

بسم الله الرحمن الرحيم



گزارش پروژه اول آزمایشگاه سیستم عامل

دکتر کارگهی

دکتر زحمتکش

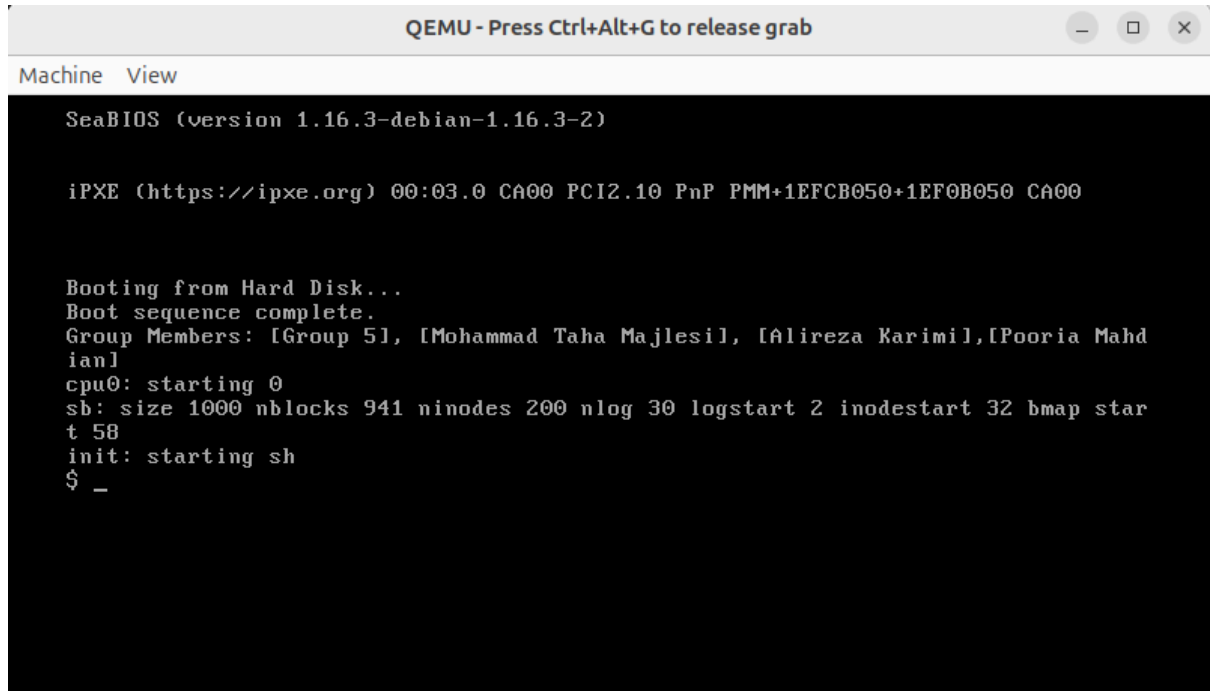
پوريا مهديان ۸۱۰۱۰۱۵۳۰

محمدطاها مجلسی کویائی ۸۱۰۱۰۱۵۰۴

علیرضا کریمی ۸۱۰۱۰۱۴۹۲

مهر ۱۴۰۳

## پرینت کردن نام افراد:



```
QEMU - Press Ctrl+Alt+G to release grab
Machine View
SeaBIOS (version 1.16.3-debian-1.16.3-2)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1EFCB050+1EF0B050 CA00

Booting from Hard Disk...
Boot sequence complete.
Group Members: [Group 5], [Mohammad Taha Majlesi], [Alireza Karimi], [Pooria Mahdian]
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ _
```

## پاسخ به سوالات

### سوال ۱: سه وظیفه اصلی سیستم عامل را نام ببرید.

1. سیستم عامل سخت افزار سطح پایین را مدیریت و از دید برنامه ها پنهان می کند، به طوری که مثلاً یک نرم افزار واژه پرداز نیازی ندارد که نگران نوع سخت افزار دیسکی که استفاده می شود باشد.
2. همچنین، سیستم عامل سخت افزار را بین چندین برنامه به اشتراک می گذارد تا آن ها به طور همزمان (یا به نظر همزمان) اجرا شوند.
3. سیستم عامل روش های کنترلی برای تعامل برنامه ها فراهم می کند تا بتوانند داده ها را با یکدیگر به اشتراک بگذارند یا با هم کار کنند.

## سوال ۲: فایل‌های اصلی سیستم عامل xv6 را توضیح دهید. نام پوشه اصلی فایل‌های هسته سیستم عامل، فایل‌های سرایند و فایل سیستم در سیستم عامل لینوکس چیست؟ در مورد محتویات آن مختصراً توضیح دهید.

فایل‌های سیستم‌عامل xv6 به دسته‌های مختلفی تقسیم می‌شوند که هر دسته مسئول مدیریت بخش خاصی از سیستم‌عامل است، مانند مدیریت فرآیندها، حافظه، سیستم فایل و فراخوانی‌های سیستمی. در اینجا توضیح مختصری از فایل‌های اصلی آمده است:

### 1. سرایندهای اصلی (Basic Headers):

این فایل‌ها شامل تعریف انواع داده‌ها، ساختارها، ثوابت سیستم و پارامترهای مورد استفاده در سراسر سیستم‌عامل هستند. همچنین جزئیات مربوط به معماری مانند ساختارهای حافظه و فرآیندها را در بر می‌گیرند.

مثال‌ها:

param.h: شامل پارامترهای سیستمی مانند حداکثر تعداد فرآیندها.

types.h: انواع پایه‌ای داده مانند uint و uchar را تعریف می‌کند.

memlayout.h: جزئیاتی درباره نحوه چیدمان حافظه و فضای آدرس‌دهی دارد.

### 2. ورود به سیستم‌عامل (Entering xv6):

این دسته شامل فایل‌های C و اسمبلی است که مسئولیت راه‌اندازی اولیه سیستم‌عامل و تنظیم CPU و حافظه را بر عهده دارند.

مثال‌ها:

entry.S: کد اسمبلی برای تنظیم CPU و تغییر به حالت محافظت شده.

main.c: تابع اصلی که سیستم را راه‌اندازی و اولین فرآیند را اجرا می‌کند.

### 3. قفل‌ها (Locks):

این فایل‌ها مکانیزم‌های همگام‌سازی را مدیریت می‌کنند تا از وقوع شرایط رقابتی (Race Conditions) جلوگیری کنند و دسترسی ایمن به منابع مشترک بین فرآیندهای متعدد را فراهم آورند.

مثال‌ها:

spinlock.c: پیاده‌سازی قفل چرخشی، که یک مکانیزم قفل ساده است.

sleeplock.c: پیاده‌سازی قفل‌های خواب که اجازه می‌دهند فرآیندها در هنگام انتظار برای آزاد شدن قفل به خواب بروند.

#### 4. فرآیندها (Processes):

این فایل‌ها مدیریت فرآیندها، شامل ایجاد، زمان‌بندی و جابجایی بین فرآیندها را انجام می‌دهند.  
مثال‌ها:

proc.c: مدیریت ایجاد، زمان‌بندی و پایان فرآیندها.

trap.c: مدیریت فراخوانی‌های سیستمی و وقفه‌ها که در طول اجرای فرآیند رخ می‌دهند.

#### 5. فراخوانی‌های سیستمی (System Calls):

این فایل‌ها فراخوانی‌های سیستمی را پیاده‌سازی می‌کنند که به برنامه‌های کاربری اجازه می‌دهند از هسته خدمات درخواست کنند.

مثال‌ها:

syscall.c: مدیریت فراخوانی‌های سیستمی.

sysproc.c: پیاده‌سازی فراخوانی‌های سیستمی مربوط به کنترل فرآیندها، مانند fork و exit.

#### 6. سیستم فایل (File System):

این فایل‌ها مسئول پیاده‌سازی سیستم فایل هستند، شامل خواندن، نوشتن و مدیریت فایل‌ها و دایرکتوری‌ها.

مثال‌ها:

fs.c: پیاده‌سازی سیستم فایل، شامل مدیریت توصیف‌گرهای فایل و inodes.

file.c: مدیریت فایل‌های تکی و توصیف‌گرهای فایل.

log.c: پیاده‌سازی ثبت عملیات برای حفظ یکپارچگی سیستم فایل.

#### 7. لوله‌ها (Pipes):

این فایل‌ها پیاده‌سازی مکانیزم‌های لوله‌ها را انجام می‌دهند که امکان تبادل داده بین فرآیندها را فراهم می‌کنند. نوشتن در یک طرف لوله، اجازه خواندن از طرف دیگر آن را می‌دهد و همچون بافر عمل می‌کند.  
مثال:

pipe.c: پیاده‌سازی لوله‌ها، شامل توابعی برای خواندن و نوشتن داده در لوله‌ها.

#### 8. عملیات روی رشته‌ها (String Operations):

این فایل‌ها شامل توابع کمکی برای عملیات روی رشته‌ها هستند، مانند کپی‌کردن، مقایسه و محاسبه طول رشته‌ها.

مثال:

string.c: شامل توابعی مانند strcmp، memmove و strlen.

#### 9. سخت‌افزار سطح پایین (Low-Level Hardware):

این فایل‌ها تعاملات سطح پایین با سخت‌افزار، مانند مدیریت وقفه‌ها و دستگاه‌های سخت‌افزاری مثل کنترلر وقفه پیشرفته (APIC) را مدیریت می‌کنند.  
مثال‌ها:

lapic.c: مدیریت Local APIC، که مسئولیت مدیریت وقفه‌ها را دارد.  
ioapic.c: مدیریت I/O APIC که وقفه‌ها را از دستگاه‌های خارجی به CPU هدایت می‌کند.

#### 10. برنامه‌های کاربری (User-Level):

این فایل‌ها تعاملات بین برنامه‌های کاربری و هسته را مدیریت می‌کنند. شامل توابعی هستند که فرآیندهای سطح کاربر و فراخوانی‌های سیستمی را مدیریت می‌کنند.  
مثال‌ها:

initcode.S: کدی که هنگام ایجاد اولین فرآیند کاربر اجرا می‌شود.  
usys.S: نقاط ورودی فراخوانی‌های سیستمی برای برنامه‌های کاربری را تعریف می‌کند.

#### 11. بوت‌لودر (Bootloader):

این فایل‌ها مسئولیت بارگذاری هسته در حافظه و آغاز اجرای سیستم‌عامل را بر عهده دارند.  
مثال‌ها:

bootmain.c: هسته را از دیسک به حافظه بارگذاری می‌کند.  
bootasm.S: کد اسمبلی که CPU را تنظیم کرده و به نقطه ورود هسته می‌پردازد.

#### 12. لینک‌دهی (Linking):

این فایل‌ها نحوه سازمان‌دهی بخش‌های مختلف هسته را در حافظه هنگام ساخت سیستم‌عامل تعیین می‌کنند.

مثال:

kernel.ld: یک اسکریپت لینک‌دهی که چیدمان حافظه هسته و مکان بارگذاری بخش‌های مختلف مانند متن، داده و bss را تعریف می‌کند.

پوشه هسته سیستم‌عامل لینوکس در مسیر /usr/src/linux/kernel/ قرار دارد و شامل فایل‌های مدیریت فرآیندها، فراخوانی‌های سیستمی، حافظه و درایورهای سخت‌افزار است.

پوشه هدر فایل‌ها در مسیر /usr/src/linux/include/ است که ساختارهای داده و پروتوتایپ‌ها را در سراسر هسته تعریف می‌کند.

پوشه سیستم فایل در مسیر /usr/src/linux/fs/ قرار دارد و مدیریت سیستم فایل، از جمله باز کردن، خواندن و نوشتن فایل‌ها را انجام می‌دهد.

## سوال ۳: دستور make -n را اجرا نمایید. کدام دستور،

## فایل نهایی هسته را می‌سازد؟

این دستور به شما نشان خواهد داد که چه فرمان‌هایی توسط دستور make اجرا خواهند شد، بدون آنکه آن فرمان‌ها را اجرا کند. پس فایل نهایی هسته توسط همان دستور make در اجراهای بعد ساخته خواهد شد.

## سوال ۴: در Makefile متغیرهایی به نامهای UPROGS و

## ULIB تعریف شده است. کاربرد آنها چیست؟

در فایل Makefile سیستم‌عامل xv6، متغیرهای ULIB و UPROGS نقش مهمی در ساخت و سازماندهی برنامه‌ها و کتابخانه‌های سطح کاربر ایفا می‌کنند.

1. ULIB (کتابخانه‌های سطح کاربر):

هدف: ULIB شامل کتابخانه‌های سطح کاربر است که به برنامه‌های کاربری لینک می‌شوند. این کتابخانه‌ها توابع پایه‌ای لازم برای تعامل برنامه‌های کاربر با هسته و انجام وظایف ضروری را فراهم می‌کنند.

مثال‌هایی از فایل‌های موجود در ULIB:

ulib.o: شامل توابع کمکی پایه‌ای است که توسط برنامه‌های کاربری استفاده می‌شوند.

printf.o: تابع printf را برای نمایش متن در کنسول فراهم می‌کند.

umalloc.o: مدیریت تخصیص حافظه پویا در برنامه‌های کاربری را انجام می‌دهد.

این کتابخانه‌ها واسطی بین برنامه‌های کاربری و سیستم ایجاد می‌کنند و قابلیت‌های ضروری مانند فراخوانی‌های سیستمی، مدیریت حافظه و عملیات روی رشته‌ها را فراهم می‌کنند.

2. UPROGS (برنامه‌های کاربری):

هدف: UPROGS شامل لیستی از برنامه‌های کاربری است که کامپایل شده و به تصویر سیستم فایل اضافه می‌شوند. این برنامه‌ها در فضای کاربر سیستم‌عامل xv6 قابل اجرا خواهند بود.

مثال‌هایی از فایل‌های موجود در UPROGS:

cat: برنامه‌ای که محتوای یک فایل را نمایش می‌دهد.

echo: برنامه‌ای که متن ورودی را به کاربر بازتاب می‌دهد.

sh: شل که به عنوان رابط خط فرمان برای تعامل کاربران با سیستم‌عامل عمل می‌کند.

این برنامه‌های کاربری در طول فرآیند ساخت کامپایل شده و سپس به تصویر سیستم فایل (معمولاً 'fs.img') اضافه می‌شوند تا در محیط شبیه‌سازی‌شده‌ی xv6 قابل اجرا باشند.

## سوال ۵: دستور `make qemu -n` را اجرا نمایید. دو دیسک به عنوان ورودی به شبیه ساز داده شده. محتوای آنها چیست؟ (راهنمایی: این دیسک ها حاوی سه خروجی اصلی فرایند بیلد هستند.)

دو دیسکی که به شبیه ساز QEMU داده می شوند، شامل اجزای ضروری برای اجرای سیستم عامل هستند:

### 1. `xv6.img`:

این دیسک شامل بوت لودر و فایل های ضروری برای راه اندازی سیستم است. `xv6 kernel` را به حافظه لود کرده و اجرای سیستم عامل را آغاز می کند. این دیسک شامل `bootblock` است که اولین کد اجرا شده می باشد و تضمین می کند که هسته در محل درست حافظه بارگذاری شود.

### 2. `fs.img`:

این دیسک شامل سیستم فایل استفاده شده در `xv6` است. داده های مهم مانند برنامه های کاربری (`cat`, `echo`, `grep`)، فایل های تنظیمات سیستم و منابع دیگر مورد نیاز برای اجرای سیستم عامل را در خود دارد. سیستم فایل همچنین شامل فایل هایی است که در طول فرآیند ساخت ایجاد شده اند، مانند فایل های `README` و برنامه های کاربری.

خروجی های اصلی فرآیند ساخت:

1. **اجرای بوت لودر:** این خروجی هنگامی تولید می شود که بوت لودر (`xv6.img`) هسته `xv6` را به حافظه بارگذاری کرده و کنترل سیستم را به آن منتقل می کند.

2. **راه اندازی هسته و سیستم فایل:** پس از بارگذاری هسته، سیستم فایل از `fs.img` گرفته می شود و اولین برنامه کاربری (`init`) شروع به اجرا می کند.

3. **اجرای برنامه های کاربری:** این شبیه سازی اجازه می دهد برنامه های کاربری از `fs.img` اجرا شوند، مانند `cat`, `grep`، و `echo`، و محیطی شبیه سازی شده برای اجرای دستورات و تعامل با سیستم عامل فراهم می آورد.

## سوال ۸: هدف از استفاده دستور `objcopy` در فرآیند اجرای make چیست؟

دستور objcopy در فرآیند make برای تبدیل فرمت فایل‌ها و حذف اطلاعات اضافی (مانند بخش‌های اضافی) از فایل‌های آجکت استفاده می‌شود. این بهینه‌سازی باعث کاهش اندازه فایل می‌شود و فایل را به فرمت باینری خام تبدیل می‌کند که برای اجرا در محیط‌هایی مانند بوت‌لودر مناسب است. بنابراین، objcopy فایل‌ها را برای استفاده در محیط‌های خاص مانند فرآیندهای بوت و سیستم‌های سطح پایین آماده می‌کند.

هدف از objcopy را می‌توان به دو بخش اصلی تقسیم کرد:

1. حذف بخش‌های اضافی: با استفاده از فلگ -S، بخش‌های غیرضروری (مانند اطلاعات دیباگ) از فایل حذف می‌شوند، که باعث کاهش اندازه فایل می‌شود.
2. تبدیل به فرمت باینری خام: با استفاده از فلگ -O binary، فایل به فرمت باینری خام تبدیل می‌شود. این فرمت برای بارگذاری مستقیم فایل‌ها در حافظه یا اجرای آن‌ها در محیط‌های خاص، مانند بوت‌لودر، ضروری است.

در Makefile، دستور objcopy برای تبدیل فایل‌های o. به فایل‌های باینری خام استفاده می‌شود.

مثال اول:

در ساخت فایل bootblock، دستور objcopy روی فایل bootblock.o با استفاده از فلگ‌های زیر استفاده می‌شود:

-S : حذف بخش‌های غیرضروری.

-O binary : تبدیل فایل به فرمت باینری خام.

.text -j : مشخص می‌کند که تنها بخش .text، که شامل کد اجرایی است، لحاظ شود.

دستور به شکل زیر است:

```
bootblock bootblock.o -S -O binary -j .text
```

مثال دوم:

برای فایل initcode.out، دستور objcopy نیز استفاده می‌شود، اما این بار فلگ .text-j- استفاده نمی‌شود، به این معنی که تمام محتوای فایل به فرمت باینری خام تبدیل می‌شود.

```
initcode initcode.out -S -O binary
```



## سوال ۱۳: کد bootmain.c هسته را با شروع از سکتور بعد از سکتور بوت خوانده و در آدرس 0x100000 قرار می‌دهد. علت انتخاب این آدرس چیست؟

هسته xv6 در آدرس 0x100000 (یک مگابایت) لود می‌شود تا از تداخل با حافظه‌ای که توسط BIOS و بوت‌لودر استفاده می‌شود جلوگیری کند. حافظه زیر 1 مگابایت برای این اجزا رزرو شده است. همچنین، این آدرس به سیستم اجازه می‌دهد تا از حالت واقعی به حالت محافظت‌شده (32 بیتی) منتقل شود و حافظه بیشتری دسترسی‌پذیر گردد. این روش به بوت‌لودر و هسته کمک می‌کند که بدون تداخل با یکدیگر عمل کنند.

## سوال ۱۸:

پرچم SEG\_USER در سیستم‌عامل xv6 نقش مهمی در جداسازی کد و داده‌های مربوط به برنامه‌های کاربری از بخش‌های حساس هسته ایفا می‌کند. این پرچم باعث می‌شود که برنامه‌های کاربری نتوانند به بخش‌های محافظت‌شده حافظه هسته دسترسی داشته باشند.

عملکرد پرچم SEG\_USER:

1. تفکیک دسترسی‌ها:

پرچم SEG\_USER تضمین می‌کند که کد و داده‌های مربوط به برنامه‌های کاربری که در حالت کاربر (user mode) اجرا می‌شوند، به بخش‌های هسته‌ای سیستم‌عامل دسترسی پیدا نکنند. این کار امنیت سیستم را افزایش می‌دهد و از برنامه‌های کاربری در برابر آسیب به سیستم محافظت می‌کند.

2. مدیریت انتقال به حالت هسته (kernel mode):

هنگامی که یک برنامه کاربری نیاز به دسترسی به خدمات هسته، مانند خواندن فایل یا دسترسی به سخت‌افزار، داشته باشد، باید به حالت هسته (kernel mode) منتقل شود. پرچم SEG\_USER به سیستم کمک می‌کند تا این انتقال به درستی مدیریت شود و اطمینان حاصل کند که تنها برنامه‌های مجاز می‌توانند به حالت هسته دسترسی پیدا کنند.

3. افزایش امنیت:

این پرچم با مدیریت دقیق دسترسی‌ها به حافظه، به حفاظت از داده‌های حساس هسته کمک می‌کند. SEG\_USER تعیین می‌کند که چه برنامه‌هایی می‌توانند به بخش‌های خاصی از حافظه دسترسی داشته باشند و از دسترسی‌های غیرمجاز به داده‌ها جلوگیری می‌کند.

4. جداسازی فضای کاربر و هسته:

با استفاده از SEG\_USER، سیستم عامل می تواند اطمینان حاصل کند که بخش های هسته ای و برنامه های کاربری از یکدیگر جدا می مانند و هیچ گونه دسترسی غیرمجاز از سمت برنامه های کاربری به اطلاعات هسته رخ نمی دهد.

## سوال ۱۹: جهت نگهداری اطلاعات مدیریتی برنامه های سطح کاربر ساختاری تحت عنوان struct proc ارائه شده است. اجزای آن را توضیح داده و ساختار معادل آن در سیستم عامل لینوکس را بیابید.

ساختار proc در xv6 تمام اطلاعات مورد نیاز برای مدیریت یک فرآیند را نگه می دارد. سیستم عامل از این ساختار برای پیگیری وضعیت، حافظه و داده های کنترلی هر فرآیند استفاده می کند. در ادامه، فیلدهای کلیدی ساختار proc و وظایف آنها توضیح داده شده است:

### 1. state (وضعیت فرآیند):

این فیلد وضعیت فعلی فرآیند را پیگیری می کند. وضعیت های ممکن عبارتند از:  
RUNNING: فرآیند در حال اجرا بر روی CPU است.  
SLEEPING: فرآیند منتظر یک منبع (مثل I/O) است.  
RUNNABLE: فرآیند آماده اجرا است و منتظر تخصیص CPU است.  
ZOMBIE: فرآیند پایان یافته، اما هنوز توسط والد جمع آوری نشده است.  
وضعیت های دیگر شامل UNUSED, EMBRYO, و WAITING.

### 2. pgdir (جدول صفحات):

این فیلد به جدول صفحات فرآیند اشاره دارد که برای مدیریت ترجمه آدرس های مجازی به فیزیکی استفاده می شود. هر فرآیند جدول صفحه خود را دارد تا فضای آدرس دهی آن به حافظه فیزیکی نگاشت شود.

### 3. kstack (پشته هسته):

به پایین پشته هسته فرآیند اشاره دارد. این پشته زمانی استفاده می شود که فرآیند به حالت هسته (مثل فراخوانی سیستمی) وارد می شود تا داده های موقتی را ذخیره کند.

### 4. context (زمینه CPU):

این فیلد مقادیر رجیسترهای CPU را هنگام تعویض زمینه (context switch) ذخیره می‌کند. وقتی فرآیند در حال اجرا نیست، وضعیت فعلی آن در این فیلد ذخیره می‌شود تا فرآیند بعداً از همان جا که متوقف شده بود، ادامه یابد.

#### 5. name (نام فرآیند):

این فیلد نام فرآیند را ذخیره می‌کند که عمدتاً برای اشکال‌زدایی استفاده می‌شود. نام به توسعه‌دهندگان یا مدیران سیستم کمک می‌کند فرآیند را پیگیری کنند.

#### 6. pid (شناسه فرآیند):

این فیلد شناسه یکتای فرآیند را ذخیره می‌کند که به هر فرآیند در هنگام ایجاد اختصاص داده می‌شود. این شناسه توسط سیستم و سایر فرآیندها برای مدیریت فرآیندها استفاده می‌شود.

#### 7. Trap Frame (tf):

این فیلد مقادیر رجیسترهای CPU را در لحظه وقوع یک وقفه یا فراخوانی سیستمی ذخیره می‌کند. این اجازه می‌دهد که سیستم پس از مدیریت وقفه یا فراخوانی سیستمی، وضعیت فرآیند را بازگرداند و فرآیند ادامه یابد.

در سیستم عامل لینوکس، ساختار مشابهی به نام task\_struct برای مدیریت فرآیندها وجود دارد. این ساختار شامل فیلدهایی برای:

state (وضعیت فرآیند)

page tables (مشابه pgdir در xv6)

stack (پشته هسته)

pid (شناسه فرآیند)

و فیلدهای دیگر برای برنامه‌ریزی و چندوظیفه‌ای.

## سوال ۲۳: کدام بخش از آمادہ سازی 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؟

در سیستم عامل xv6 برخی بخش‌ها بین تمام هسته‌های CPU مشترک هستند و برخی بخش‌ها برای هر هسته به صورت خصوصی مدیریت می‌شوند.

### بخش‌های مشترک:

**جدول صفحات (Page Table):** بین همه هسته‌ها به اشتراک گذاشته می‌شود و آدرس‌های مجازی را به فیزیکی نگاشت می‌کند تا دسترسی به حافظه برای همه هسته‌ها یکسان باشد.

**سیستم I/O:** دستگاه‌های ورودی/خروجی مانند کنترلرهای دیسک و رابط‌های شبکه برای همه هسته‌ها مشترک هستند، زیرا فرآیندها در هسته‌های مختلف ممکن است به این دستگاه‌ها نیاز داشته باشند.

**مکانیزم‌های قفل (Locks):** برای همگام‌سازی دسترسی به منابع مشترک و جلوگیری از شرایط رقابتی، قفل‌هایی مانند spinlock و sleeplock بین تمام هسته‌ها مشترک هستند.

### بخش‌های خصوصی:

**مدیریت وقفه‌ها (Interrupt Handling):** هر هسته CPU دارای APIC (Advanced Programmable Interrupt Controller) محلی خود است که وقفه‌ها را به صورت مستقل مدیریت می‌کند.

**پشته هسته (Kernel Stack):** هر هسته پشته هسته‌ای خود را دارد که هنگام ورود به حالت هسته (مثلاً در زمان فراخوانی سیستمی) از آن استفاده می‌کند.

**زمان‌بند (Scheduler):** هر هسته زمان‌بند مخصوص به خود را دارد که فرآیندهای آن هسته را مدیریت می‌کند، بدون نیاز به کنترل مرکزی.

زمان‌بند روی همه هسته‌ها اجرا می‌شود و هر هسته دارای زمان‌بند (scheduler) مستقل خودش است و می‌تواند به محض بیکار شدن، فرآیندی را از صف مشترک فرآیندها اجرا کند. این ویژگی باعث توزیع متعادل بار کاری بین هسته‌ها می‌شود و از تمرکز بار روی یک هسته جلوگیری می‌کند.