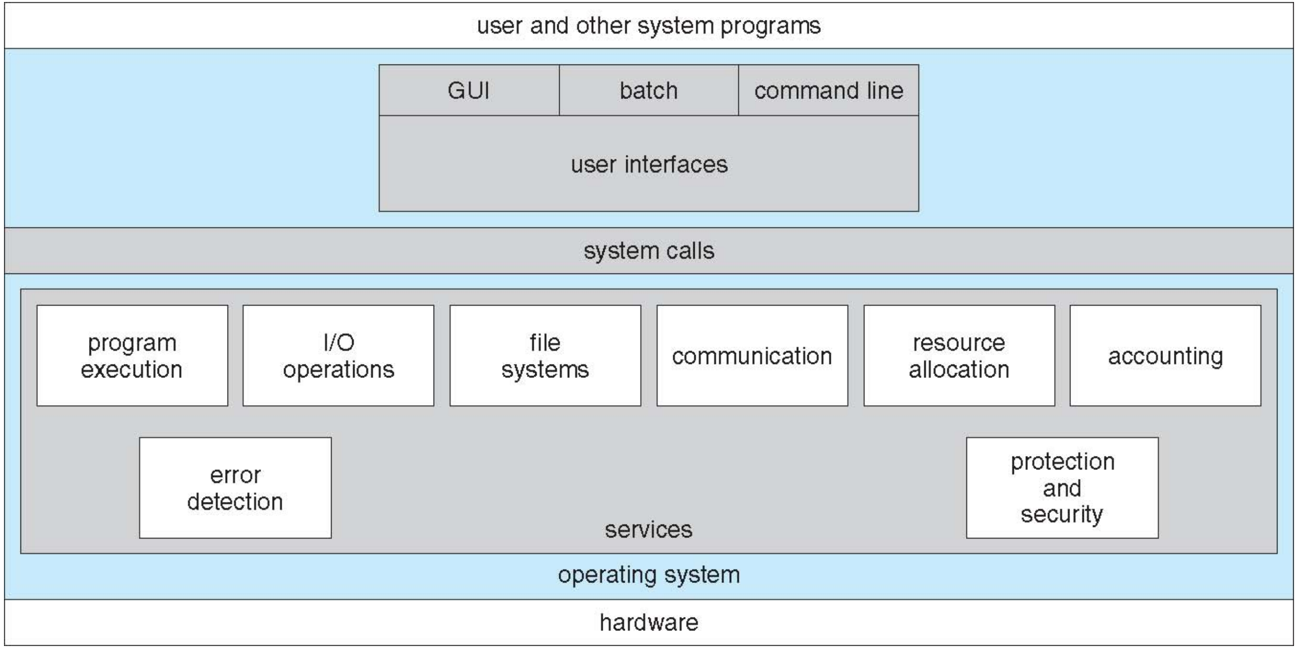
OPERATING SYSTEMS

OS STRUCTURE

Operating System Services

Operating System Services



user and other system programs

GUI

batch

command line

user interfaces

system calls

program execution

I/O operations

file systems

communication

resource allocation

accounting

error detection

protection and security

services

operating system

hardware

The Operating System is

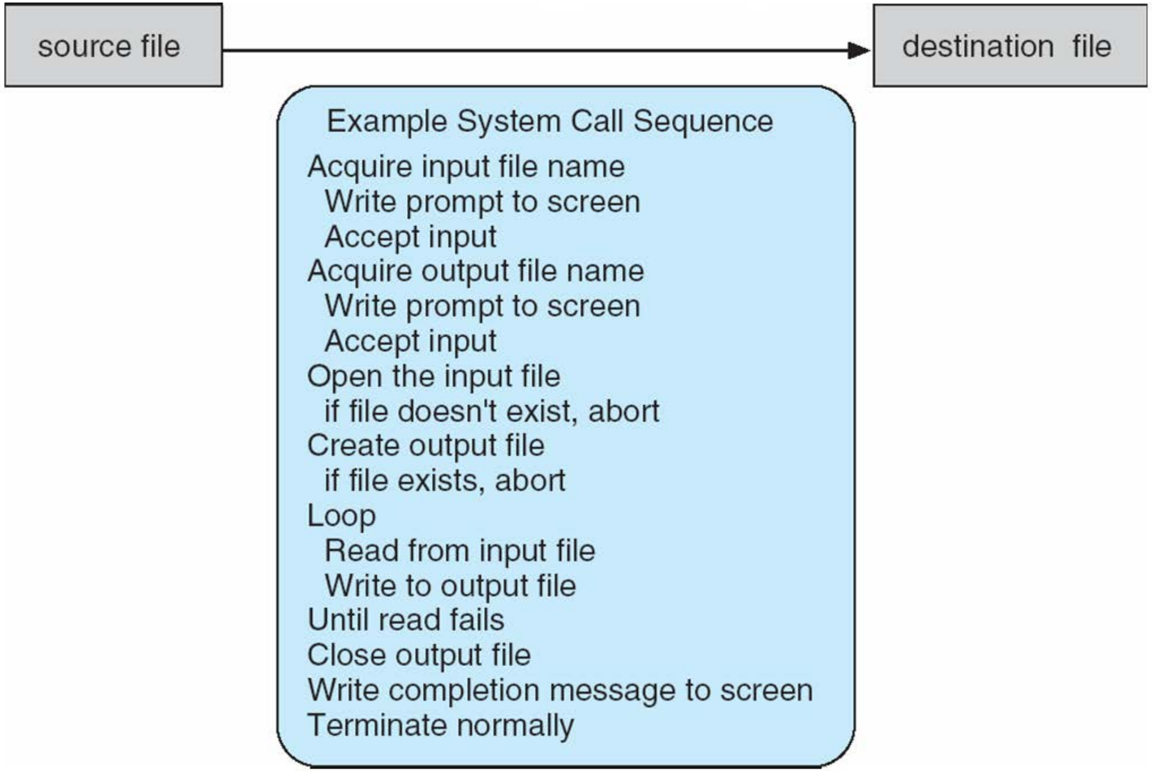
- A Human-Machine Interface
  - allows a simple interaction with the underlying hardware
- A Management Systems
  - resource sharing among users and programs
- A System that evolves with time
  - to keep up with the evolution of hardware platforms
  - to keep up with new users' requests

System Calls

What is a system call?

- A system call is a function of the OS usually part of one the system libraries
  - usually written in a high level language such as C, C++, etc.
  - small portions might be written in the assembly language
- The programmer interacts with the OS through APIs (Application Programming Interfaces)
  - a high level function performing one user task such as opening a file, etc.
  - it may contain more than one system call

Example API and system calls

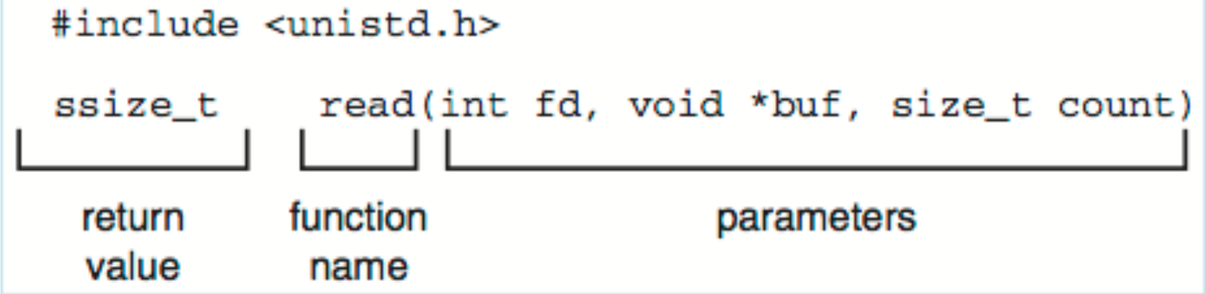


Example System Call Sequence

```
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
```

File copy

Example API



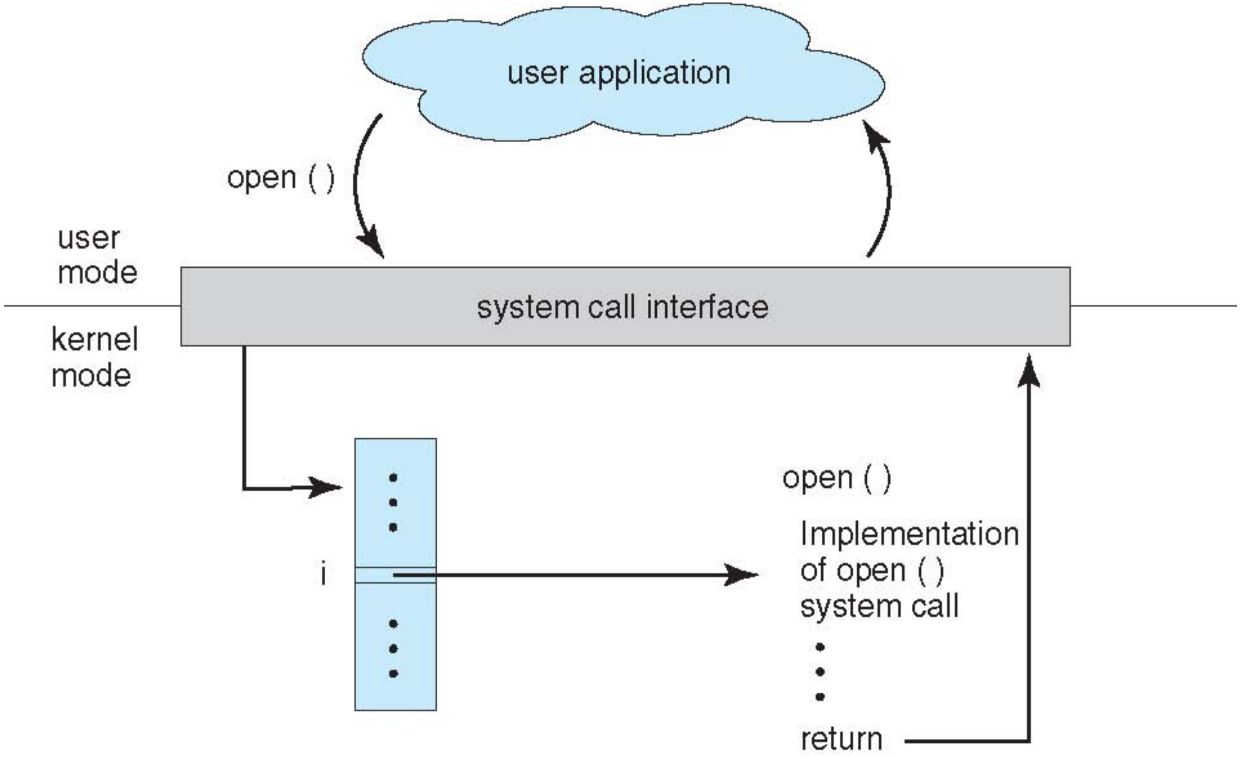
```
#include <unistd.h>

ssize_t     read(int fd, void *buf, size_t count)
```

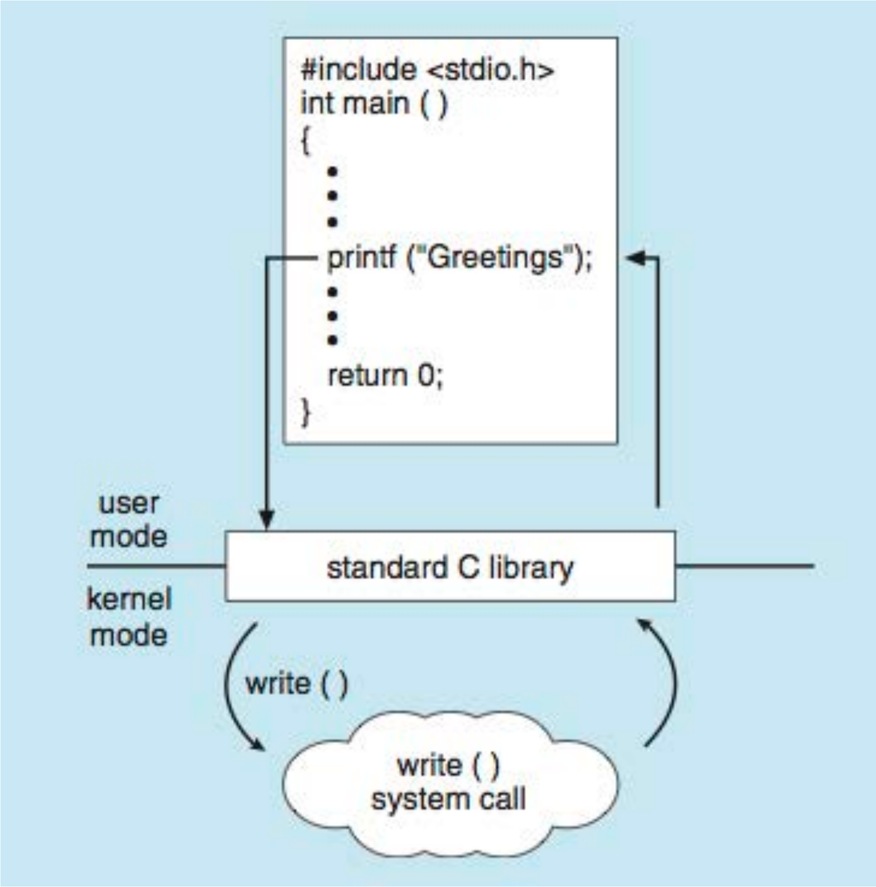
- `read()` - API UNIX - Linux
  - `int fd` the file descriptor to be read
  - `void *buf` a buffer where the data will be read into
  - `size_t count` the max number of bytes to be read into the buffer

This function returns the number of bytes that are read

The open ( ) system call



The `write()` system call



```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

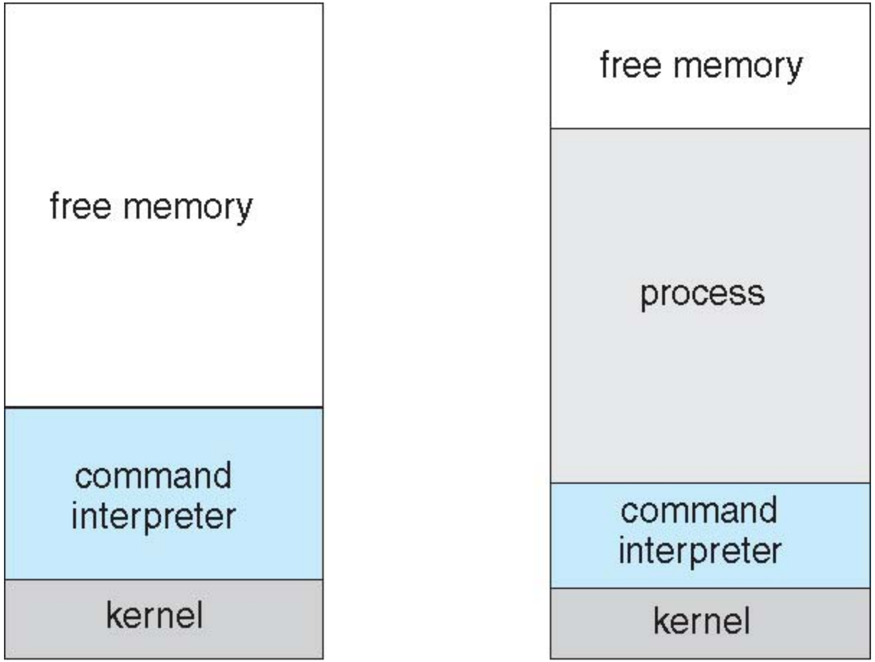
System call categories

- Process Control
- File Management
- Device Management
- OS Configuration and Settings
- Communication

UNIX Vs. Win32 System Calls

	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()`
System Info	`GetCurrentProcessID()`	`getpid()`
	`SetTimer()`	`alarm()`
	`Sleep()`	`sleep()`
Communication	`CreatePipe()`	`pipe()`
	`CreateFileMapping()`	`shmget()`
	`MapViewOfFIle()`	`nmap()`
Protection and Security	`SetFileSecurity()`	`chmod()`
	`InitializeSecurityDescriptor()`	`umask()`
	`SetSecurityDescriptorGroup()`	`chown()`

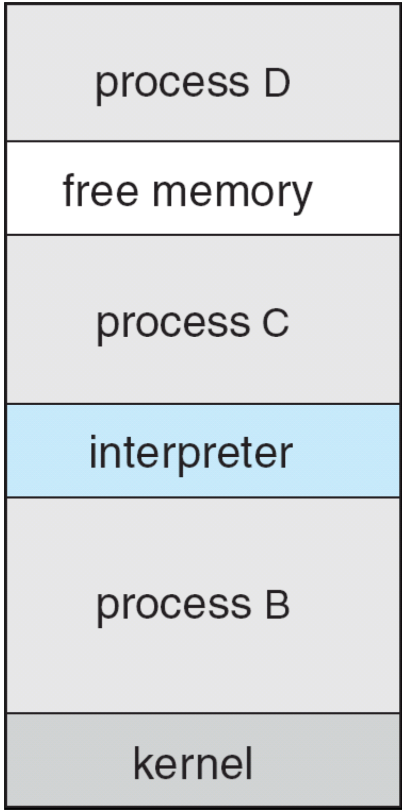
MS-DOS



(a)
at system startup

(b)
running a program

FreeBSD


process D
free memory
process C
interpreter
process B
kernel

System Programs

System Programs

- Utilities
  - File management, modification, and backup
  - Status information
  - Programming environments (text editors, compilers, debugger, etc.)
- Application programs usually distributed with the OS but nor part of the OS
  - Web browser
  - Office automation
  - Music and Video Players

Operating System Design and Implementation

Three Phases

- OS Scope
  - Users and system goals
- Policies and mechanisms
  - What the OS will have to do and how it will do it
- Implementation
  - High-level language and assembly language

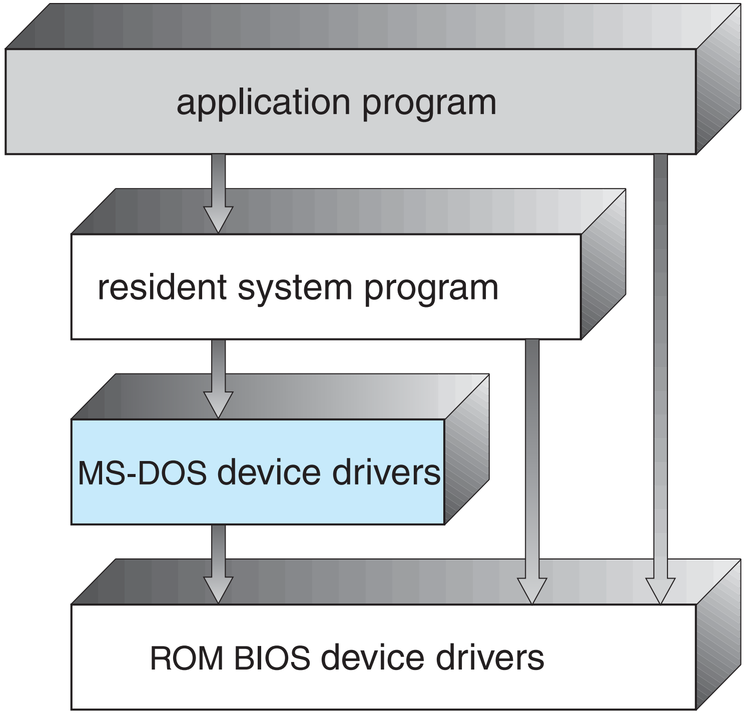
Operating System Structures

Simple Structure

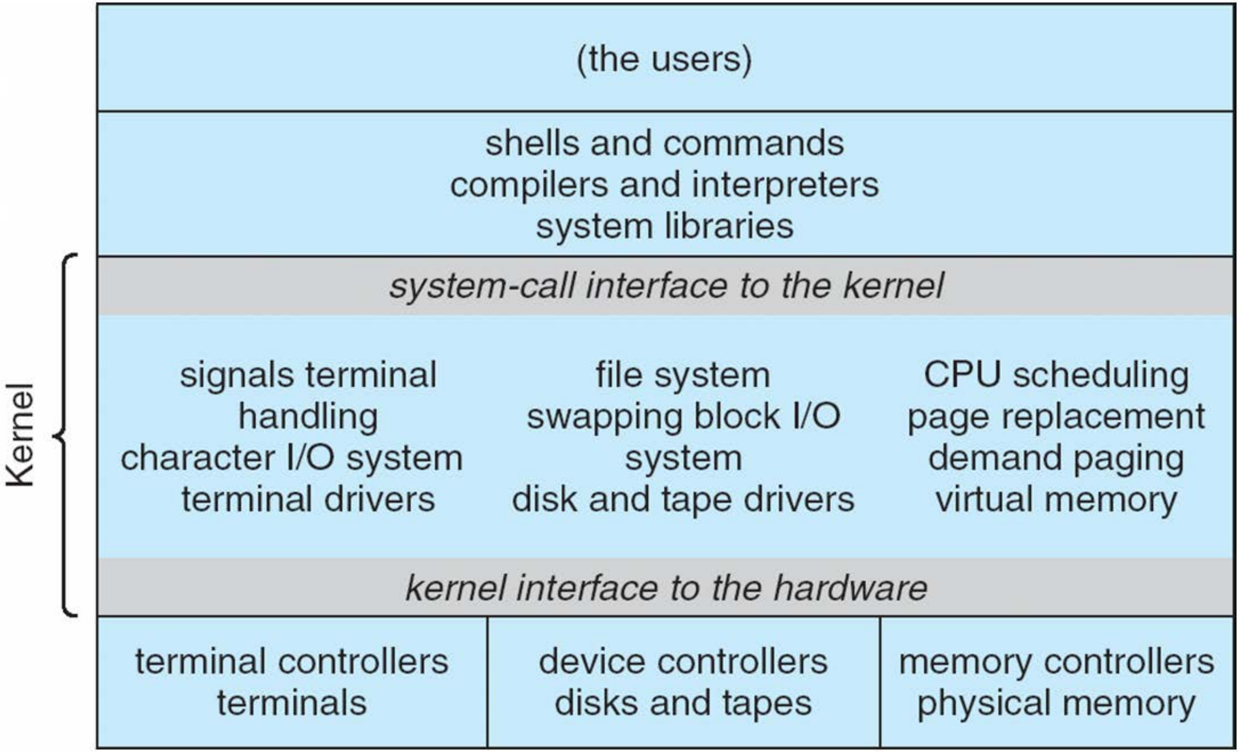
- Typical of old operating systems
  - Tightly coupled to an individual hardware architecture
  - Limited by hardware functionalities
  - MS-DOS and initial UNIX versions

- Main characteristics
  - Not divided into modules
    - Monolithic kernel
  - User programs have direct access to the I/O

Simple structure
MS-DOS



Traditional UNIX System Structure



(the users)

shells and commands
compilers and interpreters
system libraries

system-call interface to the kernel

signals terminal handling
character I/O system
terminal drivers

file system
swapping block I/O system
disk and tape drivers

CPU scheduling
page replacement
demand paging
virtual memory

kernel interface to the hardware

terminal controllers
terminals

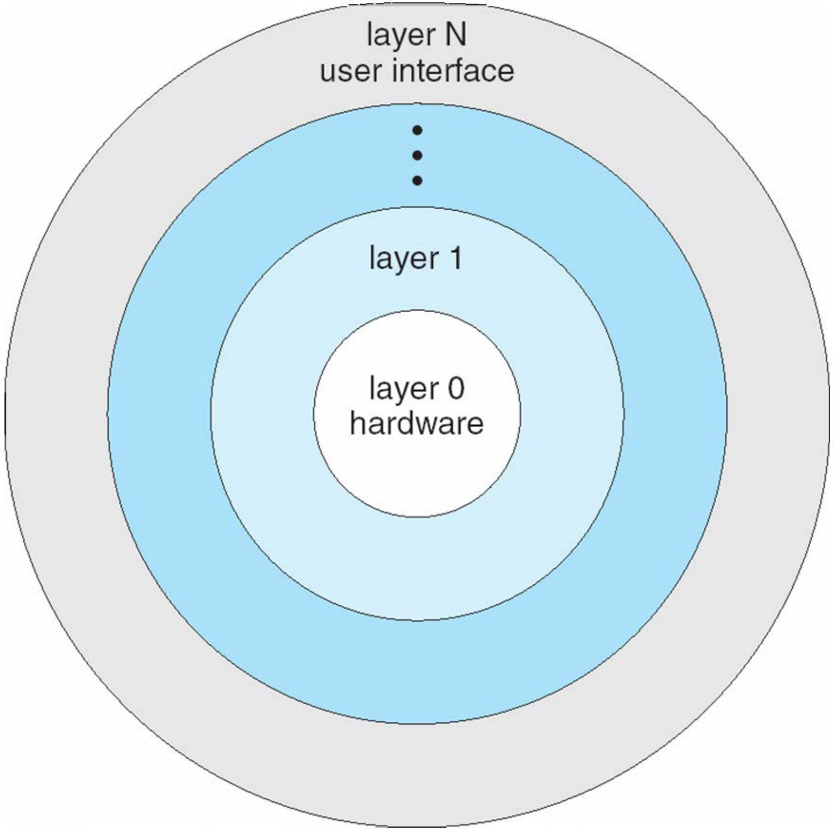
device controllers
disks and tapes

memory controllers
physical memory

Layered Approach

- Need specific hardware support
- Modularity
- The OS *hides* most of the low level functionalities
  - Protection from improper use
  - System update and upgrade is easier
  - Programmers interact with APIs

Layered Approach



How many layers?

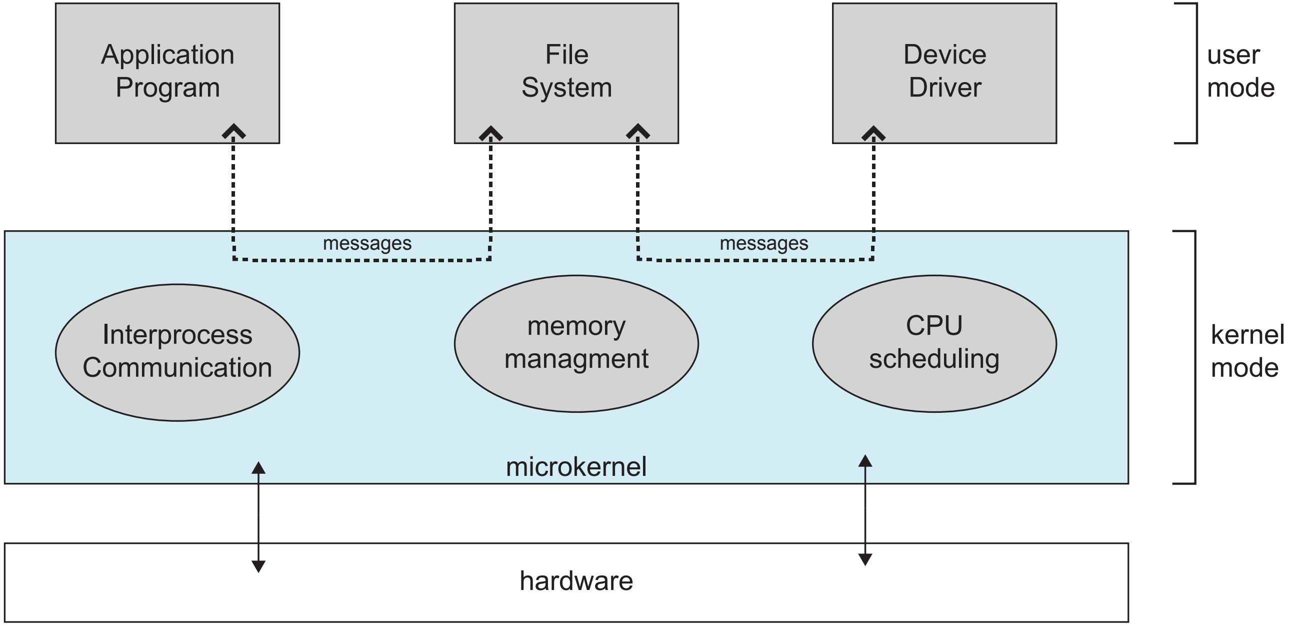
Which functions in each layer?

System effectiveness

Microkernel

- ’80s: The size of OS *kernels* was too large
- Microkernel approach
  identification of the core functions, all other functions implemented as user processes
  - e.g., the Mach OS (Carnegie Mellon)
- External components are implemented as server processes
  - they interact with each other through message passing via the kernel
    - it can slow down the system

Microkernel



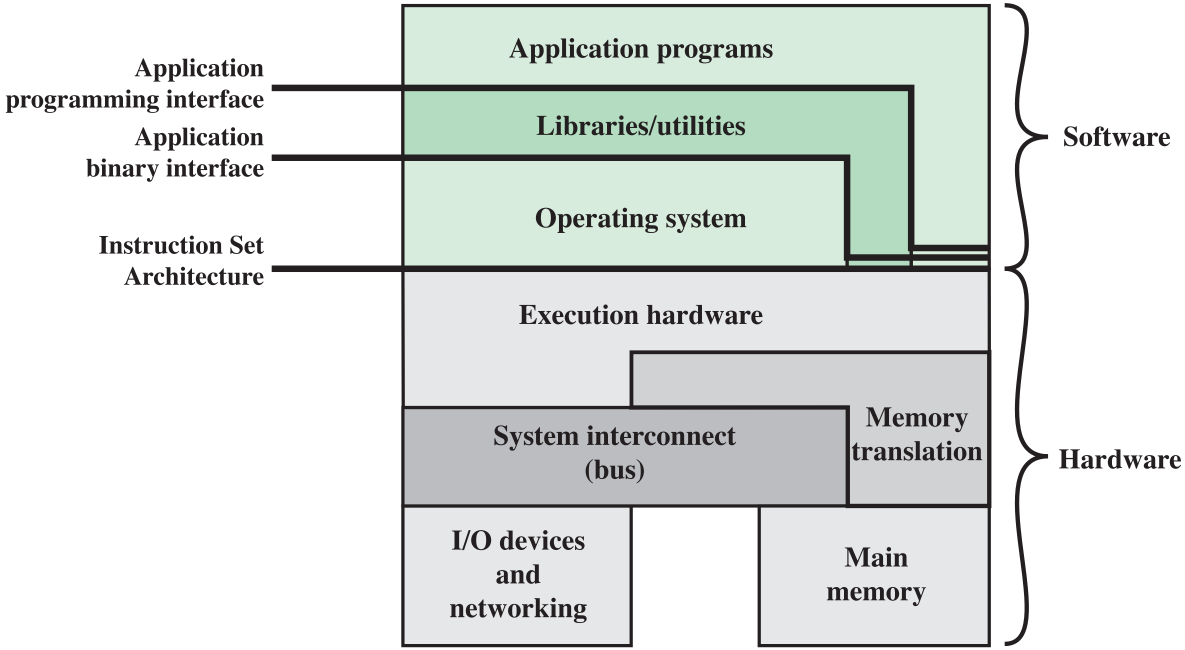
Modular structure

- Designed according to the Object Oriented Programming paradigm
  - the kernel contains only core components
  - other functions are implemented as modules that can be dynamically loaded
    - e.g., support for different file systems
  - modules can communicate to each other without calling the kernel
- This structures combines the benefits of microkernel with the layered structure

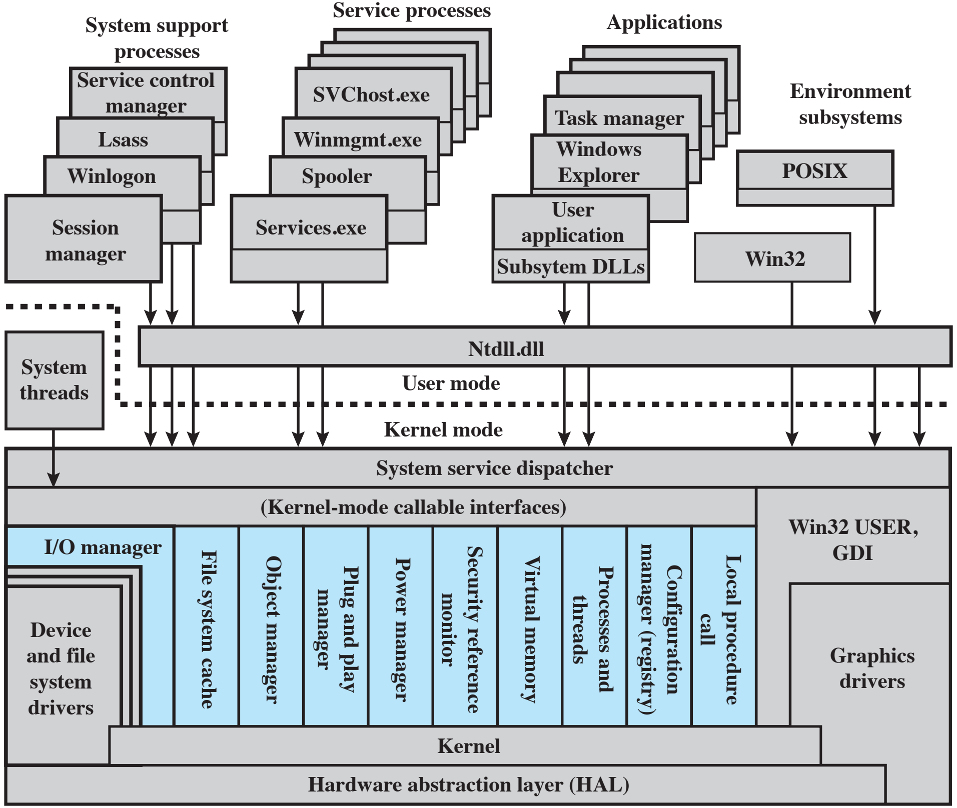
Ibrid Structures

- Modern OS does not strictly follow a particular structures
- Each function can be implemented according to one of the available structures according to the
  - goals
  - expected performances
  - user experience

Modern OS structures


Application programming interface
Application binary interface
Instruction Set Architecture
Application programs
Libraries/utilities
Operating system
Execution hardware
System interconnect (bus)
Memory translation
I/O devices and networking
Main memory
Software
Hardware

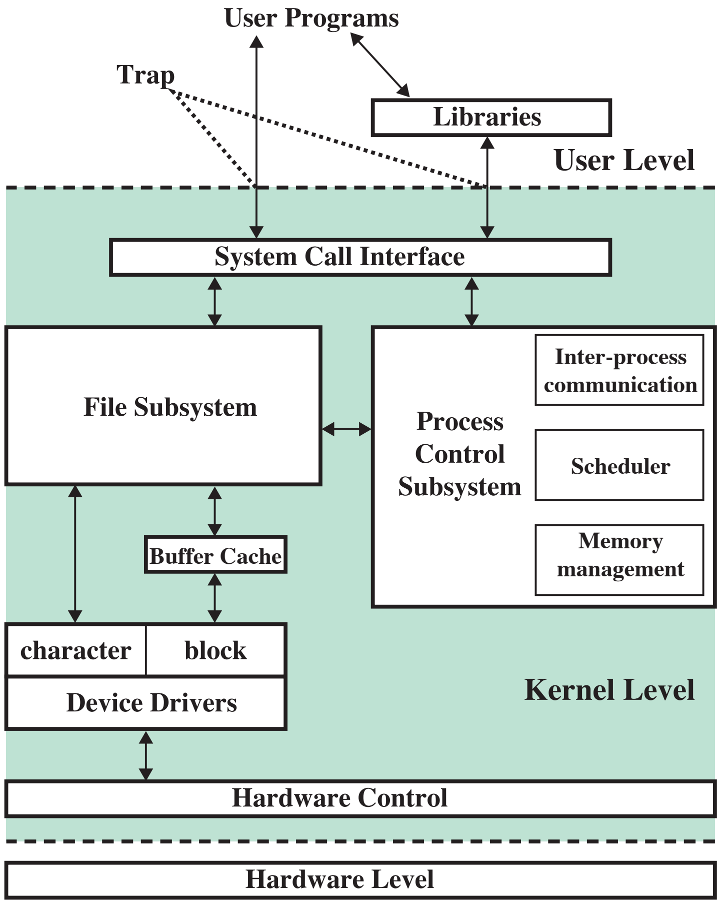
MS Windows Structure



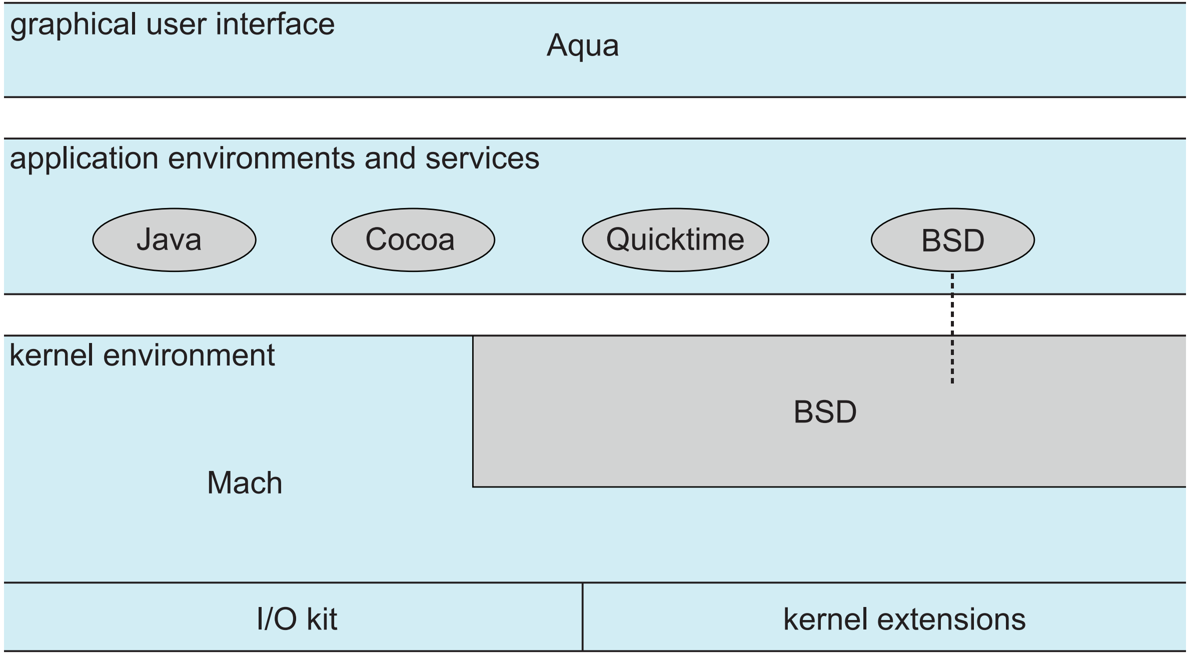
Lsass = local security authentication server
POSIX = portable operating system interface
GDI = graphics device interface
DLL = dynamic link libraries

Colored area indicates Executive

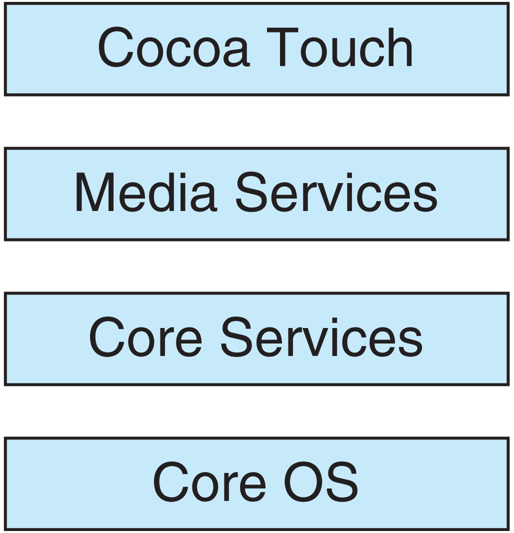
UNIX Structure



macOS Structure



iOS Structure



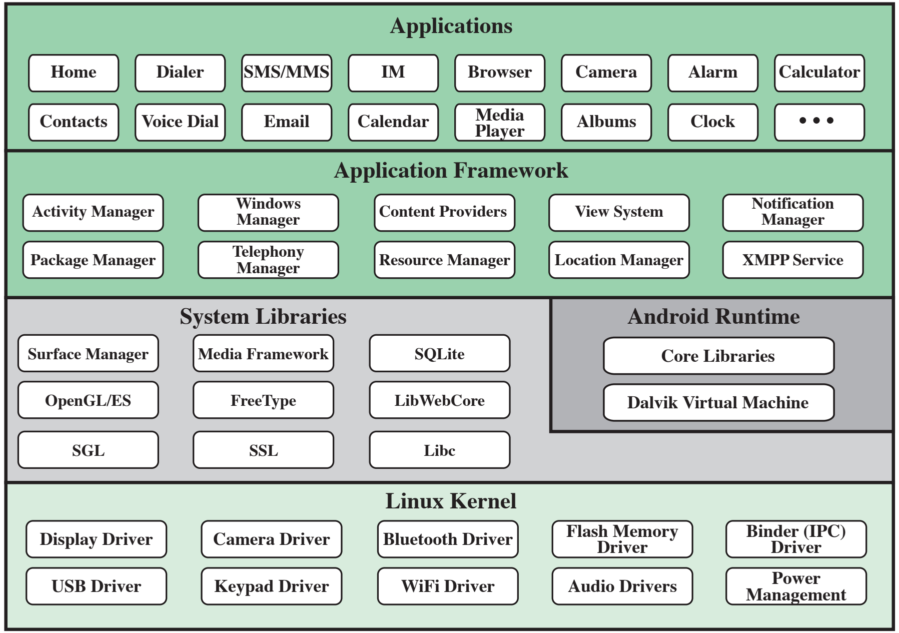
Cocoa Touch

Media Services

Core Services

Core OS

Android Structure



Applications

Home, Dialer, SMS/MMS, IM, Browser, Camera, Alarm, Calculator

Contacts, Voice Dial, Email, Calendar, Media Player, Albums, Clock, • • •

Application Framework

Activity Manager, Windows Manager, Content Providers, View System, Notification Manager

Package Manager, Telephony Manager, Resource Manager, Location Manager, XMPP Service

System Libraries

Surface Manager, Media Framework, SQLite

OpenGL/ES, FreeType, LibWebCore

SGL, SSL, Libc

Android Runtime

Core Libraries

Dalvik Virtual Machine

Linux Kernel

Display Driver, Camera Driver, Bluetooth Driver, Flash Memory Driver, Binder (IPC) Driver

USB Driver, Keypad Driver, WiFi Driver, Audio Drivers, Power Management

Implementation:

🟩 **Applications, Application Framework: Java**

⬜ ▨ **System Libraries, Android Runtime: C and C++**

⬜ **Linux Kernel: C**