

Operating Systems

Isfahan University of Technology
Electrical and Computer Engineering Department

Zeinab Zali

Session 3: Operating System Design and Implementation



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, CFS method
- **Mechanism:** **How** to do something?
 - Example: timer, Scheduling
- **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



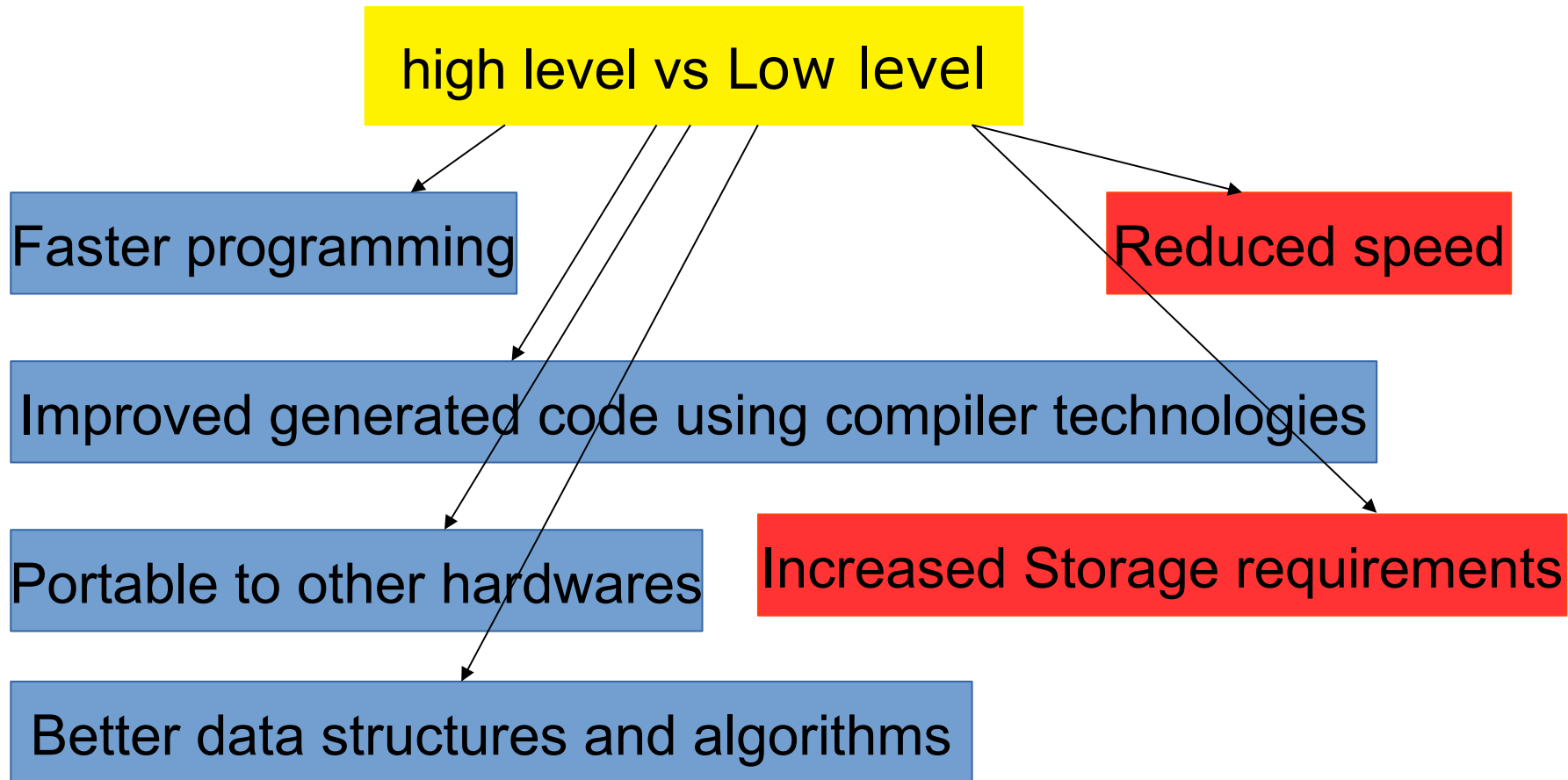


Implementation

- 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

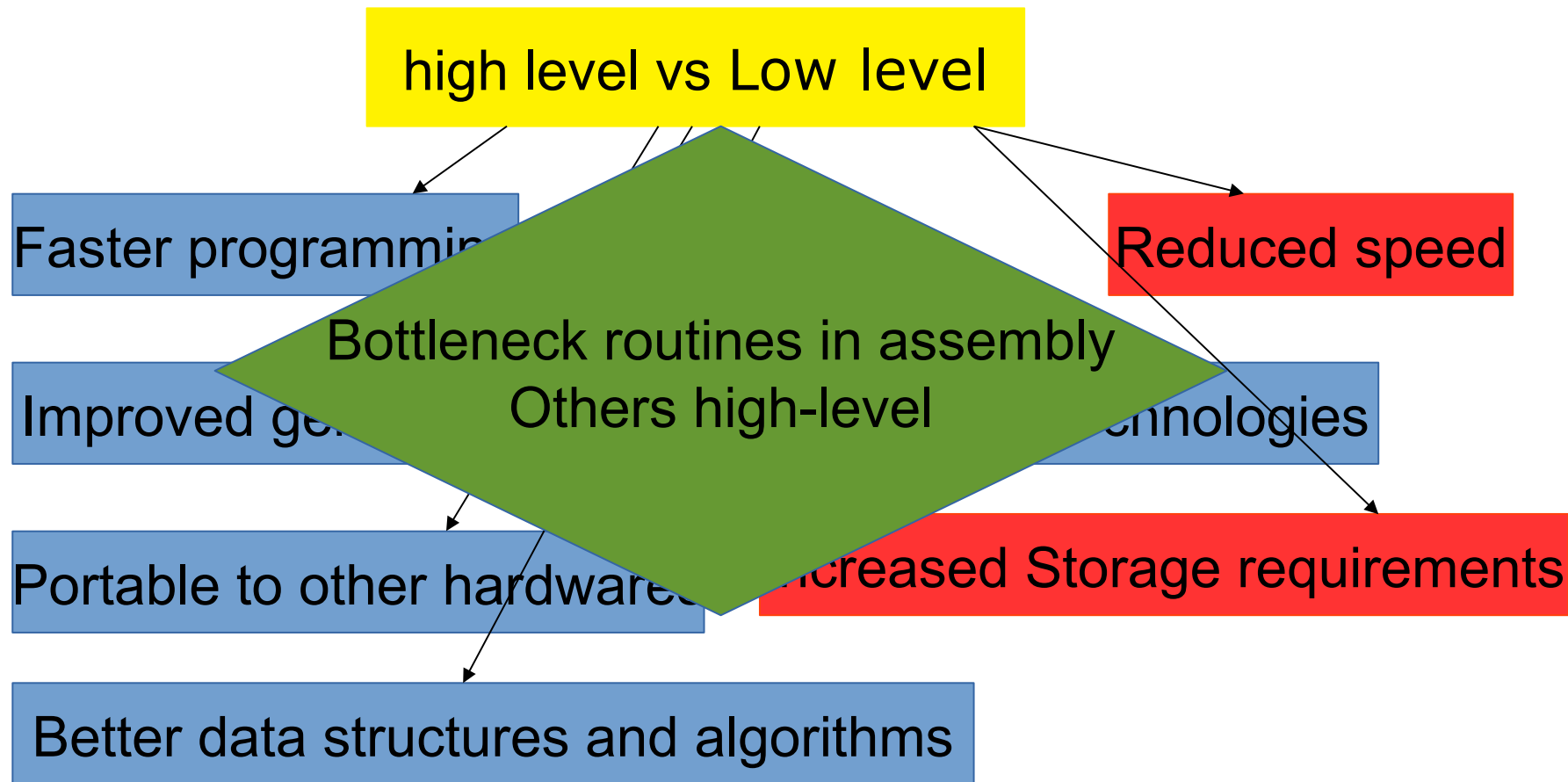


OS Implementation





OS Implementation





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

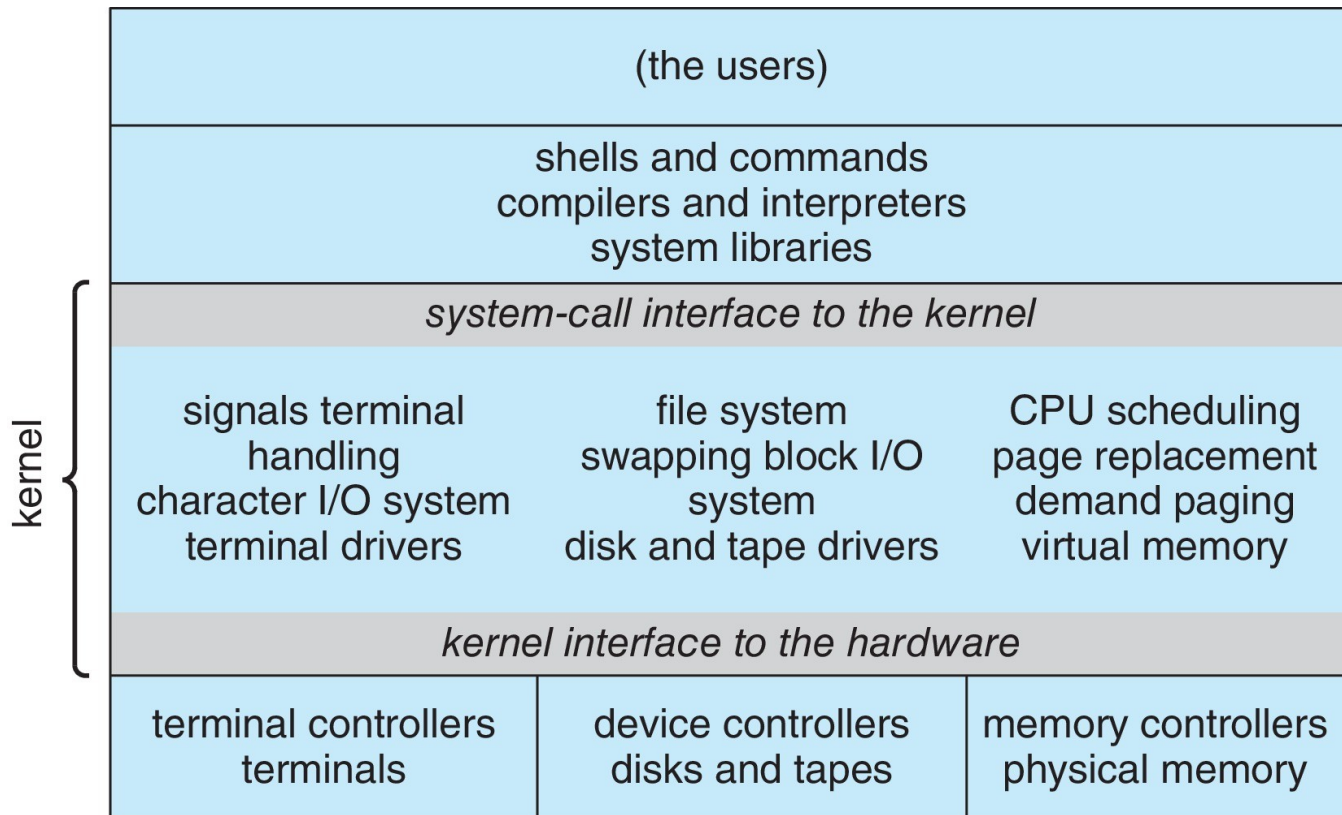
- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring.
- The UNIX OS consists of two separable parts
 - Systems programs
 - The kernel
 - ▶ Consists of everything below the system-call interface and above the physical hardware
 - ▶ Provides the file system,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





Traditional **UNIX** System Structure

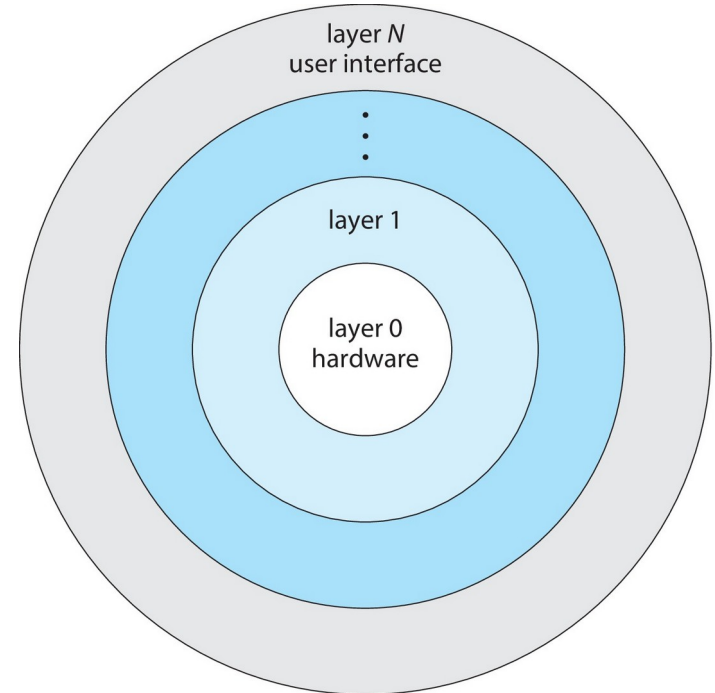
Beyond simple but not fully layered





Layered Approach

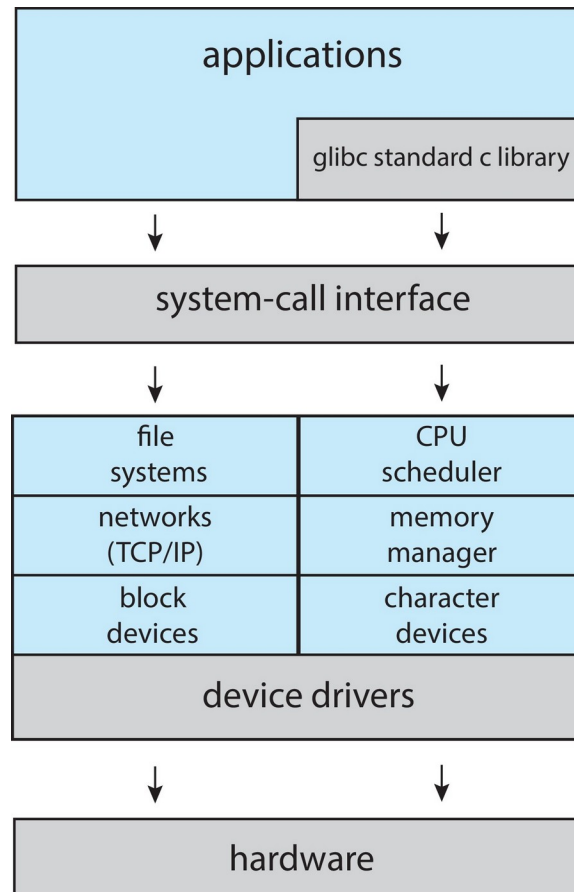
- The operating system is divided into a number of **layers** (levels), each built on top of lower layers. The bottom layer (**layer 0**), is the **hardware**; the highest (**layer N**) is the **user interface**.
- With **modularity**, layers are selected such that each uses **functions (operations)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



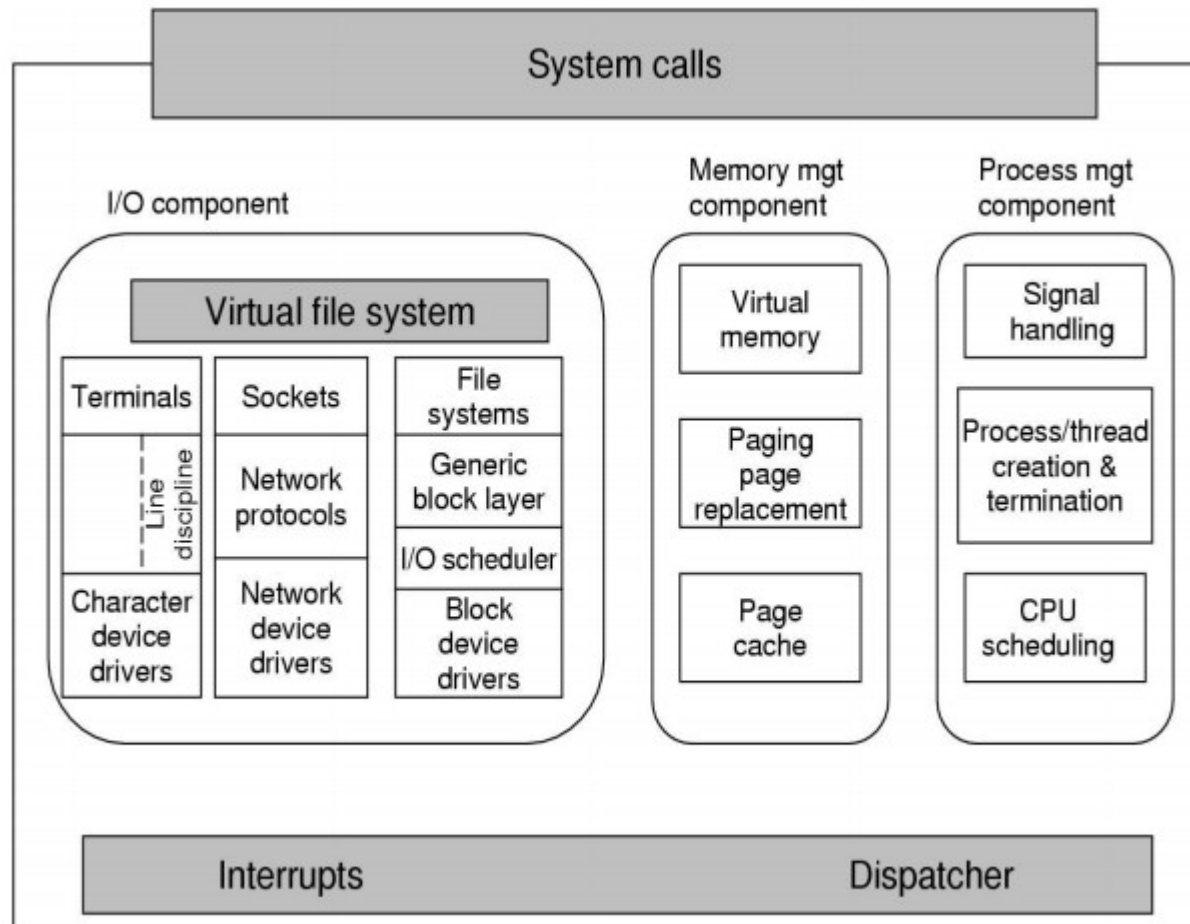


Linux System Structure

Monolithic plus **modular design**



Linux kernel structure



Structure of the Linux kernel



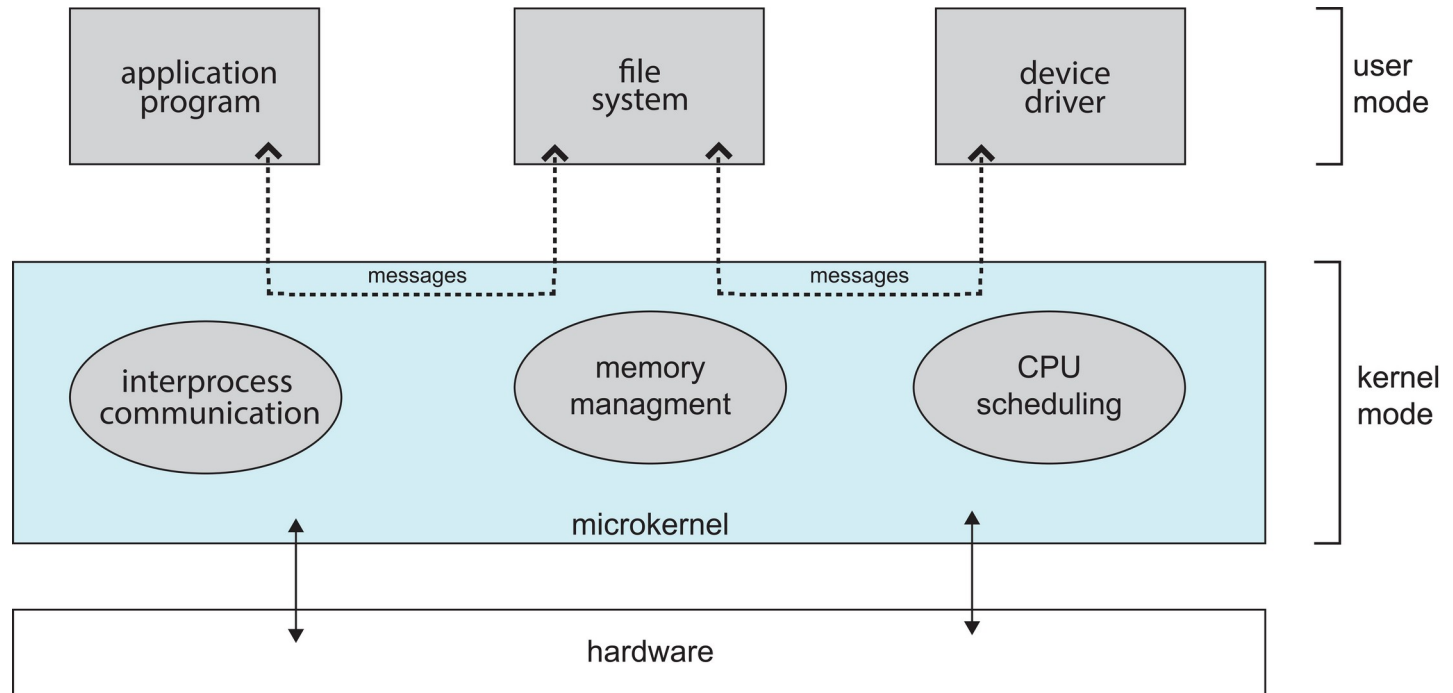
Microkernels

- 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**





Microkernel System Structure





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.





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**)





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

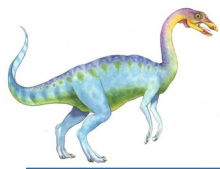




Building and Booting Linux

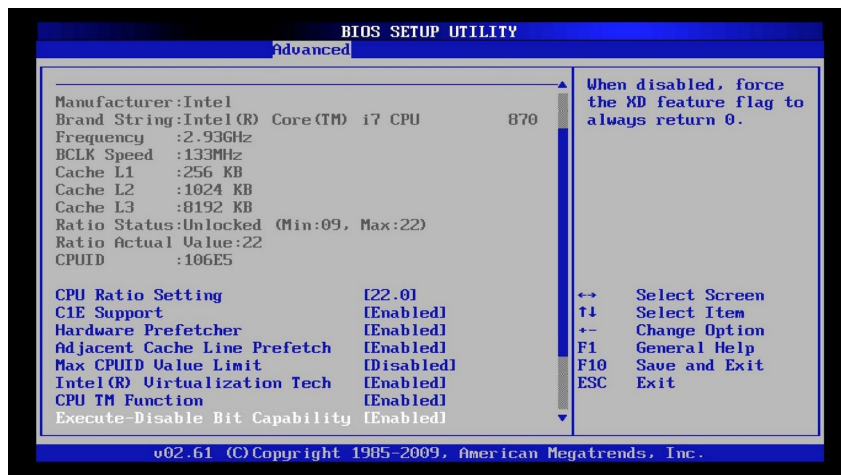
- Download Linux source code (<http://www.kernel.org>)
- Configure kernel via “make menuconfig”
- Compile the kernel using “make”
 - Produces `vmlinuz`, the kernel image
 - Compile kernel modules via “make modules”
 - Install kernel modules into `vmlinuz` via “make modules_install”
 - Install new kernel 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)**





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

```
sda [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

GNU GRUB  version 2.02~beta3-4ubuntu7

*Ubuntu
Advanced options for Ubuntu
Windows 7 (on /dev/sda1)
Arch Linux (rolling) (on /dev/sda3)
Advanced options for Arch Linux (rolling) (on /dev/sda3)
Fedora 26 (Workstation Edition) (on /dev/sdc1)
Advanced options for Fedora 26 (Workstation Edition) (on /dev/sdc1)

Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the commands
before booting or 'c' for a command-line.
```





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."





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

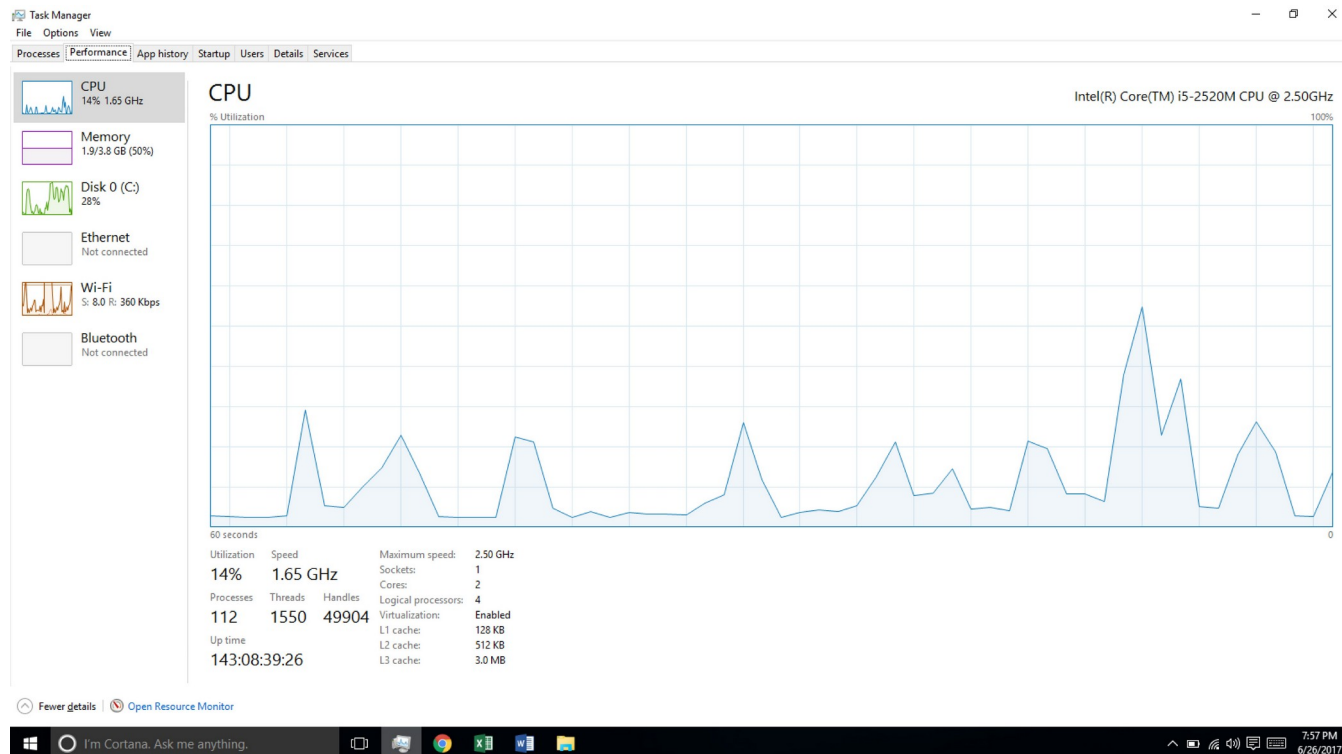
Kernighan's Law: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."





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**





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)





The diagram illustrates the Linux kernel architecture, showing the flow of data and control from Applications down to the hardware (DRAM and CPU). The architecture is organized into several layers, each with associated monitoring tools and components.

Applications Layer (Pink): This layer includes various user-space applications and their associated monitoring tools. Tools shown include: filetop, filelife, fileslower, vfscount, vfsstat, cachestat, cachetop, dcstat, dcnoop, mountsnoop, trace, argdist, funcount, funclower, funclatency, stackcount, profile, btrfsdist, btrfsslower, ext4dist, ext4slower, xfsdist, xfsslower, zfsdist, zfsslower, biotop, biosnoop, biolatency, bitesize, c* java* node* php* python* ruby*, mysqlq_d_qlower, bashreadline, gethostlatency, memleak, sslsniff, syscount, killsnoop, execsnoop, pidpersec, cpudist, runqlat, runqlen, deadlock_detector, cpuunclaimed, offcputime, wakeuptime, offwaketime, softirqs, oomkill, memleak, slabratetop, llcstat, and profile.

System Libraries Layer (Pink): This layer includes system libraries and their associated monitoring tools. Tools shown include: ucalls, uflow, ugc, uobjnew, ustat, uthreads, ucalls, uflow, ugc, uobjnew, ustat, uthreads, ucalls, uflow, ugc, uobjnew, ustat, uthreads.

System Call Interface Layer (Yellow): This layer includes the system call interface and its associated monitoring tools. Tools shown include: cachestat, cachetop, dcstat, dcnoop, mountsnoop, trace, argdist, funcount, funclower, funclatency, stackcount, profile.

VFS Layer (Blue): This layer includes the Virtual File System (VFS) and its associated monitoring tools. Tools shown include: filetop, filelife, fileslower, vfscount, vfsstat.

File Systems Layer (Blue): This layer includes various file systems and their associated monitoring tools. Tools shown include: btrfsdist, btrfsslower, ext4dist, ext4slower, xfsdist, xfsslower, zfsdist, zfsslower.

Volume Manager Layer (Blue): This layer includes the Volume Manager and its associated monitoring tools. Tools shown include: mdflush.

Block Device Interface Layer (Blue): This layer includes the Block Device Interface and its associated monitoring tools. Tools shown include: biotop, biosnoop, biolatency, bitesize.

Sockets Layer (Green): This layer includes Sockets and their associated monitoring tools. Tools shown include: tcpdist, tcpsslower, tcptop, tcpplife, tcptracer, tcpconnect, tcpaccept, tcpconnlat, tcpretrans.

TCP/UDP Layer (Green): This layer includes TCP/UDP and its associated monitoring tools. Tools shown include: hardirqs, ttysnoop.

IP Layer (Green): This layer includes IP and its associated monitoring tools. Tools shown include: tcptop, tcpplife, tcptracer, tcpconnect, tcpaccept, tcpconnlat, tcpretrans.

Ethernet Layer (Green): This layer includes Ethernet and its associated monitoring tools. Tools shown include: tcptop, tcpplife, tcptracer, tcpconnect, tcpaccept, tcpconnlat, tcpretrans.

Scheduler Layer (Orange): This layer includes the Scheduler and its associated monitoring tools. Tools shown include: cpudist, runqlat, runqlen, deadlock_detector, cpuunclaimed, offcputime, wakeuptime, offwaketime, softirqs, oomkill, memleak, slabratetop, llcstat, and profile.

Virtual Memory Layer (Orange): This layer includes Virtual Memory and its associated monitoring tools. Tools shown include: llcstat and profile.

Device Drivers Layer (Light Blue): This layer includes Device Drivers and their associated monitoring tools. Tools shown include: hardirqs and ttysnoop.

Hardware (DRAM and CPU): The hardware layer consists of DRAM and CPU. The CPU is connected to the Device Drivers layer. The CPU is also connected to the Scheduler layer via the llcstat and profile tools.

<https://github.com/iovisor/bcc#tools> 2017





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



Open Source film



LINUS TORVALDS
Creator, Linux Kernel

لینوس توروالدز
خالق هسته‌ی لینوکس

برای توضیح این که لینوکس چه باید