Operating Systems

Isfahan University of Technology
Electrical and Computer Engineering Department
1401 semester

Zeinab Zali

Session 3: Operating System Design and Implementation

جلسه 3: طراحی و پیاده سازی سیستم عامل

Design and Implementation

- Design and Implementation of OS is not “solvable”, but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- Start the design by defining goals and specifications
- Affected by choice of hardware, type of system
- **User** goals and **System** goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
- Specifying and designing an OS is highly creative task of **software engineering**

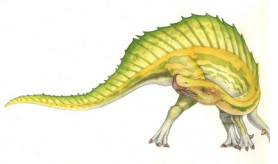
Policy and Mechanism

- **Policy**: **What** needs to be done?
  - Example: Interrupt after every 100 seconds
- **Mechanism**: **How** to do something?
  - Example: timer
- Important principle: separate policy from mechanism
- The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later.
  - Example: change 100 to 200

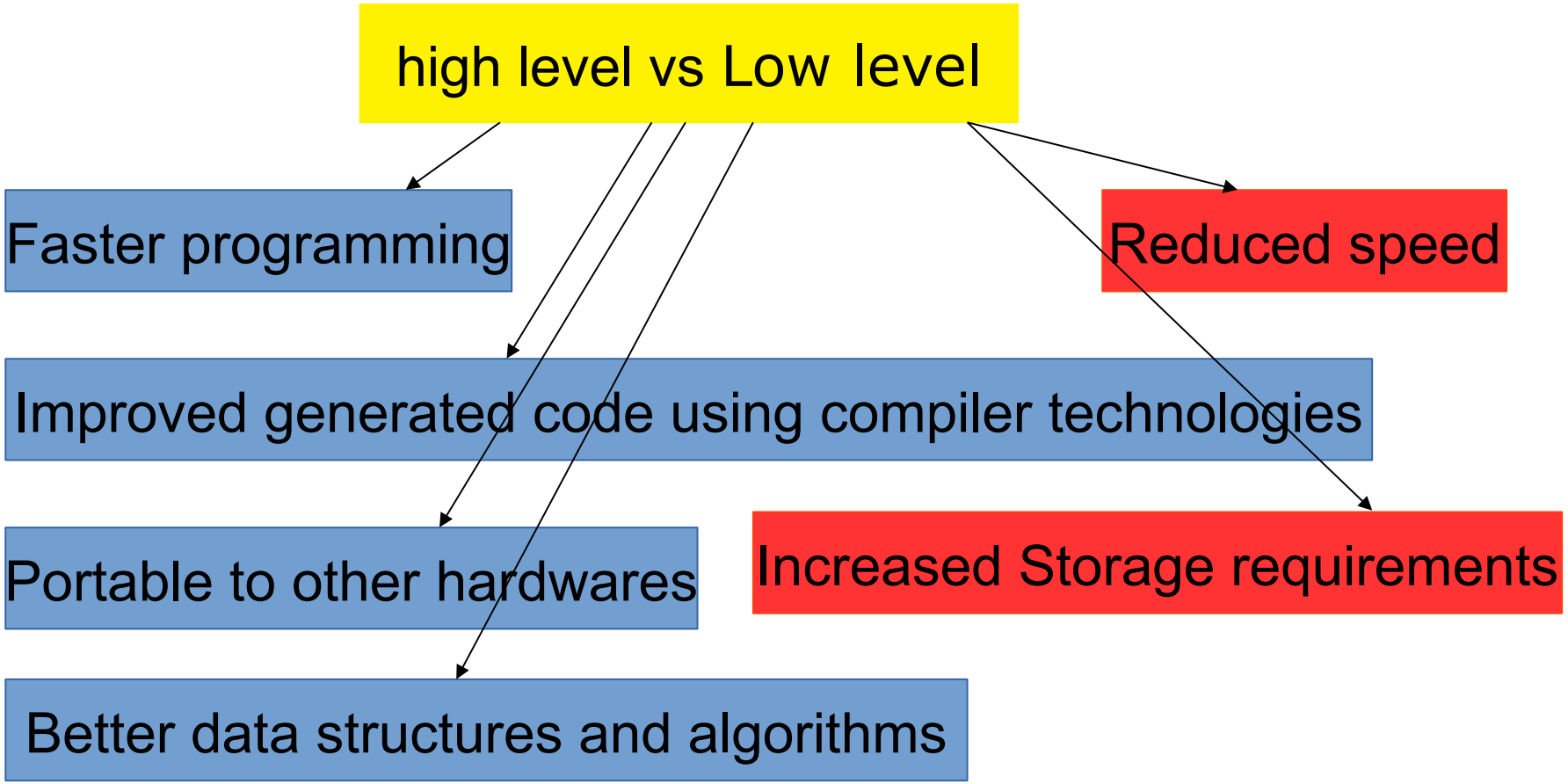
-
برای طراحی ها می تونیم یک قاعده رو در نظر بگیریم:
اینه که توی طراحی policy , mechanism رو از هم جدا کنیم
سیاست ینی این که ما سیاستمون برای یک قسمتی که میخوایم پیاده سازی کنیم مشخص باشه مثلا میخوایم یه چیزی پیاده سازی کنیم و نیاز داریم سر مواقع خاصی یه کار انجام بشه پس میگیم بعد از هر 100 ثانیه یک توقفی رخ بده که سر اون توقف اون اتفاق خاص انجام بشه
چطوری این کارو انجام میدیم؟ مثلا می تونیم از تایمر استفاده کنیم که این میشه قسمت مکانیسم
پس توی بحث طراحی این مهم است که وقتی یک هدفی داریم سیاست و مکانیسم رو ازش جدا میکنیم ینی اول فکر میکنیم برای این که این هدف رو پیاده کنیم به چه نیازی داریم ینی what که میشه policy و چطور ینی با چه روشی این کار رو انجام میدیم ینی how که میشه مکانیسم
چرا مهمه اینارو از هم جدا کنیم؟
بخاطر اینکه انعطاف پذیری توی سیستم ایجاد کنیم ینی وقتی مکانیسم رو از سیاست جدا کرده باشیم بعدا می تونیم مکانیسم های مختلفی رو استفاده کنیم براساس شرایطی که داریم و...
مثلا توی این مثال ممکنه یه روزی تصمیم بگیریم به جای هر 100 ثانیه بخوایم هر 200 ثانیه باشه

Implementation

- Much variation
  - Early OSes in assembly language
  - Then system programming languages like Algol, PL/1
  - Now C, C++
- Actually usually a mix of languages
  - Lowest levels in assembly
  - Main body in C
  - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
- More high-level language easier to **port** to other hardware
  - But slower
- **Emulation** can allow an OS to run on non-native hardware

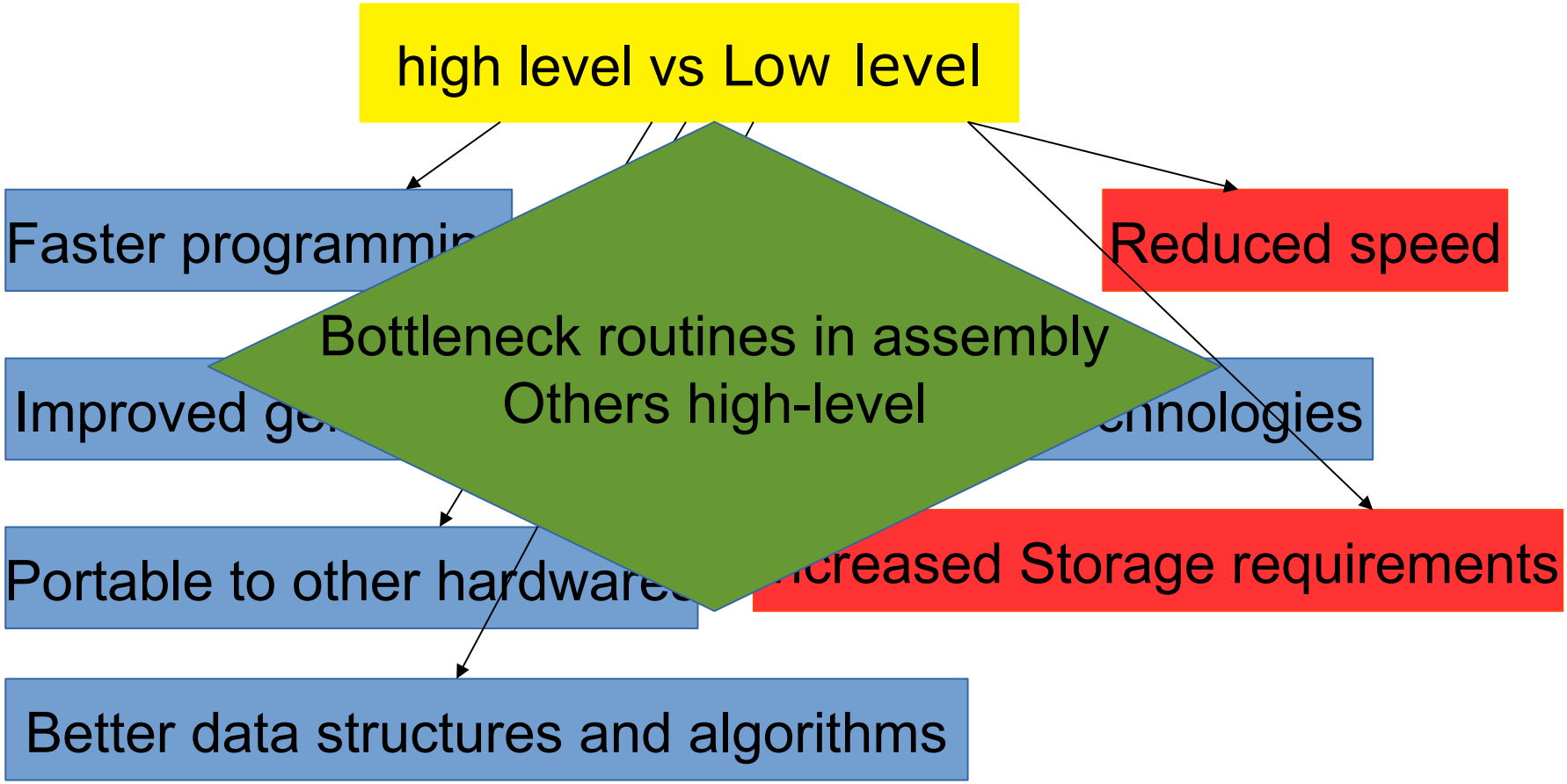


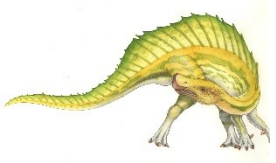
OS Implementation



مقایسه بین زبان های سطح بالا و پایین برای پیاده سازی سیستم عامل:
نکات مثبت یک زبان سطح بالا:
1- programming سریعتر است چون راحتر است چون به فهم ما نزدیک تر است
2- چون از یک کامپایلر استفاده میکنیم برای کامپایل کردنش کامپایلر خودش تکنولوژی های به کار برده که وقتی کد ماشین تولید میشه یک کد پیشرفته و خوبی تولید میشه
3- portablity زبان سطح بالا بالاتر است
4- توی زبان های سطح بالا دیتا استراکچرهای خوب و راحت برای استفاده داریم و همینطور الگوریتم ها
از طرفی سرعت اجرای برنامه توی زبان های سطح پایین بالاتر است و همینطور چون توی زبان سطح بالا از دیتا استراکچرهای پیشرفته ای استفاده می کنیم خیلی هم دقت نمیشه به میزان حافظه که استفاده میشه و در نهایت فایل های بزرگی به دست میاد و همینطور از حافظه هم خیلی بیشتر استفاده میشه تا وقتی که داریم از زبان های سطح پایین استفاده میکنیم چون توی زبان های سطح پایین همه چی دست خودمون است

OS Implementation

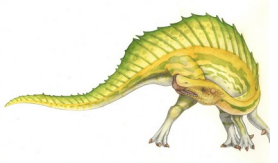




پس اون روتین هایی که از لحاظ سرعتی برامون خیلی نزدیک به سخت افزار است اونارو از زبان سطح پایین استفاده میکنیم و بقیه رو از زبان سطح بالا استفاده میکنیم

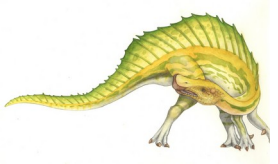
Operating System Structure

- **General-purpose** OS is very large program
- Various ways to structure ones
  - Simple structure – MS-DOS
  - More complex – UNIX
  - Layered – an abstraction
  - Microkernel – Mach



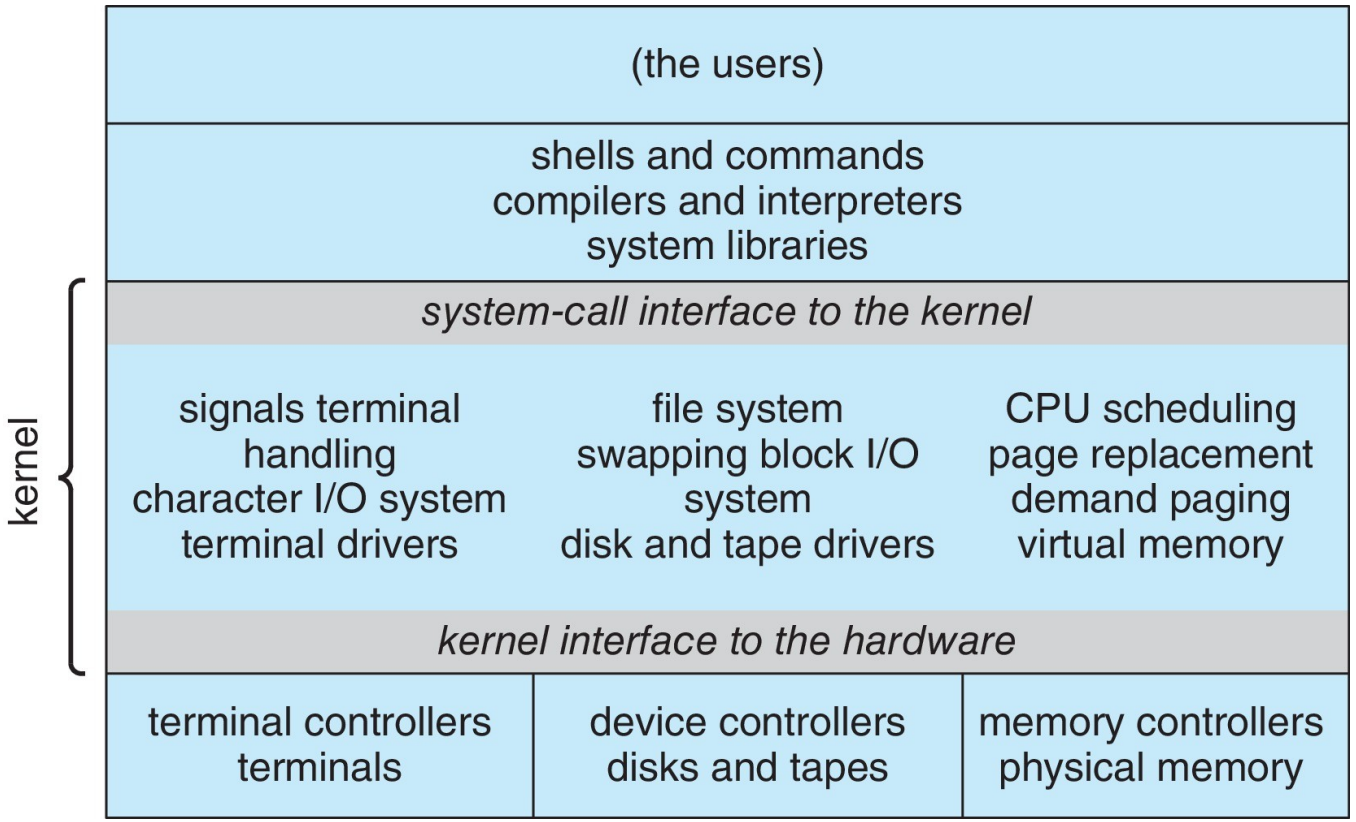
Monolithic Structure – Original UNIX

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring.
- The UNIX OS consists of two separable parts
  - Systems programs
  - The kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level



Traditional UNIX System Structure

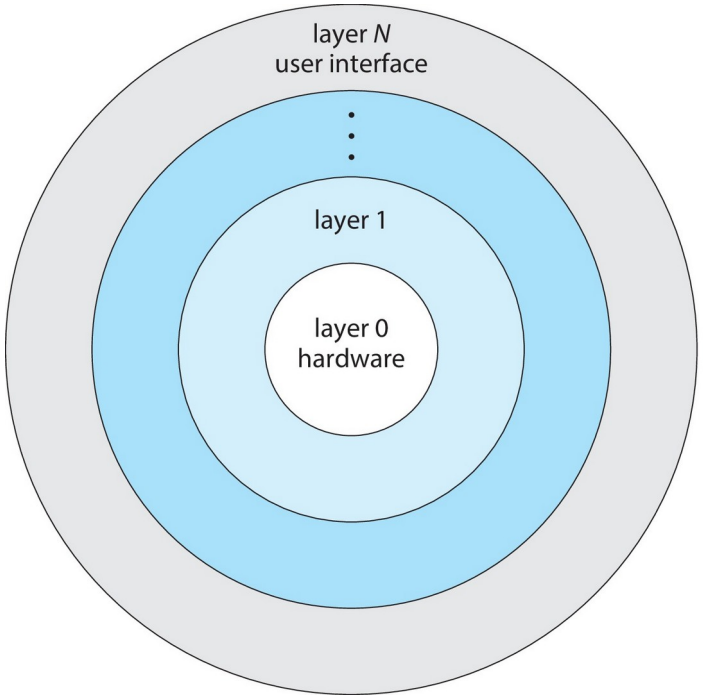
Beyond simple but not fully layered



-
هر چیزی که زیر system-call interface داریم و بالای سخت افزار میشه کرنل

Layered Approach

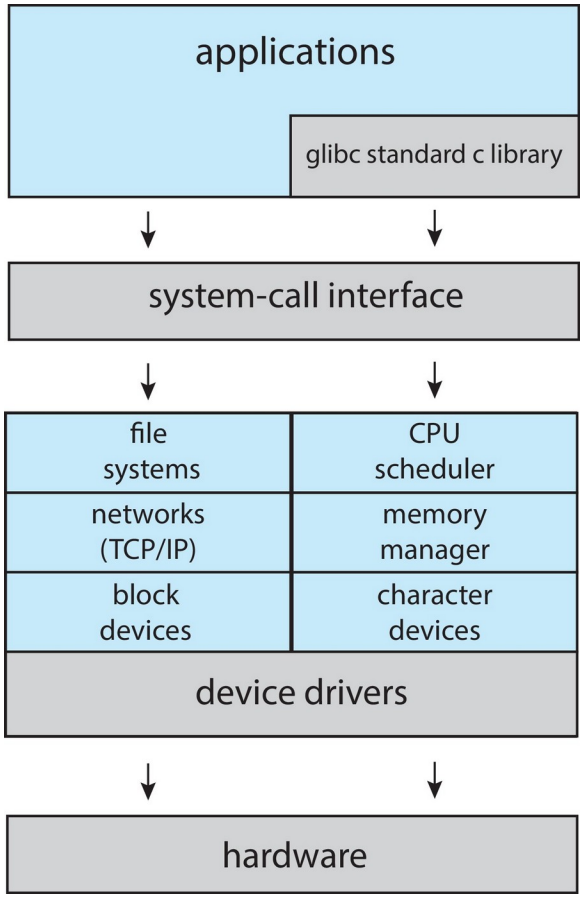
- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers



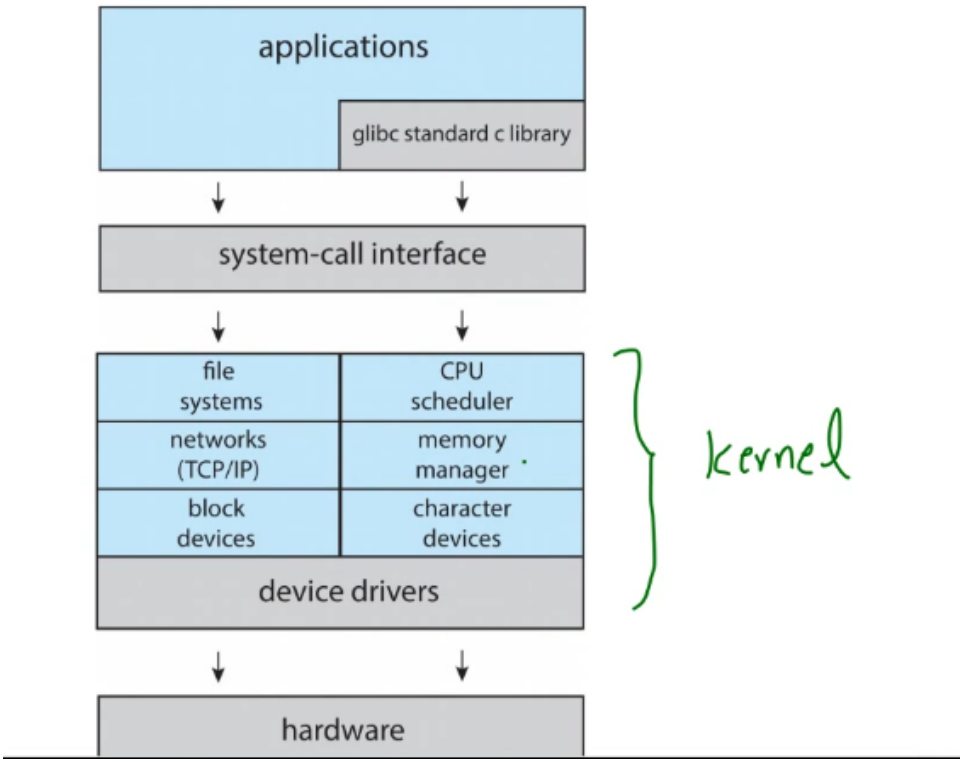
-
به طور کلی یکی از طراحی ما طراحی لایه ای است
مثلا توی این شکل سخت افزار توی لایه صفر است ینی پایین ترین لایه است
چرا به صورت دایره ای است؟ ینی هر لایه ای داره از لایه زیرش استفاده میکنه ینی مثلا لایه دو اگر بخواد از لایه صفر استفاده کنه باید از لایه یک استفاده بکنه ینی اینا رابط بین هم هستند توی این لایه ها
مشکلش: اگر ما توی لایه n یک چیزی رو بخوایم از لایه سخت افزار باید با تمام لایه های پایینی رابطه برقرار کنیم تا به لایه پایینی برسه و اینجا پرفرمنس سیستم پایین میاد

Linux System Structure

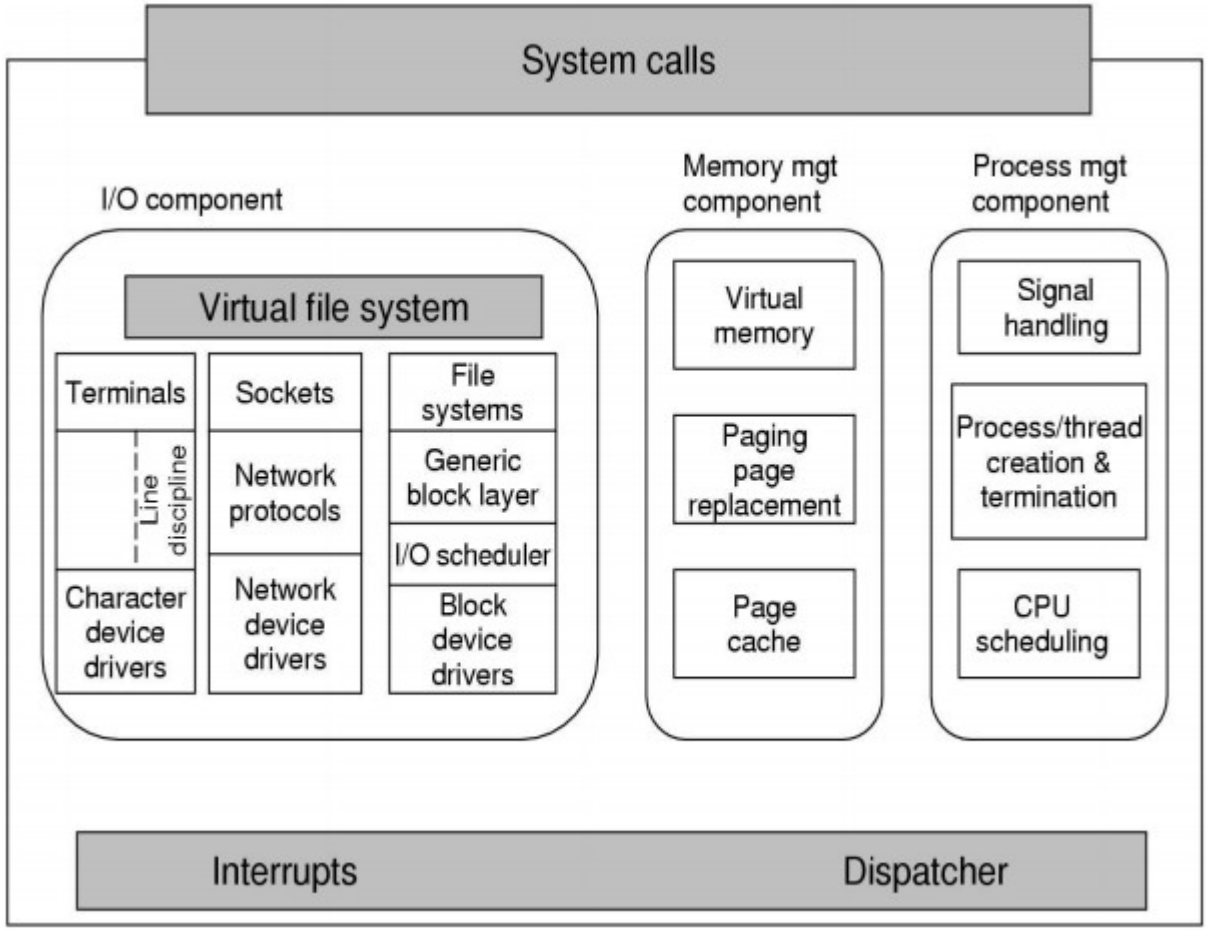
Monolithic plus **modular design**



ساختار سیستم لینوکس : طراحی بعدی
طراحی یکپارچه به علاوه ماژولار
این یک طراحی است که توی سیستم های فعلی مثل لینوکس وجود داره که این طراحی ماژولار هست به این صورت که کرنل یک برنامه پیوسته نیست ینی هر کدوم از بلوک ها به صورت یک ماژول مستقلی پیاده سازی شده است در صورتی که توی طراحی Monolithic همشون کنار هم بودن
بخش کرنل به صورت ماژول ماژول پیاده سازی شده
این چه کمکی به ما میکنه؟ انعطاف پذیری اینجا بیشتر میشه و به ما اجازه میده پیاده سازی راحتر باشه ولی طراحی monolithic این ویژگی ها رو نداشت



Linux kernel structure



Structure of the Linux kernel

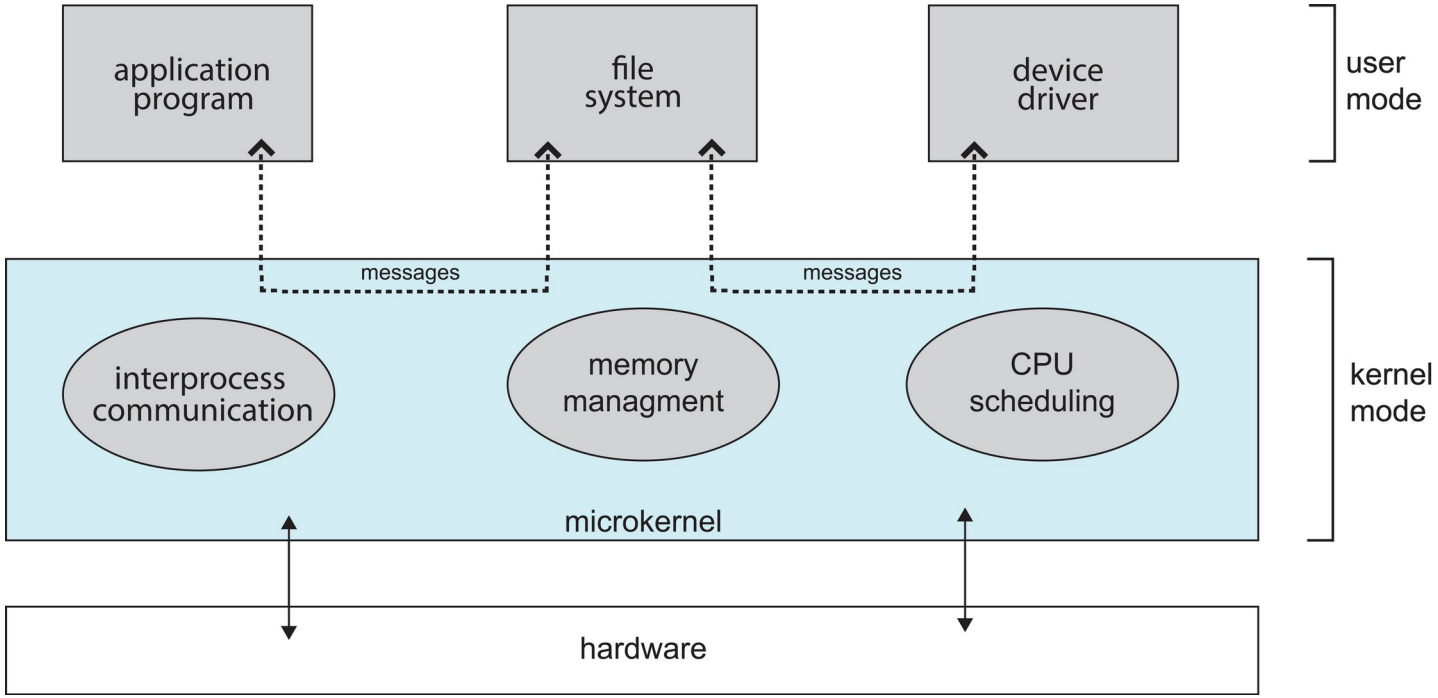
استراکچر کرنل های لینوکس فعلی:
شاید بتونیم بگیم کرنل لینوکس Monolithic است البته نه کامل Monolithic
ینی به صورت ماژولار و لایه ای پیاده سازی کردیم این بخش Monolithic رو ینی تا هر جایی که تونستیم ماژول جدا کنیم از سیستم یکپارچه کرنل و این کارو کردیم
این سه بخشی که جدا کردیم ماژول های ما هستند که درون هر ماژول طراحی لایه ای داریم و طراحی کلش هم یکپارچه است --> این طراحی ها داخل کرنل است : سوال کوییزش بود

Microkernels

- Moves as much from the kernel into user space
- **Mach** is an example of **microkernel**
  - Mac OS X kernel (**Darwin**) partly based on Mach
- Communication takes place between user modules using **message passing**
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

-
طراحی بعدی microkernels است
ینی تا حد ممکن ما بیام کرنل رو کوچیک کنیم و بیشتری بخش سیستم عامل رو منتقل کنیم به user space و حجم خیلی کمی از کار و قسمت مورد نظر توی کرنل باشه
مزیت رو بالا نوشتم
معایب:
overhead خیلی زیادی اینجا داریم و علتشو توی صفحه بعد گفتیم

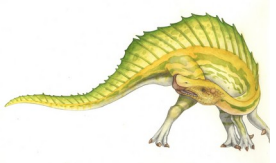
Microkernel System Structure



علت:
برای اینکه کرنل رو کوچیک کردیم مثلا بعضی از قسمت ها مثل device driver , file system که قبلا توی کرنل بودند الان اومدن توی user mode
و الان برنامه اپلیکیشنی داره ممکنه از file system و .. و از یک سری ماژول هایی که مربوط به کرنل هستند و بیرون از کرنل اند ینی توی user mode هستند الان نوع ارتباط اینا به صورت message است ینی با پیام اینا با هم ارتباط برقرار میکند که این خیلی کارامد نیست

Modules

- Many modern operating systems implement **loadable kernel modules** (**LKMs**)
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, **similar to layers but with more flexible**
  - Linux, Solaris, etc.



-
طراحی ماژولار رو بالا تر گفتیم
طراحی ماژولار غیر از اینکه هر ماژولی مستقلی و انعطاف پذیری مارو توی پیاده سازی منجر میشه یک نکته مهمی که توی سیستم های ماژولار هست loadable kernel بودن است یا LKMs
ینی ما می تونیم توی این سیستم ها یک ماژول رو اضافه کنیم یا اینکه از سیستم برش داریم
اینکه حین اجرای کرنل بدون اینکه ما کرنل رو دوباره کامپایل کنیم یا بوت کنیم بتونیم همچین کاری کنیم این خیلی انعطاف پذیری رو بالاتر می بره
یه جورایی می تونیم بگیم ماژولار بودن شبیه اون طراحی لایه ای است ولی با انعطاف پذیری بیشتر

Hybrid Systems

- Most modern operating systems are not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystem ***personalities***
- Apple Mac OS X hybrid, layered, **Aqua** UI plus **Cocoa** programming environment
  - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called **kernel extensions**)

-
ممکنه سیستم های فعلی از ترکیبی از این طراحی ها استفاده بکنند مثالاشو بالا زدیم

Building and Booting an Operating System

- Operating systems generally designed to run on a class of systems with variety of peripherals
- Commonly, operating system already installed on purchased computer
  - But can build and install some other operating systems
  - If generating an operating system from scratch
    - Write the operating system source code
    - Configure the operating system for the system on which it will run
    - Compile the operating system
    - Install the operating system
    - Boot the computer and its new operating system

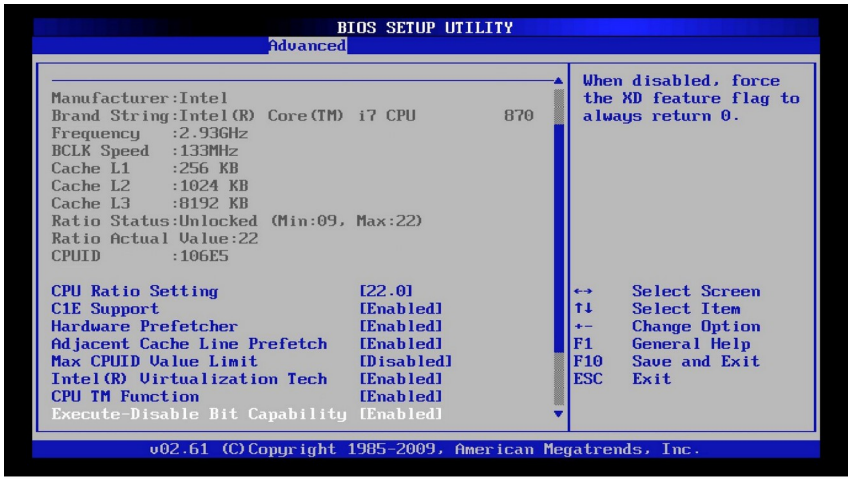
-
سیستم را طراحی کردیم وپیاده سازی کردیم چه مراحل دیگه ای باقی می مونه؟
سیستم باید یکسری از پارامترهاش کانفیگ بشه اینکه این سیستم برای چه سخت افزاری الان نصب بشه
چون سیستم عامل برای سخت افزارهای مختلفی ارائه میشه که ما باید هر سیستم عاملی که ارائه میکنیم روی سخت افزارهای مختلف قابل اجرا باشه بنابراین وقتی که میخوایم استفادش کنیم باید یکسری پارامترها مشخص کنیم که مثلا از چه cpu استفاده میشه یا چه سخت افزارهای دیگه ای
پس اصطلاحا یک کانفیگی نیاز داریم
بعد از اون کد operating system رو کامپایل میکنیم
و نهایتا الان قابل نصب میشه
بعد یک بخش هم داریم که سیستم رو بوتش کنه
مثلا مثل نصب ویندوز

Building and Booting Linux

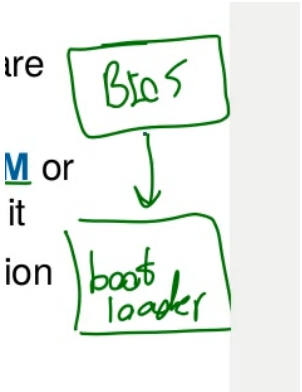
- Download Linux source code (http://www.kernel.org)
- Configure kernel via "`make menuconfig`"
- Compile the kernel using "`make`"
  - Produces `vmlinuz`, the kernel image
  - Compile kernel modules via "`make modules`"
  - Install kernel modules into `vmlinuz` via "`make modules_install`"
  - Install new kernel on the system via "`make install`"

System Boot

- When power initialized on system, execution starts at a fixed memory location
- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, **BIOS**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk
  - Modern systems replace BIOS with **Unified Extensible Firmware Interface** (**UEFI**)

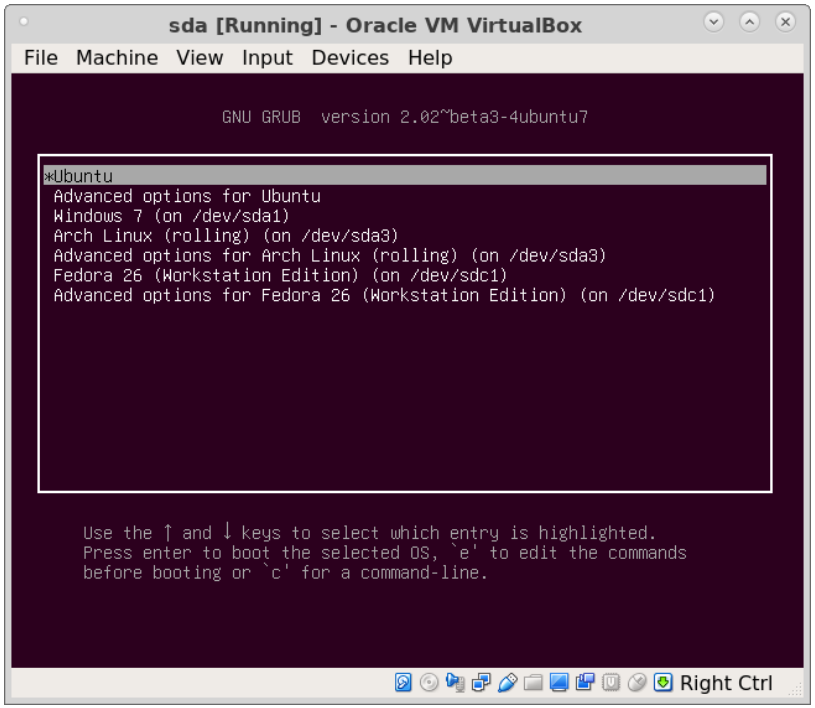


-
به سیستم عامل چطوری بفهمونیم که کامپیوتر روشن میشه این سیستم عامل بیاد بالا :
روی کامپیوتر ها یک برنامه کوچیک اولیه وجود داره که بهش bootstrap loader میگیم و این توی یک حافظه غیر فراری که بشه ROM میگیم ذخیره شده و به این برنامه BIOS میگیم ینی BIOS توی ROM ذخیره شده
حالا این BIOS خودش میاد یک boot loader دیگه رو لود میکنه حالا این boot loader بعدی boot loader همون ویندوز است یا یک boot loader جدیدی است مثل صفحه بعد
نکته: UEFI پیشرفته BIOS است که تفاوت هایی با BIOS داره:
BIOS روی ROM ذخیره شده ولی UEFI روی یک پارتیشن مشخصی از هارد دیسک ذخیره میشه و امکانات بیشتری نسبت به BIOS داره و به نسبت BIOS هم سریعتر سیستم را بوت میکنه



System Boot

- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**
- Boot loaders frequently allow various boot states, such as single user mode

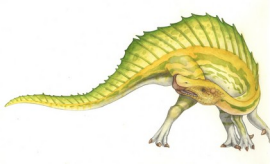


-
مثلا لینوکس که نصب میکنیم انواع boot loader داره که یکیش GRUB است
این GRUB به ما این امکان رو میده که لینوکس رو با کرنل های مختلف اگر داریم کدومشو اجرا کنیم یا اگر هم ویندوز داریم و هم لینوکس ببینیم اپشنشو ایجاد کردیم توی GRUB که با کدوم می خوایم بیایم بالا
پس برنامه GRUB یک برنامه مجزا از اون برنامه ای که به اسم BIOS می شناسیم

Operating-System Debugging

- **Debugging** is finding and fixing errors, or **bugs**
- Also **performance tuning**
- OS generate **log files** containing error information
- Failure of an application can generate **core dump** file capturing memory of the process
- Operating system failure can generate **crash dump** file containing kernel memory
- Beyond crashes, performance tuning can optimize system performance
  - Sometimes using ***trace listings*** of activities, recorded for analysis
  - **Profiling** is periodic sampling of instruction pointer to look for statistical trends

Kernighan's Law: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."



-
دیباگ کردن سیستم عامل:
یکسری یک ابزار توی سیستم عامل ها داریم:
- مثل ابزار performance tuning مثل تسک منیجر توی ویندوز مثلا می تونیم تویش ببینیم چقدر cpu در حال استفاده است و... و مشابه همین برنامه رو توی لینوکس داریم به اسم top
- ابزار دیگه log files است
- dump : یه موقعی هست که یک اپلیکیشن در حال اجرا داریم و این یهو بسته میشه یا سیستم عامل در حال اجرا بوده بعد هنگ میکنه و دیگه نمیتونه کاری بکنه و مجبور به ریپوت سیستم میشه و بعد میخوایم بفهمیم که چه اتفاقی افتاده که اینطوری شده و بعد اگر سیستم دوباره اجرا بشه بعد دیگه هیچ اطلاعاتی نداریم که اون سیستم قبلی که در حال اجرا بوده رو بتونیم دنبال کنیم و یک کاری که میشه کرد اینه که توی اون اخرین لحظه که داره این اتفاق میکنه و سیستم مجبوره همه چی رو ببنده ما می تونیم یه قسمت هایی از اون چیزی که داشته توی RAM ذخیره میشده رو توی هاردمون ذخیره کنیم که بعدا بتونیم دنبالش کنیم
اگر این فایلی که داریم ذخیره میکنیم از توی RAM مربوط به یک اپلیکیشن باشه بهش core dump میگن و اگر مربوط به خرابی سیستم عامل باشه بهش crash dump میگیم
- ابزار profiling ینی اتفاقاتی که توی سیستم عامل می افته روما دقیق تر می تونیم دنبال کنیم مثل صفحه 24

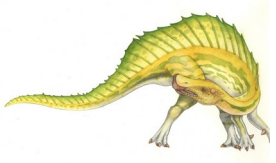
Operating-System Debugging

- **Debugging** is finding and fixing errors, or **bugs**
- Also **performance tuning**
- OS generate **log files** containing error information
- Failure of an application can generate **core dump** file capturing memory of the process
- Operating system failure can generate **crash dump** file containing

Dump: The act of copying raw data from one place to another with little or no formatting for readability.
Usually dump refers to copying data from main memory to display screen or a printer

  - **Profiling** is periodic sampling of instruction pointer to look for statistical trends

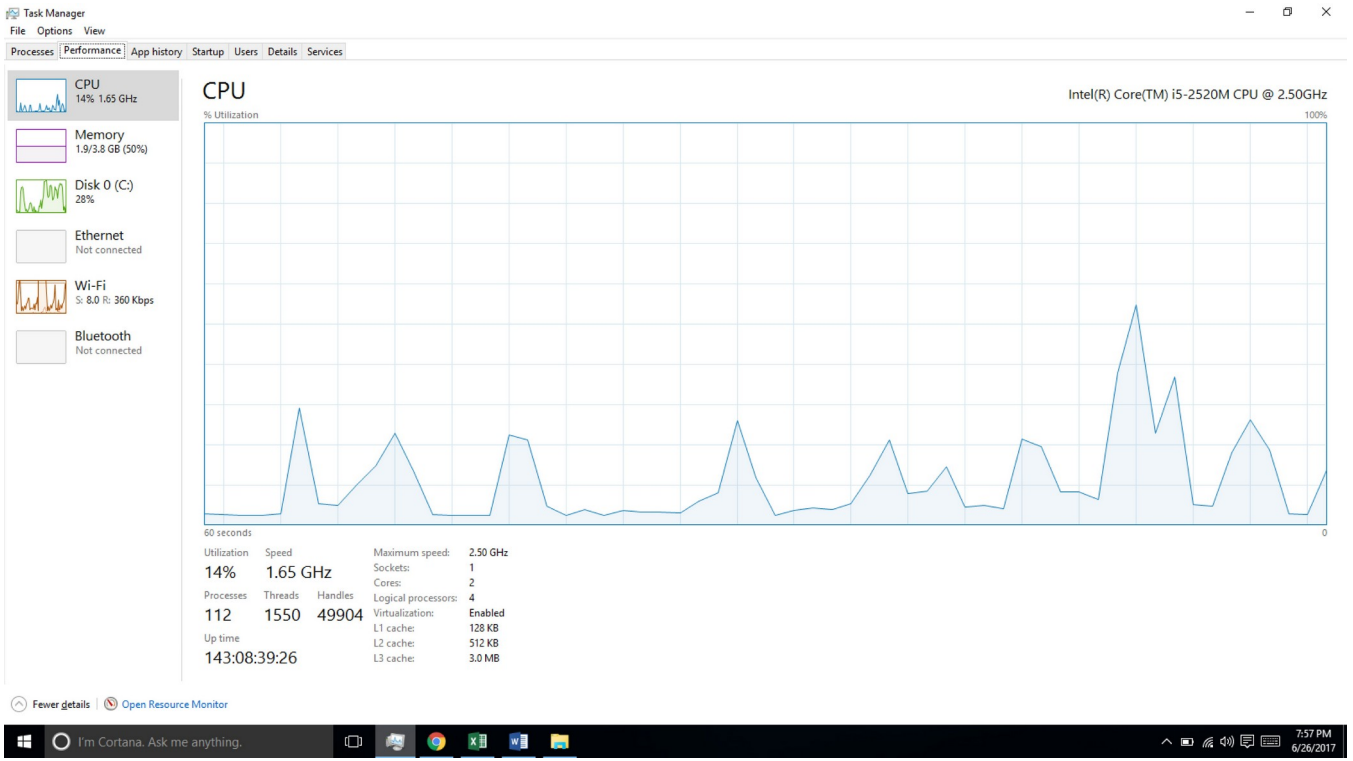
Kernighan's Law: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

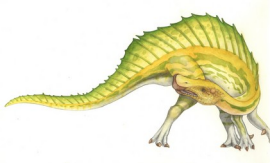


Dump: عمل کپی کردن داده های خام از یک مکان به مکان دیگر با قالب بندی کم یا بدون قالب برای خوانایی.
معمولا dump به کپی کردن داده ها از حافظه اصلی به صفحه نمایش یا چاپگر اشاره دارد

Performance Tuning

- Improve performance by removing bottlenecks
- OS must provide means of computing and displaying measures of system behavior
- For example, "top" program or Windows Task Manager





تنظیم عملکرد

+ با از بین بردن تنگناها عملکرد را بهبود بخشید
+ سیستم عامل باید ابزاری برای محاسبه و نمایش معیارهای رفتار سیستم فراهم کند
+ برای مثال، برنامه "top" یا Task Manager ویندوز

Tracing

- Collects data for a specific event, such as steps involved in a system call invocation
- Tools include
  - strace – trace system calls invoked by a process
  - gdb – source-level debugger
  - perf – collection of Linux performance tools
  - tcpdump – collects network packets

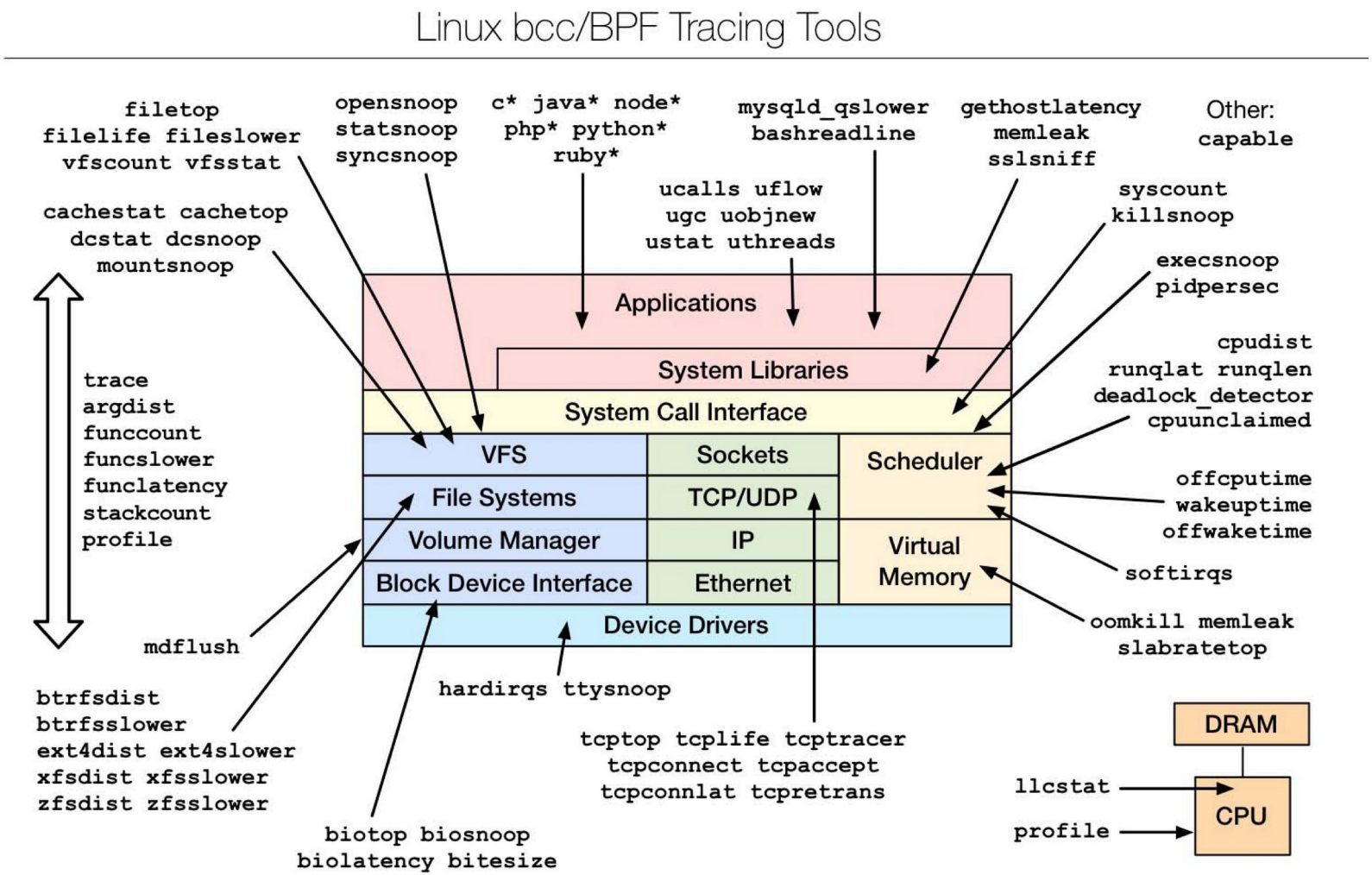
BCC

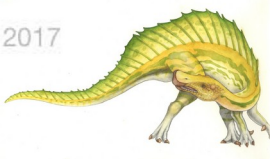
- Debugging interactions between user-level and kernel code nearly impossible without toolset that understands both and an instrument their actions
- BCC (BPF Compiler Collection) is a rich toolkit providing tracing features for Linux
  - See also the original DTrace
- For example, disksnoop.py traces disk I/O activity

```
TIME(s)          T      BYTES      LAT(ms)
1946.29186700    R      8              0.27
1946.33965000    R      8              0.26
1948.34585000    W      8192           0.96
1950.43251000    R      4096           0.56
1951.74121000    R      4096           0.35
```

- Many other tools (next slide)

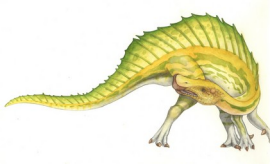
Linux bcc/BPF Tracing Tools





Free and Open-Source Operating Systems

- Operating systems made available in source-code format rather than just binary **closed-source** and **proprietary**
- Counter to the **copy protection** and **Digital Rights Management** (**DRM**) movement
- Started by **Free Software Foundation** (**FSF**), which has “copyleft” **GNU Public License** (**GPL**)
  - Free software and open-source software are two different ideas championed by different groups of people
    - **https://www.gnu.org/philosophy/open-source-misses-the-point.en.html**
- Examples include **GNU/Linux** and **BSD UNIX** (including core of **Mac OS X**), and many more
- Can use VMM like VMware Player (Free on Windows), Virtualbox (open source and free on many platforms - http://www.virtualbox.com)
  - Use to run guest operating systems for exploration



-
نکته: لینوکس یک سیستم عامل free و open-source است

Open Source film

تمرین

۱- هر یک از دسته مفاهیم زیر را با هم مقایسه کنید:

Device Driver- Device Controller

CPU scheduling- Job scheduling

۲- دو رویکرد متفاوت در برنامه نویسی CLI را بیان کرده و با هم مقایسه کنید

3- یکی از وظایف اصلی سیستم عامل، تخصیص منابع یا resource allocation است. در این رابطه دو منبع اصلی هر سیستم را نام ببرید.

تمرین

4- وظایف سیستم عامل عبارتند از
resource management، I/O handling، Process management، Memory management، Storage management، Protection ، security .

در مورد هر یک از تدابیر سخت افزاری یا نرم افزاری الف و ب، به این سؤالات پاسخ دهید

(۱) این تدبیر سخت افزاری یا نرم افزاری یا هردو است؟(توضیح مختصر) (۲) این تدبیر به منظور اجرای کدام یک از وظایف نامبرده در سیستم عامل به کار می‌رود؟ (۳) توضیح مختصر دهید که چگونه وظیفه موردنظر با تدبیر ذکرشده برآورده می‌شود.

الف) تعریف مدهای کرنل و کاربر

5- با ذکر حداقل دو مثال بنویسید در چه مواردی اپلیکیشنی که شما برنامه نویسی می کنید از سیستم عامل استفاده میکند؟ اپلیکیشن شما چگونه درخواست هایش را به سیستم عامل میفرستد؟ آیا در زبان‌های برنامه نویسی امکانی جهت ارسال دستور به سیستم عامل وجود دارد؟