



به نام خدا

آزمایشگاه سیستم عامل

# آشنایی، اجرا و اشکال زدایی هسته سیستم عامل xv6

(بخش دوم: مراحل اجرا و بوت)



## مقدمه‌ای درباره سیستم‌عامل و xv6

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

1. سه وظیفه اصلی سیستم‌عامل را بیان نمایید.
2. فایل‌های اصلی سیستم‌عامل xv6 در صفحه یک کتاب xv6 لیست شده‌اند. به طور مختصر هر گروه را توضیح دهید.

## کامپایل سیستم‌عامل xv6

یکی از روش‌های متداول کامپایل و ایجاد نرم‌افزارهای بزرگ در سیستم‌عامل‌های مبتنی بر یونیکس استفاده از ابزار Make است. این ابزار با پردازش فایل‌های موجود در کد منبع برنامه، موسوم به Makefile، شیوه کامپایل و لینک فایل‌های دودویی به یکدیگر و در نهایت ساختن کد دودویی نهایی برنامه را تشخیص می‌دهد. ساختار Makefile قواعد خاص خود را داشته و می‌تواند بسیار پیچیده باشد. اما به طور کلی شامل قواعد<sup>۱</sup> و متغیرها<sup>۲</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.

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.

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

## مراحل بوت سیستم‌عامل xv6

### اجرای بوت‌لودر

هدف از بوت آماده‌سازی سیستم‌عامل برای سرویس‌دهی به برنامه‌های کاربر است. پس از بوت، سیستم‌عامل سازوکاری جهت ارائه سرویس به برنامه‌های کاربردی خواهد داشت که این برنامه‌ها بدون هیچ مزاحمتی بتوانند از آن استفاده نمایند. کوچکترین واحد دسترسی دیسک‌ها در رایانه‌های شخصی سکتور<sup>۳</sup> است. در این‌جا هر سکتور ۵۱۲ بایت است. اگر دیسک قابل بوت باشد، نخستین سکتور آن سکتور بوت<sup>۴</sup> نام داشته و شامل بوت‌لودر<sup>۵</sup> خواهد بود. بوت‌لودر کدی است که سیستم‌عامل را در حافظه بارگذاری می‌کند.

<sup>1</sup> Rules

<sup>2</sup> Variables

<sup>3</sup> Sector

<sup>4</sup> Boot Sector

<sup>5</sup> Boot Loader

یکی از روش‌های راه‌اندازی اولیه رایانه، بوت مبتنی بر سیستم ورودی/خروجی مقدماتی<sup>۶</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.

6. در xv6 در سکتور نخست دیسک قابل بوت، محتوای چه فایل 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. 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.

9. یک ثبات عام‌منظوره<sup>۷</sup>، یک ثبات قطعه<sup>۸</sup>، یک ثبات وضعیت<sup>۹</sup> و یک ثبات کنترلی<sup>۱۰</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>۱۱</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.

10. پردازنده‌های x86 دارای مدهای مختلفی هستند. هنگام بوت، این پردازنده‌ها در مد حقیقی<sup>۱۲</sup> قرار داده می‌شوند. مدی که سیستم‌عامل ام‌اس‌داس<sup>۱۳</sup> (MS DOS) در آن اجرا می‌شد. چرا؟ یک نقص اصلی این مد را بیان نمایید؟

11. آدرس‌دهی به حافظه در این مد شامل دو بخش قطعه<sup>۱۴</sup> و افسست<sup>۱۵</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>۱۶</sup> صورت می‌گیرد. به عبارت دیگر، بوت‌لودر امکان دسترسی به تمامی قابلیت‌های سیستم را دارد. در ادامه هسته به مد حفاظت‌شده<sup>۱۷</sup> تغییر مد می‌دهد (خط ۹۱۵۳). در مد حفاظت‌شده، آدرس مورد دسترسی در برنامه (آدرس منطقی) از طریق جداولی به آدرس فیزیکی حافظه<sup>۱۸</sup> نگاشت پیدا می‌کند. ساختار آدرس‌دهی در این مد در شکل زیر نشان داده شده است.

<sup>6</sup> Basic Input/Output System

<sup>7</sup> General Purpose Register

<sup>8</sup> Segment Register

<sup>9</sup> Status Registers

<sup>10</sup> Control Registers

<sup>11</sup> Privileged Access

<sup>12</sup> Real Mode

<sup>13</sup> Microsoft Disk Operating System

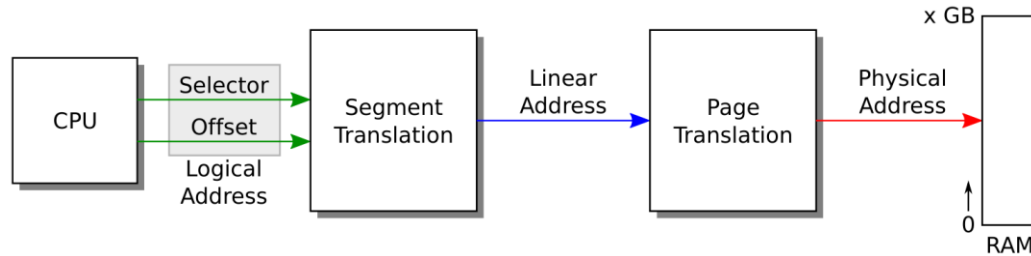
<sup>14</sup> Segment

<sup>15</sup> Offset

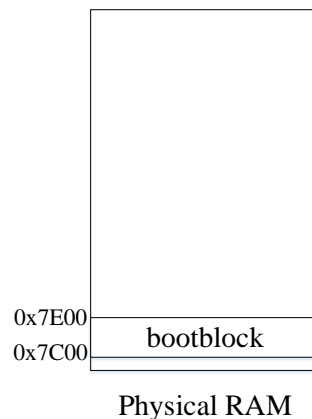
<sup>16</sup> سطوح دسترسی در ادامه پروژه توضیح داده خواهد شد.

<sup>17</sup> Protected Mode

<sup>18</sup> منظور از آدرس فیزیکی یک آدرس یکتا در سخت‌افزار حافظه است که پردازنده به آن دسترسی پیدا می‌کند.



هر آدرس در کد برنامه یک آدرس منطقی<sup>۱۹</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>۲۰</sup> و (۲) ترجمه صفحه<sup>۲۱</sup>. مفهوم ثابت‌های قطعه در این مدل تا حد زیادی با نقش آن‌ها در مدل حقیقی متفاوت است. این ثابت‌ها با تعامل با جدولی تحت عنوان جدول توصیف‌گر سراسری<sup>۲۲</sup> (GDT) ترجمه قطعه را انجام می‌دهند. به این ترتیب ترجمه آدرس در مدل 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>۲۳</sup> صرف‌نظر نموده و افسست را به عنوان آدرس منطقی در نظر گرفت و این افسست را دقیقاً به عنوان آدرس خطی<sup>۲۴</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>۲۵</sup> 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 مدل، وضعیت حافظه (فیزیکی) سیستم به صورت شکل زیر است.



<sup>19</sup> Logical Address

<sup>20</sup> Segment Translation

<sup>21</sup> Page Translation

<sup>22</sup> Global Descriptor Table

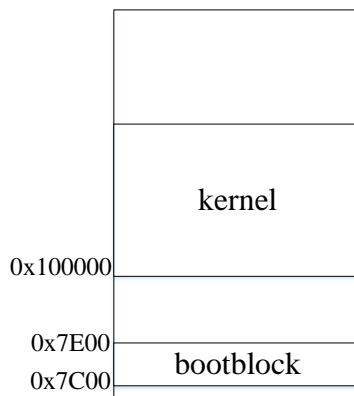
<sup>23</sup> Selector

<sup>24</sup> Linear Address

<sup>25</sup> Virtual Address

12. کد bootmain.c هسته را با شروع از سکتور بعد از سکتور بوت خوانده و در آدرس 0x100000 قرار می‌دهد.<sup>۲۶</sup> علت انتخاب این آدرس چیست؟

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



Physical RAM

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

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

### اجرای هسته xv6

هدف از entry.S ورود به هسته و آماده‌سازی جهت اجرای کد C آن است. در شرایط کنونی نمی‌توان کد هسته را اجرا نمود. زیرا به گونه‌ای لینک شده است که آدرس‌های مجازی آن بزرگتر از 0x80100000 هستند. می‌توان این مسئله را با اجرای دستور cat kernel.sym بررسی نمود. در همین راستا نگاشت مربوط به صفحه‌بندی<sup>۲۷</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>۲۸</sup> و به این ترتیب پردازنده از محل جدول نگاشت‌های جاری اطلاع خواهد داشت.

14. چرا این آدرس فیزیکی است؟

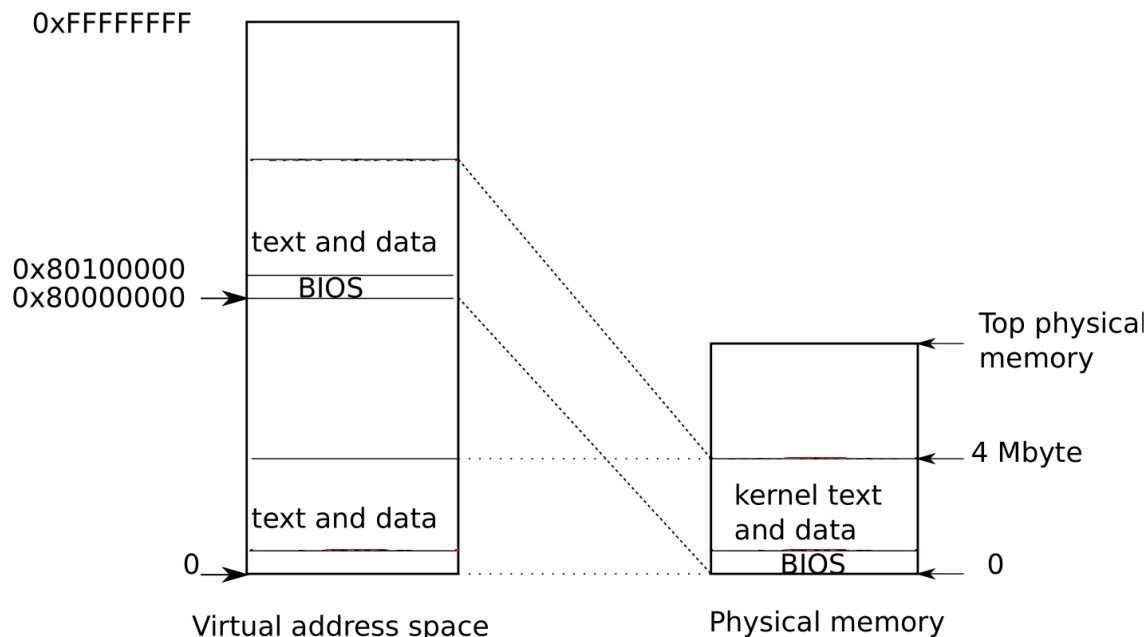
جزئیات جدول نگاشت‌ها پیچیده است. به طور ساده این جدول دارای مدخل‌هایی است که تکه‌ای پیوسته از حافظه مجازی (یا خطی) با توجه به خنثی شدن تأثیر آدرس منطقی را به تکه‌ای پیوسته به همین اندازه از حافظه فیزیکی نگاشت می‌دهد. این اندازه‌ها در هر معماری، محدود هستند. به عنوان مثال در entry.S دو تکه پیوسته چهار مگابایتی از حافظه خطی به دو تکه پیوسته چهار

<sup>26</sup> دقت شود آدرس 0x10000 تنها برای خواندن هدر فایل elf استفاده شده است و محتوای فایل هسته در 0x100000 که توسط paddr (مخفف آدرس فیزیکی) تعیین شده است، کپی می‌شود. این آدرس در زمان لینک توسط kernel.ld تعیین شده و در فایل دودویی در قالب خاصی قرار داده شده است.

<sup>27</sup> Paging

<sup>28</sup> به طور دقیق‌تر این جداول سلسله‌مراتبی بوده و آدرس اولین لایه جدول در cr3 قرار داده می‌شود.

مگابایتی از حافظه فیزیکی نگاشت داده شده است. هر تکه پیوسته یک صفحه<sup>۲۹</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 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.<sup>۳۰</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>۳۱</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>۳۲</sup> (CPL) است. سطح دسترسی در پردازنده‌های x86 از صفر تا سه متغیر بوده که صفر و سه به ترتیب ممتازترین و پایین‌ترین سطح دسترسی هستند. در سیستم عامل xv6 اگر CPL=0 باشد در هسته و اگر CPL=3 باشد در سطح کاربر هستیم<sup>۳۳</sup>. تشخیص سطح دسترسی کد کنونی مستلزم خواندن مقدار ثبات CS است.<sup>۳۴</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 ۸۳۵) 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

<sup>29</sup> Page

<sup>30</sup> در xv6 از آدرس 0x80000000 به بعد مربوط به سطح هسته و آدرس‌های 0x0 تا این آدرس مربوط به سطح کاربر هستند.

<sup>31</sup> User Level Programs

<sup>32</sup> Current Privilege Level

<sup>33</sup> دو سطح دسترسی دیگر در اغلب سیستم‌عامل‌ها بلااستفاده است.

<sup>34</sup> در واقع در مد محافظت شده، دوبیت از این ثبات، سطح دسترسی کنونی را معین می‌کند. بیت‌های دیگر کاربردهای دیگری مانند تعیین افسست مربوط به قطعه در gdt دارند.

سطح کاربر این دسترسی را صورت می دهد، جلوگیری خواهد شد. در این جا اساس تفکر این است که هسته عنصر قابل اعتماد سیستم بوده و برنامه های سطح کاربر، پتانسیل مخرب بودن را دارند.

15. به این ترتیب، در انتهای `entry.S`، امکان اجرای کد `C` هسته فراهم می شود تا در انتها تابع `main()` صدا زده (خط

۱۰۶۵) شود. این تابع عملیات آماده سازی 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 می گردد (خط ۱۲۲۰) تا آدرس های 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).

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

17. علاوه بر صفحه بندی در حد ابتدایی از قطعه بندی به منظور حفاظت هسته استفاده خواهد شد. این عملیات توسط

`seginit()` انجام می گردد. همان طور که ذکر شد، ترجمه قطعه تأثیری بر ترجمه آدرس منطقی نمی گذارد. زیرا تمامی

قطعه ها اعم از کد و داده روی یکدیگر می افتند. با این حال برای کد و داده های سطح کاربر پرچم `SEG_USER` تنظیم شده است. چرا؟ (راهنمایی: علت مربوط به ماهیت دستورالعمل ها و نه آدرس است).

### اجرای نخستین برنامه سطح کاربر

تا به این لحظه از اجرا فضای آدرس حافظه هسته آماده شده است. بخش زیادی از مابقی تابع `main()`، زیرسیستم های مختلف هسته را فعال می نماید. مدیریت برنامه های سطح کاربر مستلزم ارائه انتزاعاتی برای ایجاد تمایز میان این برنامه ها و برنامه مدیریت آن ها است. کدی که تاکنون اجرا می شد را می توان برنامه مدیریت کننده سیستم و برنامه های سطح کاربر دانست.

18. جهت نگهداری اطلاعات مدیریتی برنامه های 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 `kpgdir` است که آدرس جدول مربوط به هر برنامه سطح کاربر را نگهداری می کند. مشاهده می شود که این آدرس با آدرس مربوط به جدول کد مدیریت کننده سیستم که در `kpgdir` برای کل سیستم نگهداری شده بود، متفاوت است. تا پیش از فراخوانی `userinit()` (خط ۱۲۳۵) تقریباً تمامی زیرسیستم های هسته فعال شده اند. جهت ارائه واسطی با کاربر از طریق ترمینال و همچنین آماده سازی بخش هایی از هسته که ممکن است توأم با به خواب رفتن کد باشد، تابع `userinit()` فراخوانی می گردد. این تابع وظیفه ایجاد نخستین برنامه سطح کاربر را دارد. ابتدا توسط تابع `allocproc()` برای این برنامه یک ساختار `proc` تخصیص داده می شود (خط ۲۵۲۵). این تابع بخش هایی را که برنامه برای اجرا در سطح ممتاز (هسته) نیاز دارد، مقداردهی می کند. یکی از عملیات مهمی که در این تابع صورت می گیرد، مقداردهی `p->context->eip` به آدرس تابع `forkret()` است. این عمل منجر به این می شود که هنگام اجرای برنامه<sup>۳۵</sup> ابتدا `forkret()` اجرا گردد. آماده سازی بخش های باقی مانده سیستم در این تابع انجام می شود.

19. چرا به خواب رفتن در کد مدیریت کننده سیستم مشکل ساز است؟ (راهنمایی: به زمان بندی در ادامه توجه نمایید).

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

<sup>35</sup> دقت شود اجرا هنوز در کد مدیریت کننده سیستم است.



20. تفاوت این فضای آدرس هسته با فضای آدرس 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.

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

یک برنامه سطح کاربر می‌تواند برای دسترسی به سرویس‌های ممتاز سیستم به مد ممتاز (`CPL=0`) منتقل شود. به این ترتیب می‌تواند حتی به حافظه هسته نیز دسترسی داشته باشد. به منظور تغییر مد امن، سازوکارهایی مانند فراخوانی سیستمی<sup>۳۶</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 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>۳۷</sup> نام داشته و در ساختار `proc` ذخیره می‌شود.<sup>۳۸</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 `p->tf->cs` و `p->tf->ds` به `DPL_USER` مقداردهی شده‌اند. `p->tf->eip` برابر صفر شده است (خط ۲۵۳۸). 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 در خط ۲۵۵۰ `p->state` به `RUNNABLE` مقداردهی می‌شود. 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 `CF3` 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.

22. 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؟

23. برنامه معادل `initcode.S` در هسته لینوکس چیست؟

## نکات مهم

- در نهایت کدهای خود را در کنار گزارش با فرمت `pdf` در یک فایل `zip` آپلود نمایید.

<sup>36</sup> System Call

<sup>37</sup> Trap Frame

<sup>38</sup> تله لزوماً هنگام انتقال از مد کاربر به هسته رخ نمی‌دهد.



- همهی اعضای گروه باید به پروژه ی آپلود شده توسط گروه خود مسلط باشند و لزوماً نمره ی افراد یک گروه با یکدیگر برابر نیست.
- در صورت مشاهده ی هرگونه مشابهت بین کدها یا گزارش دو گروه، نمره ی ۰ به هر دو گروه تعلق می گیرد.
- تمامی سؤالات را در **کوتاه ترین اندازه ممکن** پاسخ دهید.