



Amirkabir University of Technology  
(Tehran Polytechnic)  
Department of Computer Engineering

---

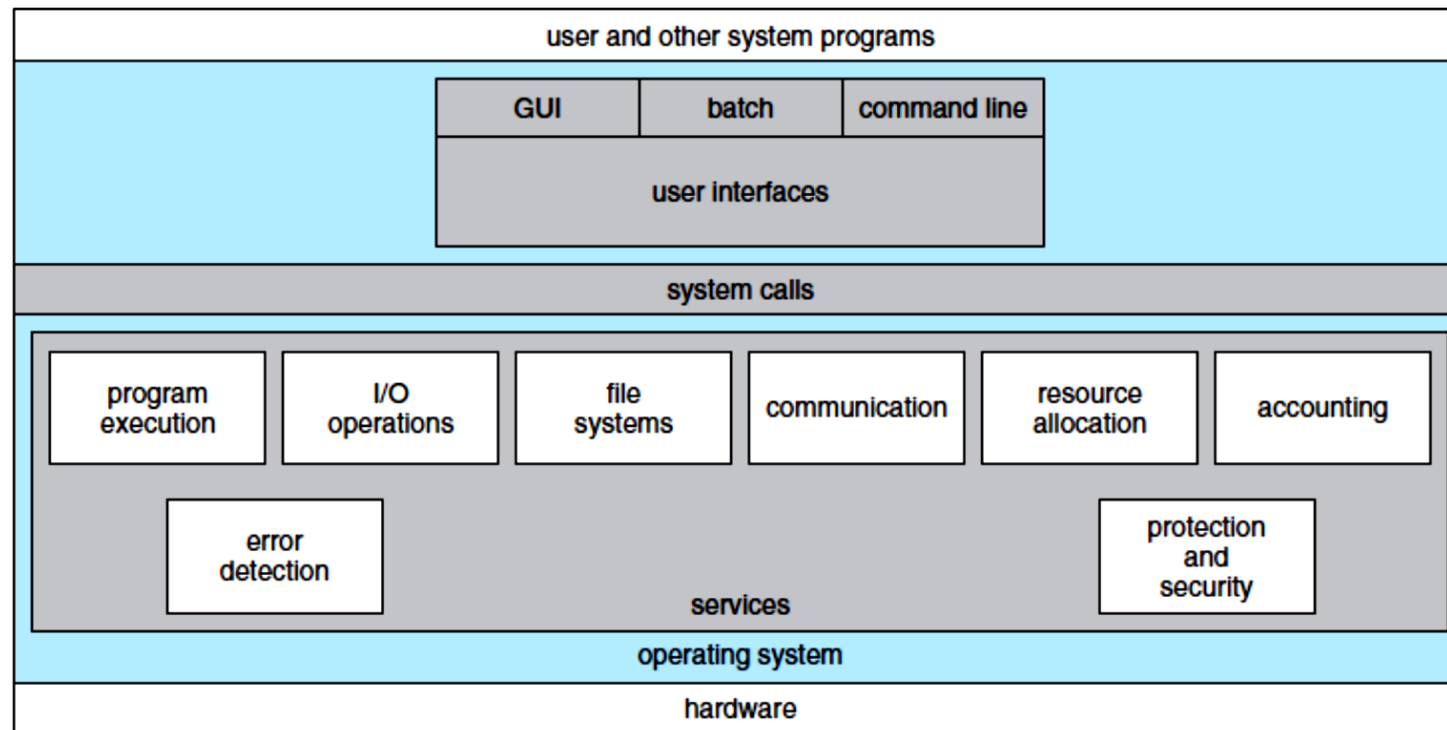
# Operating System Structure

Hamid R. Zarandi  
[h\\_zarandi@aut.ac.ir](mailto:h_zarandi@aut.ac.ir)

# Advantages of OS

## ➤ Different views from OS

- Services (**predefined comfort routines**)
- Interface to users and programs (**interactions**)
- Components & their interconnects (**SW architecture**)



# OS **services** (helpful to users)

- User interface
  - CLI (**Command-line** interface)
  - GUI (**Graphical** user interface)
  - Batch (**shell script**)
- Program execution
- I/O operations
- File-system manipulation
- Communications
  - **Shared memory**
  - **Message passing**
- Error detection

# OS services (for efficient operation)

- **Resource allocation**
- **Accounting**
- **Protection & security**

# OS interface (command line)

## ➤ Command interpreters

- Kernel-based vs. system programs

## ➤ Multiple interpreters, *shell*

- UNIX, Linux shells: *Bourne shell*
- Third party shell!
- Similar **functionality**, user **preference**

## ➤ Shell implementations

- Internal codes (make jump)
- System programs (UNIX)
  - **rm** file.txt

```

File Edit View Terminal Tabs Help
fd0      0.0    0.0    0.0    0.0  0.0  0.0    0.0  0  0
sd0      0.0    0.2    0.0    0.2  0.0  0.0    0.4  0  0
sd1      0.0    0.0    0.0    0.0  0.0  0.0    0.0  0  0
          extended device statistics
device   r/s    w/s    kr/s   kw/s wait actv  svc_t  %w  %b
fd0      0.0    0.0    0.0    0.0  0.0  0.0    0.0  0  0
sd0      0.6    0.0   38.4    0.0  0.0  0.0    8.2  0  0
sd1      0.0    0.0    0.0    0.0  0.0  0.0    0.0  0  0
(root@pbg-nv64-vn)-(11/pts)-(00:53 15-Jun-2007)-(global)
- (/var/tmp/system-contents/scripts)# swap -sh
total: 1.1G allocated + 190M reserved = 1.3G used, 1.6G available
(root@pbg-nv64-vn)-(12/pts)-(00:53 15-Jun-2007)-(global)
- (/var/tmp/system-contents/scripts)# uptime
12:53am up 9 min(s), 3 users, load average: 33.29, 67.68, 36.81
(root@pbg-nv64-vn)-(13/pts)-(00:53 15-Jun-2007)-(global)
- (/var/tmp/system-contents/scripts)# w
 4:07pm up 17 day(s), 15:24, 3 users, load average: 0.09, 0.11, 8.66
User      tty          login@ idle   JCPU   PCPU   what
root      console      15Jun07 18days    1      /usr/bin/ssh-agent -- /usr/bi
n/d
root      pts/3        15Jun07    18      4    w
root      pts/4        15Jun07 18days    w
(root@pbg-nv64-vn)-(14/pts)-(16:07 02-Jul-2007)-(global)
- (/var/tmp/system-contents/scripts)#

```

Bourne shell in Solaris 10

# Bourne Shell command interpreter

```

PBG-Mac-Pro:~ pbg$ w
15:24 up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER  TTY  FROM          LOGIN@  IDLE WHAT
pbg   console -            14:34   50 -
pbg   s000 -            15:05   - w
PBG-Mac-Pro:~ pbg$ iostat 5
            disk0      disk1      disk10      cpu      load average
      KB/t tps MB/s    KB/t tps MB/s    KB/t tps MB/s  us sy id  1m  5m  15m
      33.75 343 11.30    64.31 14  0.88    39.67  0  0.02 11  5 84  1.51 1.53 1.65
       5.27 320  1.65      0.00  0  0.00      0.00  0  0.00  4  2 94  1.39 1.51 1.65
       4.28 329  1.37      0.00  0  0.00      0.00  0  0.00  5  3 92  1.44 1.51 1.65
^C
PBG-Mac-Pro:~ pbg$ ls
Applications          Music                  WebEx
Applications (Parallels)  Pando Packages      config.log
Desktop               Pictures              getsmartdata.txt
Documents             Public                imp
Downloads             Sites                 log
Dropbox               Thumbs.db             panda-dist
Library               Virtual Machines      prob.txt
Movies                Volumes               scripts
PBG-Mac-Pro:~ pbg$ pwd
/Users/pbg
PBG-Mac-Pro:~ pbg$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.257 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.262 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.262/1.760/2.257/0.498 ms
PBG-Mac-Pro:~ pbg$

```

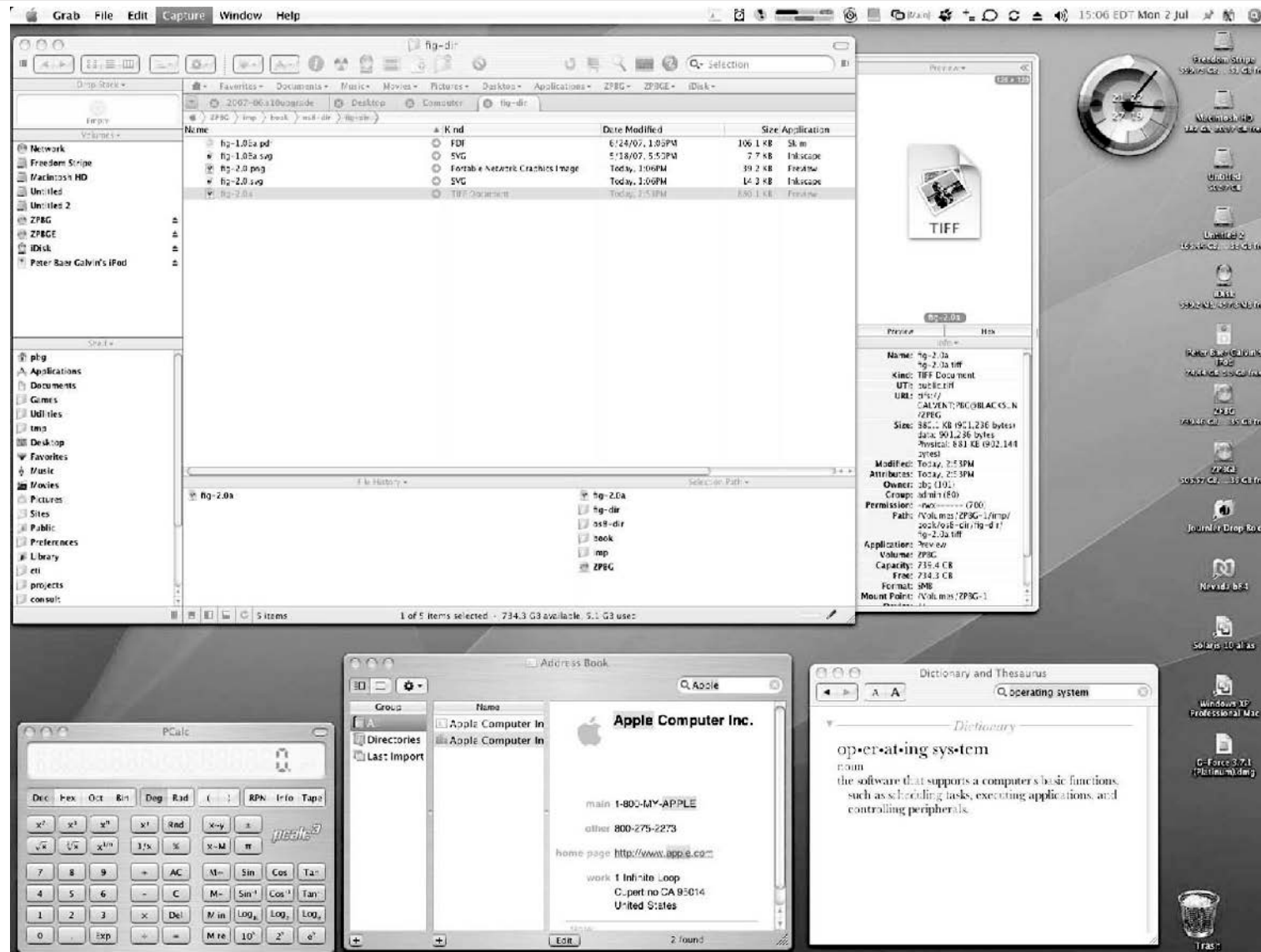
# OS interface (GUI)

- Xerox PARC (1970)
- Xerox Alto (1973)
- Apple Macintosh (1980)
- Aqua in Mac OS X
- Microsoft Windows
- UNIX
  - CDE (Common desktop environment)
  - X-Windows
  - KDE (K Desktop environment)
  - GNOME (GNU project)





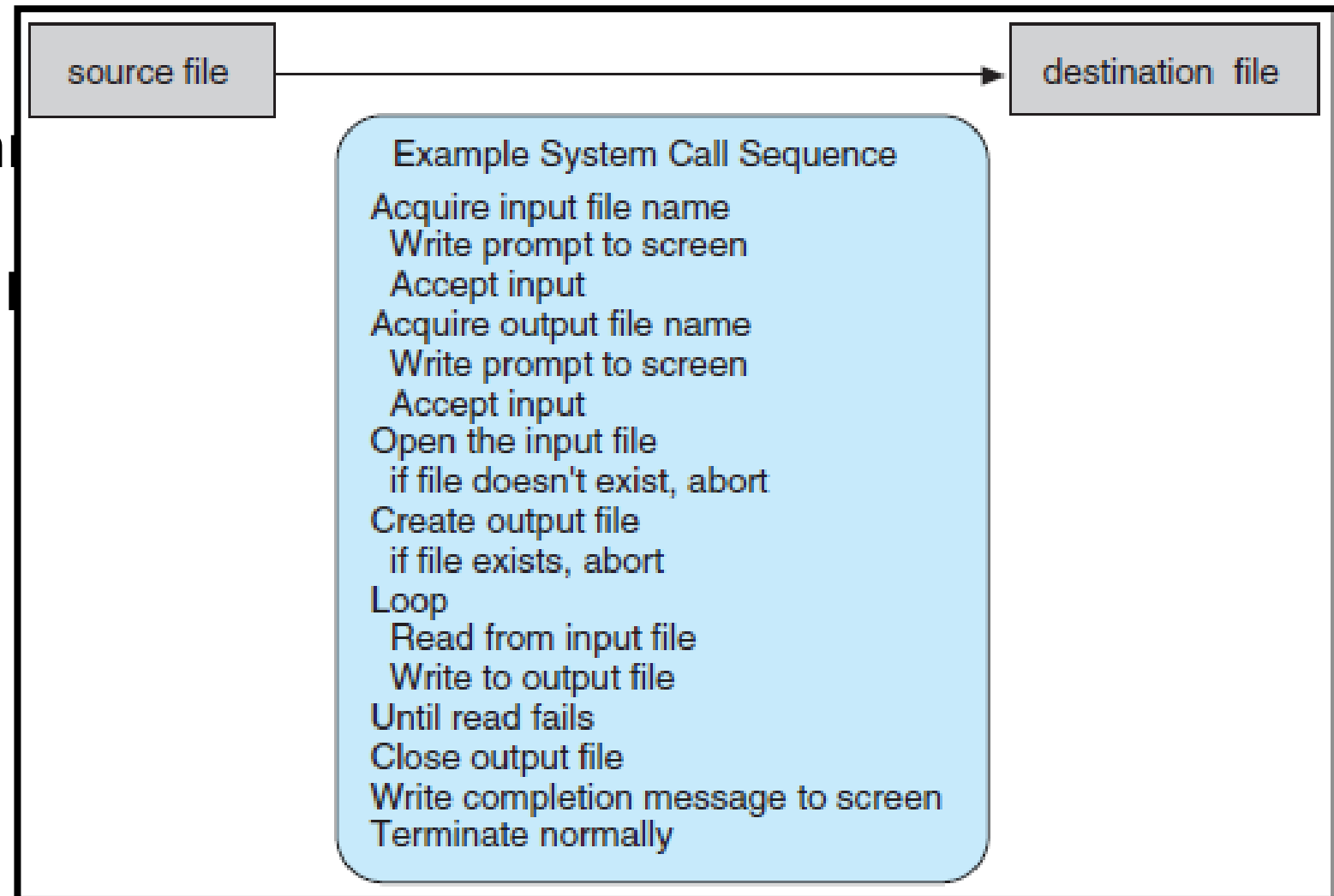
# Choice of interface (The Mac OS X GUI)





# System calls

- C, C++, Assembly
- API (Application program interface)
  - Win API
  - POSIX API (UNIX, Linux, ...)
  - JAVA API
- Example: Copy file



# Example of standard API

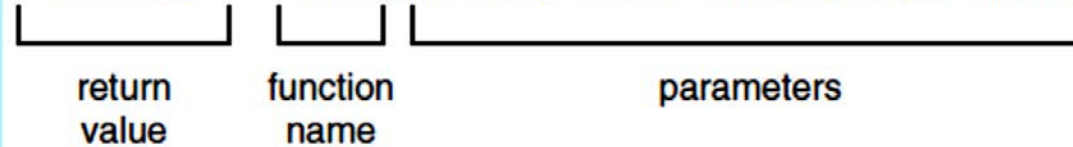
As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>
```

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



A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

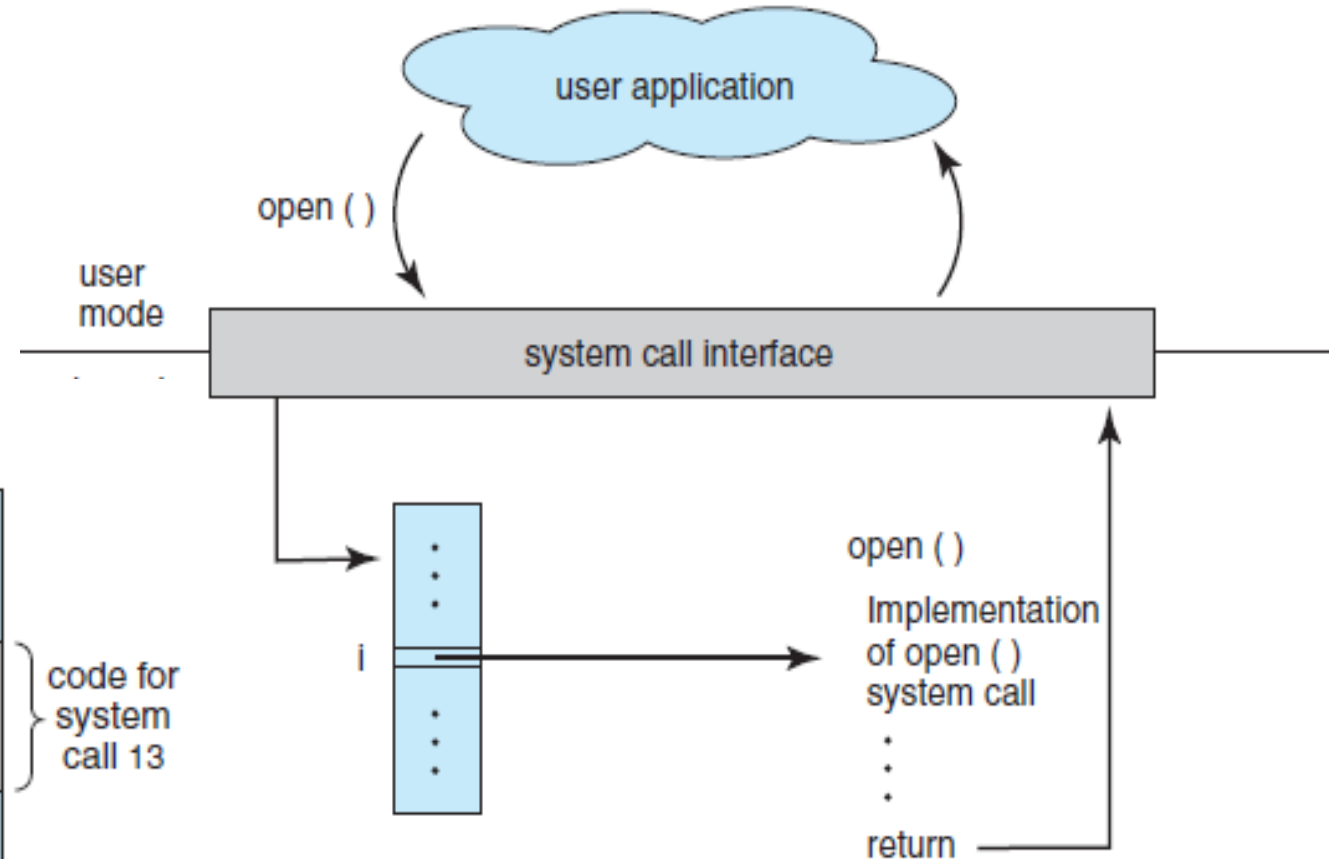
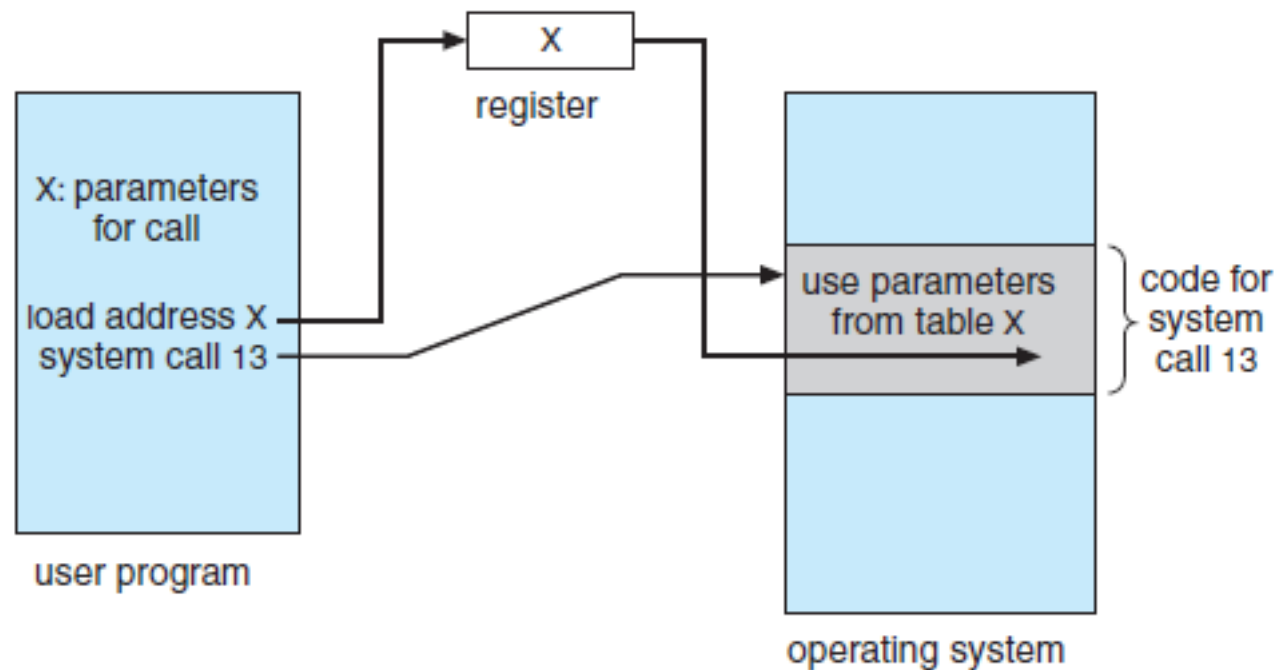
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns `-1`.

# System call interface

## ➤ Parameter passing

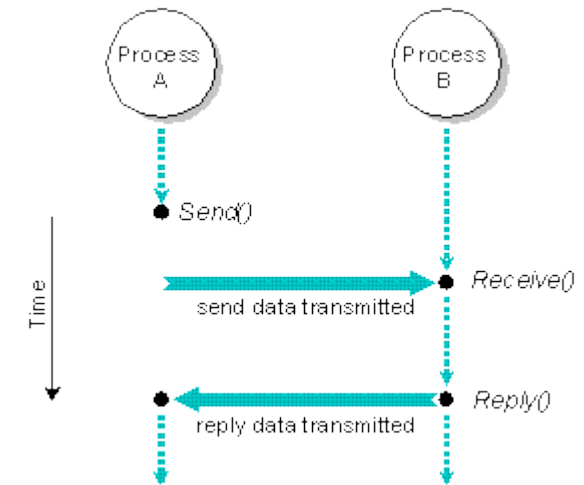
- Register
- Register pointer to mem. block
- Stack (PUSH, POP)



# Communications mechanisms

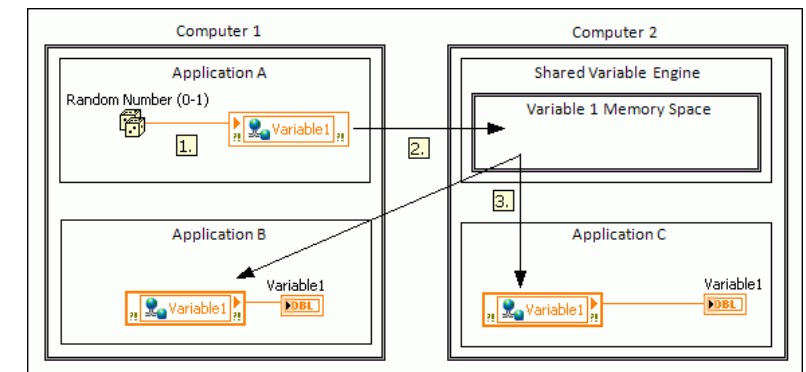
## ➤ Message passing

- Easy
- Suitable for low communication rate
- No interference



## ➤ Shared memory

- Easy
- Fast
- Points of: protection, race condition



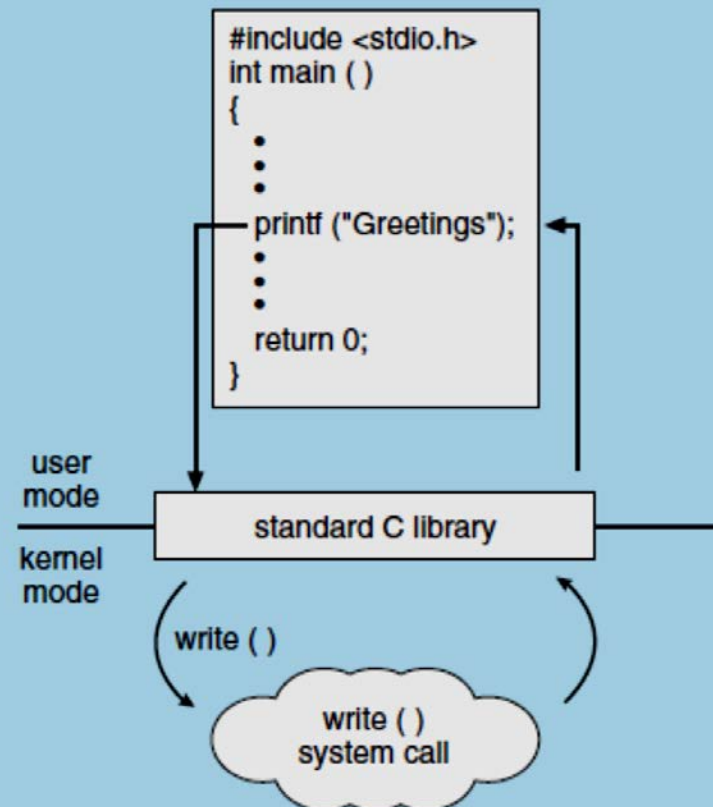
# Types of system calls

- Process control
- File manipulation
- Device manipulation
- Information maintenance
- Communications
- Protections

	Windows	Unix
<b>Process Control</b>	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
<b>File Manipulation</b>	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
<b>Device Manipulation</b>	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
<b>Information Maintenance</b>	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
<b>Communication</b>	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shm_open() mmap()
<b>Protection</b>	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

# Example of standard C library

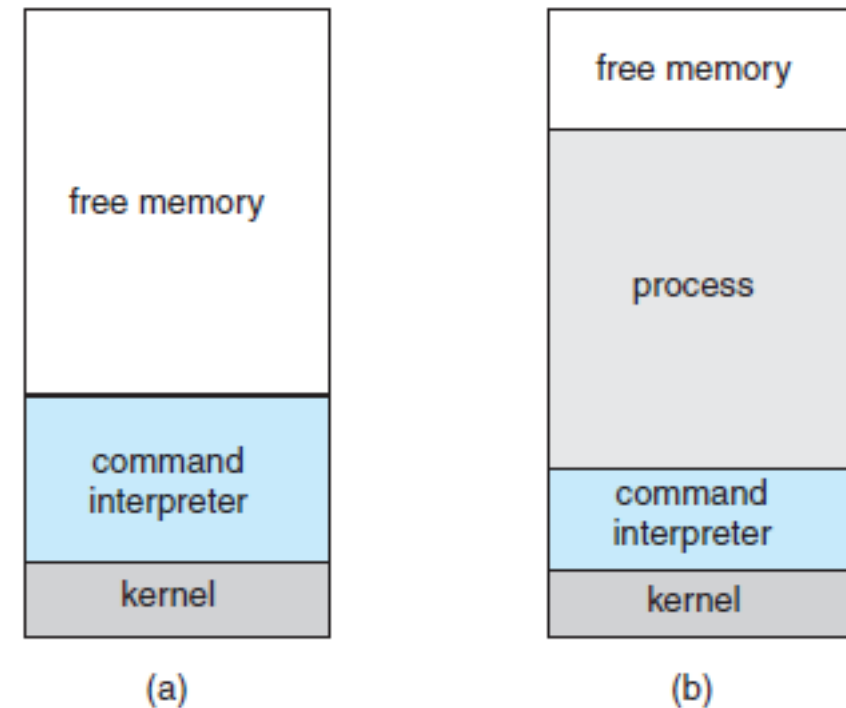
The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the `printf()` statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the `write()` system call. The C library takes the value returned by `write()` and passes it back to the user program. This is shown below:





# Example: MS-DOS

- Single-tasking
- Shell invoked when system booted
- Simple method to run program
  - No process created
- Single memory space
- Loads program into memory, overwriting all but the kernel
- Program exit -> shell reloaded

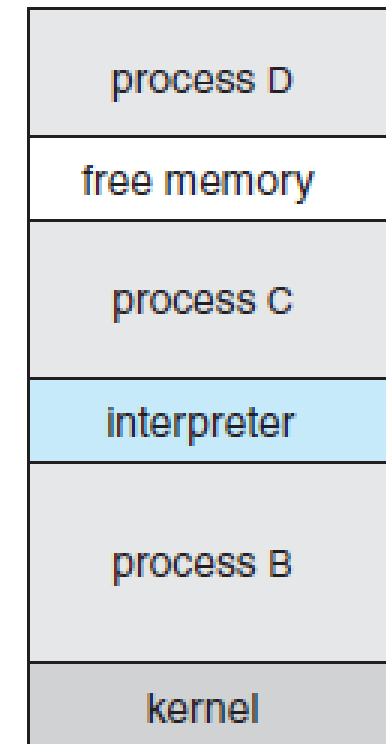


MS-DOS execution. (a) At system startup. (b) Running a program



# Example: FreeBSD

- Unix variant
- Multitasking
- User login -> invoke user's choice of shell
- Shell executes **fork()** system call to create process
  - Executes **exec()** to load program into process
  - Shell **waits** for process to terminate or continues with user commands
- Process exits with code of 0 – no error or > 0 – error code



FreeBSD running multiple programs

# OS design & implementation

## ➤ Goals

- User vs. System

## ➤ Mechanisms & policy

- how vs. what

## ➤ Implementation

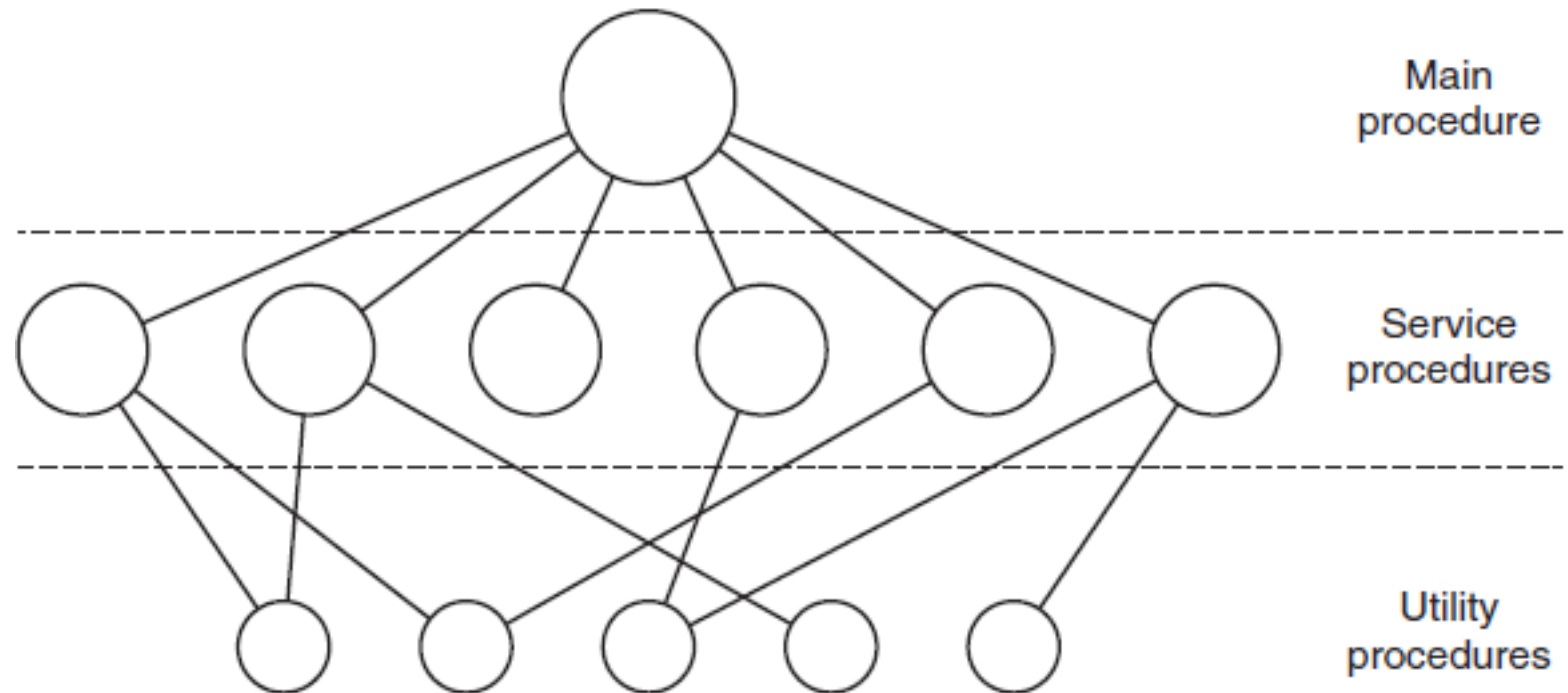
- Assembly, C or C++

- MCP: **ALGOL**
- MULTICS: **PL/1**
- Linux, Win: **C, Assembly**

- **C** is supported on diff. ISAs, CPUs

# OS structure: 1. Simple structure (Monolithic)

- The **most common** organization
- OS is a **single large program** in **kernel mode**
- **Problems**
  - **Crash** in called procedures?
  - **Unwieldy** & **difficult** to understand

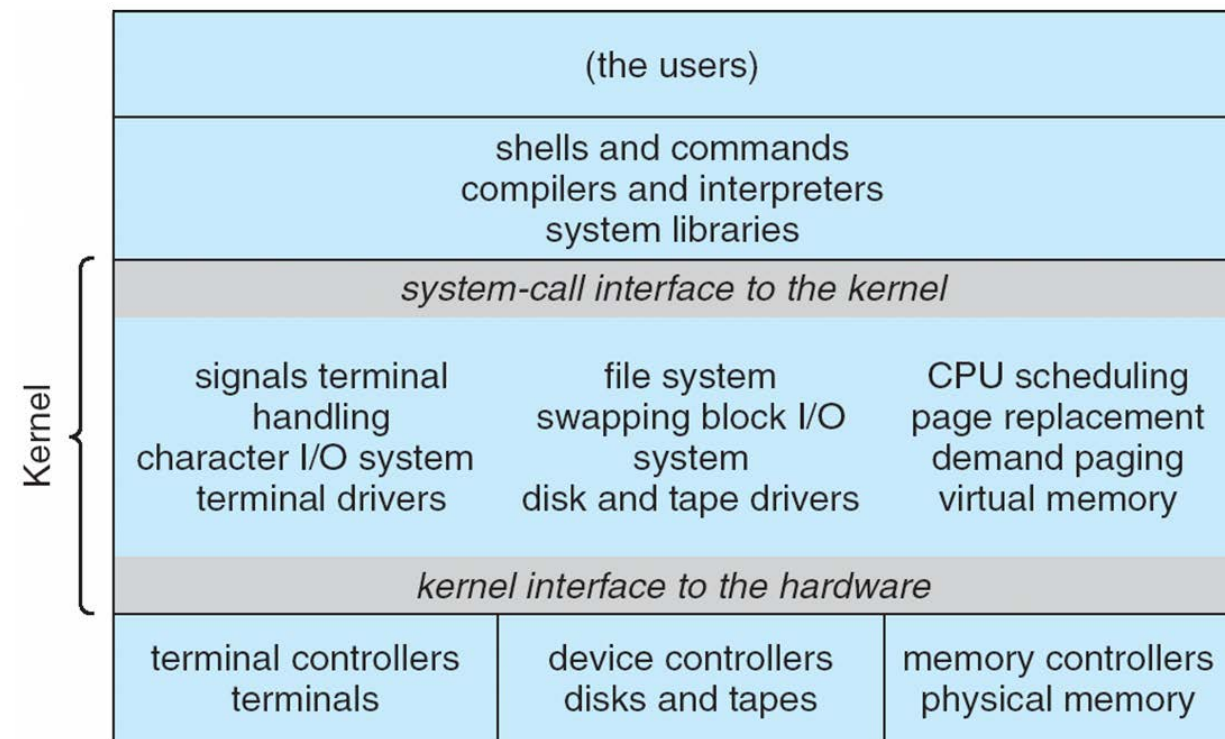
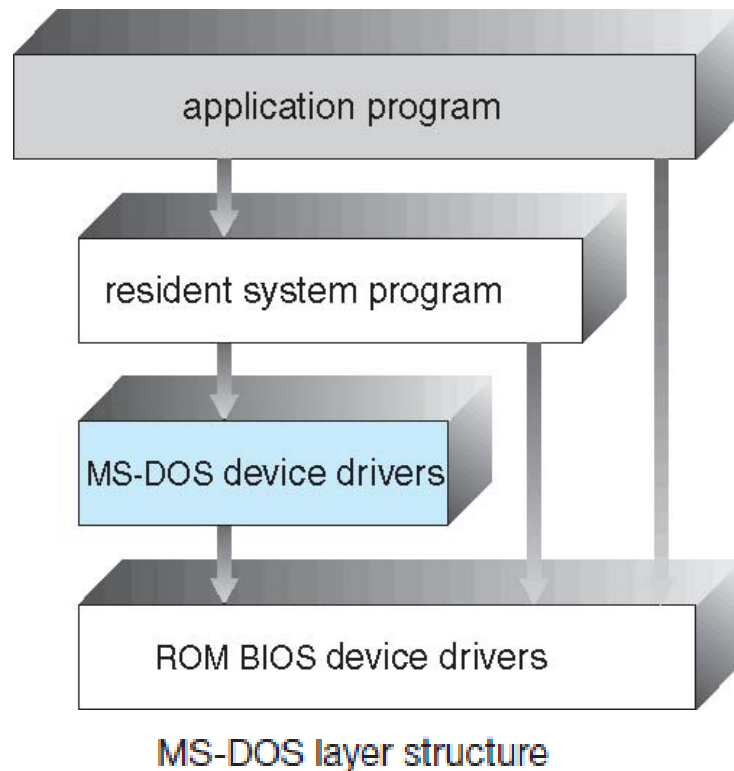


A simple structuring model for a monolithic system

# Samples

## ➤ Simple structure

### ○ MS-DOS, UNIX (Traditional)



Traditional UNIX system structure

Beyond simple but not fully layered

# OS structure: 2. Layered approach

## ➤ Layered approach

- abstractions

- adv.

  - **Simplicity**

    - ✓ Construction

    - ✓ Debugging

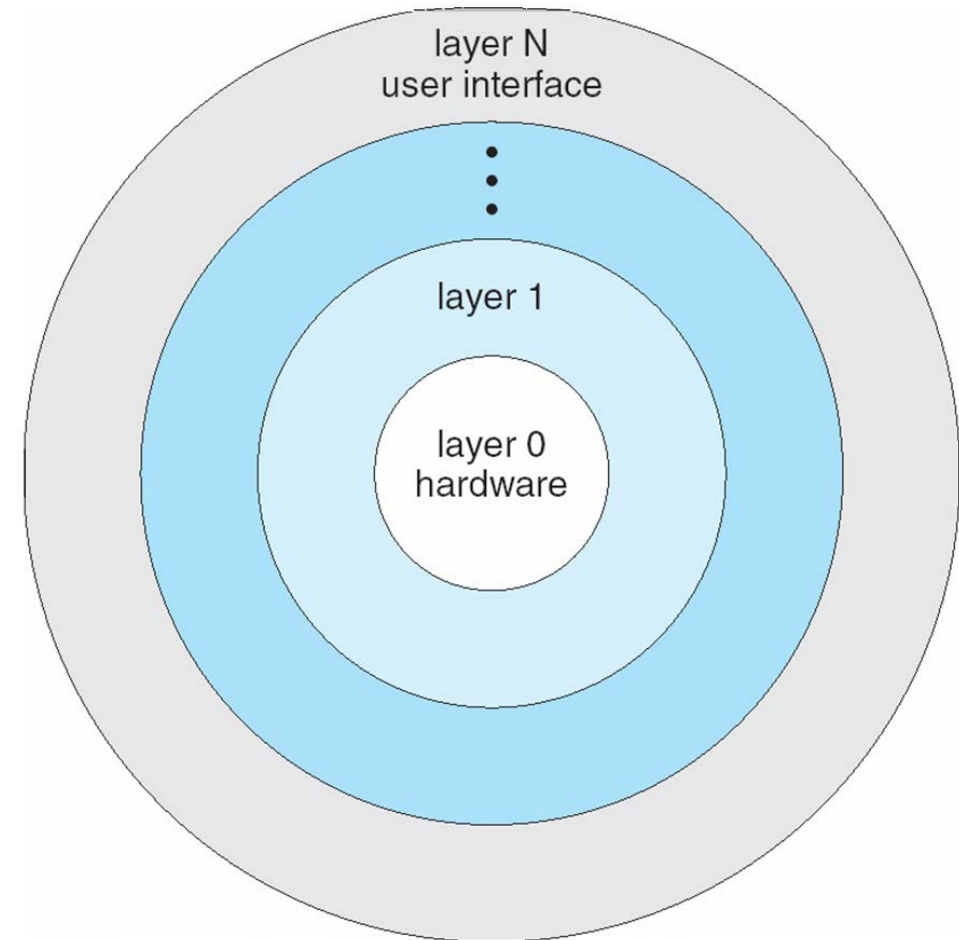
  - **Functions and operations of low layers**

- dis. adv.

  - **Layer definition problem**

    - ✓ MMU, backing store, scheduler (?)

  - **Less efficient**



# OS structure: 3. Microkernel

## ➤ Microkernel

- Moves as much from the kernel into "*user*" space
- Communication takes place between **user modules** using **message passing**

## ➤ Examples

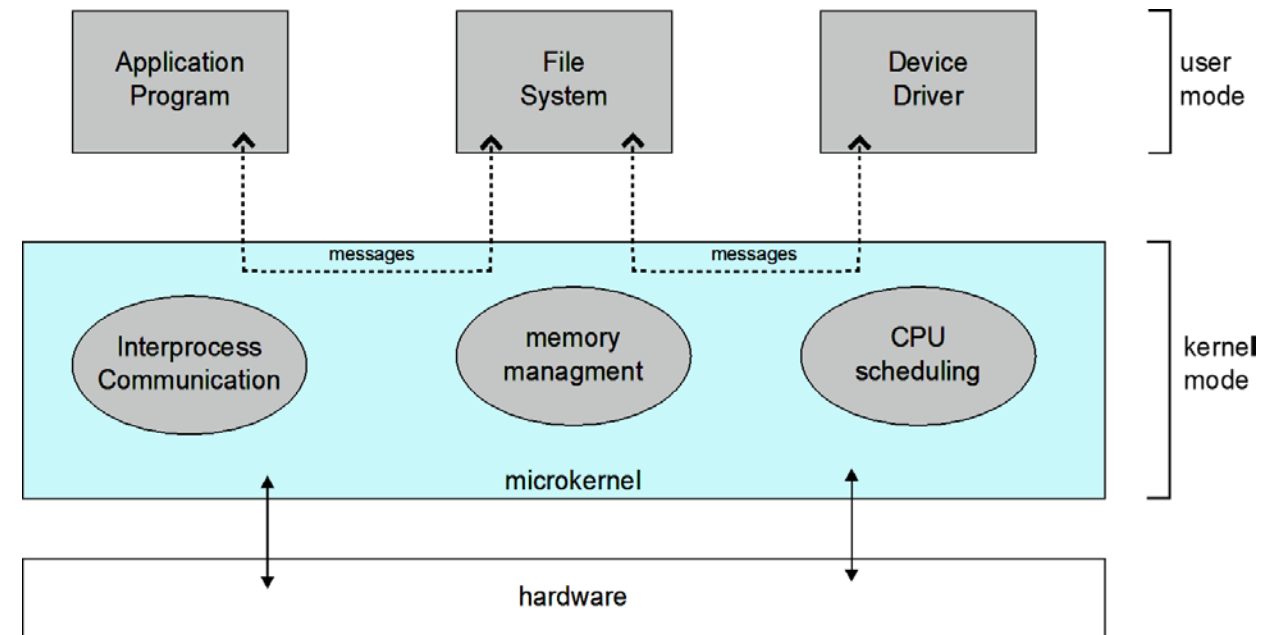
- Mach (CMU)
- Mac OS X kernel (Darwin)

## ➤ Benefits:

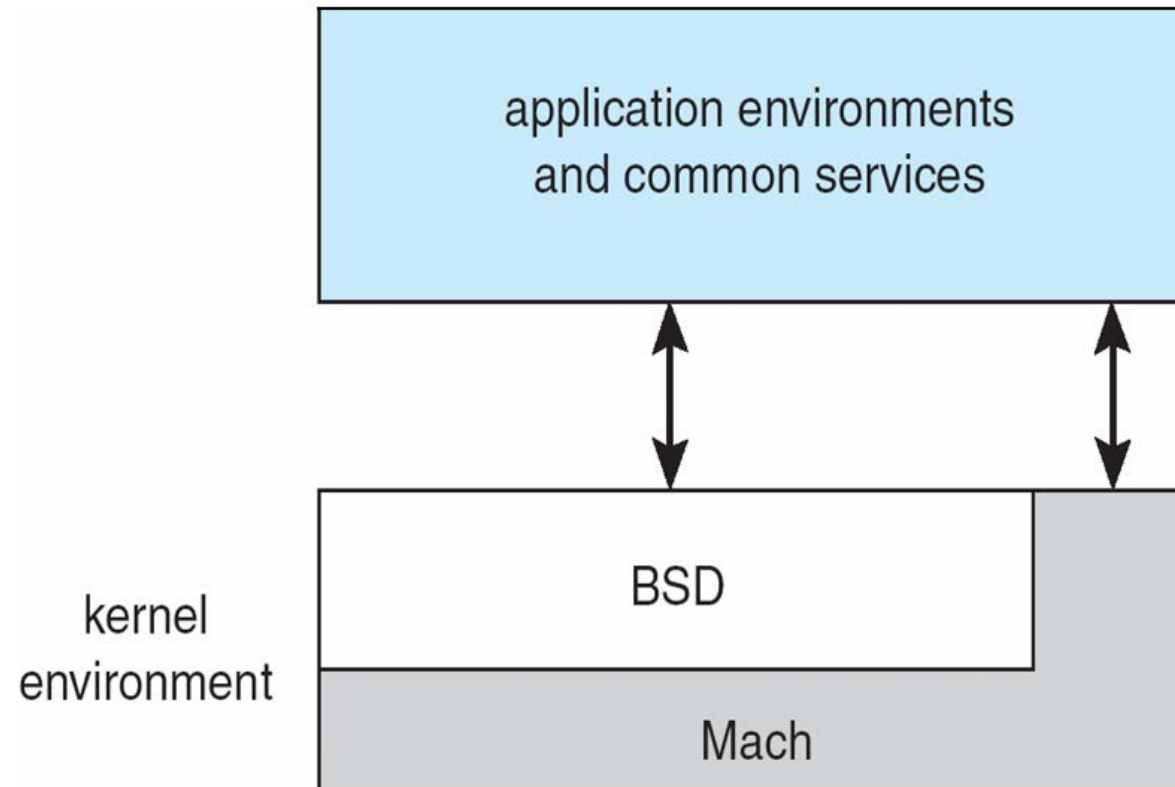
- Easier to extend a microkernel
- Easier to port the operating system to new architectures
- More reliable (less code is running in kernel mode)
- More secure

## ➤ Detriments:

- Performance overhead of user space to kernel space communication
- Windows NT 4 (microkernel) slow!
  - vs. Windows XP (monolithic) fast!



# Mac OS X structure

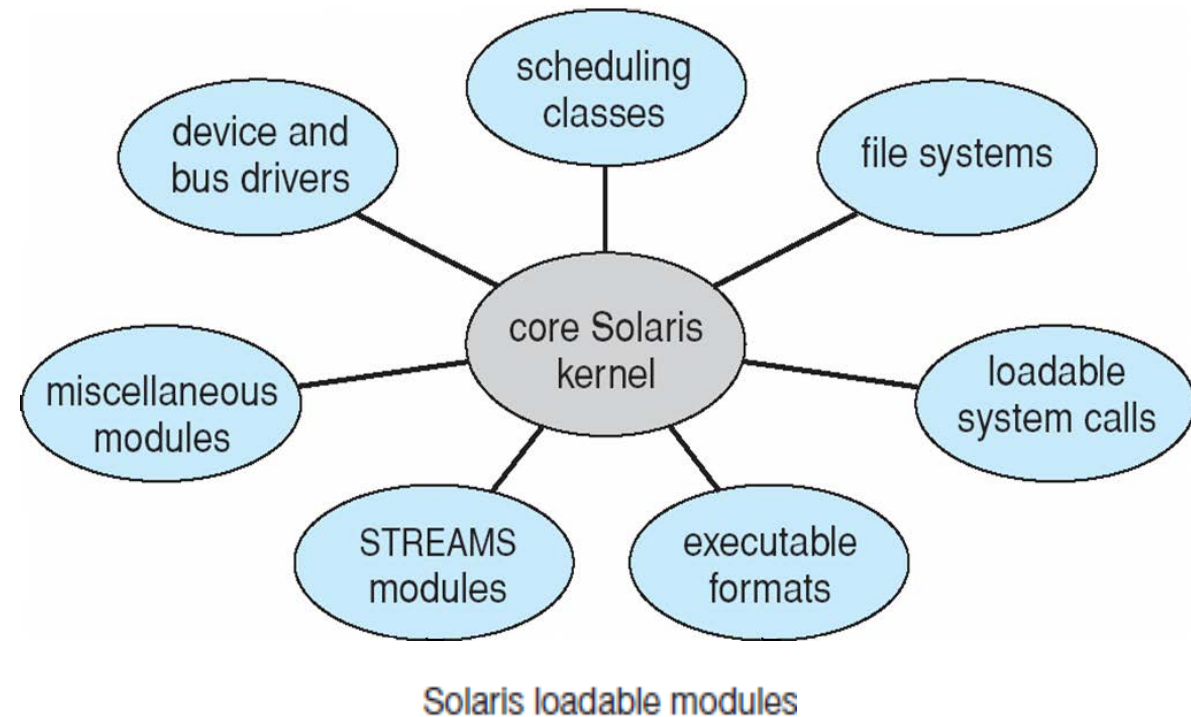




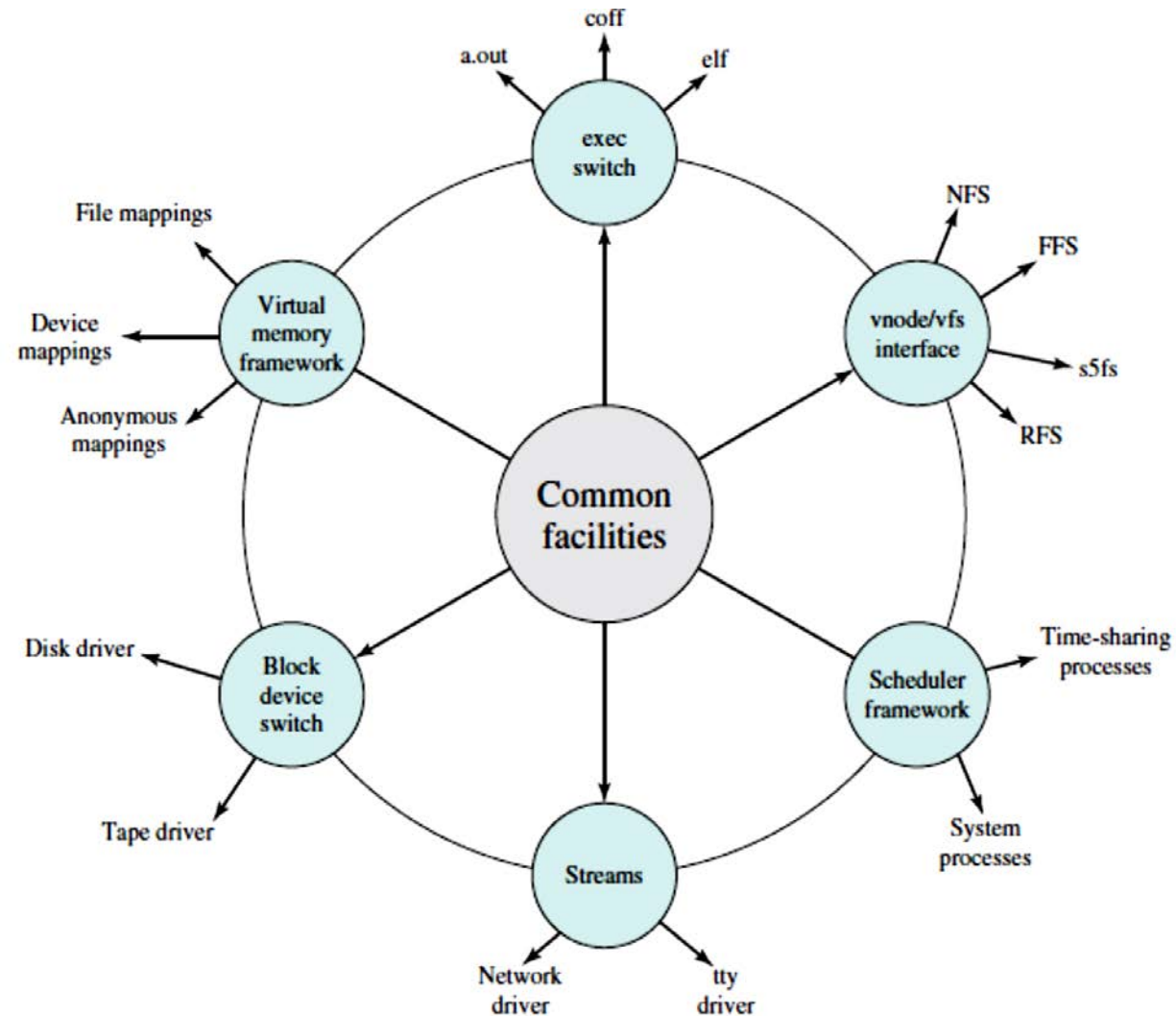
# OS structure: 4. Modules

## ➤ Modules

- Most modern operating systems implement kernel modules
  - 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
  - Faster than **microkernel**
    - ✓ No need of message passing
  - Better than **layered**
    - ✓ Direct module communications
- Overall, similar to **layers** but with more flexible
  - **Linux**, **Solaris**, etc



# Modern Unix kernel



# OS structure: 5. Hybrid

## ➤ Hybrid systems

### ○ Most modern operating systems

- Mac OS
- iOS
- Android

### ○ Better to address

- Reliability
- Security
- Usability

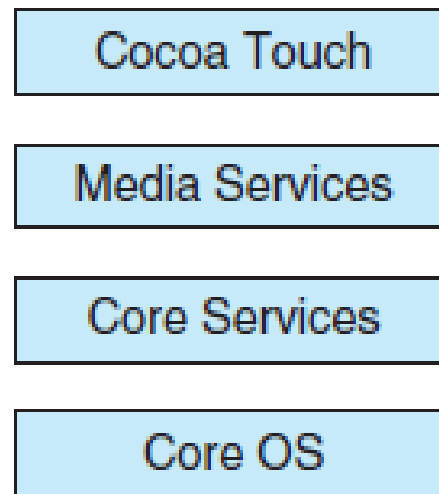
### ○ Examples

#### ■ Linux & Solaris

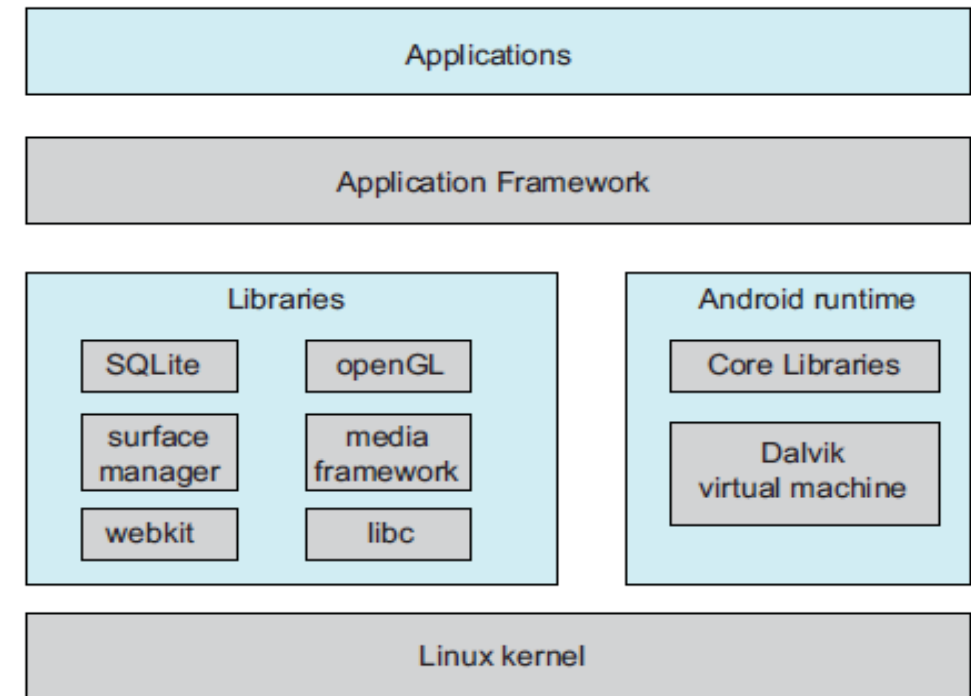
- ✓ kernel: **monolithic**
- ✓ +feature: **loadable**

#### ■ Windows

- ✓ **monolithic**+**microkernel**

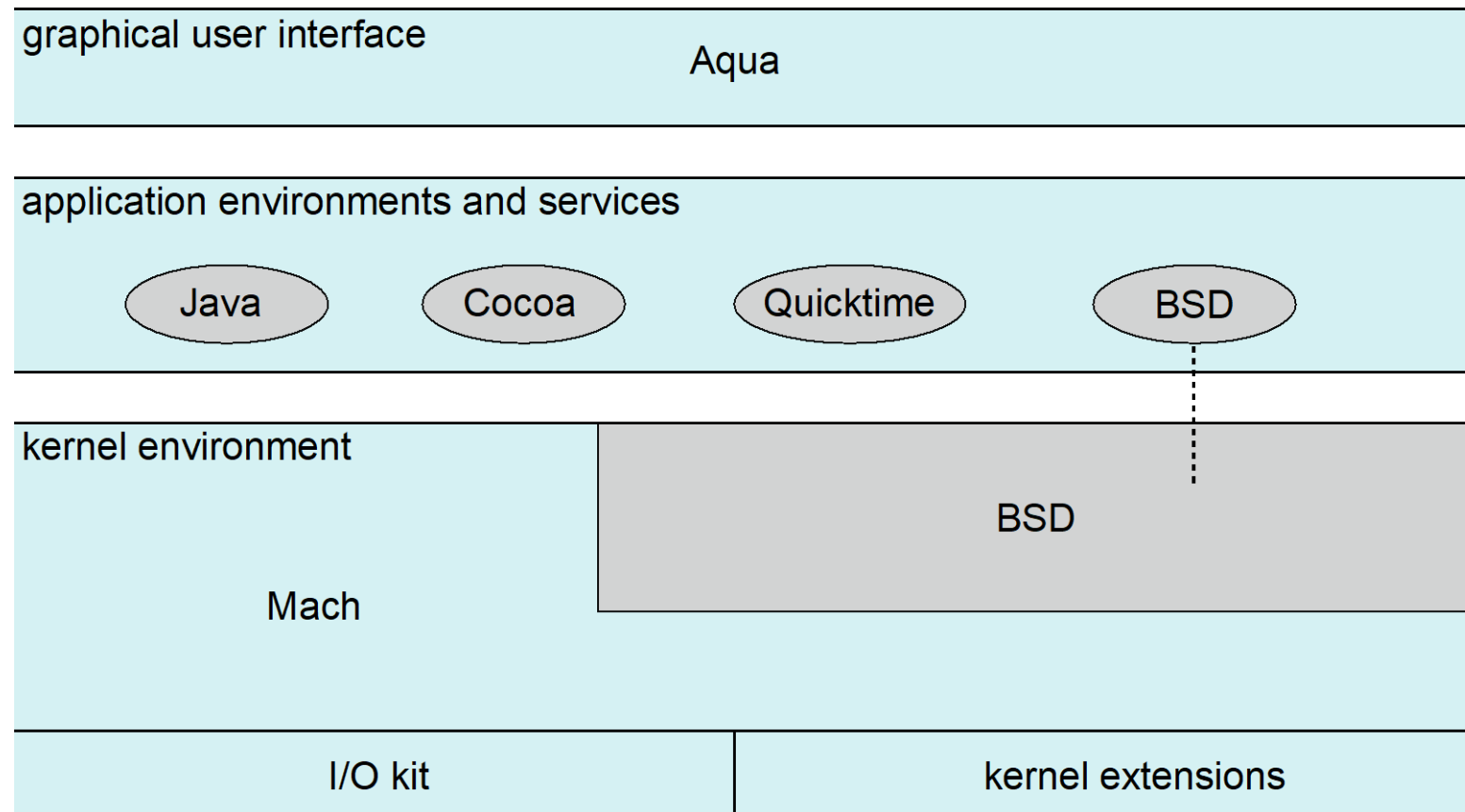


Architecture of Apple's iOS



Architecture of Google's Android

# Mac OS X structure



- Top is layered
- Below is kernel consisting of **Mach microkernel** and BSD Unix parts, plus I/O kit and **dynamically loadable modules** (called **kernel extensions**)

# 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

Cocoa Touch

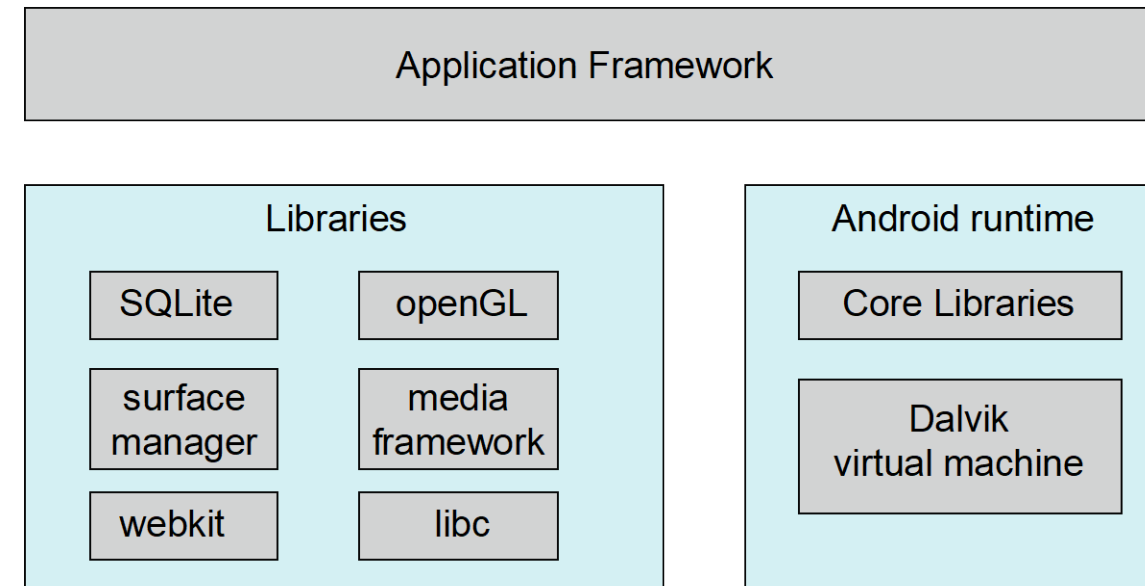
Media Services

Core Services

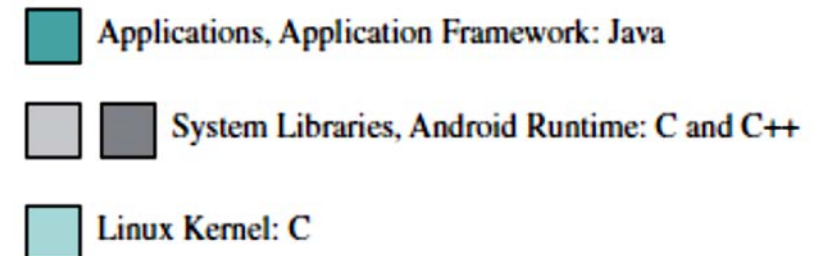
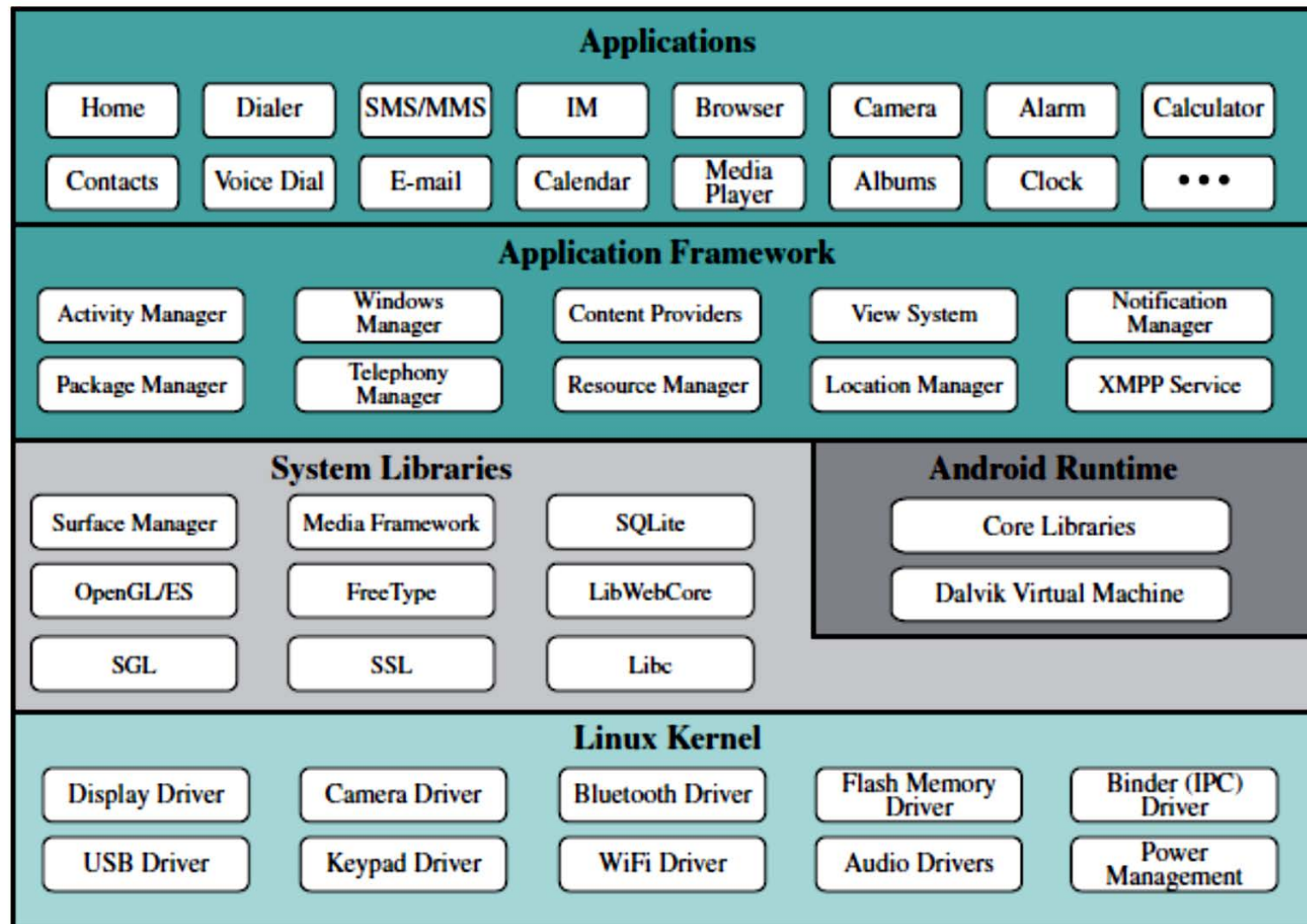
Core OS

# 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

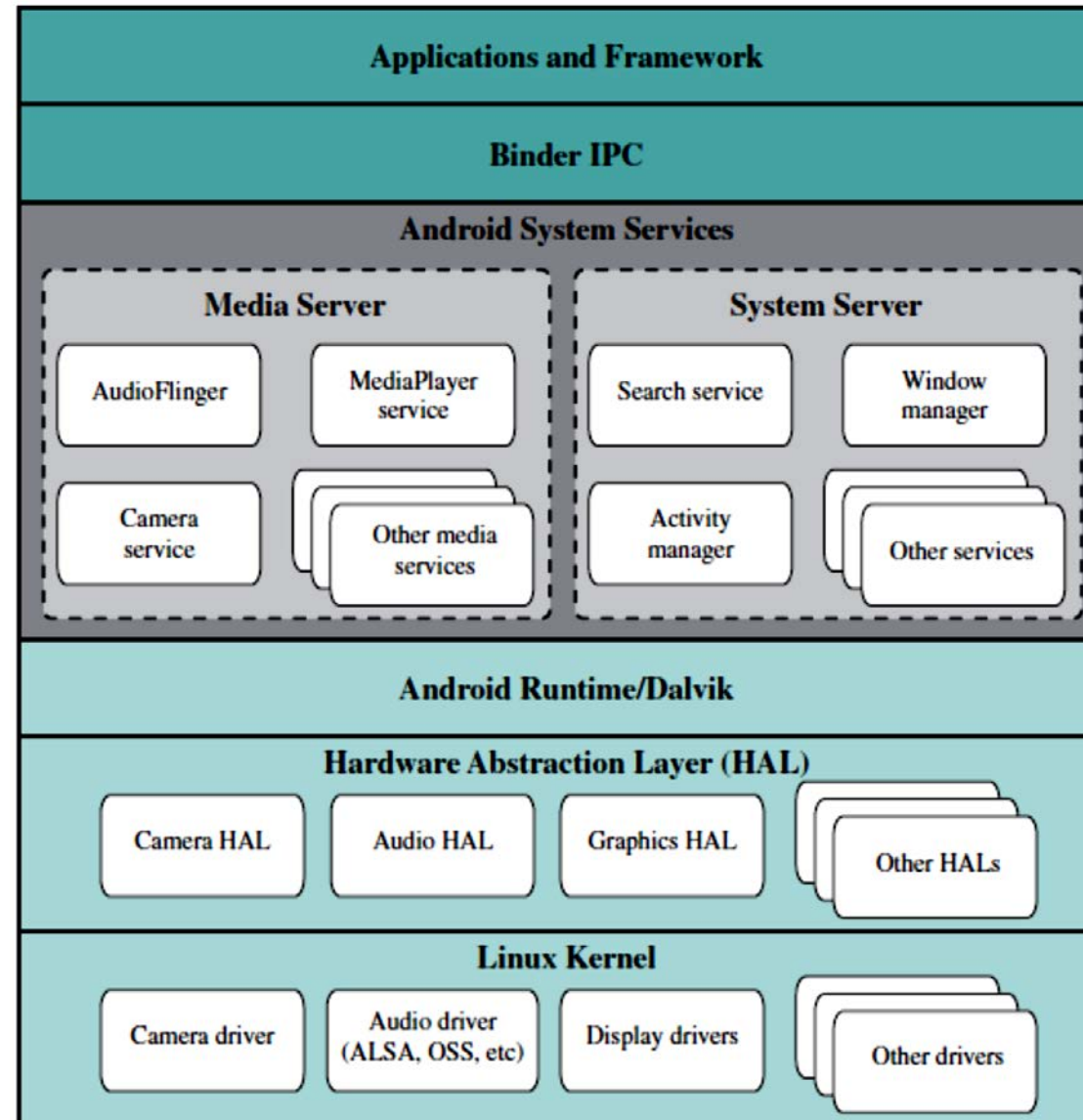


# Android SW architecture (detail)

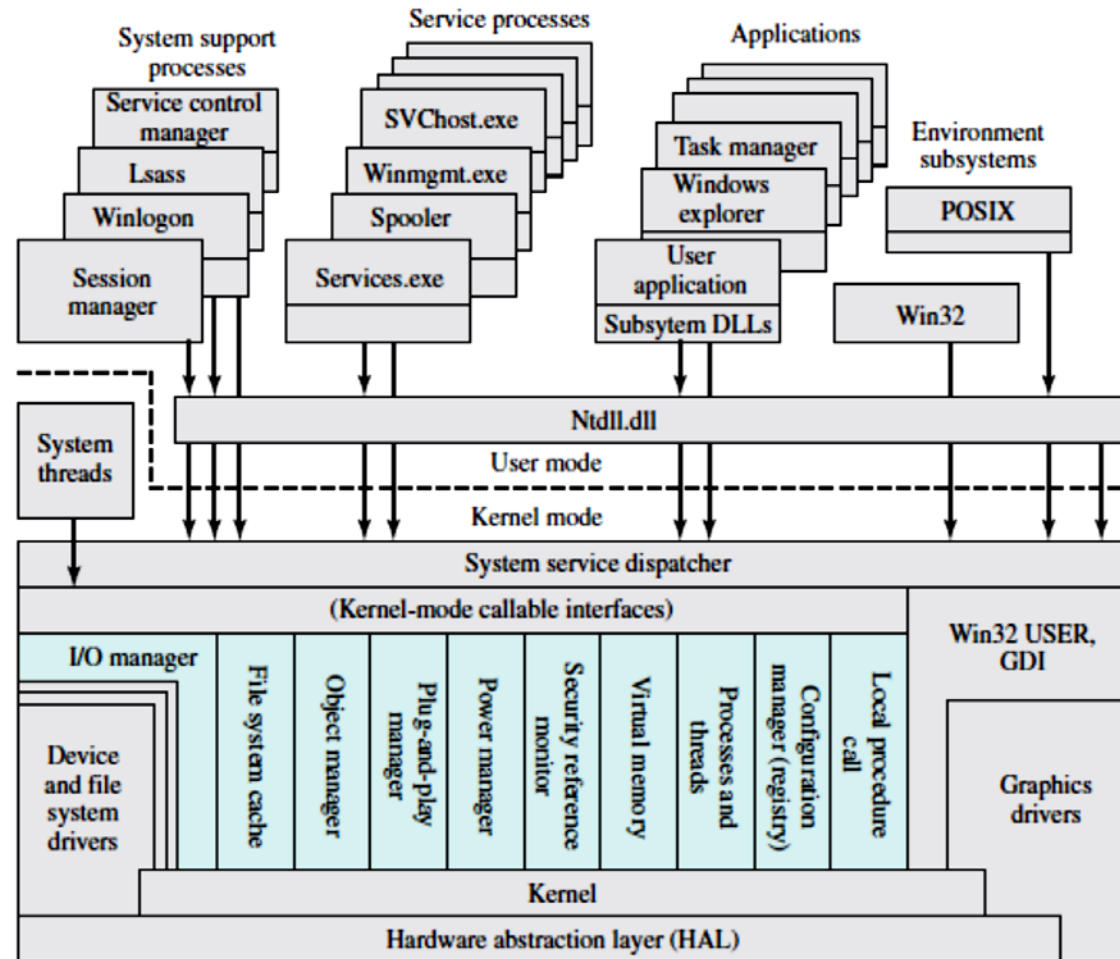




# Android sys. architecture



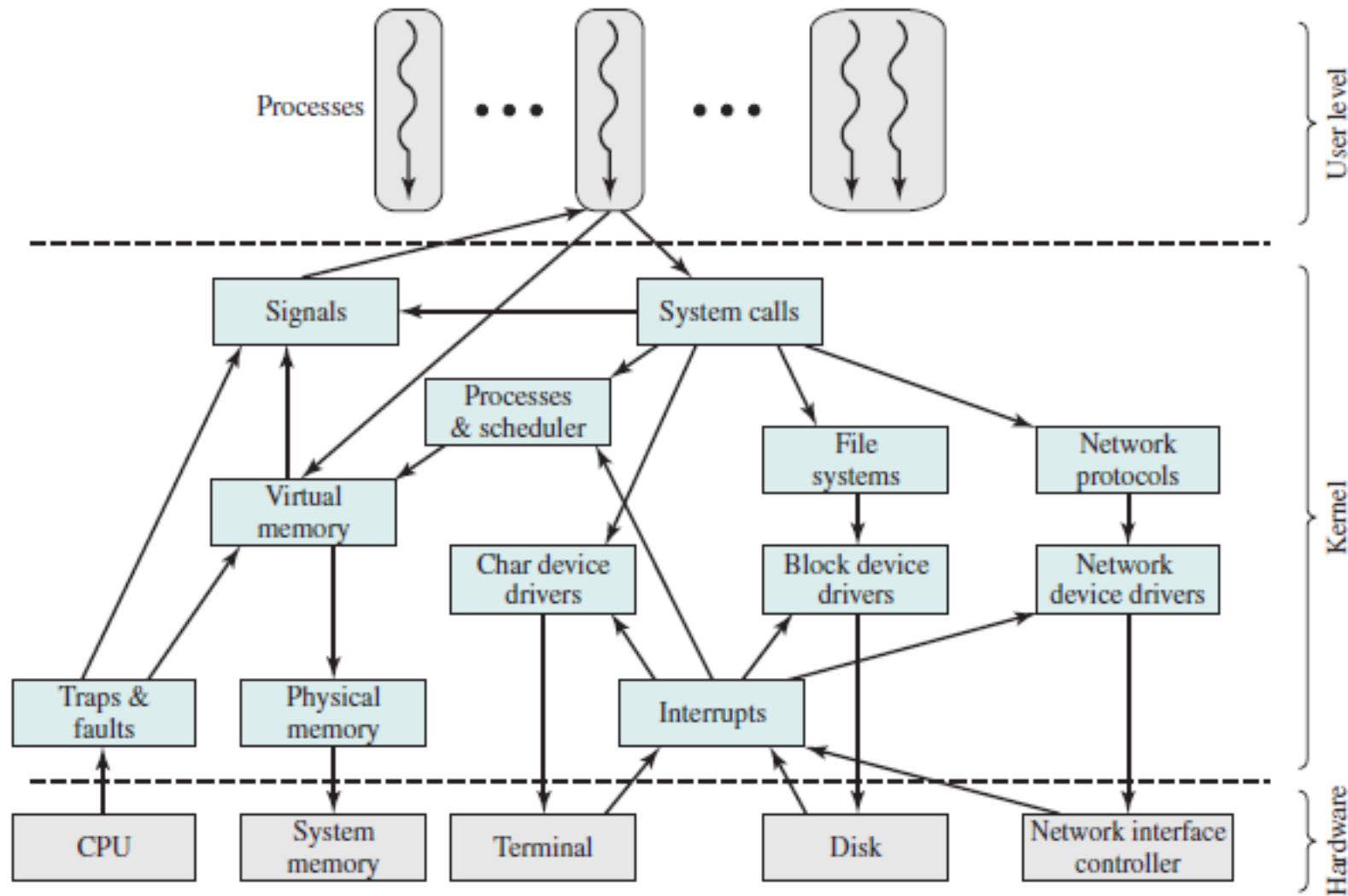
# Windows internal architecture



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

Colored area indicates Executive

# Linux kernel components



# OS debugging

- **Debugging** is finding and fixing errors, or **bugs**
- OSes 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
- 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."
- DTrace tool in Solaris, FreeBSD, Mac OS X allows live instrumentation on production systems
  - **Probes** fire when code is executed, capturing state data and sending it to consumers of those probes

# Solaris 10 *dtrace* Following System Call

```
# ./all.d 'pgrep xclock' XEventsQueued
dtrace: script './all.d' matched 52377 probes
CPU FUNCTION
0 -> XEventsQueued U
0 -> _XEventsQueued U
0 -> _X11TransBytesReadable U
0 <- _X11TransBytesReadable U
0 -> _X11TransSocketBytesReadable U
0 <- _X11TransSocketBytesreadable U
0 -> ioctl U
0 -> ioctl K
0 -> getf K
0 -> set_active_fd K
0 <- set_active_fd K
0 <- getf K
0 -> get_udatamodel K
0 <- get_udatamodel K
...
0 -> releasef K
0 -> clear_active_fd K
0 <- clear_active_fd K
0 -> cv_broadcast K
0 <- cv_broadcast K
0 <- releasef K
0 <- ioctl K
0 <- ioctl U
0 <- _XEventsQueued U
0 <- XEventsQueued U
```

# Questions?

