CSC 4320/6320: Operating Systems

Georgia State University

Chapter 02: Operating System Structures – Part II

Spring 2025

Instructor: Dr. Md Mahfuzur Rahman
Department of Computer Science, GSU

Disclaimer

The slides to be used in this course have been created, modified, and adapted from multiple sources:

- *The slides are copyright Silberschatz, Galvin and Gagne, 2018. The slides are authorized for personal use, and for use in conjunction with a course for which Operating System Concepts is the prescribed text. Instructors are free to modify the slides to their taste, as long as the modified slides acknowledge the source and the fact that they have been modified.*

Announcement

https://os-book.com/OS10/review-dir/index.html

MS/PhD Students: Submit your project proposal by February 10, 2025 (1-page summary of your title, brief outline and plan of actions)

Outline

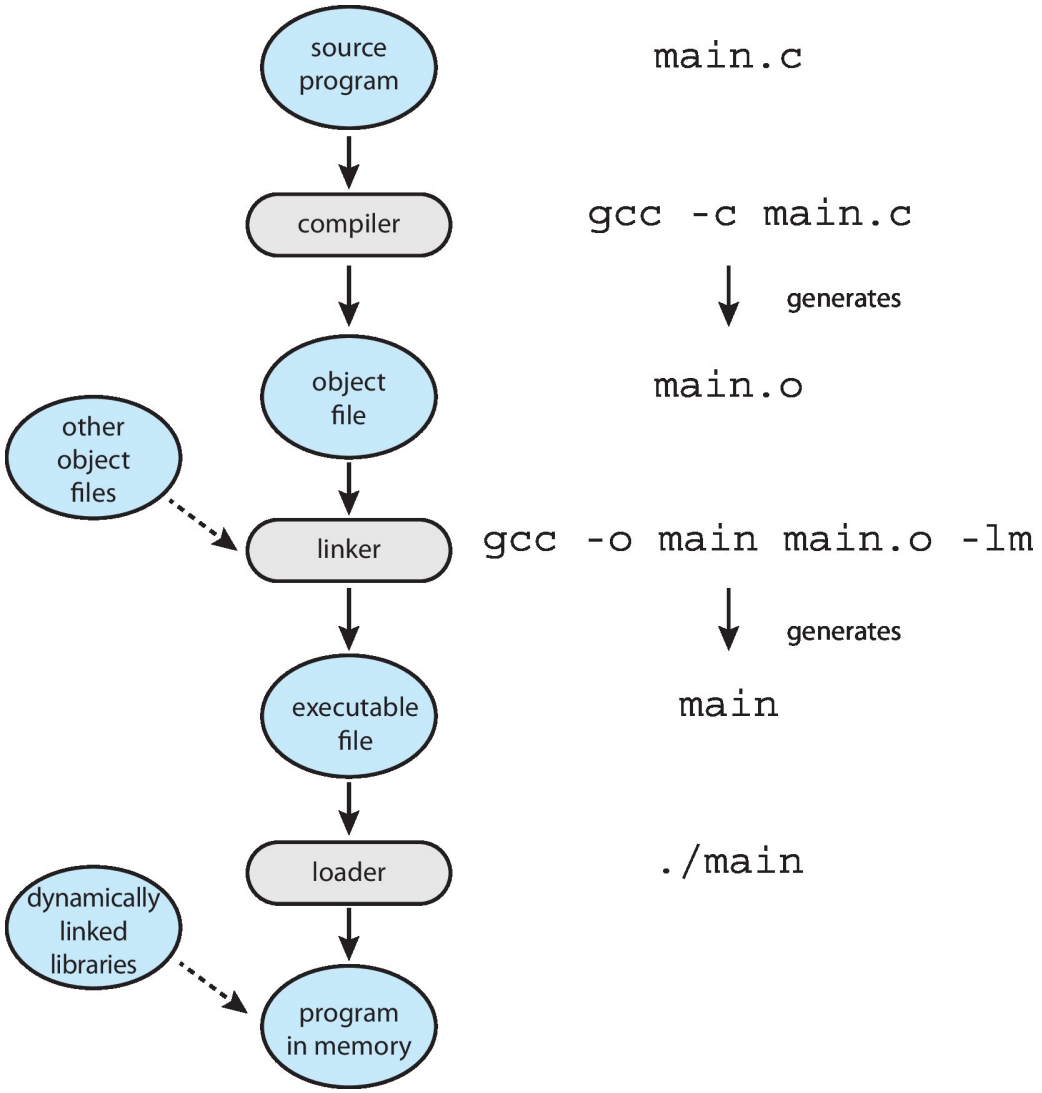
- Operating System Services (Done ✓)
- User and Operating System-Interface (Done ✓)
- System Calls (Done ✓)
- System Services (Done ✓)
- Linkers and Loaders (Done ✓)
- Why Applications are Operating System Specific
- Operating System Structure

Linkers and Loaders

Georgia State University

- 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 other 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



- The math library is not linked into the executable file main. Rather, the linker inserts relocation information
- Allows it to be dynamically linked and loaded as the program is loaded and needed during run time.

Object Code vs Executable Code

What are their differences?

An **object file** is an intermediate, non-executable file produced during the compilation process that may have unresolved dependencies.

An **executable file** is the final output after linking, fully resolved, and ready to run by the operating system.

Questions

1. Source files are compiled into object file(s) which use absolute addresses.

   True

   False ✓

2. A statically-linked library is only linked and loaded if it is conditionally required during program runtime.

   True

   False ✓

Why Applications are Operating System Specific

Georgia State University

- 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 (Unix like variants)
- **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.

Questions

Applications that are designed to work on one operating system will also work on another operating system as long as they provide the same APIs.

True

False ✓

Think about the low-level implementation – how the code will be compiled and linked; how the code will be executed on a particular CPU.

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 (mid-1980s, researchers at Carnegie Mellon University)

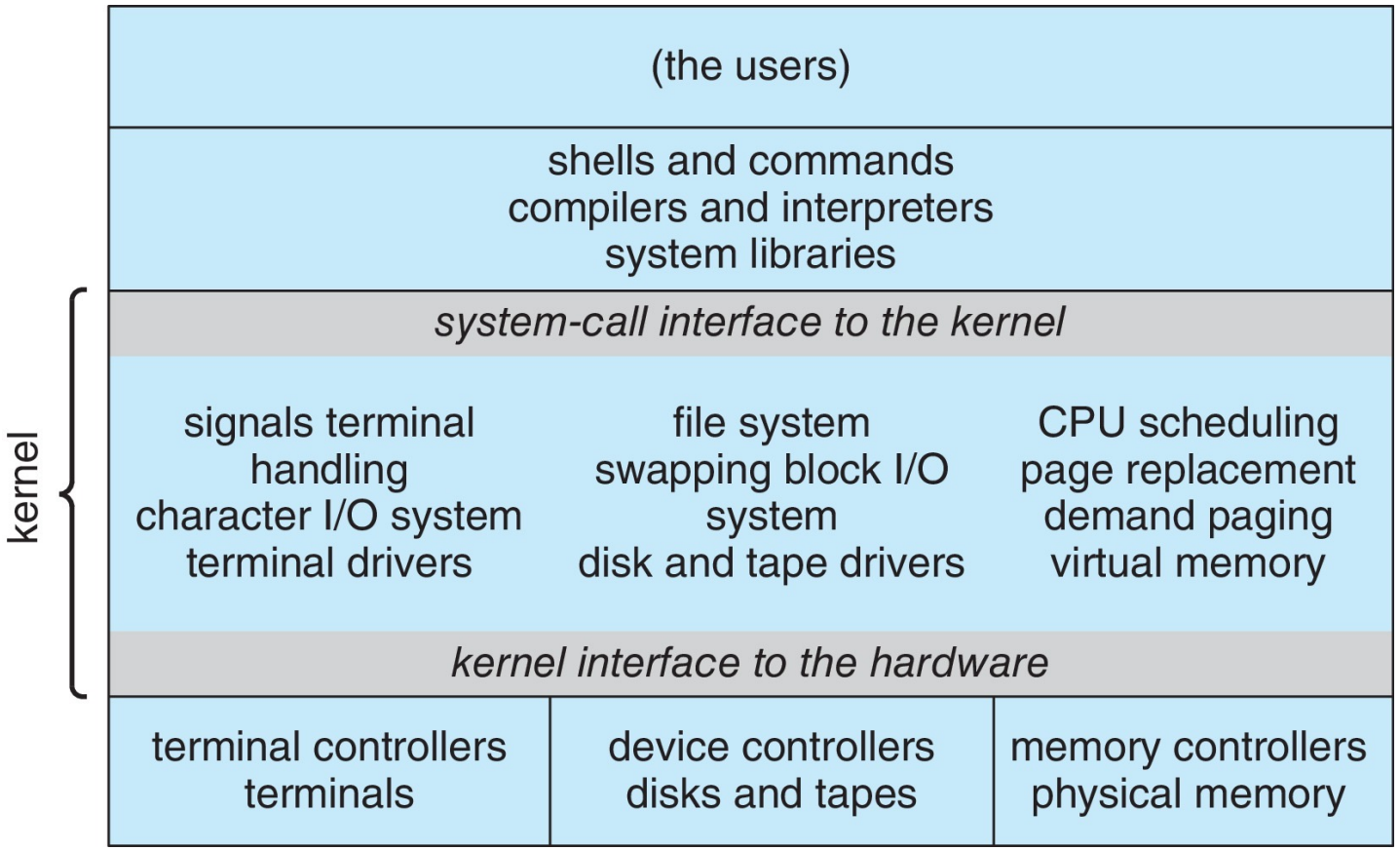
Monolithic Structure – Original UNIX

Georgia State University

- 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



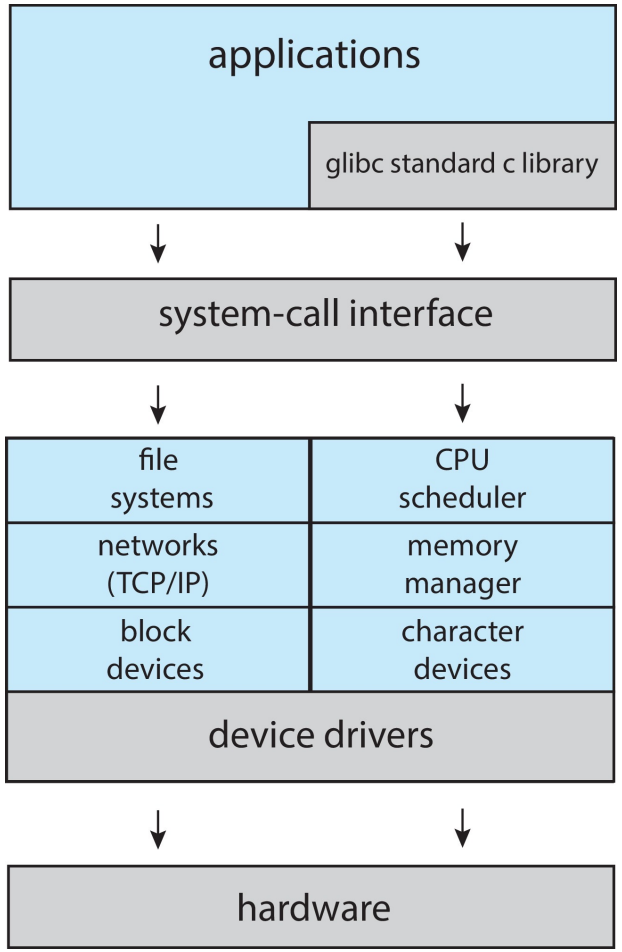
Linux System Structure

Georgia State University

Monolithic plus modular design

Monolithic kernels:

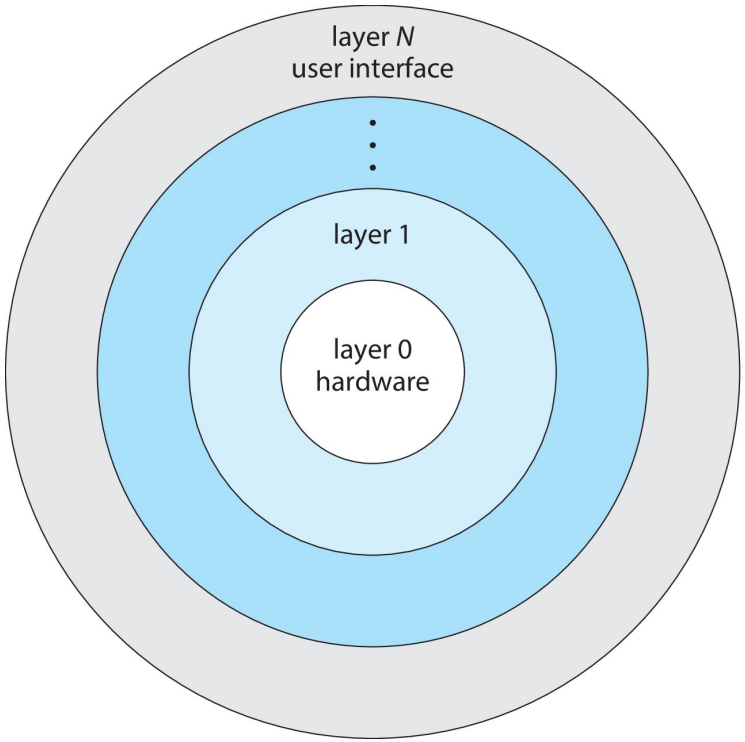
- difficult to implement and extend.
- have very little overhead in the system-call interface
- communication within the kernel is fast.



glibc (GNU C Library) provides an interface between C programs and the underlying Linux kernel

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
- Each layer is an abstract object made up of data and the operations that can manipulate those data.

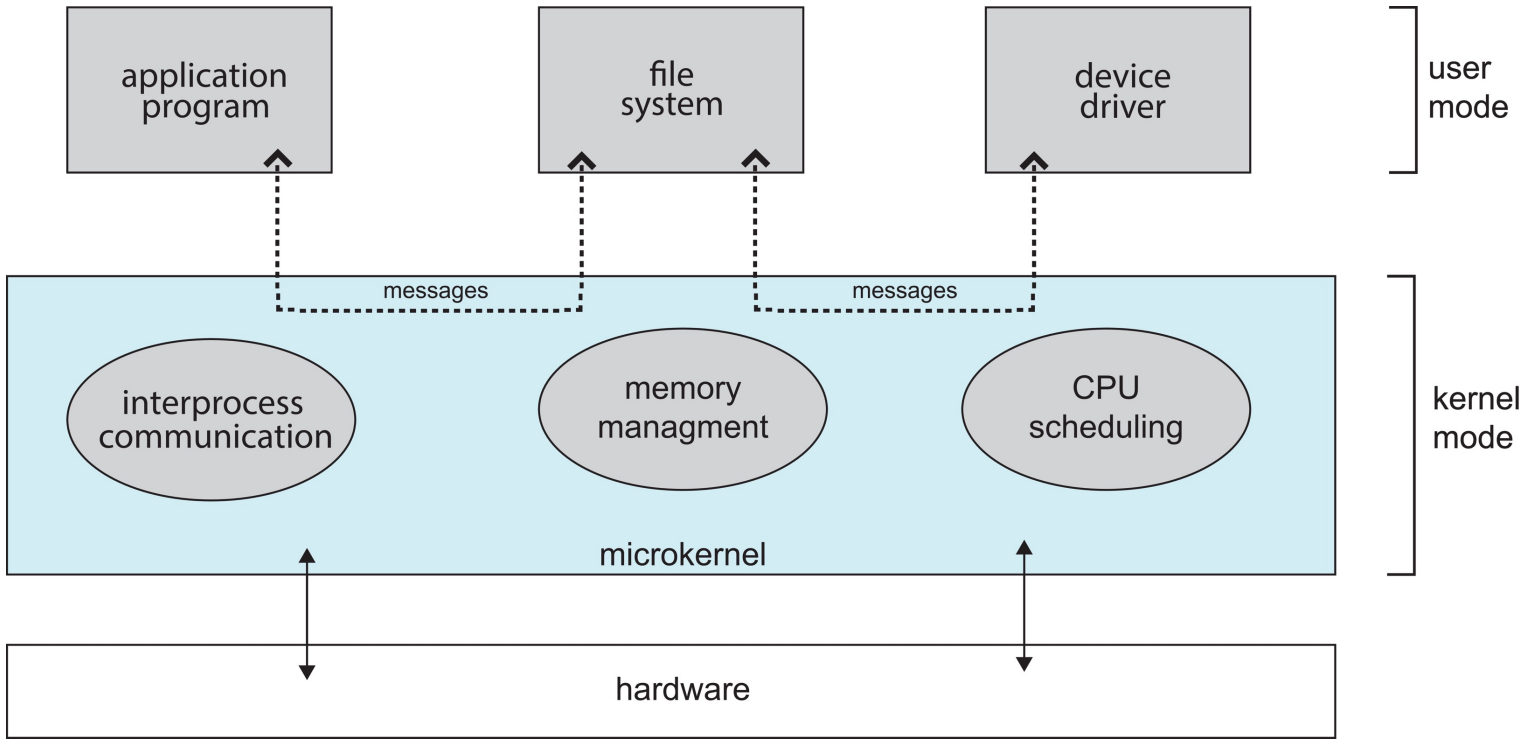


layered approach: simplicity of construction and debugging
challenging defining the functionality of each layer; slow:a user program to traverse through multiple layers to obtain an operating-system service.

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 (OS switches from one process to the next to exchange the messages)

Microkernel System Structure



- microkernel is to provide communication between the client program and the various services that are also running in user space.
- client program and service never interact directly, rather, via the microkernel.

Questions

The major difficulty in designing a layered operating system approach is ____.

A) appropriately defining the various layers ✓

B) making sure that each layer hides certain data structures, hardware, and operations from higher-level layers

C) debugging a particular layer

D) making sure each layer is easily converted to modules

A microkernel is a kernel ____.

A) containing many components that are optimized to reduce resident memory size

B) that is compressed before loading in order to reduce its resident memory size

C) that is compiled to produce the smallest size possible when stored to disk

D) that is stripped of all nonessential components ✓

Modules

- Many modern operating systems implement **loadable kernel modules** (**LKMs**)
  - the kernel has a set of core components
  - can link in additional services via modules
  - either at boot time or during run time.
  - common in modern implementations of UNIX, such as Linux, macOS, and Solaris, as well as Windows.
  - Each talks to the others over known interfaces

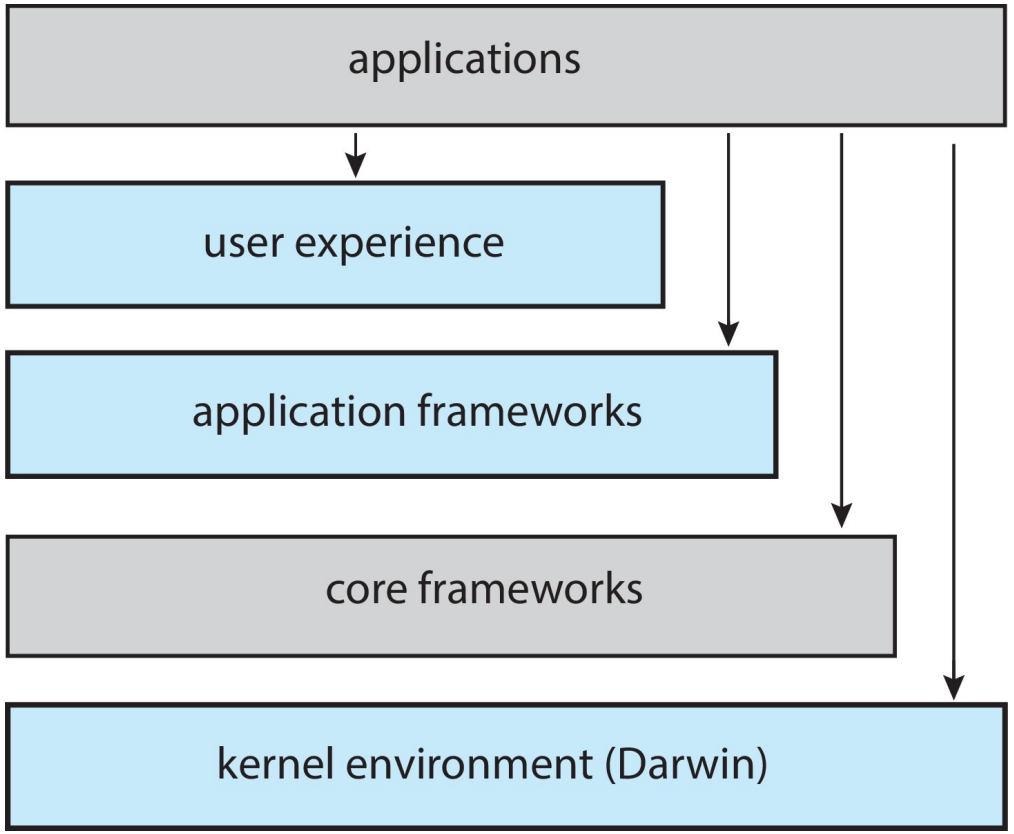
- Overall, similar to layered approach but with more flexibility (each module can call any other)
- No need to invoke message passing in order to communicate.

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 (provides API)
  - Below (the next level) 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

- UE layer defines the software interface that allows users to interact with the computing devices.
- AF layer includes the Cocoa and Cocoa Touch frameworks, which provide an API for the Objective-C and Swift programming languages
- CFs support graphics and media
- KE, also known as Darwin, includes the Mach microkernel and the BSD UNIX kernel.



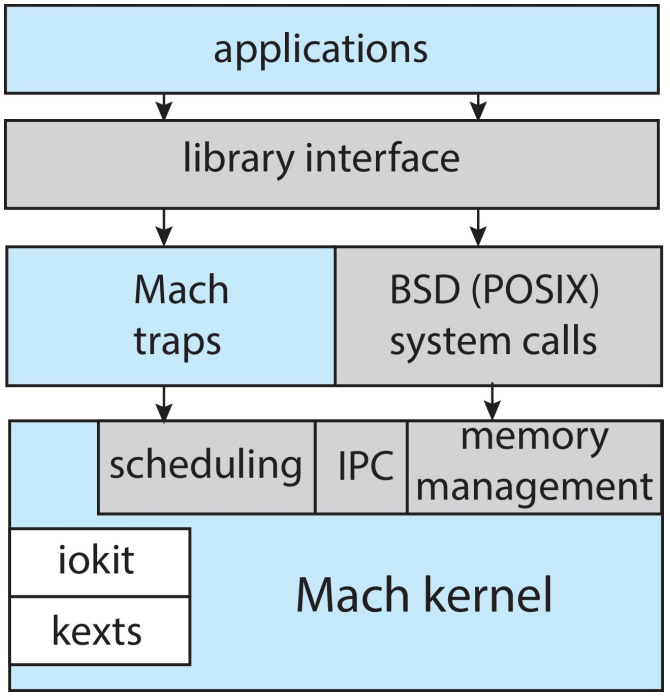
- iOS more restrictions than macOS
- Architectures they run on are different

Darwin

Darwin provides two system-call interfaces: Mach system calls (known as traps) and BSD system calls (which provide POSIX functionality).

Beneath this provides core OS services: memory management, CPU scheduling, and interprocess communication (IPC) facilities such as message passing

iokit/kexts: for development of device drivers and dynamically loadable modules

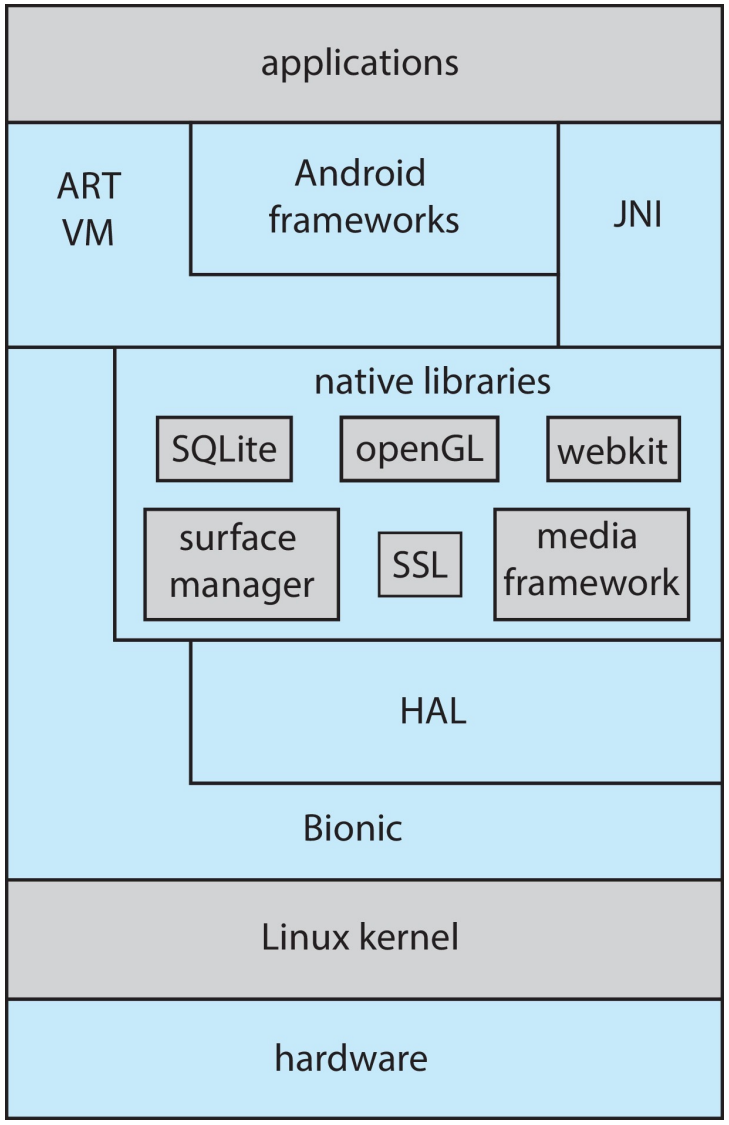


Android

- 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 then runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

Android Architecture

- A layered stack of software that provides a rich set of frameworks supporting graphics, audio, and hardware features.
- Java based development. Google has designed a separate Android API for Java development.
- Java applications are compiled into a form that can execute on the Android RunTime ART, a virtual machine
- Java native interface—or JNI – to bypass virtual machines. Codes are not portable.



- Hardware abstraction layer, or HAL to support variety of hardware/architectures
- Bionic C library (not glibc) (Google)
- Bionic supports slower CPUs
- Modified linux kernel

Question

Q. _____ allows operating system services to be loaded dynamically.

A) Virtual machines
B) Modules ✓
C) File systems
D) Graphical user interfaces

Process creation – fork Example

```c
#include <stdio.h>
#include <unistd.h>
int main() {
    printf("Before fork()\n");

    pid_t pid = fork();
    if (pid == 0) {
    // Code executed by the child process
    printf("Child process: PID = %d\n", getpid());
    }
    else if (pid > 0) {
    // Code executed by the parent process
    printf("Parent PID = %d, Child PID = %d\n", getpid(), pid);
    }
    else {
        // fork() failed
        perror("fork failed");
    }
    printf("This line is executed by both parent and child processes\n");
    return 0;
}
```