</sup>، یک ثبات وضعیت<sup>۱۲</sup> و یک ثبات کنترلی<sup>۱۳</sup> 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<sup>۱۴</sup> 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.

15. پردازنده‌های x86 دارای مدهای مختلفی هستند. هنگام بوت، این پردازنده‌ها در مد

حقیقی<sup>۱۵</sup> قرار داده می‌شوند. مدی که سیستم‌عامل ام‌اس‌داس<sup>۱۶</sup> (MS DOS) در آن اجرا

می‌شد. چرا؟ یک نقص اصلی این مد را بیان نمایید؟

<sup>۶</sup> Sector

<sup>۷</sup> Boot Sector

<sup>۸</sup> Boot Loader

<sup>۹</sup> Basic Input/Output System

<sup>۱۰</sup> General Purpose Register

<sup>۱۱</sup> Segment Register

<sup>۱۲</sup> Status Registers

<sup>۱۳</sup> Control Registers

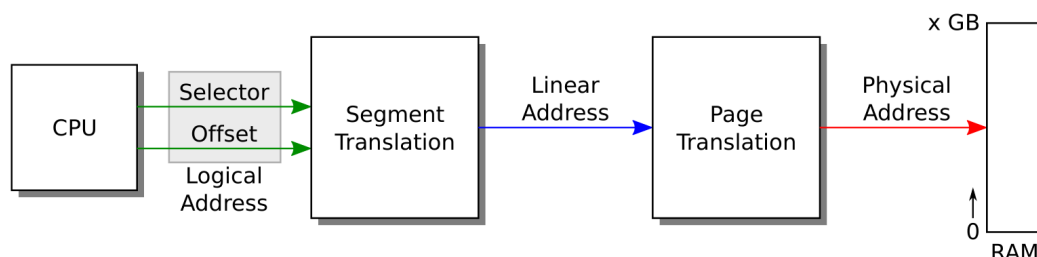
<sup>۱۴</sup> Privileged Access

<sup>۱۵</sup> Real Mode

<sup>۱۶</sup> Microsoft Disk Operating System

16. آدرس‌دهی به حافظه در این مدل شامل دو بخش قطعه<sup>17</sup> و افسست<sup>18</sup> 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<sup>19</sup> صورت می‌گیرد. به عبارت دیگر، بوت‌لودر امکان دسترسی به تمامی قابلیت‌های سیستم را دارد. در ادامه هسته به مد حفاظت‌شده<sup>20</sup> تغییر مد می‌دهد (خط ۹۱۵۳). در مد حفاظت‌شده، آدرس مورد دسترسی در برنامه (آدرس منطقی) از طریق جداولی به آدرس فیزیکی حافظه<sup>21</sup> نگاشت پیدا می‌کند. ساختار آدرس‌دهی در این مدل در شکل زیر نشان داده شده است.



هر آدرس در کد برنامه یک آدرس منطقی<sup>22</sup> 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<sup>23</sup> و ۲) ترجمه صفحه<sup>24</sup>. مفهوم ثبات‌های قطعه در این مدل تا حد زیادی با نقش آن‌ها در مد حقیقی متفاوت است. این ثبات‌ها با تعامل با جدولی تحت عنوان جدول توصیف‌گر سراسری<sup>25</sup> (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 مقداره‌ی می‌گردد (خطوط ۹۱۸۲ تا ۹۱۸۵) که می‌توان از گزینش‌گر<sup>26</sup> صرف‌نظر نموده و افسست را به عنوان آدرس منطقی در نظر گرفت و این افسست را دقیقاً به عنوان آدرس خطی<sup>27</sup> 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 مقداره‌ی 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<sup>28</sup> اطلاق خواهد شد. نگاشت دوم یا ترجمه صفحه در کد بوت فعال نمی‌شود. لذا در این‌جا نیز نگاشت

<sup>17</sup> Segment

<sup>18</sup> Offset

<sup>19</sup> سطوح دسترسی در ادامه پروژه توضیح داده خواهد شد.

<sup>20</sup> Protected Mode

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

<sup>22</sup> Logical Address

<sup>23</sup> Segment Translation

<sup>24</sup> Page Translation

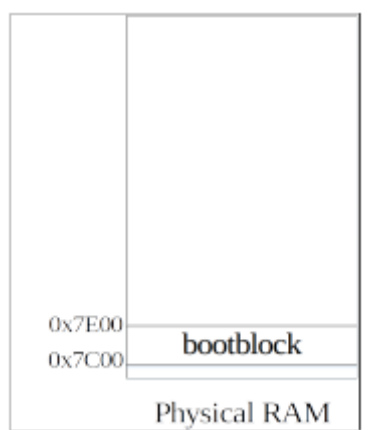
<sup>25</sup> Global Descriptor Table

<sup>26</sup> Selector

<sup>27</sup> Linear Address

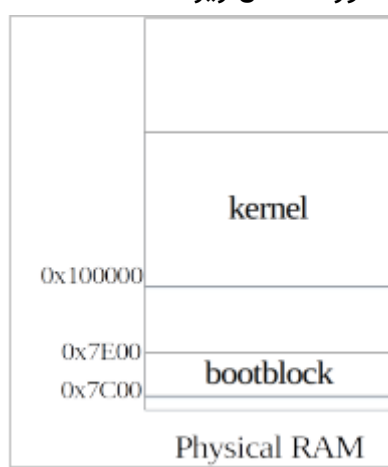
<sup>28</sup> Virtual Address

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.



17. کد bootmain.c هسته را با شروع از سکتور بعد از سکتور بوت خوانده و در آدرس 0x100000 قرار می‌دهد.<sup>29</sup> 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.

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

<sup>29</sup> 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.

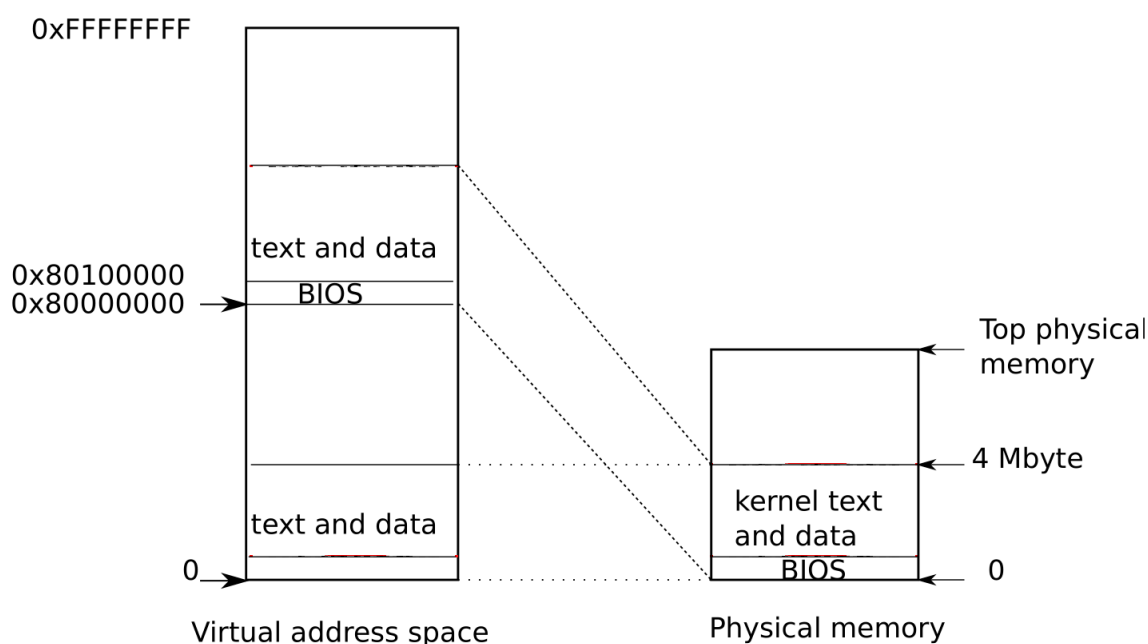


## اجرای هسته xv6

هدف از entry.S ورود به هسته و آماده‌سازی جهت اجرای کد C آن است. در شرایط کنونی نمی‌توان کد هسته را اجرا نمود. زیرا به گونه‌ای لینک شده است که آدرس‌های مجازی آن بزرگتر از 0x80100000 هستند. می‌توان این مسئله را با اجرای دستور cat kernel.sym بررسی نمود. در همین راستا نگاشت مربوط به صفحه‌بندی<sup>30</sup> (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 اجرا این آدرس در ثبات کنترلی cr3 بارگذاری شده<sup>31</sup> و به این ترتیب پردازنده از محل جدول نگاشت‌های جاری اطلاع خواهد داشت.

19. 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<sup>32</sup> نام دارد. یعنی حالت حافظه مطابق شکل زیر خواهد بود.



نیمه چپ شکل، فضای آدرس مجازی را نشان می‌دهد. جدول آدرس‌های نیمه چپ را به نیمه راست نگاشت می‌دهد. در این‌جا دو صفحه چهار مگابایتی به یک بخش چهار مگابایتی از حافظه فیزیکی نگاشت شده‌اند. یعنی برنامه می‌تواند با استفاده از دو آدرس به یک محتوا دسترسی یابد. این یکی دیگر از قابلیت‌های صفحه‌بندی است. در ادامه اجرا قرار است هسته تنها از بخش بالایی فضای

<sup>30</sup> Paging

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

<sup>32</sup> Page

آدرس مجازی استفاده نماید.<sup>33</sup> به عبارت دیگر، نگاشت پایینی حذف خواهد شد. علت اصلی این است که باید حافظه مورد دسترسی توسط هسته از دسترسی برنامه‌های کاربردی یا به عبارت دقیق‌تر برنامه‌های سطح کاربر<sup>34</sup> 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<sup>35</sup> (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 سیستم عامل xV6 اگر CPL=0 باشد در هسته و اگر CPL=3 باشد در سطح کاربر هستیم.<sup>36</sup> تشخیص سطح دسترسی کد کنونی مستلزم خواندن مقدار ثبات cs است.<sup>37</sup>

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.

20. 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).

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

22. 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.)

<sup>33</sup> در xV6 از آدرس 0x80000000 به بعد مربوط به سطح هسته و آدرس‌های 0x0 تا این آدرس مربوط به سطح کاربر هستند.

<sup>34</sup> User Level Programs

<sup>35</sup> Current Privilege Level

<sup>36</sup> دو سطح دسترسی دیگر در اغلب سیستم عامل‌ها بلااستفاده است.

<sup>37</sup> در واقع در مد محافظت شده، دو بیت از این ثبات، سطح دسترسی کنونی را معین می‌کند. بیت‌های دیگر کاربردهای دیگری مانند تعیین افسس 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.

23. 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 برنامه<sup>38</sup> ابتدا `forkret()` اجرا گردد. آماده‌سازی بخش‌های باقی‌مانده سیستم در این تابع انجام می‌شود.

24. 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.

25. 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.

26. 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 مد امن، سازوکارهایی مانند فراخوانی سیستمی<sup>39</sup> 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

<sup>38</sup> دقت شود اجرا هنوز در کد مدیریت‌کننده سیستم است.

<sup>39</sup> System Call

سطح کاربر به فضای آدرس مجازی هسته منتقل می‌شود. لذا باید وضعیت اجرای برنامه سطح کاربر در فضای آدرس مجازی سطح کاربر در مکانی ذخیره گردد. این مکان قاب تله<sup>40</sup> نام داشته و در ساختار proc ذخیره می‌شود.<sup>41</sup>

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 است 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 سیستم‌عامل را فراهم می‌آورد.

27. 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؟

28. 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

<sup>40</sup> Trap Frame

<sup>41</sup> تله لزوماً هنگام انتقال از مد کاربر به هسته رخ نمی‌دهد.

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<sup>42</sup> 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 ردگیری<sup>43</sup> 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<sup>44</sup> (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<sup>45</sup>، چندریسه‌ای<sup>46</sup> و حتی هسته‌های سیستم عامل توسط این ابزار ممکن است. جهت اشکال‌زدایی xv6 با GDB، در گام نخست باید سیستم عامل 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 سیستم عامل 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.

<sup>42</sup> Memory Leak

<sup>43</sup> Tracing

<sup>44</sup> GNU Debugger

<sup>45</sup> Single-Thread

<sup>46</sup> Multithread

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 اشکال زدا بازگشت.
- این عملیات در میان اجرا، آن را متوقف نموده و کنترل را به خط فرمان اشکال زدا 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 سیستم عامل 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 دستورهایی 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 برنامه نویسی اطلاع داده می‌شود. اگر هم آدرسی از فضای پشته<sup>47</sup> داده شود، ممکن است در حین اجرا متغیرها یا داده‌های نامرتب دیگر 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.

<sup>47</sup> Stack

<sup>48</sup> یعنی فضای آدرسی که داده‌هایی از جمله مقادیر متغیرهای محلی و آدرس‌های برگشت مربوط به توابع فراخوانی شده در آن قرار دارد.

یک مزیت مهم Watch، تشخیص وضعیت مسابقه<sup>49</sup> است که در فصول بعدی درس با آن آشنا خواهید شد. در این شرایط می‌توان تشخیص داد که کدام ریس<sup>50</sup> یا پردازنده 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 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 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. 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.

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

```
$ bt
```

(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؟ متغیرها محلی چگونه؟ نتیجه این دستور را در گزارش کار 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)

<sup>49</sup> Race Condition  
<sup>50</sup> Thread



## 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<sup>51</sup>) قابل مشاهده نیست، نیاز به اشکال زدایی در سطح کد اسمبلی خواهد بود. به عنوان مثال بهینه سازی های ممکن است ترتیب اجرا در کد سطح بالا را تغییر داده یا بخشی از کد را حذف نماید. به عنوان مثال دیگر می توان از شیوه دسترسی به جداول لینکر نام برد. جزئیات دسترسی به یک تابع کتابخانه ای 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<sup>52</sup> 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 می توانید 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 می شود؟

دستورهای `stepi` و `nexti` معادل های سطح اسمبلی `step` و `next` بوده و به جای یک دستور سی، در ریزدانگی یک دستورالعمل ماشین عمل می کنند. در شرایطی که کد مورد اشکال زدایی از ابتدا در زبان اسمبلی نوشته شده باشد، چاره ای جز استفاده از این دستورها وجود نخواهد داشت.

## نکات پایانی

با توجه به کاستی هایی که در اشکال زدها وجود دارد، همچنان برخی از تکنیک ها در کدزنی می تواند بسیار راهگشا باشد. ساده ترین راه برای اشکال زدایی این است که تغییرها را اندک انجام داده و گام به گام از صحت اجرای کد، اطمینان حاصل شود. به عنوان مثال اگر آرایه ای ۱۰۰ عنصری تخصیص داده شده و در نقطه ای فراتر از مرز انتهایی آن نوشتن صورت گیرد، حافظه ای غیر از حافظه مربوط به آرایه دستکاری می گردد. چندین حالت ممکن است رخ دهد. از جمله اینکه:

۱. اقدام به نوشتن در حافظه ای 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 نامرتب نوشته شده و مشکلی پیش نیاید.
۳. 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 اشکال زدا راحتی راهگشا نخواهد بود. چک کردن اندازه آرایه و احتمال دسترسی به خارج آن در سطح کد، می توانست راحت تر باشد. البته در برخی موارد به سادگی و یا با تکنیک هایی مانند استفاده از Watch، ضبط اجرا و حرکت رو به عقب از حالت نادرست، می توان اشکال را یافت<sup>53</sup>. اما تکنیک قبلی بهتر بود.

<sup>51</sup> C

<sup>52</sup> Text user interface

<sup>53</sup> GDB در برنامه های عادی قادر به ضبط و اجرای رو به عقب برنامه است. همچنین ابزار RR که توسط شرکت موزیلا برای اشکال زدایی فایرفاکس ارائه شده است امکان انجام همین عملیات را به صورت قطعی دارد. این قطعیت، در اشکال زدایی کدهای همروند و وضعیت مسابقه بسیار کمک کننده است.



- 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.

## نکات مهم

- برای تحویل پروژه ابتدا یک مخزن خصوصی در سایت GitHub 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 0 به هر دو گروه تعلق می‌گیرد.
- سؤالات را در کوتاه‌ترین اندازه ممکن پاسخ دهید.