Repository Link: <https://github.com/SamanEN/Operating-System-Lab-Projects>

Latest Commit Hash: <hash>

# آشنایی با سیستم‌عامل xv6

## معماری سیستم‌عامل xv6

سیستم عامل xv6 مشابه Unix v6 نوشته شده و معماری و ساختاری شبیه به آن دارد. این سیستم عامل برای پردازنده‌های مبتنی بر x86 نوشته شده (مطابق با داکیومنت این سیستم عامل؛ xv6-rev11). علاوه بر آن در دفاع از این سخن می‌توان به فایل x86.h اشاره کرد که از دستورات پردازنده‌های x86 استفاده شده است. در دیگر فایل‌های “basic headers”، نظیر asm.h و mmu.h نیز می‌توان اشاراتی به معماری x86 مشاهده کرد.

معماری کلی سیستم عامل Unix بصورت زیر می‌باشد:

همانطور که گفته شد، معماری xv6 نیز از Unix پیروی می‌کند. این موضوع از دسته‌بندی فایل‌ها که شامل file systems، system calls، user-level و... می‌شود نیز قابل مشاهده است.

## بخش های پردازه و چگونگی اختصاص پردازنده به پردازه های مختلف

یک پردازه در xv6 از حافظۀ فضای کاربری (user-space) (شامل دستورات، داده‌ها و استک)، و وضعیت پردازه که فقط برای هسته قابل رؤیت است تشکیل شده است.

xv6 زمان را بین پردازه‌ها تقسیم می‌کند و به صورت نامحسوس پردازنده‌ها را برای اجرای دستورات به پردازه‌ها اختصاص می‌دهد. هروقت یک پردازه قرار است اجرا نشود، سیستم‌عامل register های CPU که حاوی مقادیر مورد نیاز آن پردازه بوده را ذخیره می‌کند تا دفعه بعدی که آن پردازه قرار است اجرا شود، آنها را بازگرداند.

هسته xv6 به هر پردازه یک شناسه یکتا PID (Process Identifier) اختصاص می‌دهد. با استفاده از system call ء getpid() می‌توان PID پردازۀ کنونی را دریافت کرد.

## مفهوم file descriptor و عملکرد pipe در xv6

# اضافه کردن یک متن به Boot Message

# اضافه کردن چند قابلیت به کنسول xv6

## دستور ctrl + n برای پاک کردن همه اعداد خط کنونی

## دستور ctrl + r برای برعکس کردن خط کنونی

## دستور tab برای تکمیل کردن خط کنونی

# اجرا و پیاده‌سازی یک برنامه سطح کاربر

## برنامه prime\_numbers

# مقدمه‌ای درباره سیستم‌عامل و xv6

## سه وظیفه اصلی سیستم‌عامل

## گروه های فایل های اصلی xv6

# کامپایل سیستم‌عامل xv6

## دستور make -n و کدام دستور فایل نهایی را می‌سازد؟

## متغیر های UPROGS و ULIB در Makefile

# اجرا بر روی شبیه‌ساز QEMU

## محتوای دو دیسک ورودی QEMU

# مراحل بوت سیستم‌عامل xv6

# اجرای بوت‌لودر

## محتوای سکتور نخست دیسک قابل بوت

## مقایسه فایل باینری بوت با بقیه فایل های باینری xv6 و تبدیل آن به اسمبلی

## علت استفاده از objcopy در هنگام make

## چرا برای بوت کردن فقط از فایل C استفاده نشده و اسمبلی هم هست؟

## وظیفه ثبات های x86

## نقص اصلی real mode پردازنده x86

## آدرس دهی حافظه در real mode

## کد bootmain.c چرا هسته را در آدرس 0x100000 قرار می‌دهد؟

## کد معادل entry.s در هسته لینوکس

# اجرای هسته xv6

## چرا آدرس فیزیکی است؟

## توابع entry.s را توضیح دهید و تابع معادل در هسته لینوکس را بیابید

## مختصری راجع به محتوای فضای آدرس مجازی هسته

## چرا برای کد و داده های سطح کاربر پرچم SEG\_USER تنظیم شده است؟

# اجرای نخستین برنامه سطح کاربر

## اجزای struct proc و معادل آن در لینوکس

## چرا به خواب رفتن در کد مدیریت‌کننده سیستم مشکل ساز است؟

## تفاوت فضای آدرس هسته با فضای آدرس توسط kvmalloc

## تفاوت فضای آدرس inituvm با فضای آدرس کاربر در کد مدیریت سیستم

## کدام بخش از آماده‌سازی سیستم بین تمامی هسته های پردازنده مشترک و کدام بخش اختصاصی است؟

## برنامه معادل initcode.s در هسته لینوکس

# اشکال زدایی

# روند اجرای GDB

## دستور مشاهده breakpoint ها

## دستور حذف یک breakpoint

# کنترل روند اجرا و دسترسی به حالت سیستم

## خروجی bt

## تفاوت دستور x و print

## نمایش وضعیت ثبات ها و متغیر های محلی؛ رجیستر های edi و esi

## ساختار struct input

# اشکال زدایی در سطح کد اسمبلی

## خروجی دستور های layout src و layout asm در TUI

## دستور های جابجایی میان توابع زنجیره فراخوانی جاری (نقطه توقف)