Operating Systems
[2. Operating-System Structures]

Chung-Wei Lin
cwlin@csie.ntu.edu.tw
CSIE Department
National Taiwan University

From Operating System Concepts (10th Edition)

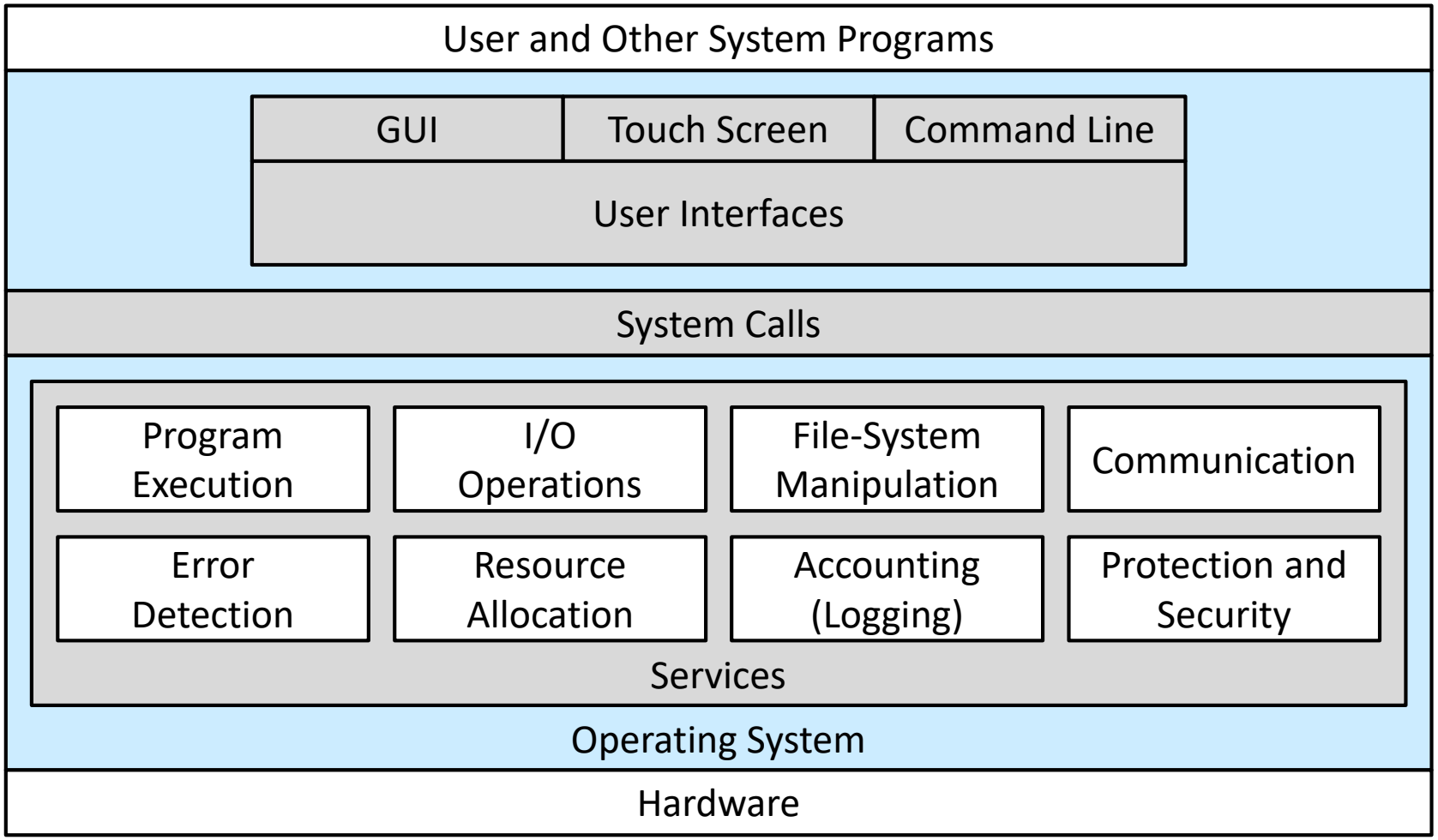
Objectives

- Identify services provided by an operating system
- Illustrate how system calls are used to provide operating system services
- Compare and contrast monolithic, layered, microkernel, modular, and hybrid strategies for designing operating systems
- Illustrate the process for booting an operating system

Outline

- **Operating-System Services**
- User and Operating-System Interface
- System Calls
- System Services
- Linkers and Loaders
- Why Applications Are Operating-System Specific
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

View of Operating System Services



Operating-System Services (1/3)

- Provide functions that are helpful to the user
  - **User interface**
    - Almost all operating systems have a **user interface** (UI)
    - Examples: graphics user interface (GUI), touch-screen, command-line interface
  - **Program execution**
    - The system must be able to load a program into memory and run the program
    - The program must be able to end its execution, either normally or abnormally
  - **I/O operations**
    - A running program may require I/O, which may involve a file or an I/O device

Operating-System Services (2/3)

- Provide functions that are helpful to the user
  - **File-system manipulation**
    - Programs need to read and write files and directories, create and delete them by name, search for a given file, and list file information
    - Some operating systems include permissions management to allow or deny access to files or directories based on file ownership
  - **Communication**
    - One process needs to exchange information with another process on the same computer or on different computer systems
    - Communications may be implemented via shared memory or message
  - **Error detection**
    - The operating system needs to be detecting and correcting errors constantly
    - Errors may occur in the CPU and memory hardware, in I/O devices, and in the user program

Operating-System Services (3/3)

- Ensure the efficient operation of the system itself
  - **Resource allocation**
    - When there are multiple processes running at the same time, resources must be allocated to each of them
    - Examples: CPU cycles, main memory, file storage, I/O devices
  - **Accounting (logging)**
    - Keep track of which programs use how much and what kinds of computer resources
  - **Protection and security**
    - Concurrent processes should not interfere with each other
    - Protection involves ensuring that all access to system resources is controlled
    - Security of the system from outsiders requires user authentication and extends to defending external I/O devices from invalid access attempts

Outline

- Operating-System Services
- **User and Operating-System Interface**
- System Calls
- System Services
- Linkers and Loaders
- Why Applications Are Operating-System Specific
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

User and Operating-System Interface

- Command interpreters
  - On systems with multiple command interpreters to choose from, the interpreters are known as **shells**
  - A **shell script** is not compiled into executable code but rather is interpreted by the command-line interface
- Graphical user interface
- Touch-screen interface

Outline

- Operating-System Services
- User and Operating-System Interface
- **System Calls**
  - Example, Application Programming Interface, Types of System Calls
- System Services
- Linkers and Loaders
- Why Applications Are Operating-System Specific
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

System Calls

- **System calls** provide an interface to the services made available by an operating system
- Generally available as functions written in C and C++
  - Although certain low-level tasks may have to be written using assembly-language instructions

Outline

- Operating-System Services
- User and Operating-System Interface
- **System Calls**
  - **Example**, Application Programming Interface, Types of System Calls
- System Services
- Linkers and Loaders
- Why Applications Are Operating-System Specific
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

Example of System Calls

- Copy the contents of one file to another file
  - Example command: `cp in.txt out.txt`
  - Example sequence:
- Many other system calls are there

```
Acquire input file name
    Write prompt to screen
    Accept input
Acquire output file name
    Write prompt to screen
    Accept input
Open the input file
    if file doesn't exist, abort
Create output file
    if file exists, abort
Loop
    Read from input file
    Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

Outline

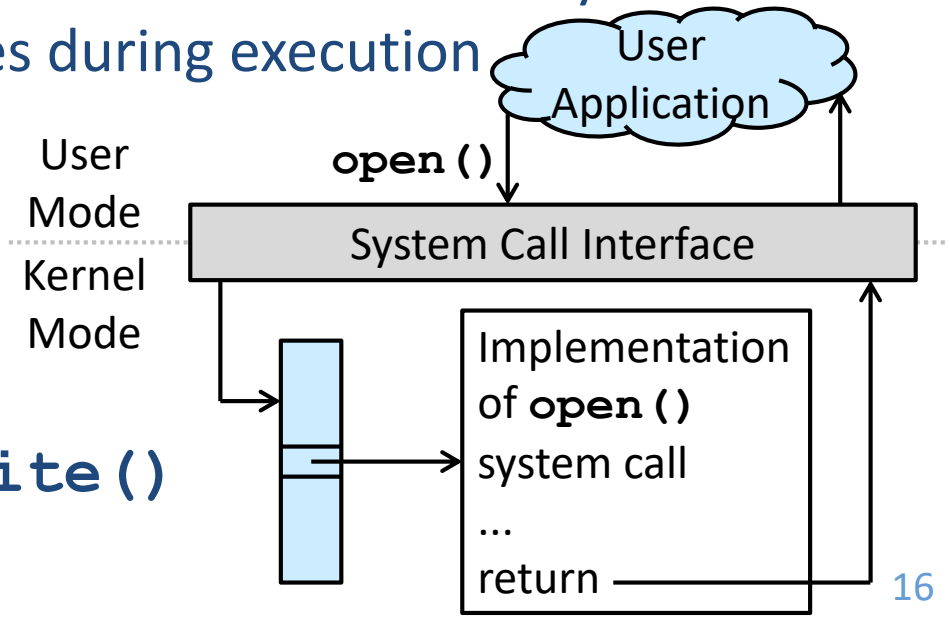
- Operating-System Services
- User and Operating-System Interface
- **System Calls**
  - Example, **Application Programming Interface**, Types of System Calls
- System Services
- Linkers and Loaders
- Why Applications Are Operating-System Specific
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

Application Programming Interface

- Typically, application developers design programs according to an **application programming interface** (API)
  - Rather than direct system call use
  - Reasons: portability and easiness
  - Example: the Windows function `CreateProcess()` actually invokes the `NTCreateProcess()` system call in the Windows kernel
- Most common APIs are
  - Windows API for Windows systems
  - POSIX API for POSIX-based systems
    - Virtually all versions of UNIX, Linux, and Mac OS X
  - Java API for the Java virtual machine
- Note that the system-call names used throughout this text are mostly generic

System-Call Interface

- The run-time environment (RTE) provides **system-call interface**
  - Intercept function calls in the API
  - Invoke the necessary system calls within the operating system
    - Typically, a table is indexed according to numbers associated with system calls
  - Return the status of the system call
- The caller needs to know NOTHING about how the system call is implemented or what it does during execution
  - Only need to
    - Obey the API
    - Understand what the operating system will do as a result of the execution of that system call
- Example: `printf()` and `write()`



Parameter Passing

- Approach 1
  - Pass the parameters in registers
    - There may be more parameters than registers
- Approach 2
  - The parameters are generally stored in a block (or table) in memory
    - The address of the block is passed as a parameter in a register
- Linux uses a combination of Approaches 1 and 2
- Approach 3
  - Parameters also can be placed (or pushed) onto a stack by the program and popped off the stack by the operating system
    - Some operating systems prefer this because those approaches do not limit the number of parameters being passed

Outline

- Operating-System Services
- User and Operating-System Interface
- **System Calls**
  - Example, Application Programming Interface, **Types of System Calls**
- System Services
- Linkers and Loaders
- Why Applications Are Operating-System Specific
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

Types of System Calls (1/3)

- Process control
  - create process, terminate process
  - load, execute
  - get process attributes, set process attributes
  - wait event, signal event
  - allocate and free memory
  - end, abort
  - **Debugger**
    - A system program designed to aid the programmer in finding and correcting errors or bugs
  - **Lock** shared data

Types of System Calls (2/3)

- File management
  - create file, delete file
  - open, close
  - read, write, reposition
  - get file attributes, set file attributes
- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices

Types of System Calls (3/3)

- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get process, file, or device attributes
  - set process, file, or device attributes
- Communications
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach or detach remote devices
  - **Message-passing model** and **shared-memory model**
- Protection
  - get file permissions, set file permissions

Windows and Unix System Calls

Types	Windows	Unix
Process Control	`CreateProcess()`	`fork()`
	`ExitProcess()`	`exit()`
	`WaitForSingleObject()`	`wait()`
File Management	`CreateFile()`	`open()`
	`ReadFile()`	`read()`
	`WriteFile()`	`write()`
	`CloseHandle()`	`close()`
Device Management	`SetConsoleMode()`	`ioctl()`
	`ReadConsole()`	`read()`
	`WriteConsole()`	`write()`
Information Maintenance	`GetCurrentProcessID()`	`getpid()`
	`SetTimer()`	`alarm()`
	`Sleep()`	`sleep()`
Communications	`CreatePipe()`	`pipe()`
	`CreateFileMapping()`	`shm_open()`
	`MapViewOfFile()`	`mmap()`
Protection	`SetFileSecurity()`	`chmod()`
	`InitlializeSecurityDescriptor()`	`umask()`
	`SetSecurityDescriptorGroup()`	`chown()`

Outline

- Operating-System Services
- User and Operating-System Interface
- System Calls
- **System Services**
- Linkers and Loaders
- Why Applications Are Operating-System Specific
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

System Services

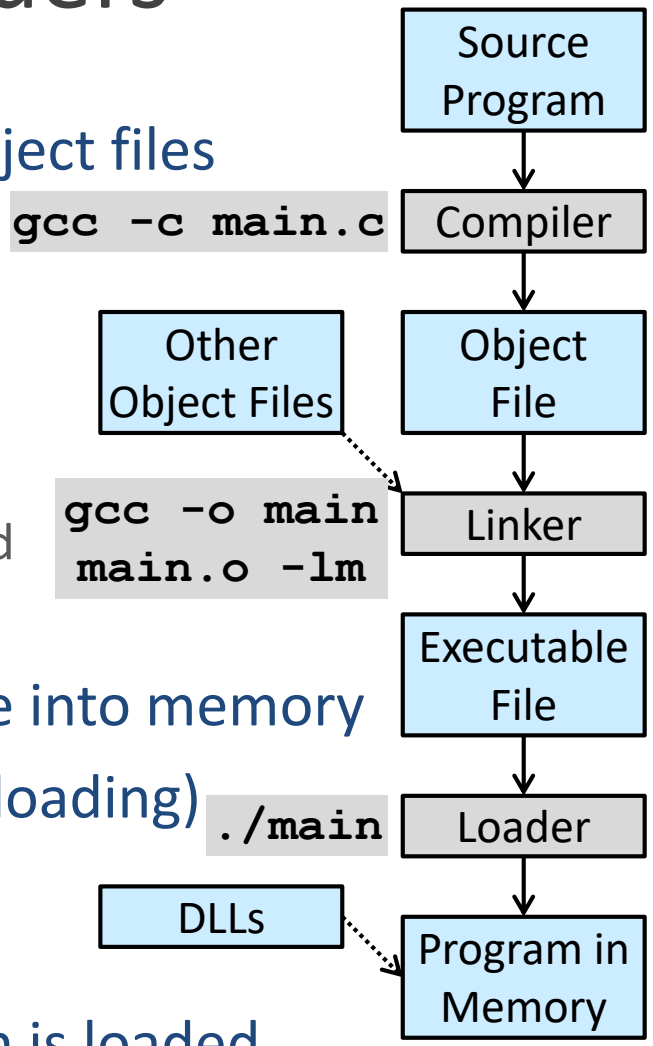
- Provide a convenient environment for program development and execution
  - File management
  - Status information
  - File modification
  - Programming-language support
  - Program loading and execution
  - Communications
  - Background services (services, subsystems, or daemons)
  - Application programs
- The view of the operating system seen by most users is defined by the system and application programs
  - Rather than by the system calls

Outline

- Operating-System Services
- User and Operating-System Interface
- System Calls
- System Services
- **Linkers and Loaders**
- Why Applications Are Operating-System Specific
- Operating-System Design and Implementation
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

Linkers and Loaders

- A compiler compiles source files into object files
  - **Relocatable object file** to be loaded into any physical memory location
- A **linker** combines relocatable object files into a binary **executable** file
  - Other object files or libraries may be included
  - Examples: standard C or math library (`-lm`)
- A **loader** loads the binary executable file into memory
- Relocation (associated with linking and loading)
  - Assign final addresses to the program parts
- Most systems allow a program to dynamically link libraries as the program is loaded
  - Example: Windows dynamically linked libraries (DLLs)



Outline

- Operating-System Services
- User and Operating-System Interface
- System Calls
- System Services
- Linkers and Loaders
- **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

- An application can be made to run on multiple OSs
  - Written in an interpreted language with interpreters on multiple OSs
  - Written in a language that includes a virtual machine
  - Use a standard language or API and compile separately
- Challenges
  - Different APIs
  - Different file formats
  - Different CPU instruction sets
  - Different system calls

Outline

- Operating-System Services
- User and Operating-System Interface
- System Calls
- System Services
- Linkers and Loaders
- Why Applications Are Operating-System Specific
- **Operating-System Design and Implementation**
- Operating-System Structure
- Building and Booting an Operating System
- Operating-System Debugging

Design and Implementation

- Design goals
  - At the highest level, the design will be affected by the choice of hardware and the type of systems
  - The lower-level requirements may be much harder to specify
    - User goals and system goals
- Policies and mechanisms
  - Respectively determine what will be done and how to do something
- Implementation
  - Most in higher-level languages and part in assembly language
  - Example: Android
    - Kernel mostly in C with some assembly language; system libraries in C or C++; application frameworks mostly in Java
- Software engineering

Outline

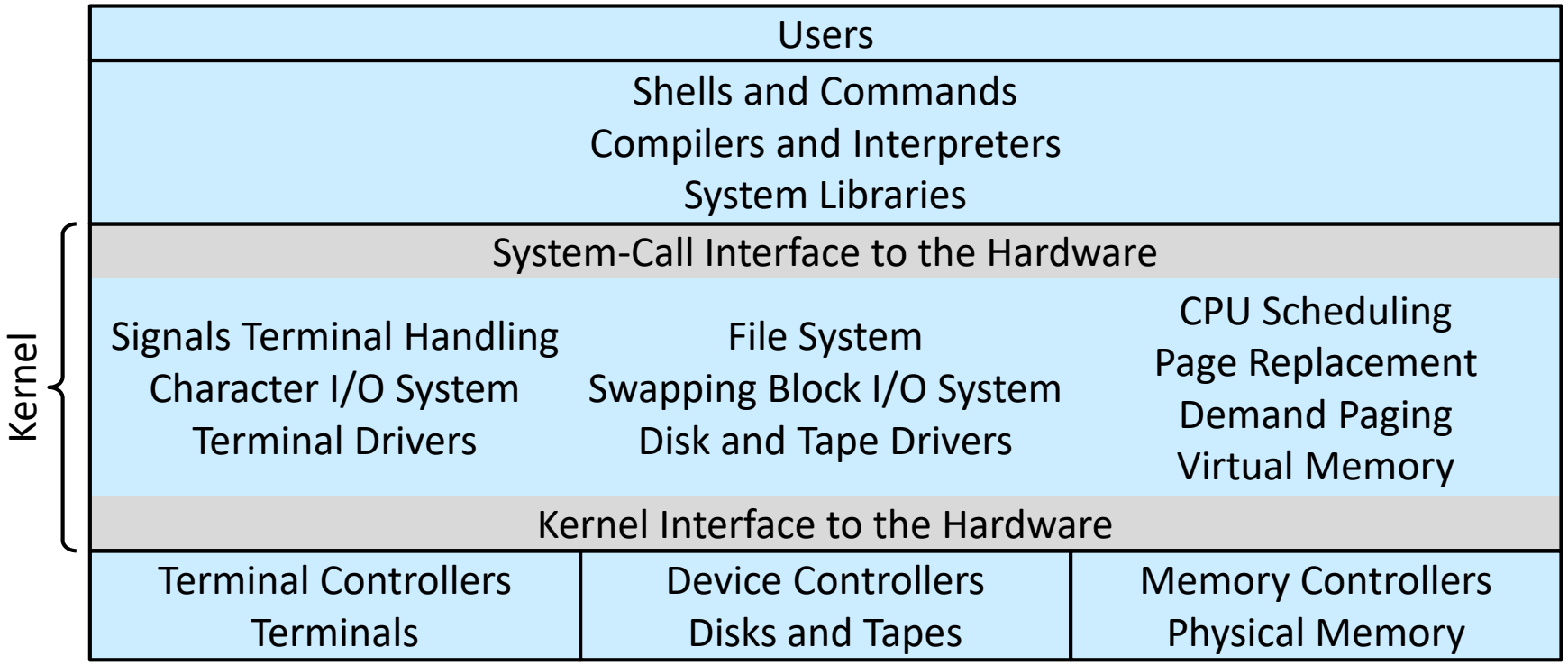
- Operating-System Services
- User and Operating-System Interface
- System Calls
- System Services
- Linkers and Loaders
- Why Applications Are Operating-System Specific
- Operating-System Design and Implementation
- **Operating-System Structure**
  - **Monolithic Structure**, Layered Approach, Microkernels, Modules, Hybrid Systems
- Building and Booting an Operating System
- Operating-System Debugging

Monolithic Structure

- The simplest structure for organizing an operating system is no structure at all
  - Speed and efficiency
    - Little overhead in the system-call interface
    - Fast communication within the kernel
  - Simple
  - Difficult to implement and extend

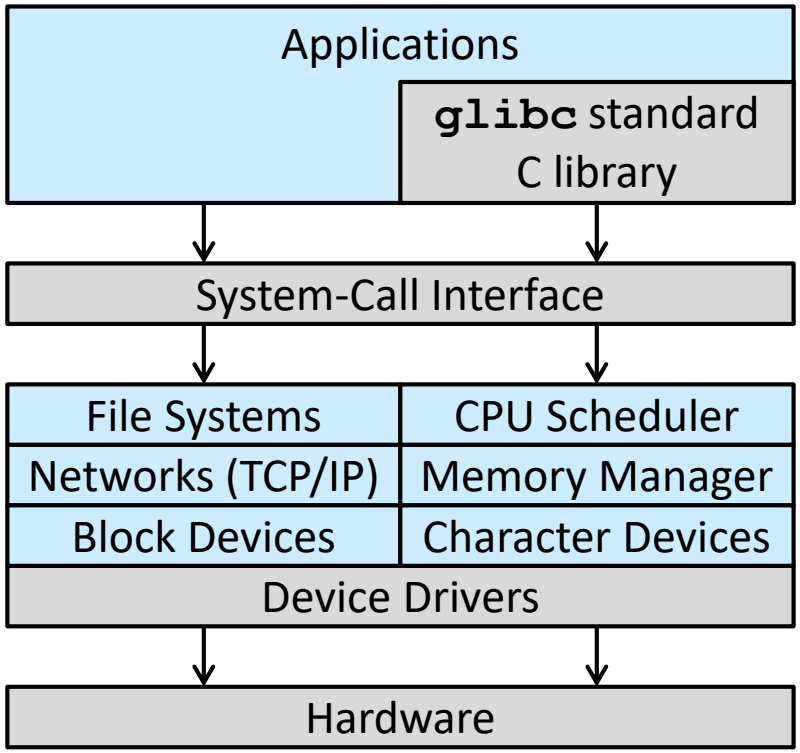
Traditional UNIX System Structure

- Two separable parts
  - The kernel (further separated into a series of interfaces and device)
  - System programs
- Everything below the system-call interface and above the physical hardware is the kernel



Linux System Structure

- The Linux kernel is monolithic in that it runs entirely in kernel mode in a single address space
  - It does have a modular design that allows the kernel to be modified during run time
- Applications typically use the **`glibc`** standard C library when communicating with the system call interface to the kernel

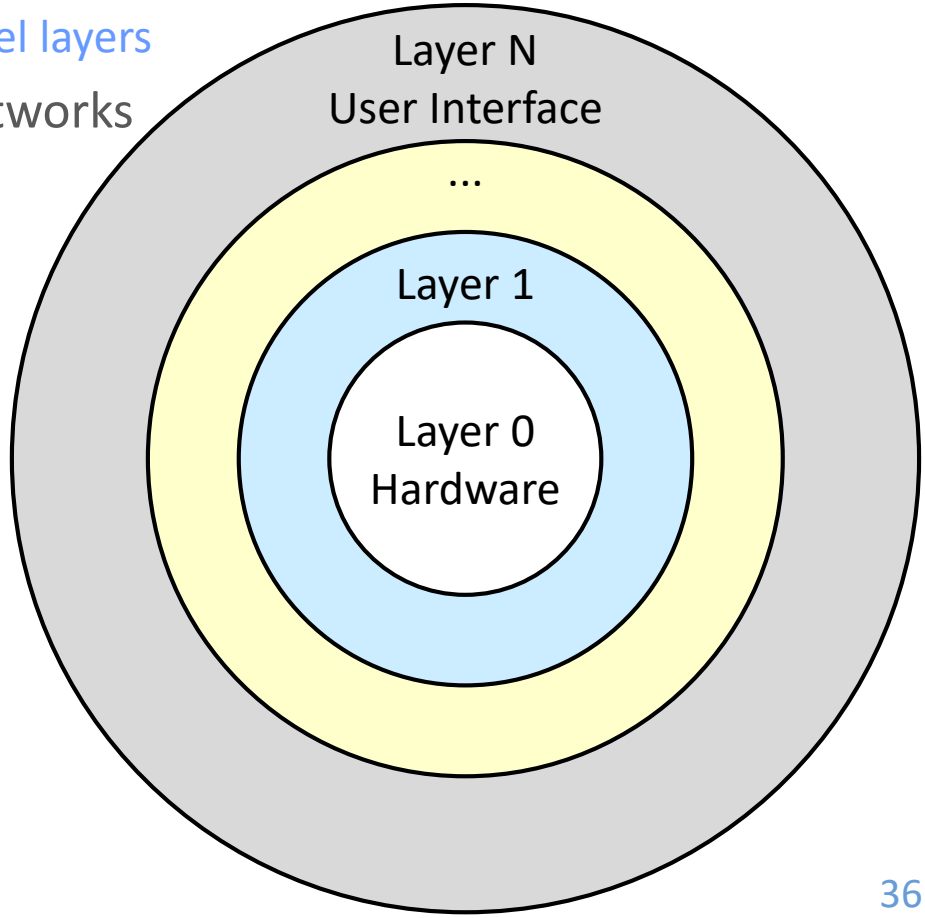


Outline

- Operating-System Services
- User and Operating-System Interface
- System Calls
- System Services
- Linkers and Loaders
- Why Applications Are Operating-System Specific
- Operating-System Design and Implementation
- **Operating-System Structure**
  - Monolithic Structure, **Layered Approach**, Microkernels, Modules, Hybrid Systems
- Building and Booting an Operating System
- Operating-System Debugging

Layered Approach

- The operating system is broken into a number of layers (levels)
  - A layer
    - Can be invoked by higher-level layers
    - Can invoke operations on lower-level layers
  - Successfully used in computer networks
  - Challenges
    - Layer functionality definition
    - Poor overall performance

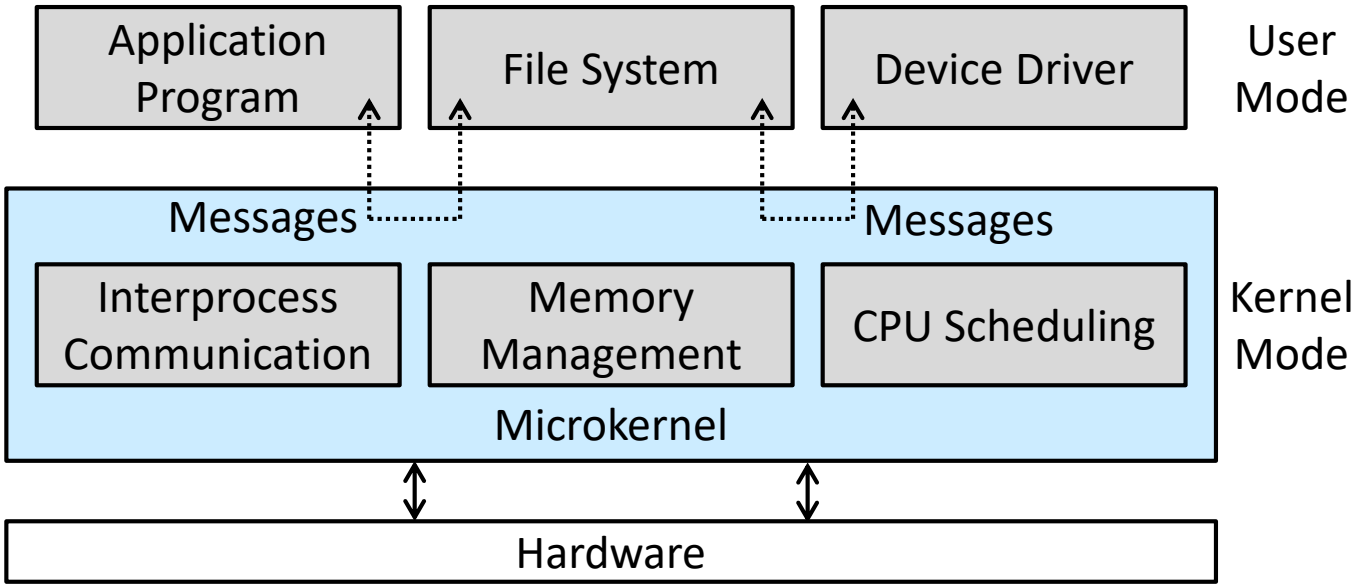


Outline

- Operating-System Services
- User and Operating-System Interface
- System Calls
- System Services
- Linkers and Loaders
- Why Applications Are Operating-System Specific
- Operating-System Design and Implementation
- **Operating-System Structure**
  - Monolithic Structure, Layered Approach, **Microkernels**, Modules, Hybrid Systems
- Building and Booting an Operating System
- Operating-System Debugging

Microkernels

- Remove all nonessential components from the kernel and implement them as user level programs
  - Examples: Mach, part of Darwin, QNX
  - Communication through message passing
  - Better extensibility, portability, security, and reliability
  - Poor performance



Outline

- Operating-System Services
- User and Operating-System Interface
- System Calls
- System Services
- Linkers and Loaders
- Why Applications Are Operating-System Specific
- Operating-System Design and Implementation
- **Operating-System Structure**
  - Monolithic Structure, Layered Approach, Microkernels, **Modules**, Hybrid Systems
- Building and Booting an Operating System
- Operating-System Debugging

Modules

- **Loadable kernel modules** (LKMs)
  - The kernel
    - Have a set of core components
    - Link in additional services via modules, either at boot time or during run time
  - Similar to the layered approach but more flexible
  - Similar to the microkernel approach but more efficient
  - Common in modern implementations
    - UNIX (Linux, macOS, Solaris) as well as Windows

Outline

- Operating-System Services
- User and Operating-System Interface
- System Calls
- System Services
- Linkers and Loaders
- Why Applications Are Operating-System Specific
- Operating-System Design and Implementation
- **Operating-System Structure**
  - Monolithic Structure, Layered Approach, Microkernels, Modules, **Hybrid Systems**
- Building and Booting an Operating System
- Operating-System Debugging

Hybrid Systems

- Most modern operating systems are hybrid systems
- Linux
  - Monolithic
    - Having it in a single address space provides very efficient performance
  - Modular
    - New functionality can be dynamically added to the kernel
- Windows
  - Monolithic (again for performance reasons)
  - Some microkernel systems
    - Provide support for separate subsystems (known as operating-system personalities) that run as user-mode processes
  - Modular
    - Provide support for dynamically loadable kernel modules

Outline

- Operating-System Services
- User and Operating-System Interface
- System Calls
- System Services
- Linkers and Loaders
- 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-system generation
  - Write or obtain 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
- System boot
  - A small piece of code known as the bootstrap program or boot loader locates the kernel
  - The kernel is loaded into memory and started
  - The kernel initializes hardware
  - The root file system is mounted

Outline

- Operating-System Services
- User and Operating-System Interface
- System Calls
- System Services
- Linkers and Loaders
- 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

- Failure analysis
  - If a process fails, most OSs write the error information to a **log file** and save the memory state of the process to a **core dump**
  - If a **crash** (a failure in the kernel) occurs, error information is written to a log file, and the memory state is saved to a **crash dump**
- Performance monitoring and tuning
  - Per-process or system-wide observations
  - Counters and tracing
- Debugging the interactions between user-level and kernel code is nearly impossible without a toolset
  - BPF compiler collection (BCC) provides tracing features for Linux systems

Objectives

- Identify services provided by an operating system
- Illustrate how system calls are used to provide operating system services
- Compare and contrast monolithic, layered, microkernel, modular, and hybrid strategies for designing operating systems
- Illustrate the process for booting an operating system

Q&A