CMPT 300 D100
Operating System I

Lecture 2-1 – OS Structure
Chapter 2

Dr. Hazra Imran

Spring 2022

Try out activity - Sol

Program 1 and 2 run concurrently. Whenever a timeout interrupt occurs, the kernel switches control between the programs. Show the order of instruction execution, assuming program 1 is currently running.

Application 1	Application 2
… instruction i (timeout interrupt) instruction i+1 … instruction k (timeout interrupt) instruction k+1 …	instruction 0 … instruction j (timeout interrupt) instruction j+1 …

1. instruction i
2. instruction 0
3. instruction j
4. Instruction i+1
5. instruction k
6. Instruction j+1

Interrupt

Learning Objectives

- To describe the services an operating system provides to users, processes, and other systems.
- To discuss the various ways of structuring an operating system.
- To be able to debug a sample C program

Operating System Services

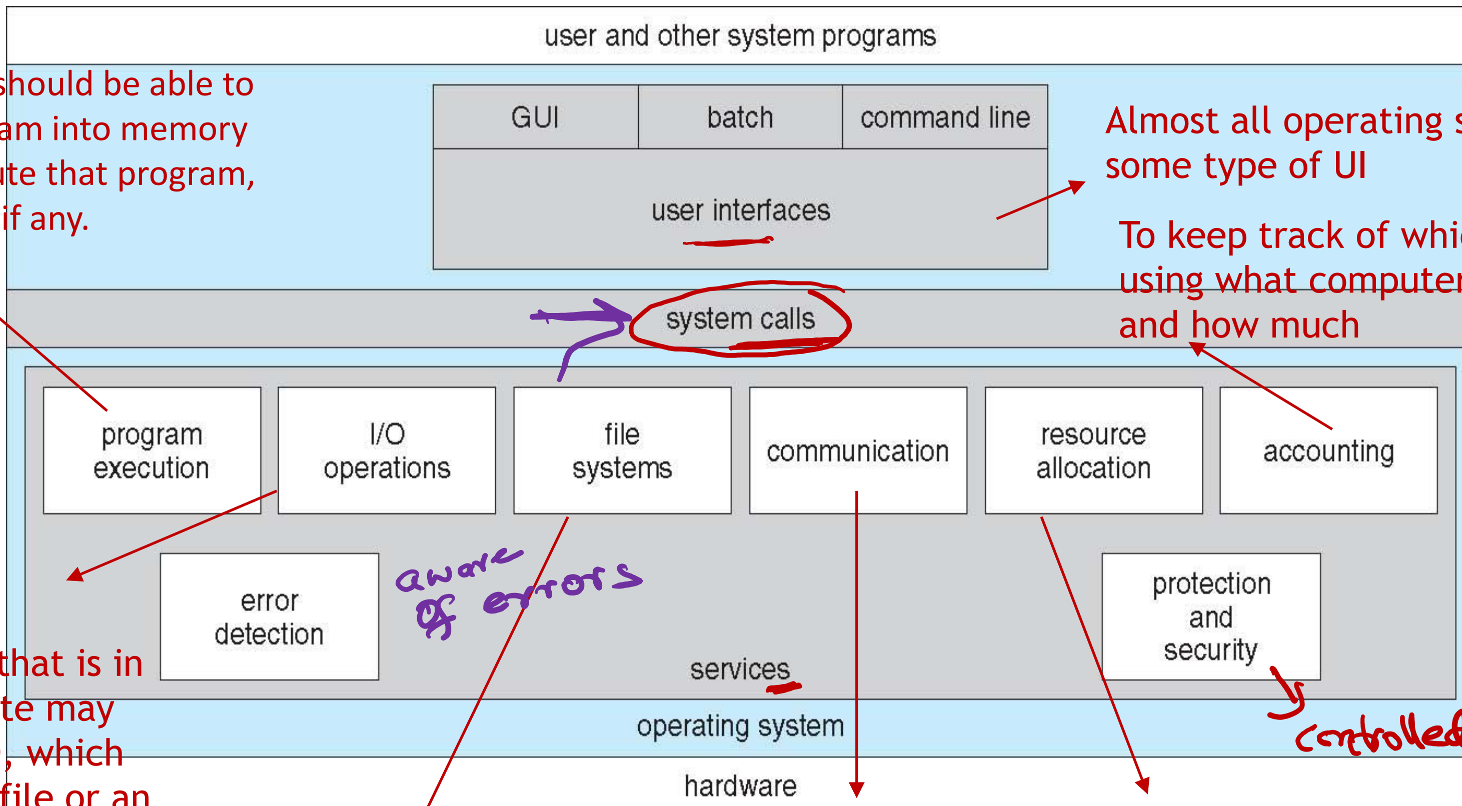
users
efficient operations

Operating systems provide an environment for execution of programs and services to programs and users.

The system should be able to load a program into memory and to execute that program, show errors if any.

Almost all operating systems have some type of UI

To keep track of which users is using what computer resources and how much



aware of errors

controlled

P2

SM

P1

A program that is in running state may require I/O, which could be a file or an I/O device

Programs need to read and write files and directories, create and delete them

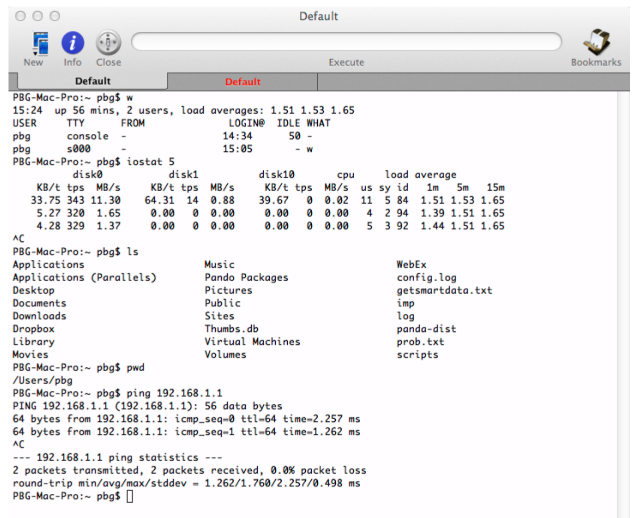
Processes may exchange information which could be through shared memory or message passing.

Example of resources are CPU cycle, I/O devices

CLI vs Shell vs GUI

- Textual interface for user to interact with the system
- Parses text input, which includes commands and parameters and executes the command.

- A shell program is an implementation of a CLI



- User-friendly desktop metaphor interface
- Usually mouse, keyboard, and monitor

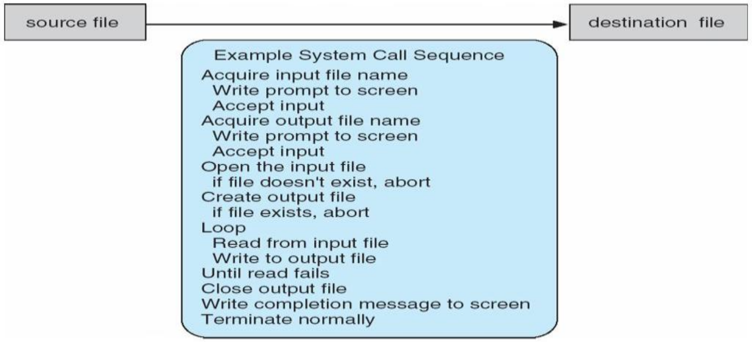
Many systems now include both CLI and GUI interfaces

- Microsoft Windows is GUI with CLI “command” shell
- Apple Mac OS X is “Aqua” GUI interface with UNIX kernel underneath and shells available
- Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)

System Calls

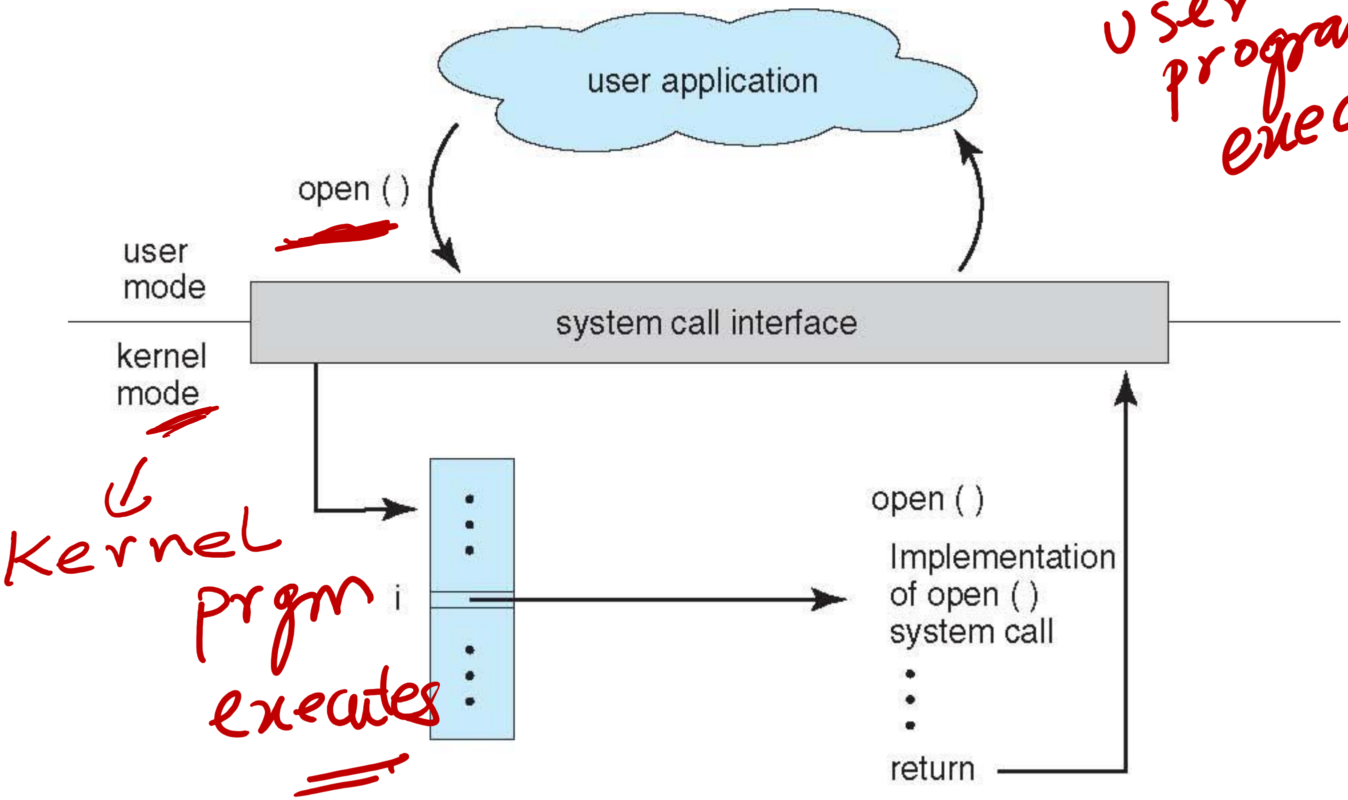
read()

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use
- Three most common APIs are
  - Win32 API for Windows,
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and
  - Java API for the Java virtual machine (JVM)



API – System Call – OS Relationship

user program execute



Question : How the switch between user and kernel mode occurs?

Interupts

System Call

- Typically, a number associated with each system call
  - **System-call interface** maintains a table indexed according to these numbers.
  - The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values

- The caller need know nothing about how the system call is implemented. (usage)

System Call Parameter Passing

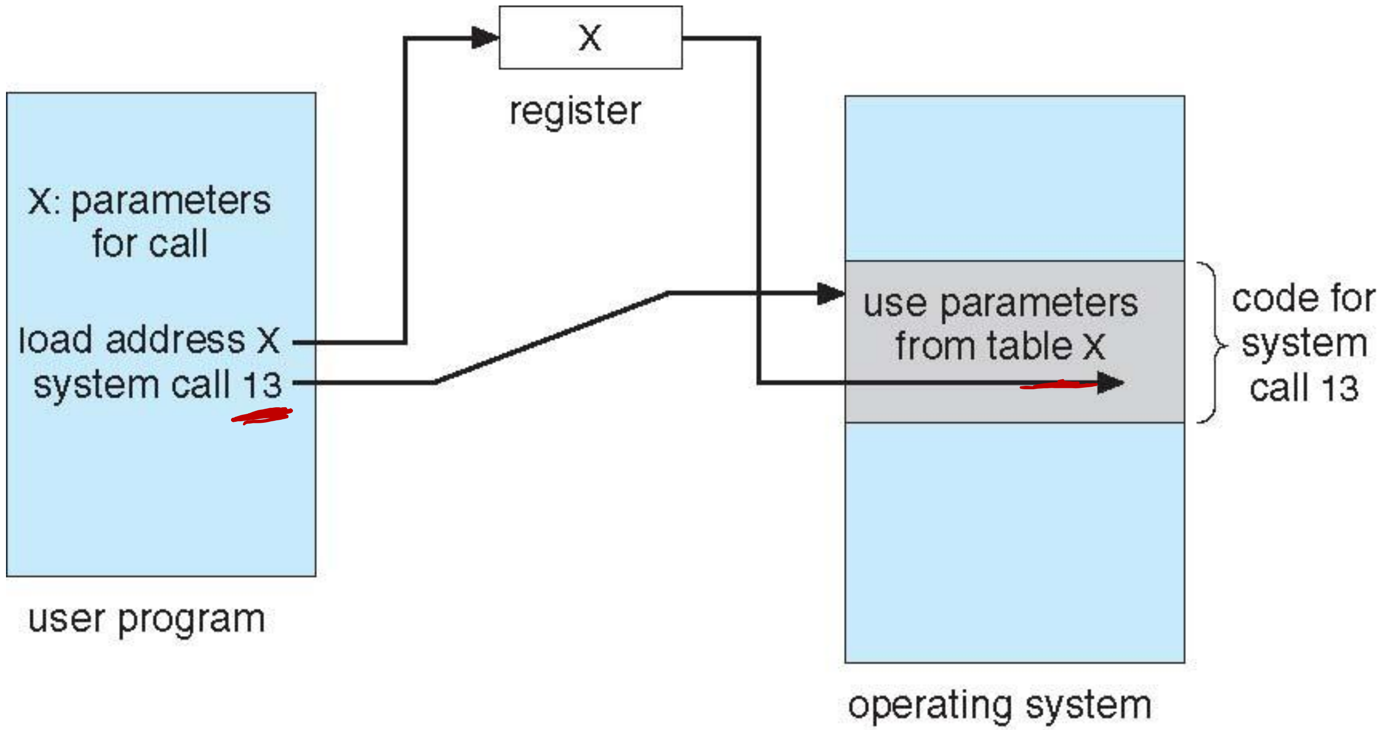
Three general methods used to pass parameters to the OS

1. Simplest: pass the parameters in registers
   - In some cases, may be more parameters than registers

2. Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register
   - This approach taken by Linux and Solaris

3. Parameters placed, or **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system

Parameter Passing via Table



Types of System Calls

1. Process control

- create process, terminate process
- end, abort
- load, execute
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory
- Dump memory if error
- Debugger for determining bugs, single step execution
- Locks for managing access to shared data between processes

2. File management

- create file, delete file
- open, close file
- read, write, reposition
- get and set file attributes

Types of System Calls

3. Device management
- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

4. Information maintenance
- get time or date, set time or date
- get system data, set system data
- get and set process, file, or device attributes

5. Communications
- create, delete communication connection
- send, receive messages if message passing model to host name or process name
- Shared-memory model create and gain access to memory regions
- transfer status information
- attach and detach remote devices

6. Protection
- Control access to resources
- Get and set permissions
- Allow and deny user access

roles

Examples of Windows and Unix System Calls

	Windows	Unix

Process Control	`CreateProcess()`
	`ExitProcess()`
	`WaitForSingleObject()`
		`fork()`
		`exit()`
		`wait()`

File Manipulation	`CreateFile()`
	`ReadFile()`
	`WriteFile()`
	`CloseHandle()`
		`open()`
		`read()`
		`write()`
		`close()`

Device Manipulation	`SetConsoleMode()`
	`ReadConsole()`
	`WriteConsole()`
		`ioctl()`
		`read()`
		`write()`

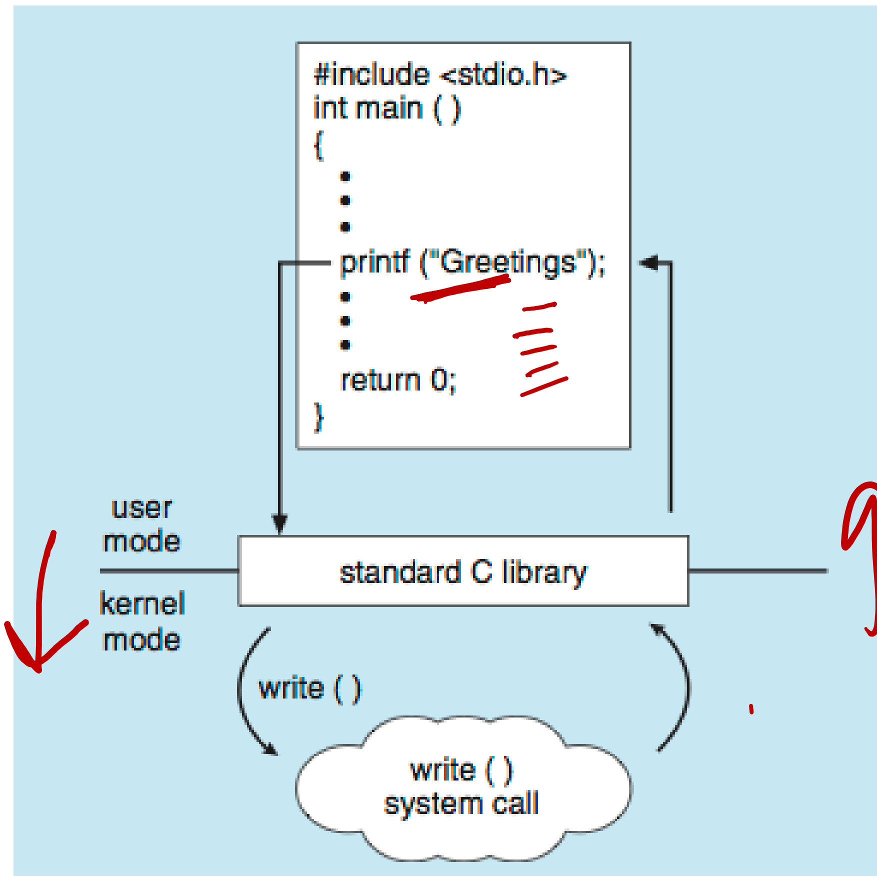
Information Maintenance	`GetCurrentProcessID()`
	`SetTimer()`
	`Sleep()`
		`getpid()`
		`alarm()`
		`sleep()`

Communication	`CreatePipe()`
	`CreateFileMapping()`
	`MapViewOfFile()`
		`pipe()`
		`shmget()`
		`mmap()`

Protection	`SetFileSecurity()`
	`InitlializeSecurityDescriptor()`
	`SetSecurityDescriptorGroup()`
		`chmod()`
		`umask()`
		`chown()`

Standard C Library Example

- C program invoking printf() library call, which calls write() system call



bit = 0
1

PCB
status

Example: strace on Linux

strace -c ls

- Displays summary info on system calls invoked during the execution of the command ‘ls’.
- Numerous system calls are typically invoked for even simple tasks

```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 22.47    0.000202           7        27           mmap
 15.35    0.000138          15         9           openat
 14.24    0.000128          14         9           mprotect
  7.68    0.000069           9         7           read
  5.34    0.000048           4        10           fstat
  5.34    0.000048          24         2         2 statfs
  5.23    0.000047           4        11           close
  5.23    0.000047          47         1           munmap
  3.00    0.000027           9         3           brk
  2.67    0.000024           3         8           pread64
  2.56    0.000023          11         2           rt_sigaction
  2.11    0.000019           9         2           ioctl
  1.33    0.000012           6         2         1 arch_prctl
  1.22    0.000011          11         1           rt_sigprocmask
  1.22    0.000011           5         2           getdents64
  1.22    0.000011          11         1           set_tid_address
  1.22    0.000011          11         1           set_robust_list
  1.22    0.000011          11         1           prlimit64
  0.78    0.000007           7         1           write
  0.56    0.000005           2         2         2 access
  0.00    0.000000           0         1           execve
------ ----------- ----------- --------- --------- ----------------
100.00    0.000899                   103         5 total
```

man strace

- Displays info (manual) on strace

Dual Mode – Transition between User/Kernel Modes

Allow only the OS to perform privileged instructions

Privileged instructions: operations that may cause harm

- Examples: I/O control, timer management, interrupts

User and kernel modes:

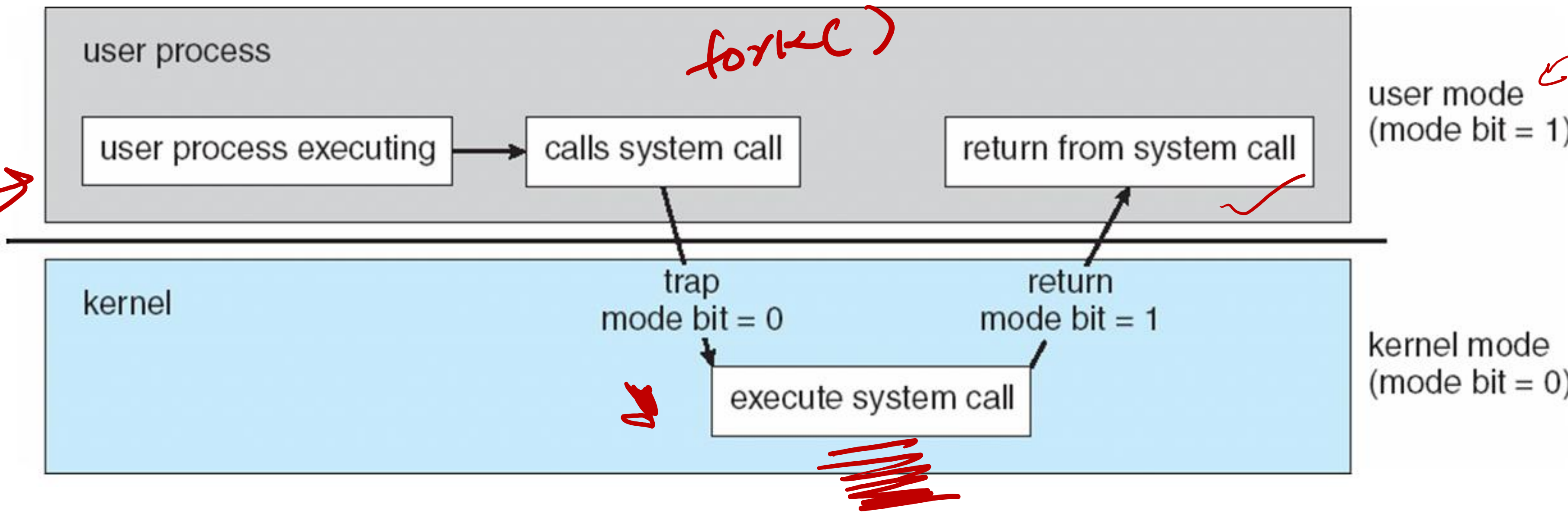
- User mode: process executes on behalf of the user
- Kernel mode: process executes on behalf of the kernel
- Process in user mode cannot execute privileged instructions

control

Mode bit: a bit in hardware to switch between modes

System boot – h/w K mode
Load OS → h/w U mode

Dual Mode – Transition between User/Kernel Modes



1. User code calls a system call
2. OS checks that everything (e.g., parameter values) is in order and legal, switch to kernel mode
3. OS executes system call which may contain multiple privileged instructions
4. Mode again set to user mode

Clicker

Changing the mode bit of the CPU ______ can only be done by a privileged instruction

(A) From user mode to kernel mode

(B) From Kernel mode to user mode – without any constraints

(C) in any way

- A ***supervisor call*** (***kernel call***) is a privileged instruction that automatically transfers execution control to a well-defined location within the OS kernel.
- Thus supervisor calls provide the interface between the OS kernel and the higher-level software.

sudo/su vs kernel/user mode

- User types
  - Superuser: can do anything
  - "Normal" user: can only access their own files, programs
- su – switch to super user account
- sudo – run program on behalf of the superuser
  - Example: change system configuration (modify global config files under /etc/), run another user's program
  - Not necessarily switches to the kernel mode!

Clicker

A system call is a kernel function.

(A) True
(B) False

A system call is a service function that invokes kernel functions, but is itself more of a high-level operation than a kernel function.

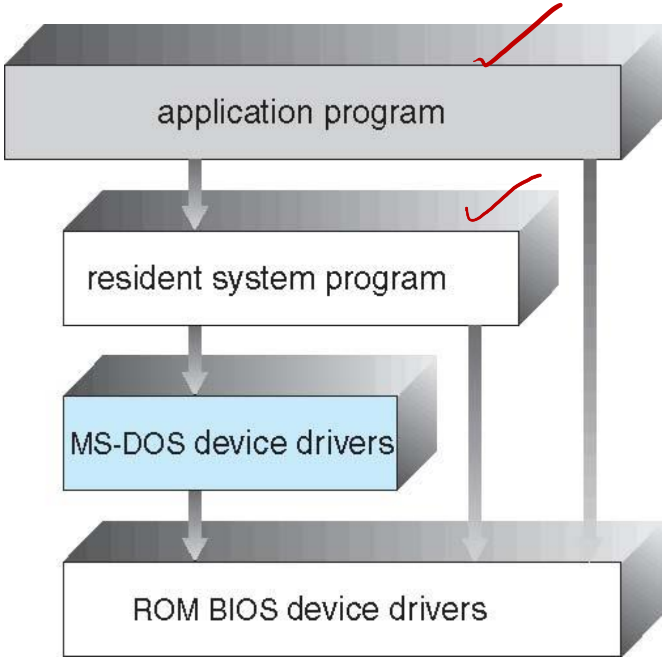
Operating System Structure

- General-purpose OS is very large program

- Various ways to structure ones
  - Simple structure – MS-DOS
  - Monolithic structure
  - Layered approach – an abstraction
  - Microkernel
  - Modular
  - Hybrid

Simple Structure - MS-DOS

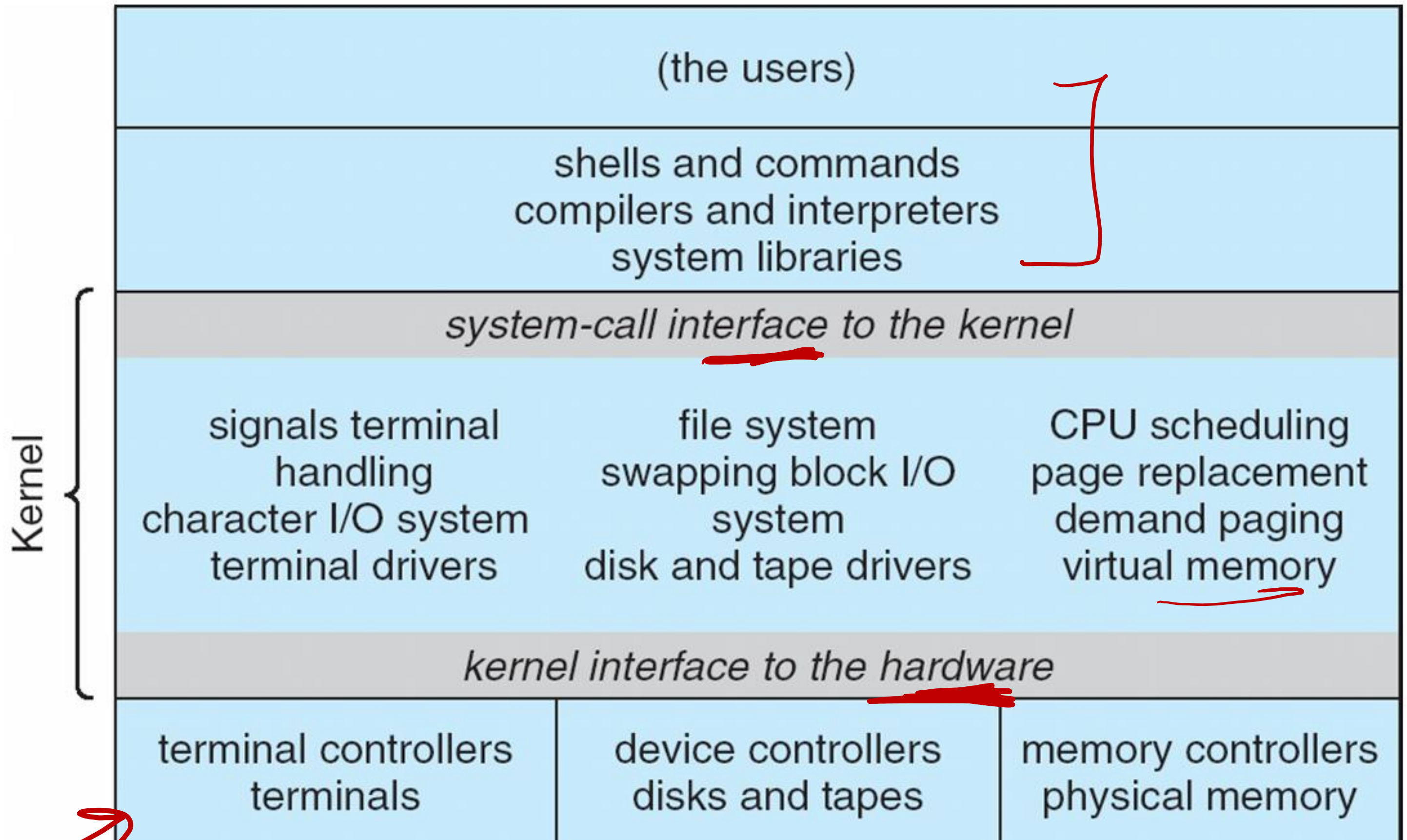
- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules
  - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated



Monolithic

Unix

No structure – entire kernel in a single binary file



Beyond simple but not fully layered

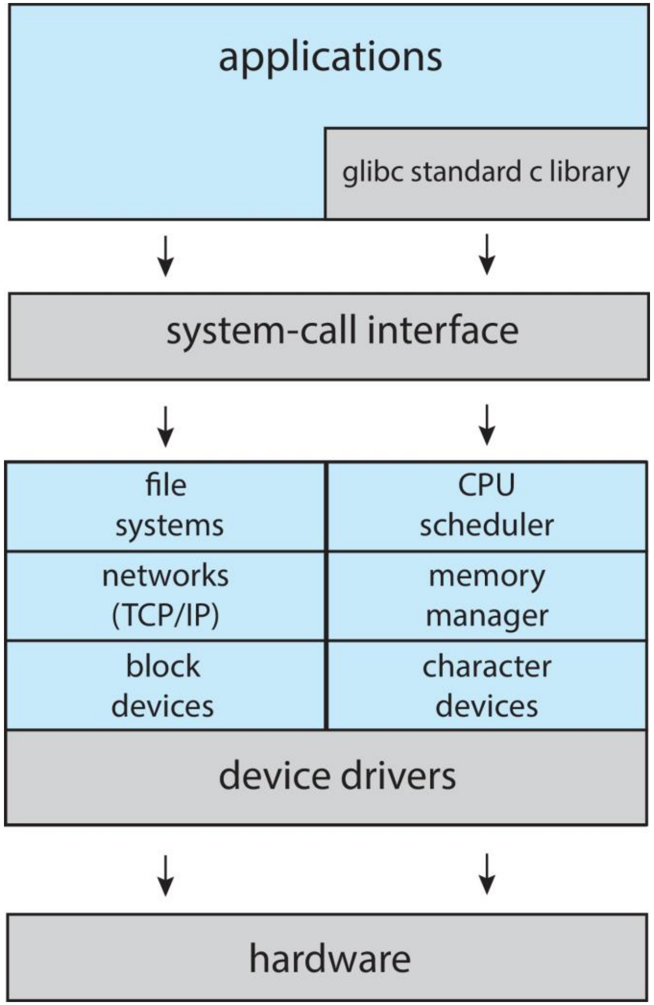
Monolithic Structure

Advantage:

- High performance
  - Little overhead in system call interface
  - Communication within the kernel is fast

Problems:

- Difficult to implement & maintain: Too much in one layer
- Examples: Traditional UNIX, Linux

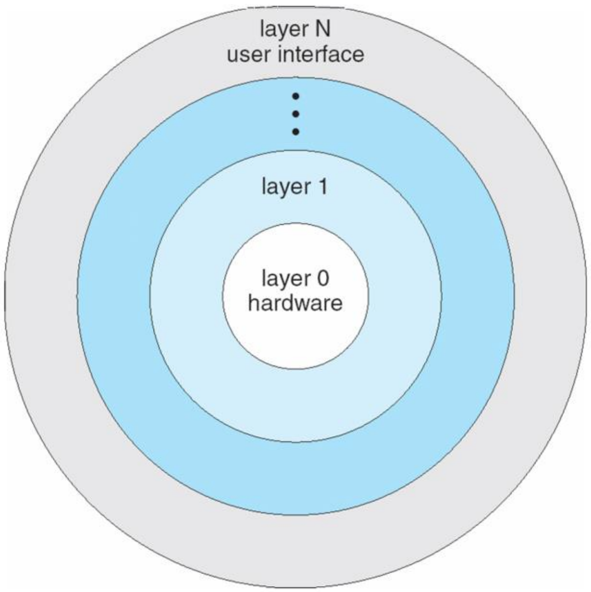


Linux: Monolithic plus modular design

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

Advantage:
- Easier to develop, debug, and update
- Focus on one layer at a time

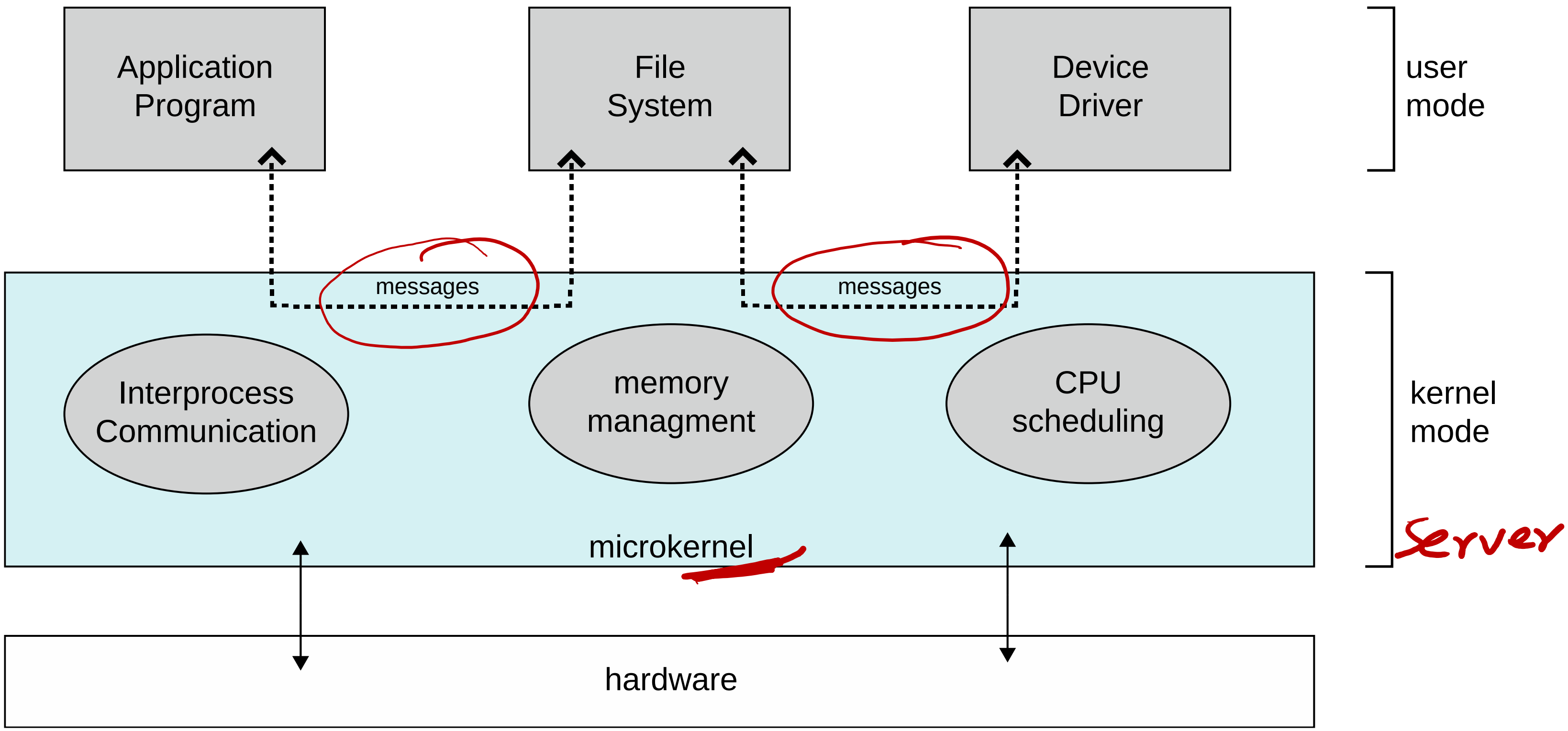


Problems :
- Less efficient - Each layer adds some overhead
- Tricky to define layers

Microkernel System Structure

- Moves as much from the kernel into user space
- Mach example of microkernel
  - Mac OS X kernel (Darwin) partly based on Mach
- Communication takes place between user modules using message passing
- **Advantages:**
  - 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
- **Problem**
  - Performance overhead of user space to kernel space communication

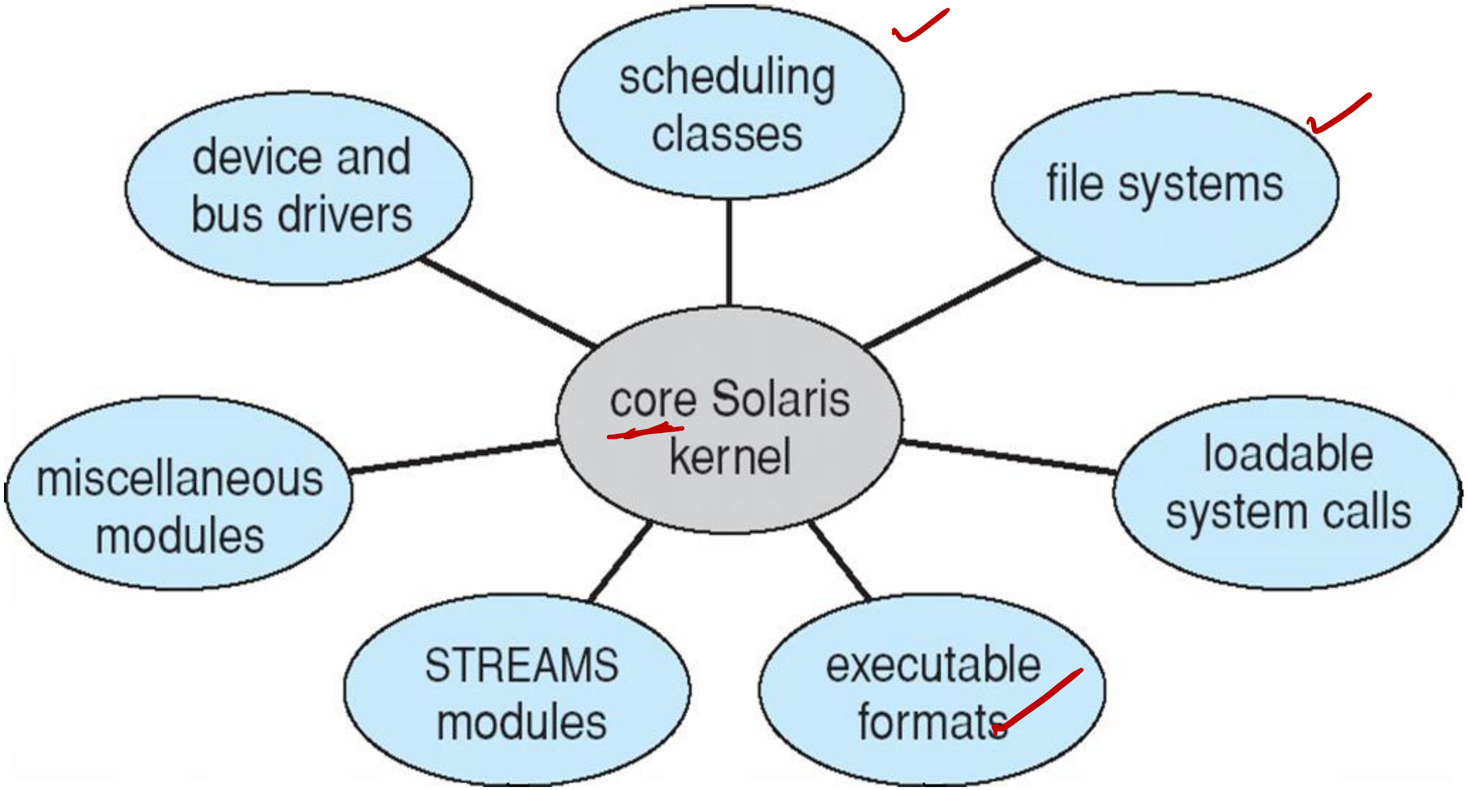
Microkernel System Structure



Modules

- Many modern operating systems implement **loadable kernel modules**
  - Uses object-oriented approach
  - Each core component is separate
  - Each communicates 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

Solaris Modular Approach



Modular Structure

Advantages:

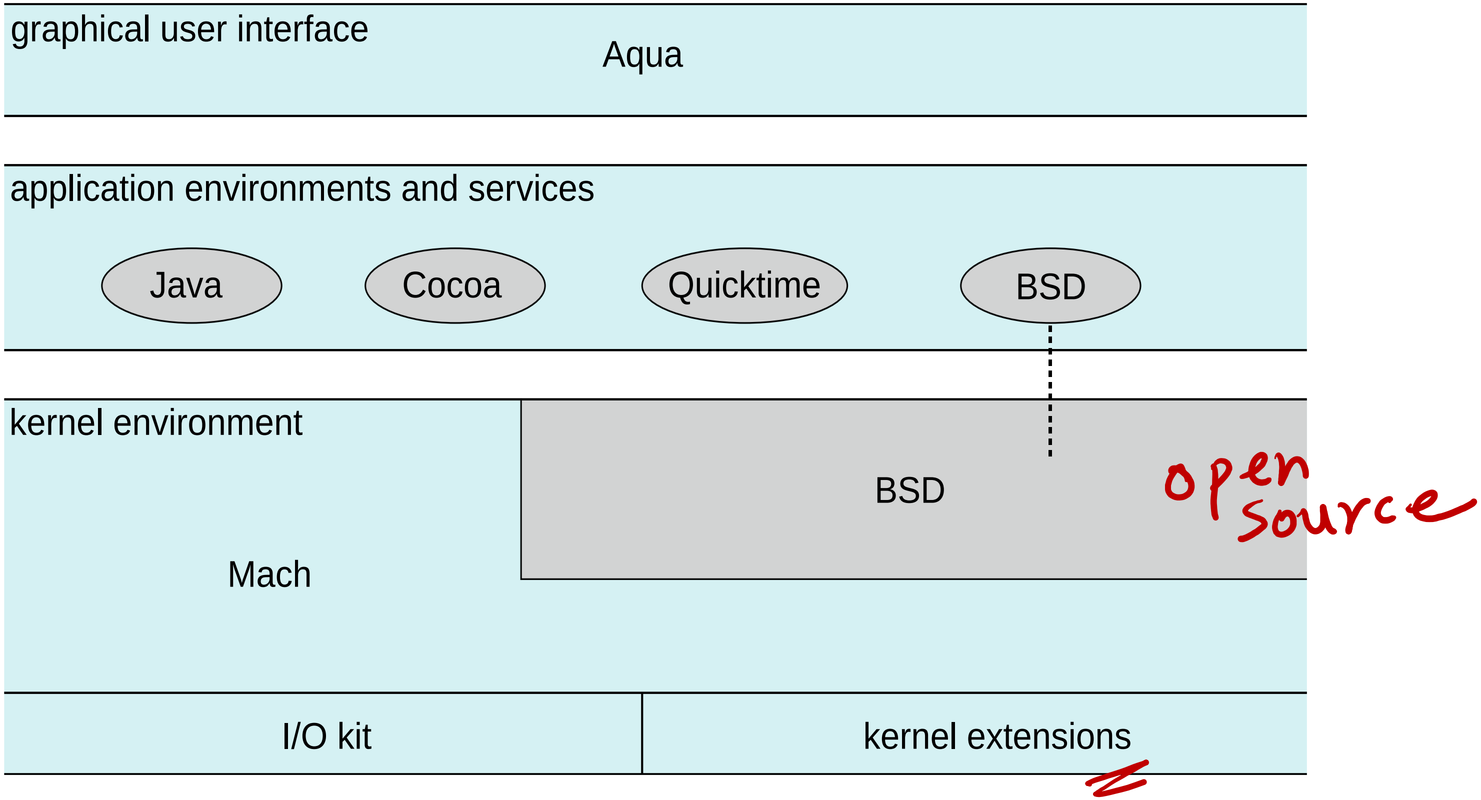
- Easy to maintain, update, and debug
- Similar to layers but more flexible
- Similar to microkernel but more efficient

Problems :

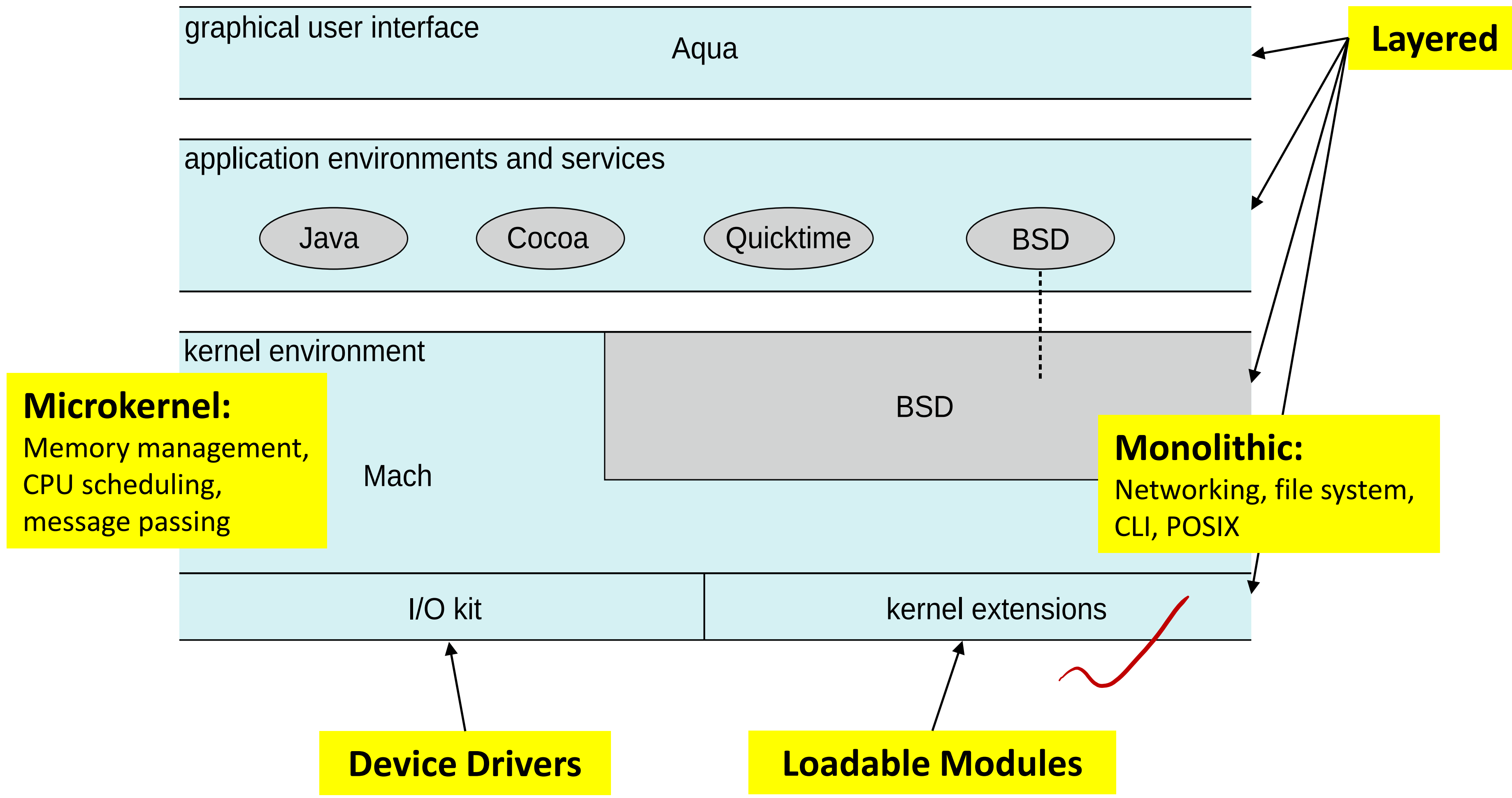
- Modules are part of the kernel with **full permission**
  - Stability: buggy modules may cause the OS to crash
  - Security: a malicious module can compromise the whole system

Hybrid Structures: Most Current OSes

Mac OS X Structure:



Hybrid Structures: Most Current OSes



Mac OS X Structure