Operating-System Structures (Part-2)

System Services

- Provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
- File management - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- Status information
  - Some ask the system for info - date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugging information
  - Typically, these programs format and print the output to the terminal or other output devices
  - Some systems implement a registry - used to store and retrieve configuration information

System Services (Cont.)

- File modification
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- Programming-language support - Compilers, assemblers, debuggers and interpreters sometimes provided
- Program loading and execution- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- Communications - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

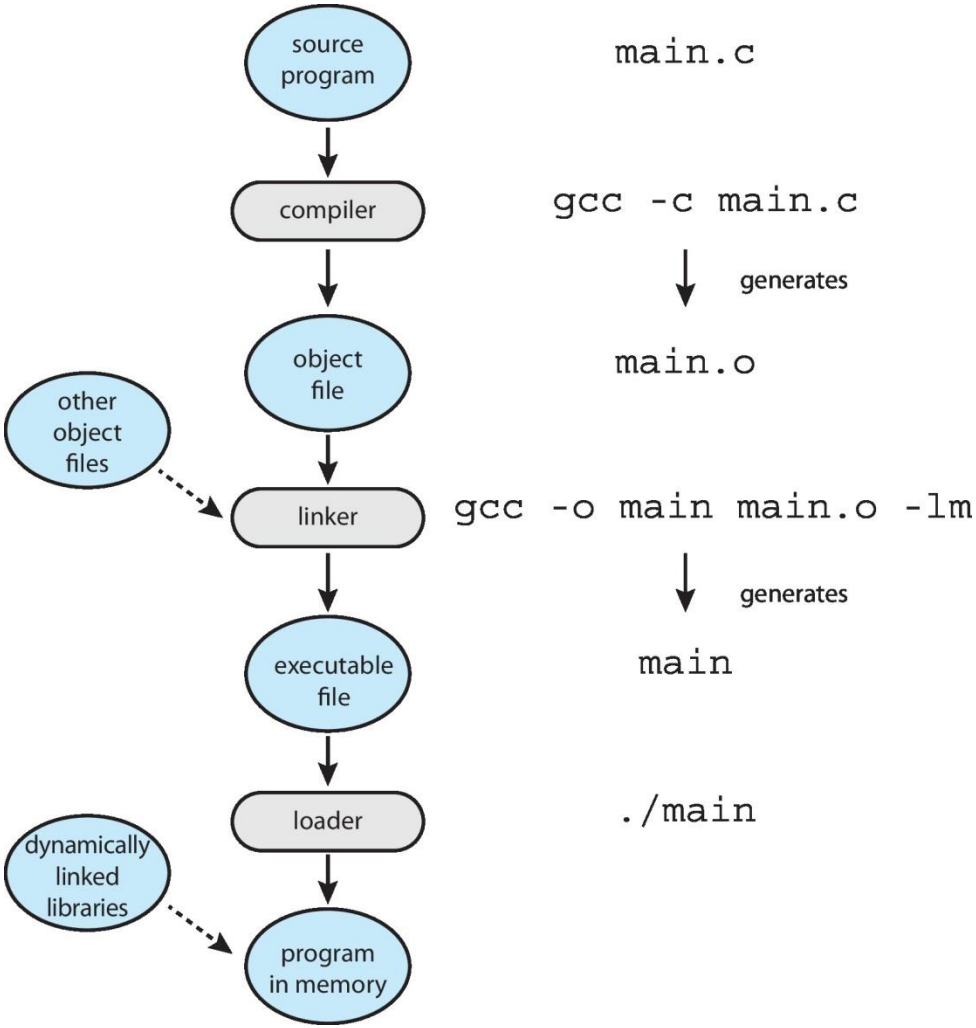
System Services (Cont.)

- Background Services
  - Launch at boot time
    - Some for system startup, then terminate
    - Some from system boot to shutdown
  - Provide facilities like disk checking, process scheduling, error logging, printing
  - Run in user context not kernel context
  - Known as services, subsystems, daemons
- Application programs
  - Don’t pertain to system
  - Run by users
  - Not typically considered part of OS
  - Launched by command line, mouse click, finger poke

Linkers and Loaders

- Source code compiled into object files designed to be loaded into any physical memory location – relocatable object file
- Linker combines these into single binary executable file
  - Also brings in libraries
- Program resides on secondary storage as binary executable
- Must be brought into memory by loader to be executed
  - Relocation assigns final addresses to program parts and adjusts code and data in program to match those addresses
- Modern general purpose systems don't link libraries into executables
  - Rather, dynamically linked libraries (in Windows, DLLs) are loaded as needed, shared by all that use the same version of that same library (loaded once)
- Object, executable files have standard formats, so operating system knows how to load and start them

The Role of the Linker and Loader



Why Applications are Operating System Specific

- Apps compiled on one system usually not executable on other operating systems
- Each operating system provides its own unique system calls
  - Own file formats, etc.
- Apps can be multi-operating system
  - Written in interpreted language like Python, Ruby, and interpreter available on multiple operating systems
  - App written in language that includes a VM containing the running app (like Java)
  - Use standard language (like C), compile separately on each operating system to run on each
- **Application Binary Interface** (**ABI**) is architecture equivalent of API, defines how different components of binary code can interface for a given operating system on a given architecture, CPU, etc.

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

Implementation

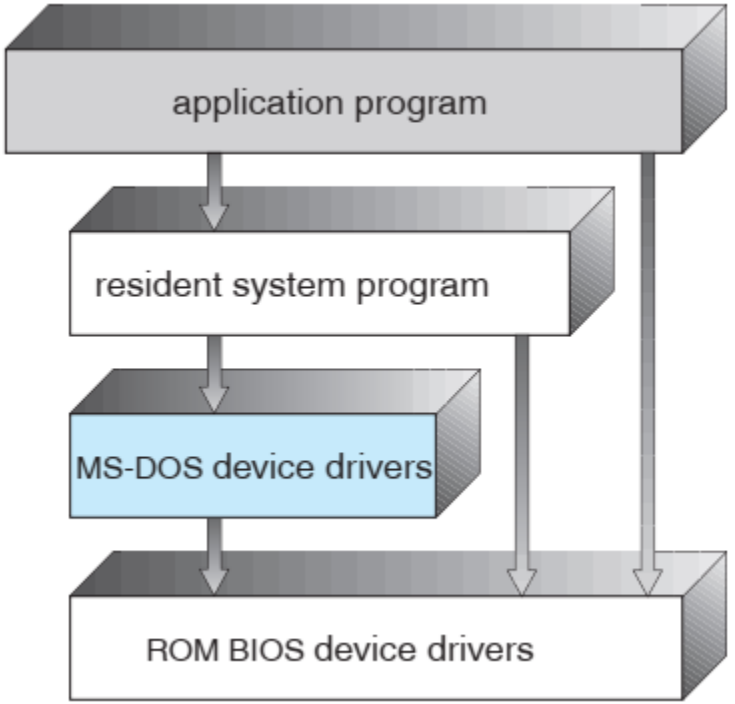
- Much variation
  - Early OSes in assembly language
  - Then system programming languages like Algol, PL/1
  - Now C, C++
- Actually usually a mix of languages
  - Lowest levels in assembly
  - Main body in C
  - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
- More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

Operating System Structure

- General-purpose OS is very large program.
- Various ways to structure ones
  - Simple structure
  - Monolithic structure
  - Layered approach
  - Microkernel
  - Modules
  - Hybrid systems

Simple Structure

- Followed by the most of the operating system that was designed in very beginning.
- Does not have well-defined structures.
- Used by MS-DOS.



Simple Structure

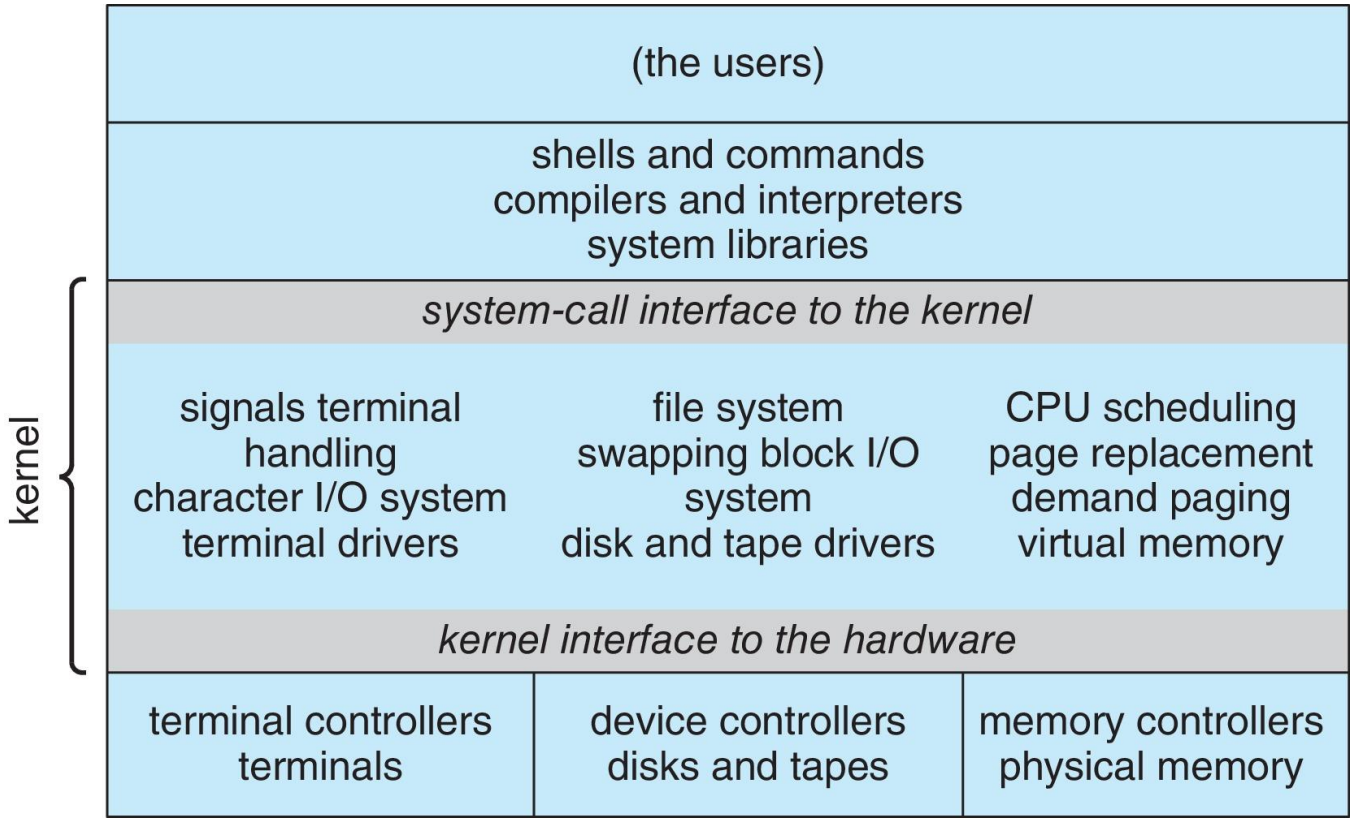
- Interfaces and levels of functionality are not well separated.
  - Application programs are able to access the hardware directly.
- Such freedom leaves MS-DOS vulnerable to errant (or malicious) programs, causing entire system crashes when user programs fail.
- Not well protected, not well structured, and not well defined.
- MS-DOS was written on Intel 8088 which does not provide dual mode and hardware protection.

Monolithic Structure

- Place all of the functionality of the kernel into a single, static binary file that runs in a single address space.
- Followed by the earlier UNIX OSes.
- Limited by hardware functionality
- 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

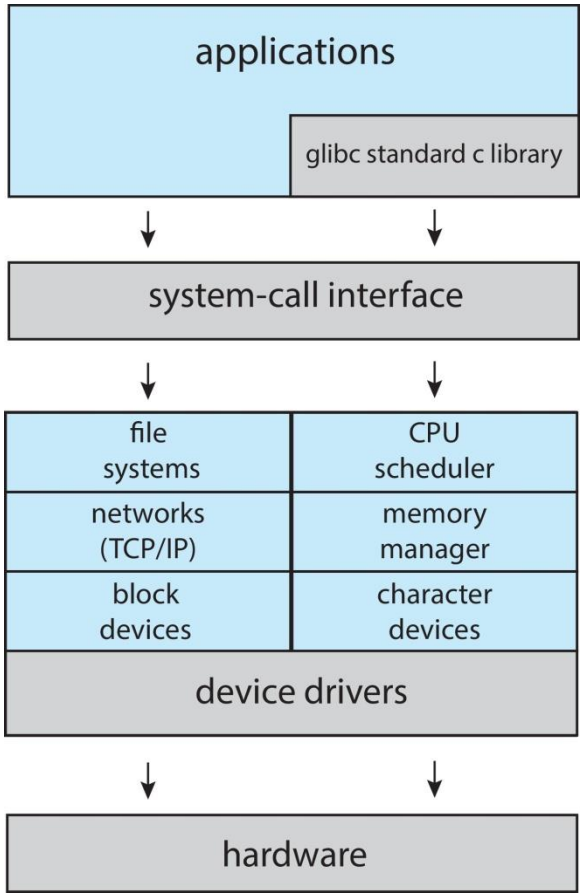
Monolithic Structure (cont.)

- Traditional UNIX System Structure



Monolithic Structure (cont.)

- Monolithic plus modular design that allows the kernel to be modified during run time.
- Linux System Structure

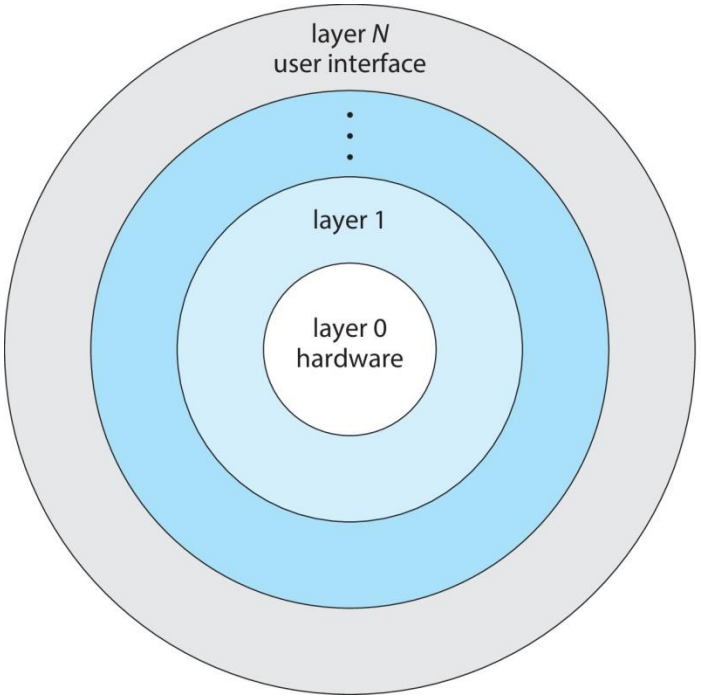


Monolithic Structure (cont.)

- Drawbacks
  - Too many functionalities are packed into one level i.e. kernel.
  - Difficult to maintain, implement and extend.
    - For instance, to fix the cpu scheduling, we have to touch entire kernel.
- Advantage
  - Speed and efficiency

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



Layered Approach (cont.)

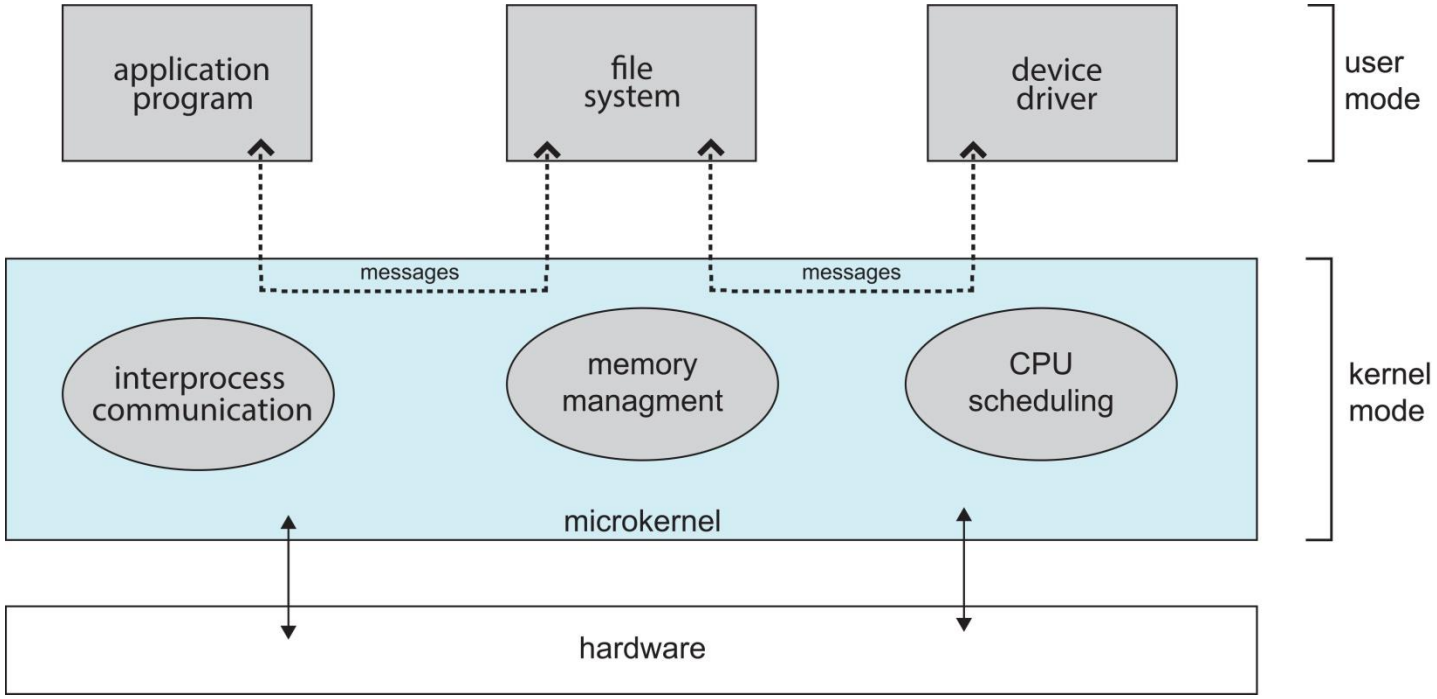
- Advantage:
  - All the layers can be defined separately and interact with each other as required.
  - Also, it is easier to create, maintain and update the system if it is done in the form of layers.
  - Change in one layer specification does not affect the rest of the layers.
  - Hardware are protected from the layers above.

- Disadvantages:
  - OS tends to be less efficient than other implementations.

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



Microkernel System Structure (cont.)

- Advantages:
  - Extending the operating system becomes much easier.
  - Any changes to the kernel tend to be fewer, since the kernel is smaller.
  - The microkernel also provides more security and reliability.

- Disadvantages:
  - Poor performance due to increased system overhead from message passing.

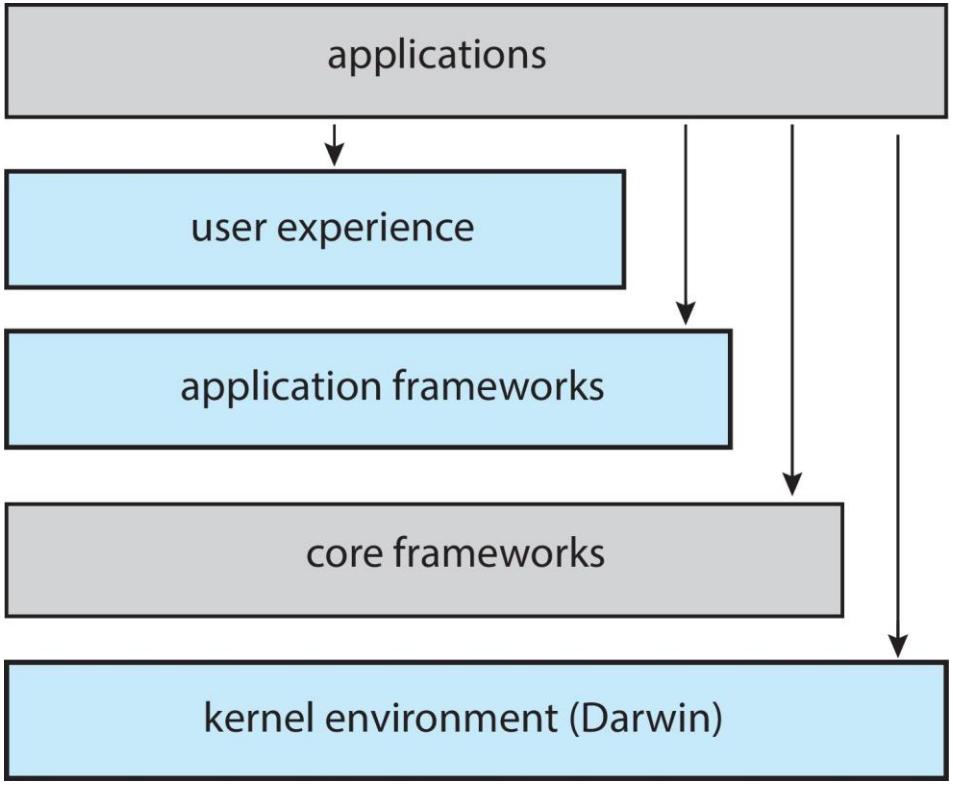
Modules

- 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

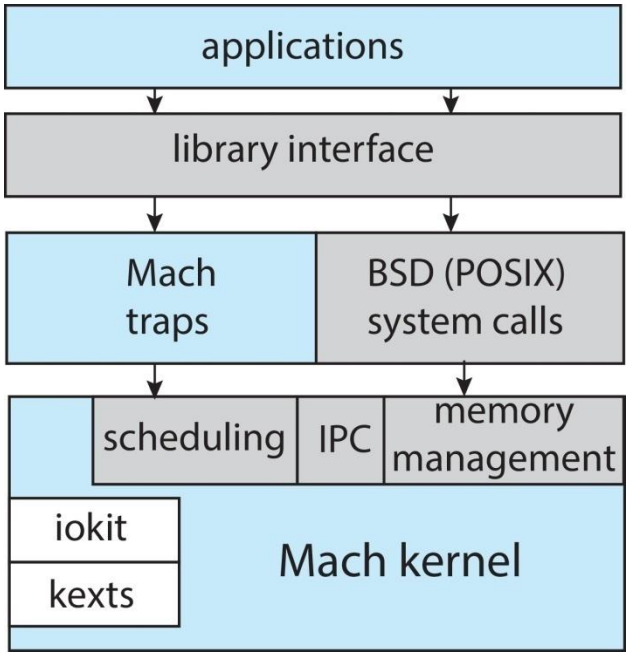
Hybrid Systems

- 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)

macOS and iOS Structure

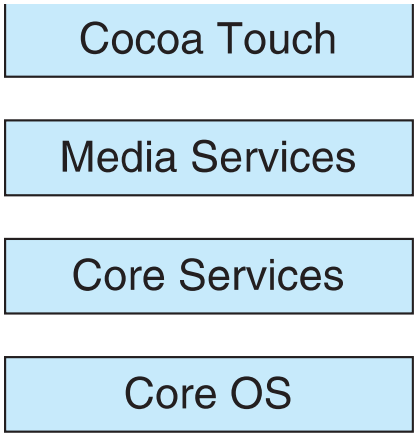


Darwin


applications
library interface
Mach traps
BSD (POSIX) system calls
scheduling
IPC
memory management
iokit
kexts
Mach kernel

iOS

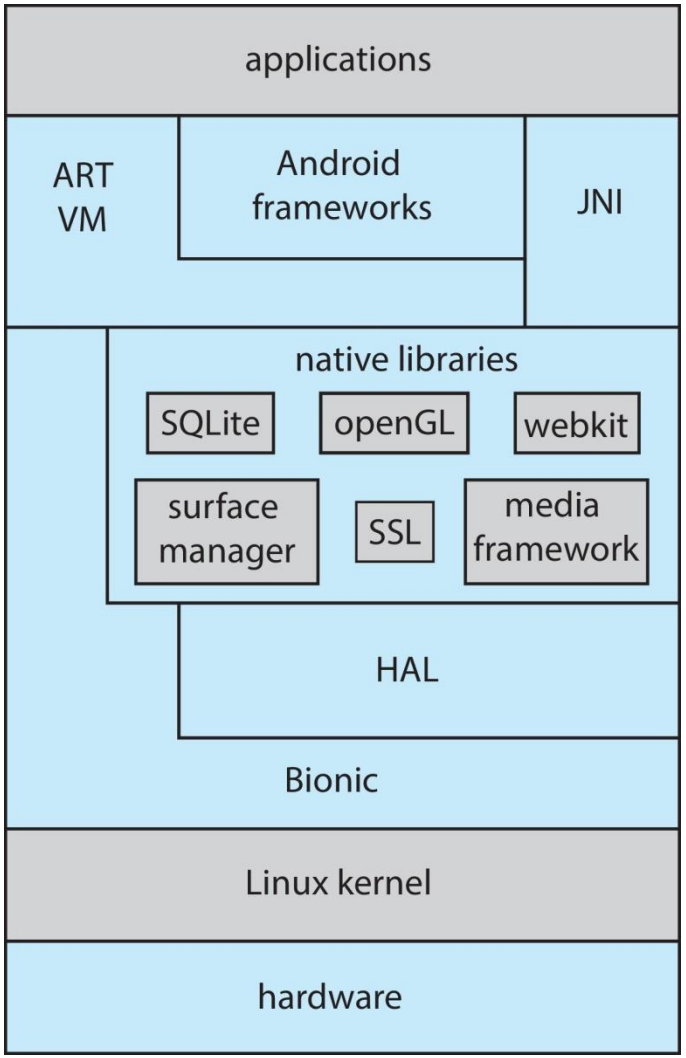
- Apple mobile OS for *iPhone*, *iPad*
  - Structured on Mac OS X, added functionality
  - Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs. Intel)
  - Cocoa Touch Objective-C API for developing apps
  - Media services layer for graphics, audio, video
  -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



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

Android Architecture



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)
- 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

- 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."