1) سیستم عامل Unix توسط Thompson Ken و Dennis Ritchie نوشته شده است. سیستم عامل xv6 بر اساس Unix(نسخه 6) پیاده سازی شده است. xv6 از ساختار Unix v6 استفاده میکند ولی با ANSI C برای پردازنده های مبتنی بر x86 پیاده سازی شده است.

دلایل: در فایل x86.h میتوانیم دستورات assembly مختص پردازنده های مبتنی بر x86 را مشاهده کنیم. در فایل asm.h نیز استفاده از معماری x86 ذکر شده است. در فایل mmu.h از mmu مربوط به x86 استفاده شده است. در traps.h نیز میتوانیم مشاهده کنیم که trap ها مخصوص معماری x86 پیاده سازی شده اند.

2) هر پردازه در xv6 از user space memory که شامل stack و data و instruction و همینطور از استیت پیش پردازش که مربوط به کرنل است، تشکیل شده است. xv6 می تواند پردازه های share-time داشته باشد. بین آن CPU های موجود که برای مجموعه ای از پردازش ها منتظر اجرا هستند، سوییچ می کند. زمانی که پردازه ای اجرا نمی شود ، xv6 رجیستر های CPU آن را ذخیره می کند تا برای پردازش بعدی استفاده کند. کرنل به هر پردازه یک pidنسبت می دهد.

3)

a) descriptor file در سیستم عامل‌های مبتنی بر UNIX یک اشاره‌گر به منبع یا مقصد داده‌ها در سیستم فایل است. این فایل‌ها معمولاً با یک عدد صحیح شناسایی می‌شوند و توسط سیستم عامل برای دسترسی به داده‌ها استفاده می‌شوند. به عنوان مثال، یک فایل متنی معمولی در سیستم عامل UNIX به عنوان یک descriptor file باز می‌شود و یک اشاره‌گر به آن در اختیار برنامه‌ی کاربردی قرار می‌گیرد.

b) Pipe به دو فرآیند امکان مبادله‌ی داده‌ها با یکدیگر بدون نیاز به file descriptor را می دهد. یک pipe معمولاً شامل دو file descriptor است که به‌صورت یک‌طرفه بین دو فرآیند ایجاد می‌شود و به‌عنوان یک کانال ارتباطی بین آن‌ها عمل می‌کند. فرآیندها می‌توانند داده‌ها را در pipe بنویسند و فرآیند دیگر می‌تواند آن‌ها را از pipe بخواند. این روش معمولاً برای ارتباط بین دو فرآیند مختلف که به‌صورت همزمان اجرا می‌شوند، استفاده می‌شود. به‌عنوان مثال، فرآیند پدر می‌تواند داده‌هایی را در pipe بنویسد و فرآیند فرزند آن‌ها را بخواند.

4) fork یک فرآیند جدید را شروع می کند که یک کپی از چیزی است که آن را فراخوانی می کند، در حالی که exec تصویر فرآیند فعلی را با تصویر دیگری (متفاوت) جایگزین می کند. هر دو پردازش والد و فرزند به طور همزمان در مورد ()fork اجرا می شوند در حالی که Control هرگز به برنامه اصلی باز نمیگردد مگر اینکه یک خطای ()execوجود داشته باشد.
ادغام نکردن فراخوانی ها برای ساختن یک پردازش و لود کردن پردازش استفاده های هوشمندانه ای در I/O redirection دارد. برای جلوگیری از پردازش های تکراری و سریع جایگزین کردن آن ، کرنل ها با استفاده از تکنیک های virtual memory ، پیاده سازی fork را برای این زمینه بهبود می بخشند.

5) مدیریت منابع – ایجاد بستری اجرای برنامه های کاربردی – ارائه خدمات سیستمی مثل خدمات شبکه و مدیریت فایل ها

6)

a) Basic header ها یه سری کتابخانه هستند که شامل تابع های مختلف هستند. xv6 entering شامل یه سری فایل هست که مربوط به اجرا شدن هسته xv6 است. Processes مربوط به مدیریت پروسس ها هست.

b) نام پوشه‌ی اصلی فایل‌های هسته سیستم عامل لینوکس به طور عمومی /boot نامیده می‌شود. در این پوشه فایل‌هایی مانند vmlinuz و initrd.img وجود دارند که برای بارگذاری و اجرای سیستم عامل لینوکس در مراحل اولیه‌ی بوت استفاده می‌شوند.
پوشه‌ی /lib/modules شامل فایل‌های سرایند هسته است. این فایل‌ها ماژول‌های هسته لینوکس هستند که در حین استفاده از سیستم عامل بارگذاری و به کار می‌روند.
فایل سیستم (File System) در سیستم عامل لینوکس بیانگر ساختار دایرکتوری‌ها و فایل‌های موجود در سیستم فایل لینوکس است. سیستم فایل لینوکس بر پایه‌ی ساختار سلسله‌مراتبی دایرکتوری‌ها (مانند /var/ ,usr, /etc و ...) ساختاردهی شده است و دسترسی به فایل‌ها و دایرکتوری‌ها در آن با استفاده از مسیرهایی مانند / و /home/user صورت می‌گیرد.

7) -

8) UPROGS : شامل لیستی از برنامهء کاربر هایی که باید ساخته شوند و کاربر می تواند از آنها استفاده کند را نگه می دارد. به همین دلیل اگر بخواهیم برنامهء کاربر ای بسازیم و اضافه کنیم باید در این قسمت نام اش را ذکر کنیم.
برای مثال برنامه یا دستور هایی که کاربر میتوانند اجرا کنند شامل
_cat_echo\ _forktest\ _grep\ _init\ _kill_In\ _Is\ _mkdir\ _rm\ _sh\ _stressfs
که می بینیم نام شان همگی در UPROG لیست شده اند.
ULIB : برای اینکه کاربران بتوانند برنامه هایی توسعه دهند، یکسری دستورات و کتابخانه هایی برای آنها ساخته شده است که این متغیر مسئول نگه داشتن آنهاست مثل فایل ulib.o که دستورهای strapy و memset و ... را دارد فایل usys.o که سیستم کال ها را شامل می شود، umaloco که شامل دستورهایdynamic memory allocation ها است و printf.o .

9) دیسک اول: حاوی فایل های bootblock, kernel, initcode است. فایل bootblock، کد بوتلاژ اولیه برای بارگذاری هسته سیستم عامل xv6 است. فایل kernel، فایل اصلی هسته سیستم عامل xv6 است که شامل کد اصلی سیستم عامل و دستورالعمل‌های مربوط به بوت سیستم است. فایل initcode، فایل اجرایی اولیه است که در هنگام بوت کردن سیستم عامل اجرا می‌شود و تمامی پردازش‌ها از طریق آن شروع می‌شوند.
دیسک دوم: حاوی فایل fs.img است. فایل fs.img، یک فایل سیستم فایل کامل است که حاوی ساختارهای دایرکتوری، فایل‌ها و سایر منابع سیستم فایل است. این فایل برای تست عملکرد سیستم فایل xv6 استفاده می‌شود.

10) دو فایل bootmain.c و bootasm.S وجود دارد.

11) همه آبجکت فایل ها ELF اند ولی فایل bootblock این هدر را ندارد و در واقع یک فایل raw binary هدری ندارد و این فایل در سکتور اول ممووری لود می شود سکتور اول مربوط به بوت است و کرنل از سکتور دوم به بعد است از آنجا که وقتی سیستم بالا می آید CPU از سکتور اول که مربوط به بوت است شروع به اجرای دستورات می کند و چون هنوز سیستم عامل اجرا نشده است نمی تواند فایل ELF را متوجه شود این سیستم عامل است که ELF را میشناسد به همین دلیل فقط قسمت تکست bootblock.o که مربوط به instruction ها است را جدا می شود و در حافظه قرار می گیرد تا CPU بتواند آن را

اجرا کند همچنین این کد به معماری CPU وابسته است. به همین دلیل برای قابل فهم کردن انسان آن باید معماری آن را که i386 است و ۱۶ بیت است ذکر کنیم.

12) برای اینکه قسمت تکست فایل bootblock.o را که همان instruction ها هستند را بتواند کپی کند و فایل خام باینریِ bootblockرا بسازد از objcopy استفاده شده است.

13) زیرا برای بعضی ویژگی ها نیاز داریم از قابلیتهای سطح سیستم استفاده کنیم. همچنین اجرای کد اسمبلی کم حجم تر و سریع تر است. کد bootasm.S پردازنده را به حالت محافظت شده ۳۲ بیت برده و پس از آن تابع bootmain از فای bootmain.c صدا زده می شود.

14) **ثبات عام منظوره:** EAX, EBX, ECX, EDX, ESI و EDI رجیسترهایی هستند که برای نگهداری متغیر ها حین محاسبات است و همچنین رجیستر EIP (PC).

ثبات قطعه: اشاره گر به قطعه های مختلف مثل استک ، داده و کد در آنها نگهداری می شود مثل SS که پوینتر به استک یا CS که پوینتر به کد را نگه می دارند.

ثبات وضعیت: اطلاعات وضعیت کنونی پردازنده را نگه می دارند مثل EFLAGS که اطلاعتی درباره flag های zero و overflowو ... را نگه می دارد.

ثبات کنترلی: مسئول تغییر و یا کنترل پردازنده را بر عهده دارند مثل CRO که وظیفه های مثل فعال کردن protected moder و یا سوییچ کردن بین تسک ها را دارد.

15) _
16) _
17) _

18) https://github.com/torvalds/linux/blob/master/arch/arm64/kernel/entry.S

19) اگر این بخش را به صورت مجازی در نظر می گرفتیم باز باید یک بخش فیزیکی در نظر می گرفتیم تا این بخش مجازی را مشخص کند، یعنی در نهایت نیاز به بخش فیزیکی بود.

20) _
21) _

22) زیرا بتوان تفاوتی بین پردازه های سطح کاربر و پردازه های سطح کرنل ایجاد نمود. از آنجایی که محتوای هر دو این پردازنده ها در یک فضای فیزیکی قرار گرفته اند با این کار می توان تشخیص داد که آن داده ها یا کد ها، همان داده ها و کد های سطح کاربر می باشند و اجازه دسترسی به کرنل را ندارند.

23)

- SZ :سایز حافظه متعلق به پردازه به بایت.
- pgdir :پوینتر به table page است.
- kstack :پایین استک کرنل برای این پردازه را مشخص می کند.
- state :وضعیت این پردازه را مشخص میکند.
- pid :عدد اختصاص داده شده به این پردازه.
- parent :پدر این پردازه یا به عبارت دیگر سازنده این پردازه را مشخص می کند.
- tf :چارچوب trap برای call system حال حاضر
- context :برای switching context نگهداری شده است.
- chan :اگر صفر نباشد به معنای خوابیدن پردازه است.
- killed :اگر غیر صفر باشد یعنی پردازه kill شده است.
- ofile :فایل های باز شده توسط این پردازه.
- cwd :پوشه ی کنونی را مشخص میکند.
- name :نام این پردازه.

معادل این ساختار در کرنل لینوکس در لینک زیر آمده است. (task struct)

https://github.com/torvalds/linux/blob/master/include/linux/sched.h

24) _

25) در kvmalloc یک صفحه برای آدرس های کرنل برای مدیریت پردازه ها ساخته می شود و برای کل سیستم است در صورتی که setupkvm یک صفحه برای هر پردازه می سازد که مثلا در userinit برای اولین پردازه این صفحه ساخته می شود.

26) _

27) همانطور که در کد main.c مشخص است یکسری دستورات مثل allocate کردن phsycal page و ساختن trap vector ها ، ساختن لینک لیستی از بافرها و به عهده Bootstrap processor است ولی کارهایی مثل Map کردن آدرس منطقی به آدرس مجازی که توسط تابع seginit انجام می شود بین آنها مشترک است.

28) https://github.com/torvalds/linux/blob/master/arch/arm/boot/bootp/init.S

اشکال زدایی

۱) برای مشاهده Breakpoint ها از چه دستوری استفاده میشود؟

از دستور breakpoints info maintenance استفاده میشود. همچنین دستور (break info)

۲) برای حذف یک Breakpoint از چه دستوری و چگونه استفاده میشود؟

می توان با دستور clear <filename>:<linenumber> بر یک نقطه قرار داده شده در خط linenumber از فایل filename را حذف کرد. همچنین دستورات مشاهده Breakpoint ها به هر کدام یک شماره نسبت می دهند که به فرمت del<number> می توان Breakpoint مورد نظر را حذف کرد.

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

این دستور همان back trace و یک لیست از زنجیره توابع فراخوانی شده تا رسیدن به آن Breakpoint را نشان میدهد. این کار را به کمک stack فراخوانی توابع انجام می دهد و باعث می شود بدانیم که تابع چگونه به جایگاه کنونی محل Breakpointکه می تواند یک تابع خاص یا یک آدرس از حافظه و یا خطی در منع کد باشد رسیده است.

۴) دو تفاوت دستورهای x و print را توضیح دهید. چگونه میتوان محتوای یک ثبات خاص را چاپ کرد؟

دستور print یک عبارت (EXP) می گیرد که مقدار (value) آن را با در نظر گرفتن ویژگی های مشخص شده و به فرمت مشخص شده نمایش می دهد. مبنای کارکرد دستور ، آدرس می باشد و محتوای (content) آن را نمایش می دهد. همچنین بخش FMT در این دستور (برخلاف print) لازم است و به صورت تعداد تکرار همراه با حرف فرمت (مثل u,t,a,l,c,s,f,x) و حرف اندازه (مثل h,w,g,b)

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؟

دستور info register <registerName> و info registers و info locals و info variables

بعضی عملیات ها هستند که فقط با استفاده از esi و edi قابل انجام هستند .Esi در این عملیات ها به source و edi به مکان destination اشاره می کند.

۶) به کمک استفاده از GDB ،درباره ساختار input struct 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)**

در این struct سه متغیر w, r, e وجود دارد که دو متغیر r و w ابتدای این خط ورودی را در buffer نشان می دهند. با استفاده از print مقدار این دو را مشاهده می کنیم. برای متغیر e نیز در ادامه توضیح داده شده است .به عنوان مثال در فایل console.c روی خط ۴۰۸ breakpoint قرار دادیم . بعد از اینکه یکی از cursor ها را به راست بردیم مقدار input.e یکی بیشتر شده است .پس input.e نمایانگر مقدار انتهایی خط در حال تایپ شدن در buffer است.

7) خروجی دستور های layout asm , layout src در TUI چیست؟

دستور layout src برنامه را در حالت کد منبع آن نمایش می دهد.

دستور layout asm برنامه را در حالت کد اسمبلی آن نمایش می دهد.

8) برای جابجایی میان توابع زنجیره فراخوانی جاری (نقطه توقف) از چه دستور هایی استفاده می شود؟

دستور up می تواند به فریم قبلی یا بیرونی برود. دستور down می تواند به فریم بعدی یا داخلی برود. در دستور bt با up به سمت فریم بزرگتر و با down به فریم کوچکتر می رود.