UNIVERSITY of SAN CARLOS
SCIENTIA • VIRTUS • DEVOTIO

DCISM
DEPARTMENT OF COMPUTER, INFORMATION SCIENCES AND MATHEMATICS

CS 3104
OPERATING SYSTEMS

2

CHAPTER 2

OPERATING-SYSTEM STRUCTURES

OUTLINE

- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

- Outline
- ✓ Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

WHY APPLICATIONS ARE OPERATING-SYSTEM SPECIFIC?

- **Fundamental Fact:**
  - Apps compiled on one operating system are not usually executable on other operating systems
- **Problem:**
  - Each operating system provides its own unique system calls
    - Own file formats, etc.

- Outline
- ✓ Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

WHY APPLICATIONS ARE OPERATING-SYSTEM SPECIFIC?

- **Apps can be multi-operating system:**
  1. Apps written in interpreted language (like Python or Ruby) that has an interpreter available on multiple operating systems
  2. Apps written in language that includes a VM containing the running app (like Java)
  3. Application developer can use a standard language (like C) or API, compile separately on each operating system to run on each

- **Application Binary Interface (ABI):**
  - architecture equivalent of API (**API:** application level)
  - defines how different components of binary code can interface for a given operating system on a given architecture, CPU, etc.

- Outline
- ✓ Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

WHY APPLICATIONS ARE OPERATING-SYSTEM SPECIFIC?

Important Notes

- Unless an interpreter, RTE, or binary executable file is written for and compiled on a specific operating system on a specific CPU type (such as Intel x86 or ARMv8), the application will fail to run.

- Imagine the amount of work that is required for a program such as the Firefox browser to run on Windows, macOS, various Linux releases, iOS, and Android, sometimes on various CPU architectures.

- Outline
- Why Applications are Operating-System Specific?
- ✓ Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

OPERATING-SYSTEM DESIGN AND IMPLEMENTATION

- Problems in the design and implementation of OS that are 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 and type of system
  - traditional desktop/laptop, mobile, distributed, or real time

- Outline
- Why Applications are Operating-System Specific?
- ✓ Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

OPERATING-SYSTEM DESIGN AND IMPLEMENTATION

- **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
- **Important Notes:**
  - requirements are vague and may be interpreted in various ways
  - no unique solution to the problem of defining the requirements for an operating system
  - wide range of systems in existence shows that different requirements can result in a large variety of solutions for different environments

- Outline
- Why Applications are Operating-System Specific?
- ✓ Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

OPERATING-SYSTEM DESIGN AND IMPLEMENTATION

- Important principle to separate:
  - **Policy:** ***What*** will be done?
  - **Mechanism:** ***How*** to do it?
- **Mechanisms** determine how to do something
- **Policies** decide what will be done
- The **separation of policy from mechanism** is a very important principle, it allows maximum flexibility if policy decisions are to be changed later (example: timer)
- **Specifying and designing an OS** is **highly creative task** of **software engineering**

- Outline
- Why Applications are Operating-System Specific?
- ✓ Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

OPERATING-SYSTEM DESIGN AND IMPLEMENTATION

OS Implementation

- **Operating Systems** are **collections of many programs**
  - written by many people over a long period of time
  - difficult to make general statements on their implementation

- **Much variation:**
  - Early Operating Systems were written in Assembly Language
  - Then system programming languages like Algol, PL/1
  - Now in C, C++

- Outline
- Why Applications are Operating-System Specific?
- ✓ Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

OPERATING-SYSTEM DESIGN AND IMPLEMENTATION

OS Implementation

- Actually, usually a **mix of languages**:
  - Lowest levels of the kernel in assembly
  - Main body in C
  - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
- **More high-level language** are easier to **port** to other hardware
  - but slower
- **Emulation** can allow an OS to run on non-native hardware

- Outline
- Why Applications are Operating-System Specific?
- ✓ Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

OPERATING-SYSTEM DESIGN AND IMPLEMENTATION

Important Notes

- Major performance improvements in operating systems are more likely to be the result of **better data structures and algorithms** than of **excellent assembly-language code**.
- In addition, although operating systems are large, **only a small amount of the code is critical to high performance**; the **interrupt handlers, I/O manager, memory manager, and CPU scheduler** are probably the most critical routines.
- After the system is written and is working correctly, bottlenecks can be identified and can be refactored to operate more efficiently

- **Outline**
- **Why Applications are Operating-System Specific?**
- **Operating-System Design and Implementation**
- ✓**Operating-System Structure**
- **Building and Booting an Operating System**
- **Operating-System Debugging**

OPERATING-SYSTEM STRUCTURE

- General-purpose OS is a very large and complex program
- Various **ways to structure** Operating Systems:
  - **Simple structure:** MS-DOS
  - **More complex:** UNIX
  - **Layered:** An abstraction
  - **Microkernel:** Mach

Note: Not all versions of Mach are microkernels

- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- ✓Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

OPERATING-SYSTEM STRUCTURE

Monolithic Structure

- **Monolithic Structure**:
  - common technique for designing OS
  - place all of the functionality of the kernel into a single, static binary file that runs in a single address space
  - often known as a tightly coupled system because changes to one part of the system can have wide-ranging effects on other parts
  - **example: Original UNIX OS**

- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- ✓ Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

OPERATING-SYSTEM STRUCTURE

Monolithic Structure

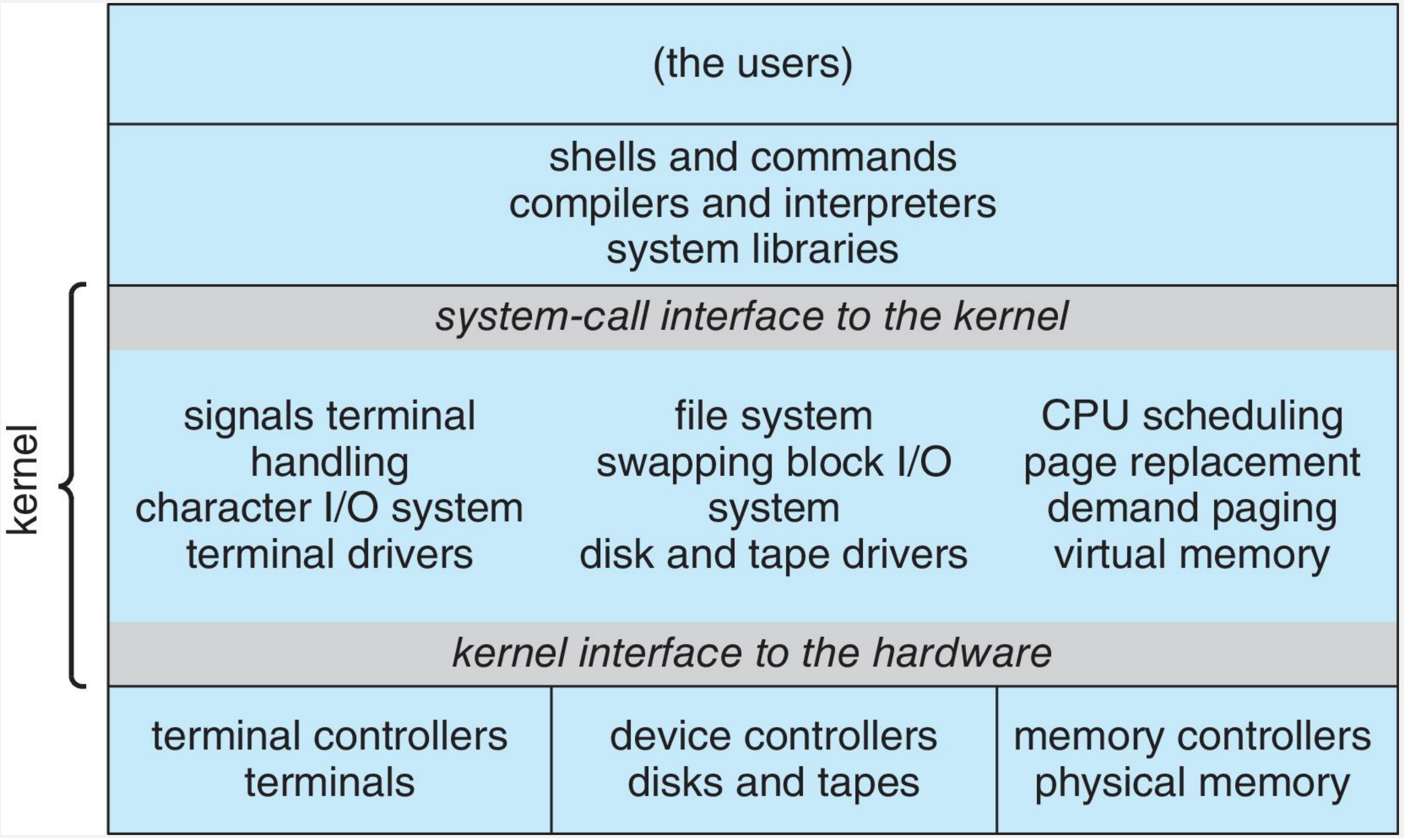
- **UNIX OS**
  - limited by hardware functionality
    - the original UNIX operating system had limited structuring
  - **Consists of two separable parts:**
    - **Systems programs**
    - **The kernel**
      - ✓ Consists of everything below the system-call interface and above the physical hardware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

- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- ✓Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

OPERATING-SYSTEM STRUCTURE

Traditional UNIX System Structure

Beyond simple but not fully layered

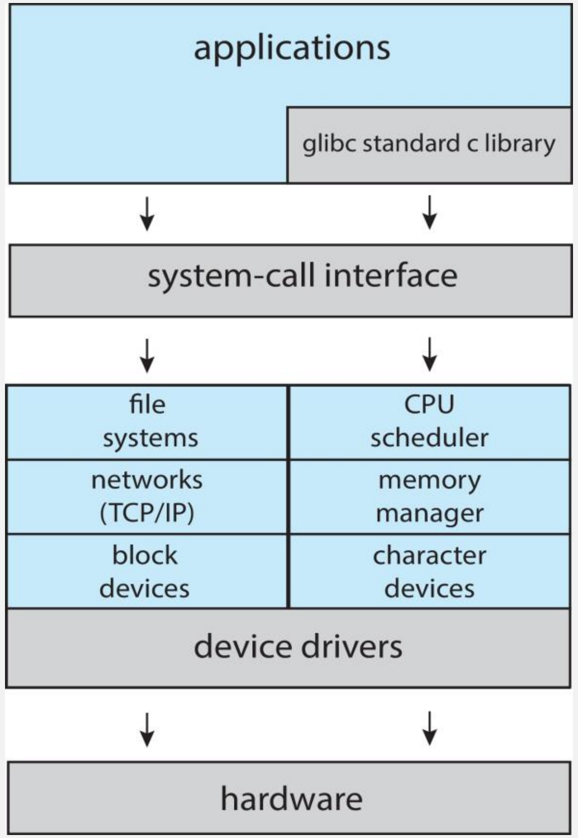


- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- ✓Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

OPERATING-SYSTEM STRUCTURE

Linux System Structure

Monolithic plus modular design

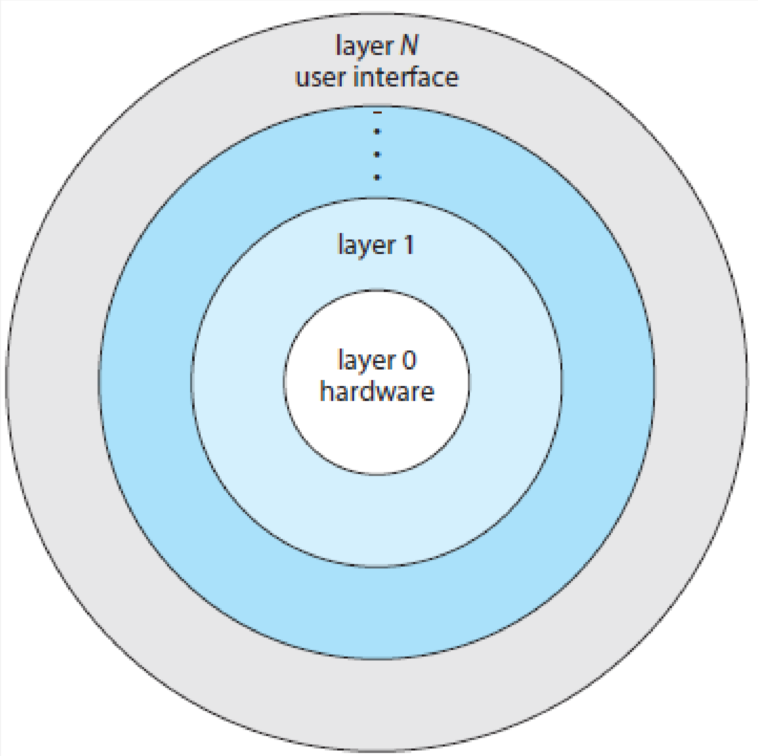


- **Outline**
- **Why Applications are Operating-System Specific?**
- **Operating-System Design and Implementation**
- ✓ **Operating-System Structure**
- **Building and Booting an Operating System**
- **Operating-System Debugging**

OPERATING-SYSTEM STRUCTURE

Layered Approach

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



- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- ✓ Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

OPERATING-SYSTEM STRUCTURE

Microkernels

- A **piece of software or even code** that contains the **near-minimum amount of functions and features** required to **implement an operating system**.
- **Moves** as much **from the kernel** into **user space**
- **Mach:** example of **microkernel**
  - Mac OS X kernel (**Darwin**) is partly based on Mach
- **Communication takes place between user modules** using **message passing**

- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- ✓Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

OPERATING-SYSTEM STRUCTURE

Microkernels

- **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

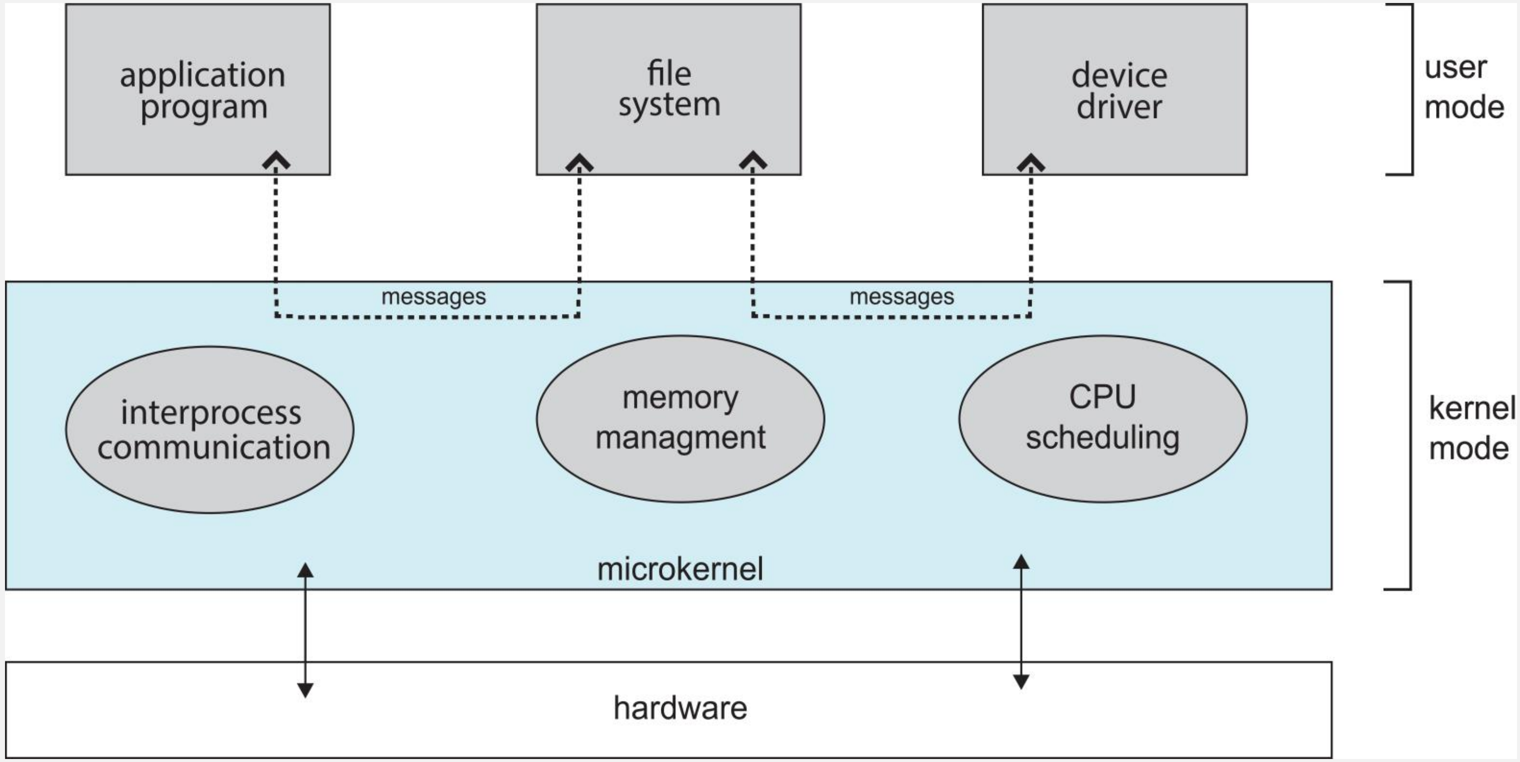
OPERATING-SYSTEM STRUCTURE

- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation

✓ Operating-System Structure

- Building and Booting an Operating System
- Operating-System Debugging

Microkernel System Structure



- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- ✓Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

OPERATING-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 layered approach** but **more flexible** because **any module can call any other module**
  - Linux, Solaris, etc.

- **Outline**
- **Why Applications are Operating-System Specific?**
- **Operating-System Design and Implementation**
- ✓**Operating-System Structure**
- **Building and Booting an Operating System**
- **Operating-System Debugging**

OPERATING-SYSTEM STRUCTURE

Hybrid Systems

- Most modern operating systems are actually 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**)

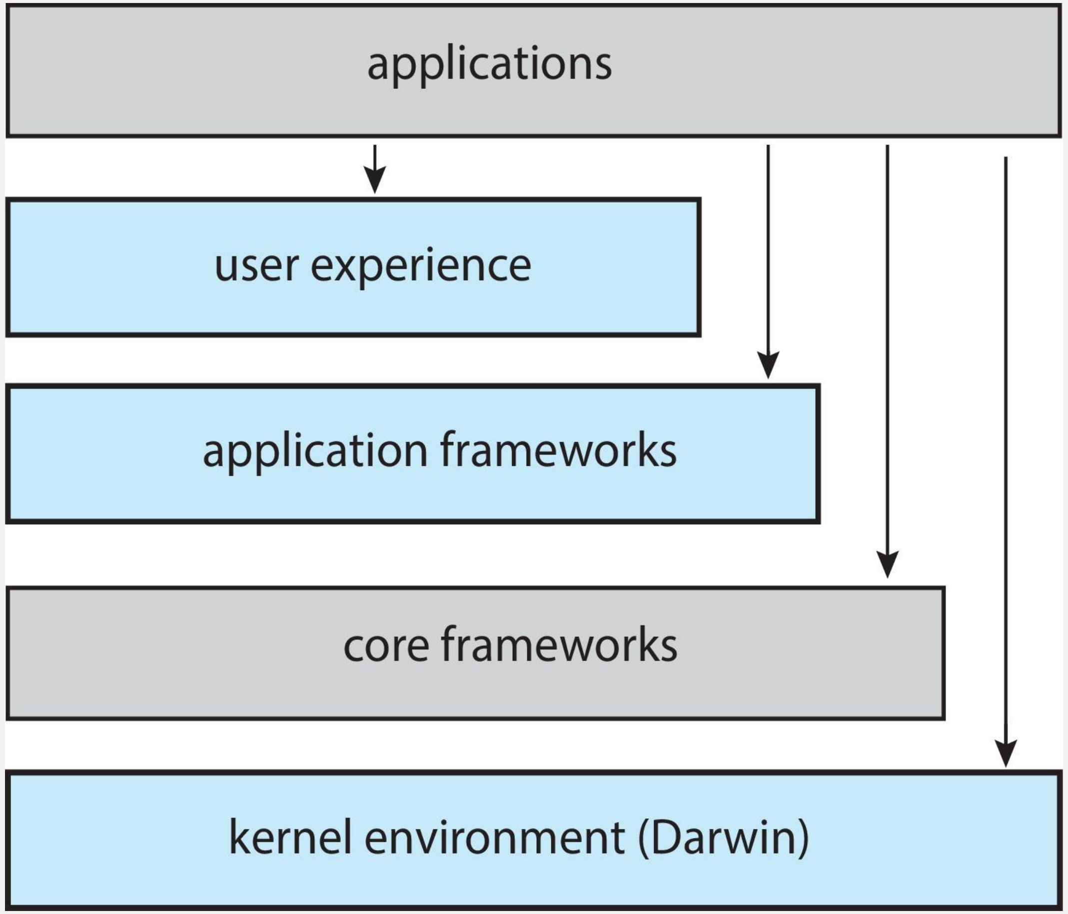
OPERATING-SYSTEM STRUCTURE

- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation

✓ Operating-System Structure

- Building and Booting an Operating System
- Operating-System Debugging

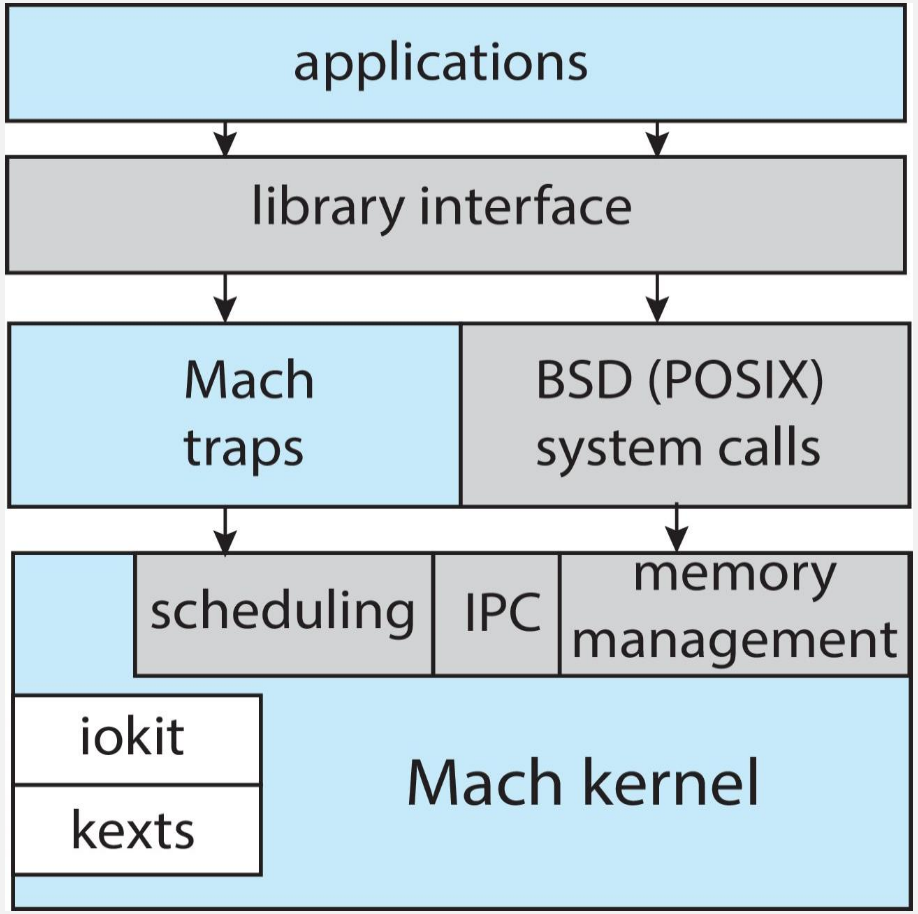
macOS and iOS Structure



OPERATING-SYSTEM STRUCTURE

- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- ✓Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

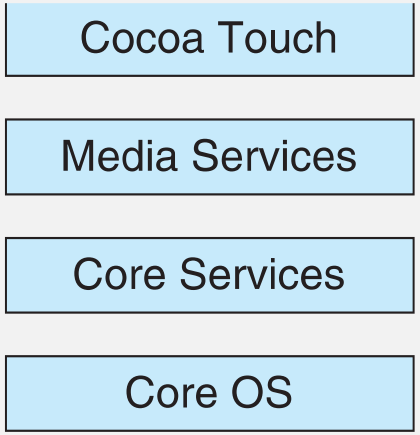
Darwin



OPERATING-SYSTEM STRUCTURE

iOS

- Apple mobile OS for ***iPhone***, ***iPad***
  - Structured on Mac OS X, added functionality
  - Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs. Intel)
  - **Cocoa Touch** Objective-C API for developing apps
  - **Media services** layer for graphics, audio, video
  - **Core services** provides cloud computing, databases
  - **Core O**perating **S**ystem, based on Mac OS X kernel



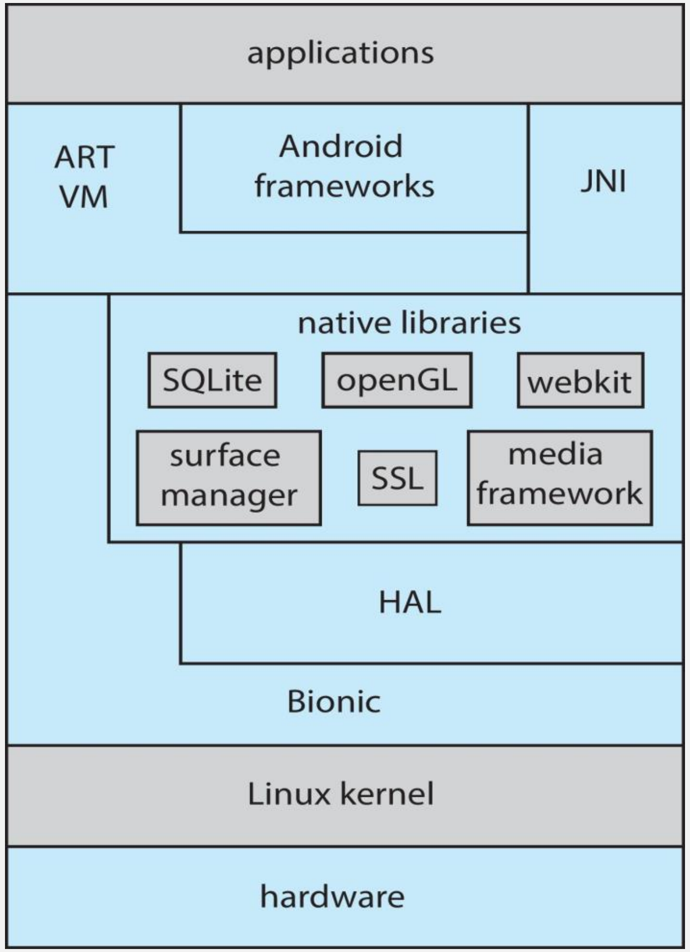
OPERATING-SYSTEM STRUCTURE

Android

- Developed by Open Handset Alliance (mostly Google)
  - Open Source
- Similar stack to IOS
- Based on Linux kernel but modified
  - Provides process, memory, device-driver management
  - Adds power management
- **Runtime environment** includes core set of libraries and Dalvik virtual machine
  - Apps developed in Java plus Android API
    - ✓ Java class files compiled to Java bytecode then translated to executable than runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

OPERATING-SYSTEM STRUCTURE

Android Architecture



- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- Operating-System Structure
- ✓Building and Booting an Operating System
- Operating-System Debugging

BUILDING AND BOOTING AN OPERATING SYSTEM

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

- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- Operating-System Structure
- ✓Building and Booting an Operating System
- Operating-System Debugging

BUILDING AND BOOTING AN OPERATING SYSTEM

Building and Booting Linux

- Download Linux source code (https://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`"

- **Outline**
- **Why Applications are Operating-System Specific?**
- **Operating-System Design and Implementation**
- **Operating-System Structure**

✓**Building and Booting an Operating System**

- **Operating-System Debugging**

BUILDING AND BOOTING AN OPERATING SYSTEM

System Boot

- When power is 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 a 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)**
- 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

OPERATING-SYSTEM DEBUGGING

- **Outline**
- **Why Applications are Operating-System Specific?**
- **Operating-System Design and Implementation**
- **Operating-System Structure**
- **Building and Booting an Operating System**
- ✓ **Operating-System Debugging**

- **Debugging** is finding and fixing **errors**, or **bugs**
- Can also include **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

- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- ✓Operating-System Debugging

OPERATING-SYSTEM DEBUGGING

- Beyond crashes, **performance tuning** can optimize system performance
  - Sometimes using ***trace listings*** of activities, recorded for analysis
  - **Profiling** is the 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."

- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- ✓Operating-System Debugging

OPERATING-SYSTEM DEBUGGING

Performance Tuning

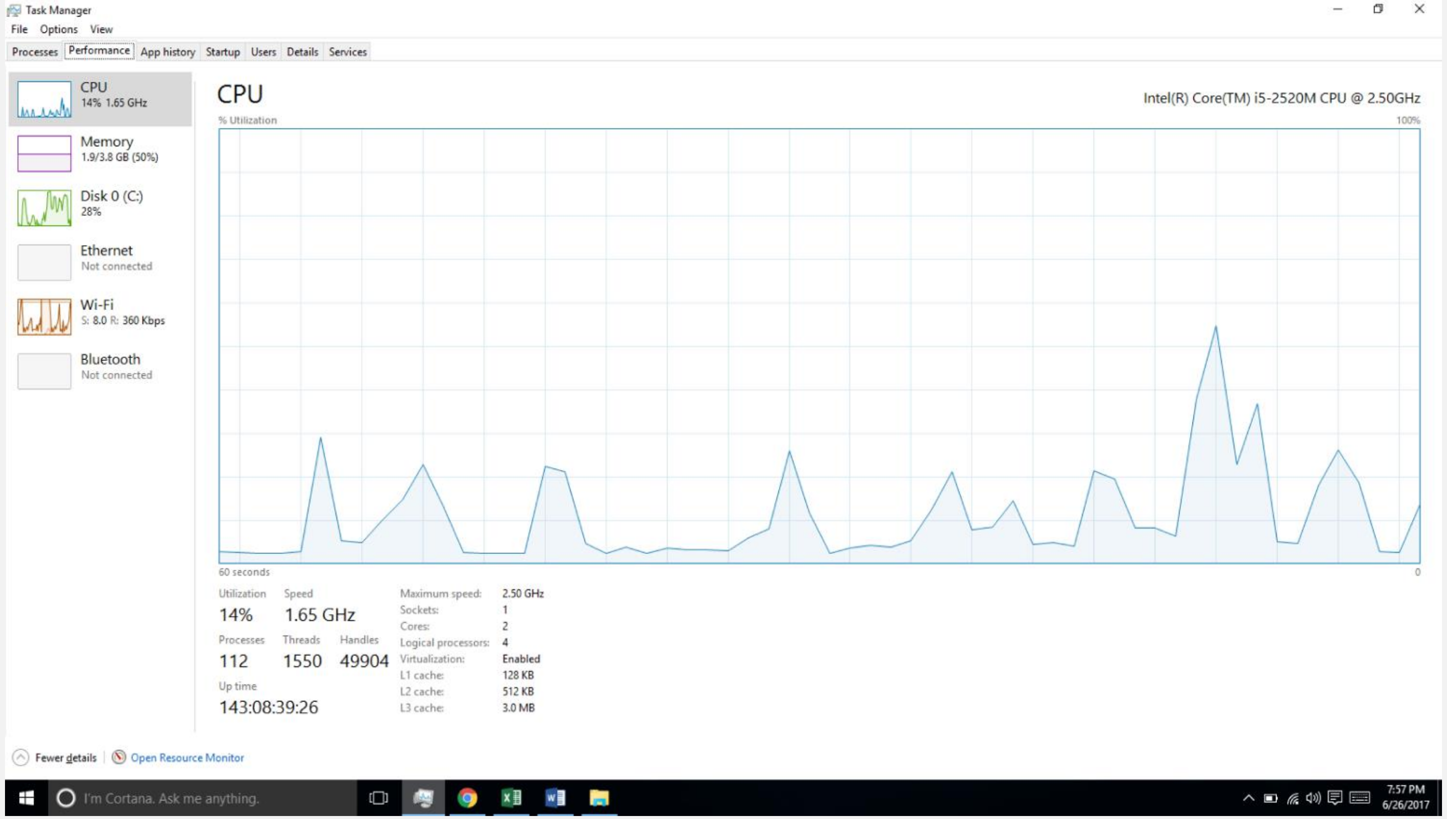
- Improve performance by removing bottlenecks
- OS must provide means of computing and displaying measures of system behavior
- **Example:** “top” program or Windows Task Manager

- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- ✓ Operating-System Debugging

OPERATING-SYSTEM DEBUGGING

Performance Tuning

The Windows 10 Task Manager



OPERATING-SYSTEM DEBUGGING

Counters

- Operating systems **keep track of system activity** through a series of **counters**
  - the number of system calls made, or
  - the number of operations performed to a network device or disk

- **Counter-based tools** are tools that simply **inquire on the current value of certain statistics that are maintained by the kernel**

- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- ✓ Operating-System Debugging

OPERATING-SYSTEM DEBUGGING

Counters

- **Tools include:**
  - **ps** — reports information for a single process or selection of processes
  - **top** — reports real-time statistics for current processes
  - **vmstat** — reports memory-usage statistics
  - **netstat** — reports statistics for network interfaces
  - **iostat** — reports I/O usage for disks

OPERATING-SYSTEM DEBUGGING

- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- ✓ Operating-System Debugging

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

- **Outline**
- **Why Applications are Operating-System Specific?**
- **Operating-System Design and Implementation**
- **Operating-System Structure**
- **Building and Booting an Operating System**

✓ **Operating-System Debugging**

OPERATING-SYSTEM DEBUGGING

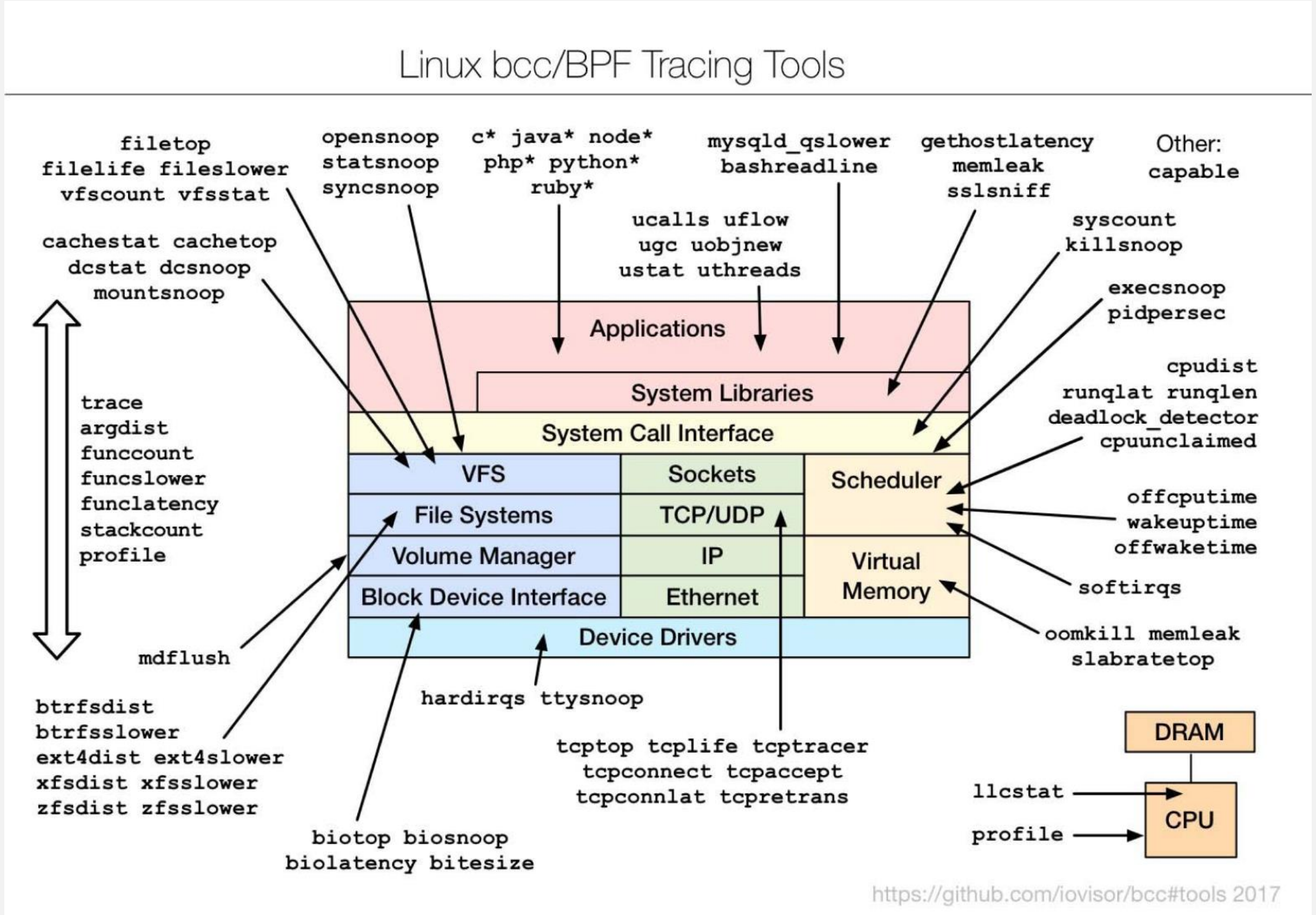
BCC

- **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**
- **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

OPERATING-SYSTEM DEBUGGING

- Outline
- Why Applications are Operating-System Specific?
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- ✓ Operating-System Debugging



2

CS 3104
OPERATING SYSTEMS

***** END *****

THANK YOU!