

# Finish Your Group Info

- Fill in group info by midnight today
- Fill in even if you plan to have a single-person group



**Group Questionnaire (before 09/18)**



國立陽明交通大學

NATIONAL YANG MING CHIAO TUNG UNIVERSITY

Institute of Artificial Intelligence Innovation

Department of Computer Science

# Operating System

## Lecture 02: OS Structure

Shuo-Han Chen 陳碩漢

[shch@nycu.edu.tw](mailto:shch@nycu.edu.tw)

Wed. 10:10 - 12:00 EC115 +

Fri. 11:10 – 12:00 Online

# Course Schedule

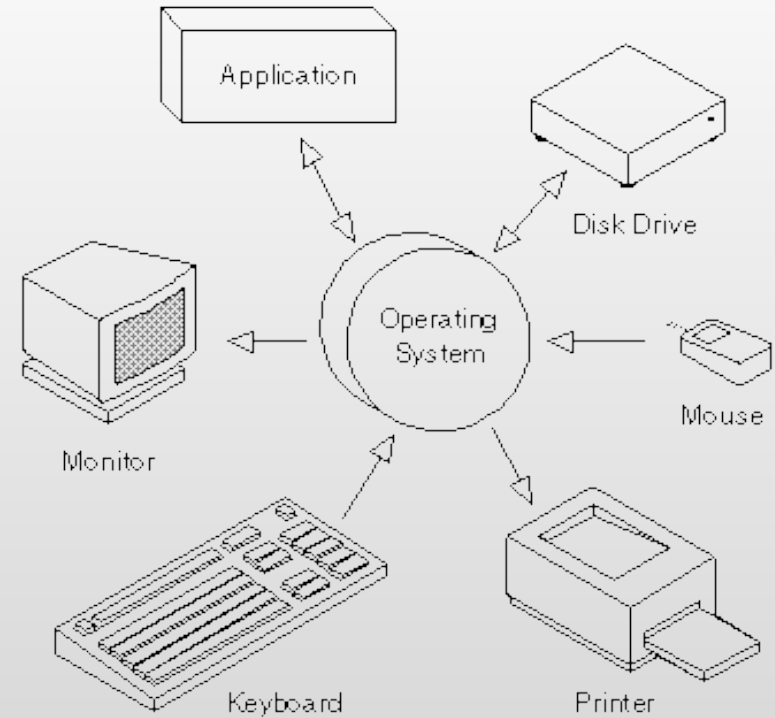
W	Date	Lecture	Online	Homework
1	Sept. 4	Lec01: Course Overview & Historical Prospective		
2	Sept. 11	Lec02: Introduction	V	
3	Sept. 18	Lec03: OS Structure	V	HW01
4	Sept. 25	Lec04: Processes Concept	V	
5	Oct. 2	Lec08: Memory Management	V	
6	Oct. 9	Lec09: Virtual Memory Management	V	HW02
7	Oct. 16	Lec05: Process Scheduling	V	
8	Oct. 23	School Midterm Exam		
9	Oct. 30	Lec06: Process Synchronization	V	
10	Nov. 6	Lec07: Deadlocks	V	HW03
11	Nov. 13	Lec10: File System Interface	V	
12	Nov. 20	School Event – No class		
13	Nov. 27	Lec11: File System Implementation	V	HW04
14	Dec. 4	Lec12: Mass Storage System	V	
15	Dec. 11	Lec13: IO Systems	V	
16	Dec. 18	School Final Exam		

# Outline

- OS Services
- OS-Application Interface
- OS Structure

# OS Services

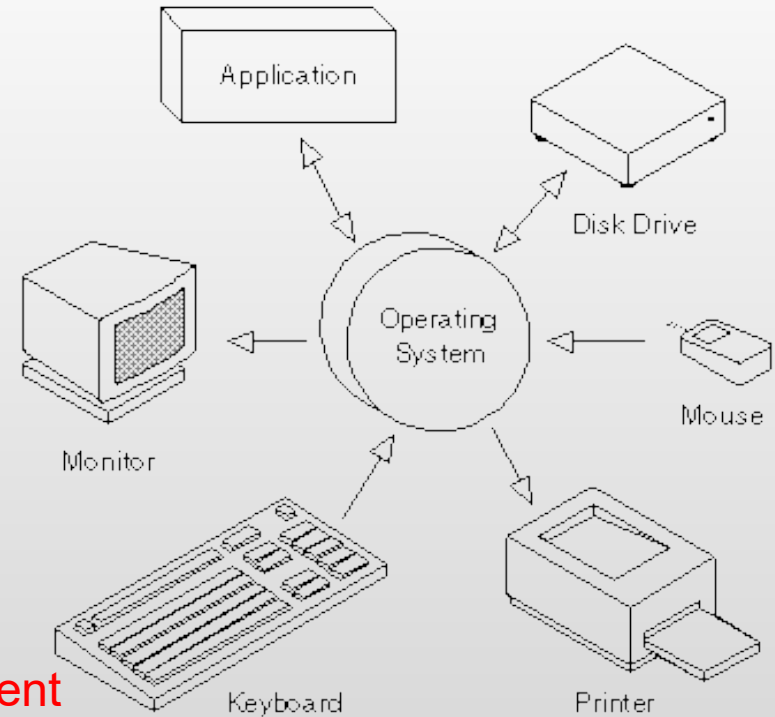
- User interface
- Program Execution
- I/O operations
- File-system manipulation
- Communication
- Error detection
- Resource allocation
- Accounting
- Protection and security



# OS Services

- User interface
- Program Execution
- I/O operations
- File-system manipulation
- Communication
- Error detection
- Resource allocation
- Accounting
- Protection and security

ensuring the efficient  
operation of the system itself

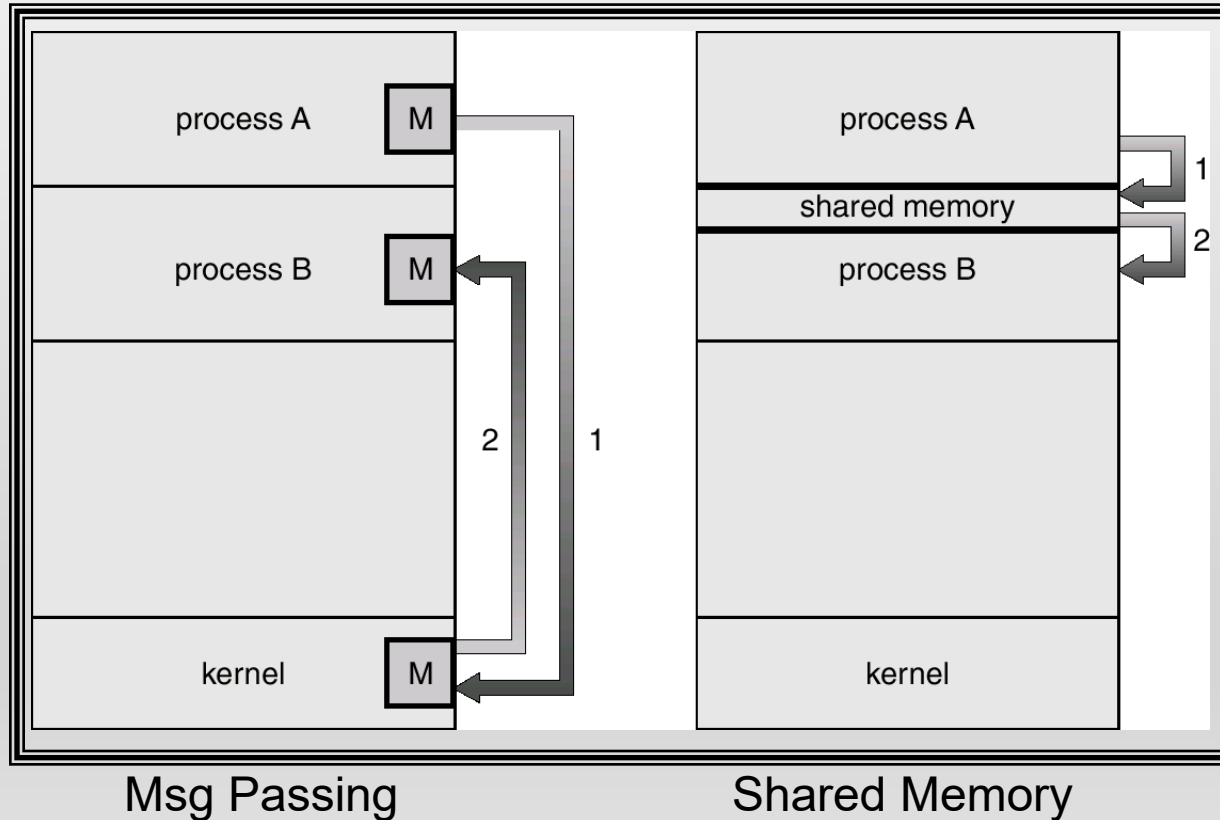


# User Interface

- CLI (Command Line Interface)
  - Fetches a command from user and executes it
  - **Shell: Command-line interpreter** (CSHELL, BASH)
    - Adjusted according to user behavior and preference
- GUI (Graphic User Interface)
  - Usually mouse, keyboard, and monitor
  - **Icons** represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions
- Most systems have both **CLI** and **GUI**

# Communication Models

- Communication may take place using either **message passing** or **shared memory**.





# Applications-OS Interface

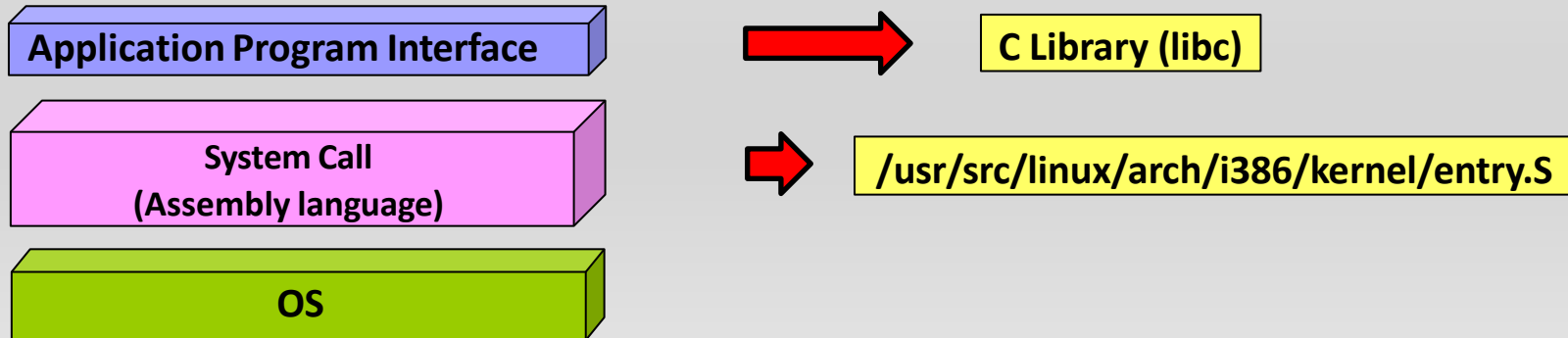
- System calls
- API

# System Calls

- Request OS services
  - **Process control**—abort, create, terminate process allocate/free memory
  - **File management**—create, delete, open, close file
  - **Device management**—read, write, reposition device
  - **Information maintenance**—get time or date
  - **Communications**—send receive message

# System Calls & API

- System calls
  - The **OS interface** to a running program
  - An explicit request to the **kernel** made via a **software interrupt**
  - Generally available as **assembly-language** instructions
- API: Application Program Interface
  - **Users mostly program against API instead of system call**
  - Commonly implemented by language libraries, e.g., **C Library**
  - An API call could involve **zero or multiple system call**
    - Both malloc() and free() use system call brk()
    - Math API functions, such as abs(), don't need to involve system call



# Interface vs. Library

- User program:

```
printf("%d", exp2(int x, int y));
```

- Interface:

```
int exp2(int x, int y);
```

i.e. return the value of  $X \cdot 2^y$

- Library:

```
Imp1:      int exp2(int x, int y) { for (int i=0; i<y; i++) x=x*2; return x;}
```

```
Imp2:      int exp2(int x, int y) { x = x << y; return x;}
```

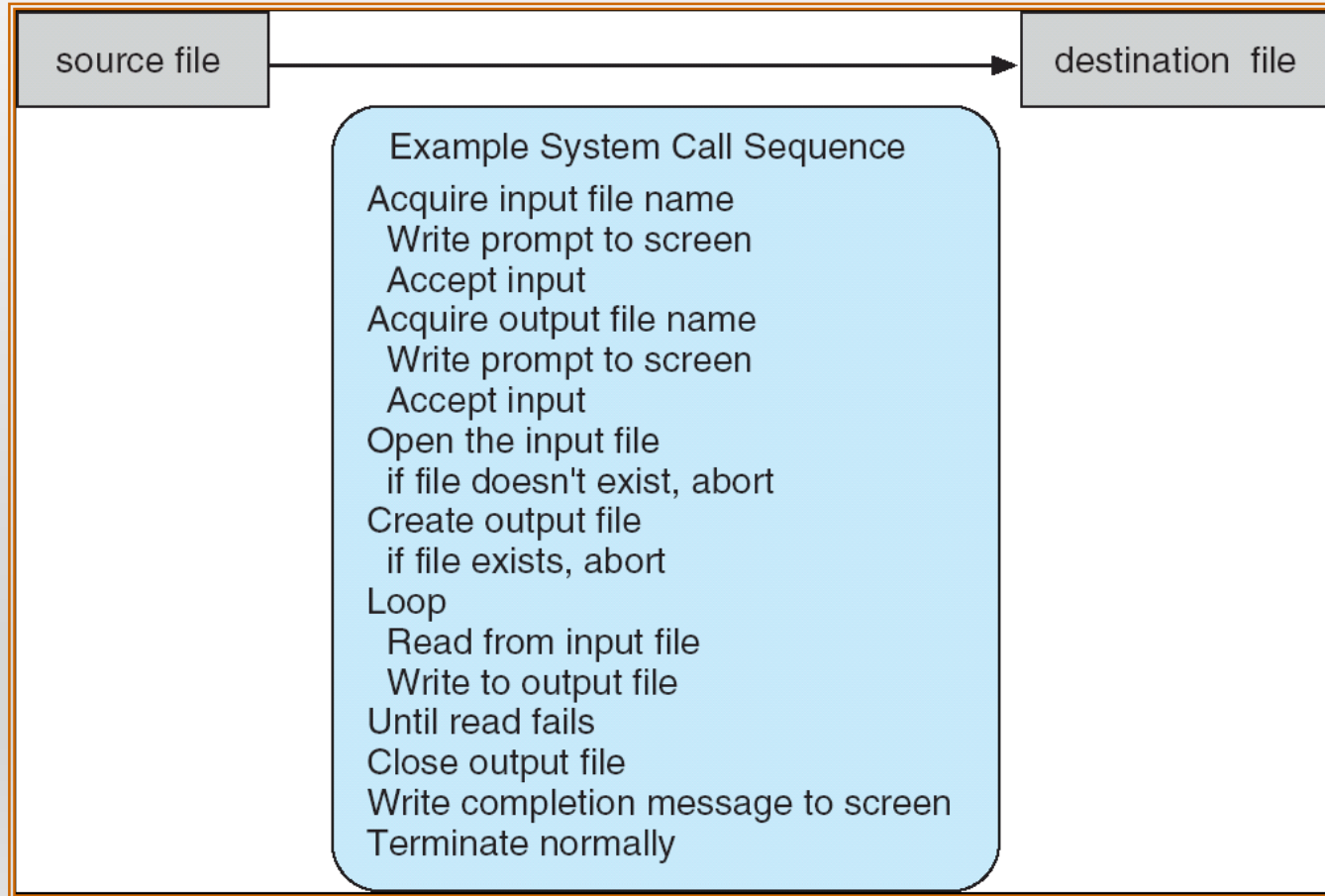
```
Imp3:      int exp2(int x, int y) { return HW_EXP(x,y);}
```

# API: Application Program Interface

- Three most common APIs:
  - **Win32 API** for **Windows**
    - [http://en.wikipedia.org/wiki/Windows\\_API](http://en.wikipedia.org/wiki/Windows_API)  
<http://msdn.microsoft.com/en-us/library/windows/desktop/ff818516%28v=vs.85%29.aspx>
  - **POSIX API** for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X)
    - POSIX -> “**P**ortable **O**perating **S**ystem **I**nterface for Unix”  
<http://en.wikipedia.org/wiki/POSIX>  
[http://www.unix.org/version4/GS5\\_APIs.pdf](http://www.unix.org/version4/GS5_APIs.pdf)
  - **Java API** for the Java virtual machine (JVM)

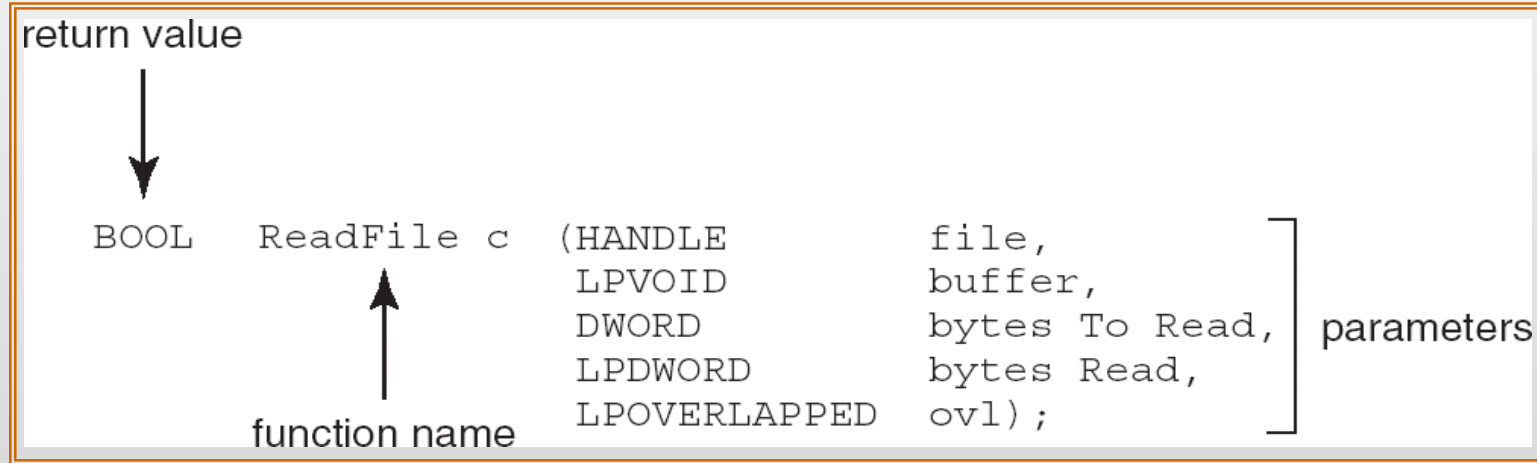
# An Example of System Calls

- System call sequence to copy the contents of one file to another file



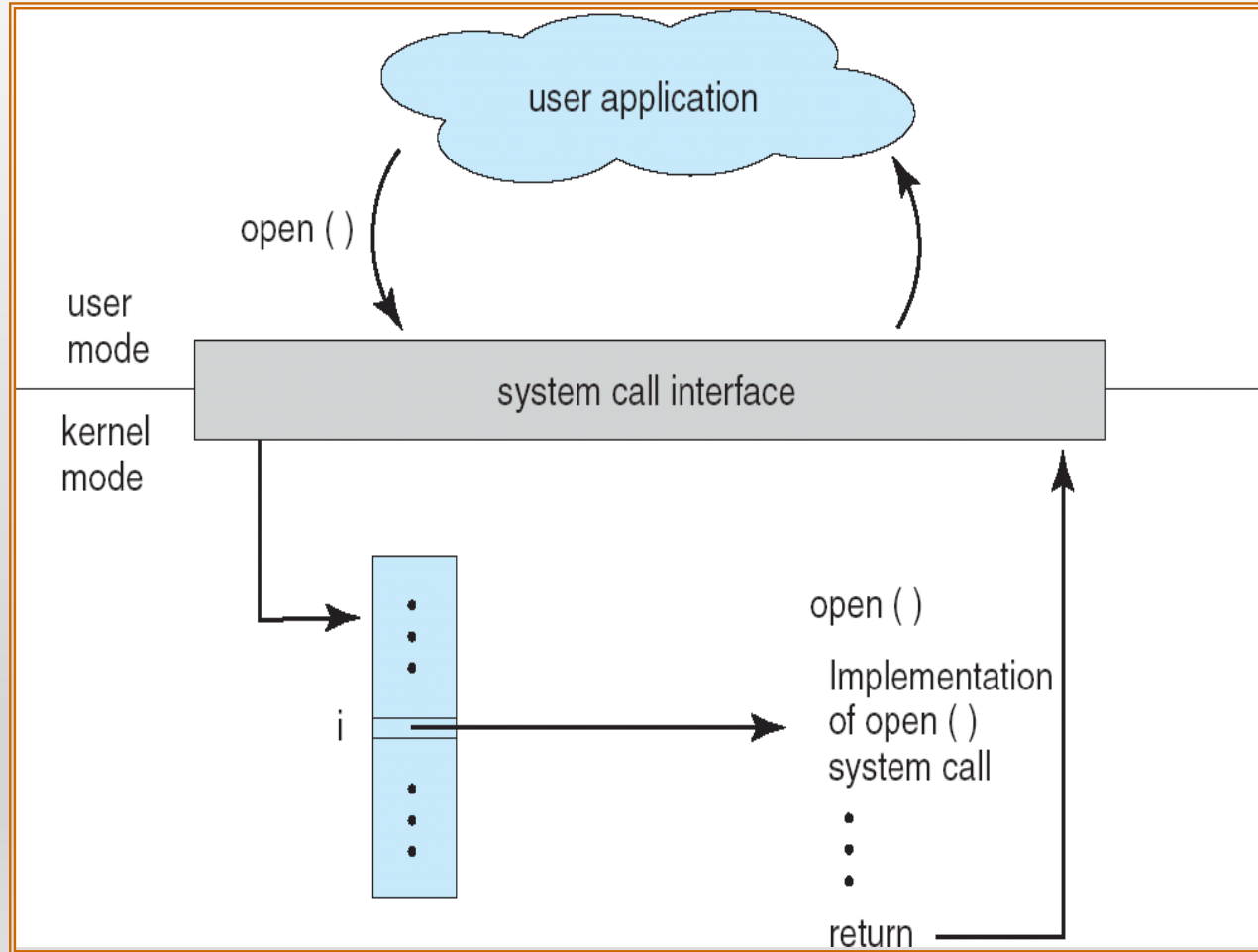
# An Example of Standard API

- Consider the ReadFile() function in the Win32 API—a function for reading from a file



- A description of the parameters passed to `ReadFile()`
  - `HANDLE file`—the file to be read
  - `LPVOID buffer`—a buffer where the data will be read into and written from
  - `DWORD bytesToRead`—the number of bytes to be read into the buffer
  - `LPDWORD bytesRead`—the number of bytes read during the last read
  - `LPOVERLAPPED ovl`—indicates if overlapped I/O is being used

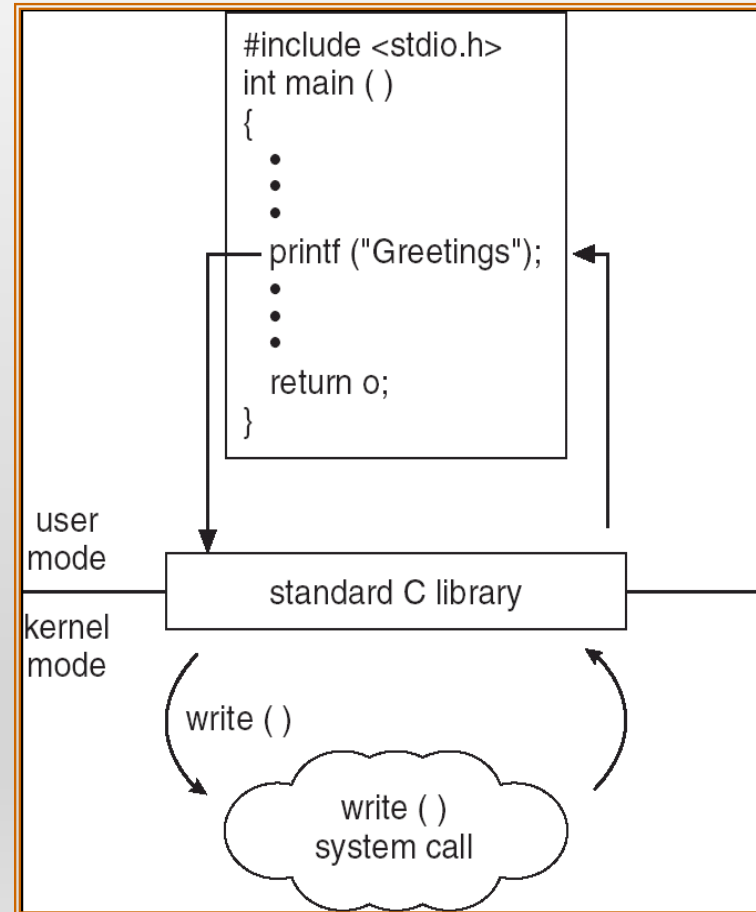
# API - System Call - OS Relationship





# Standard C Library Example

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



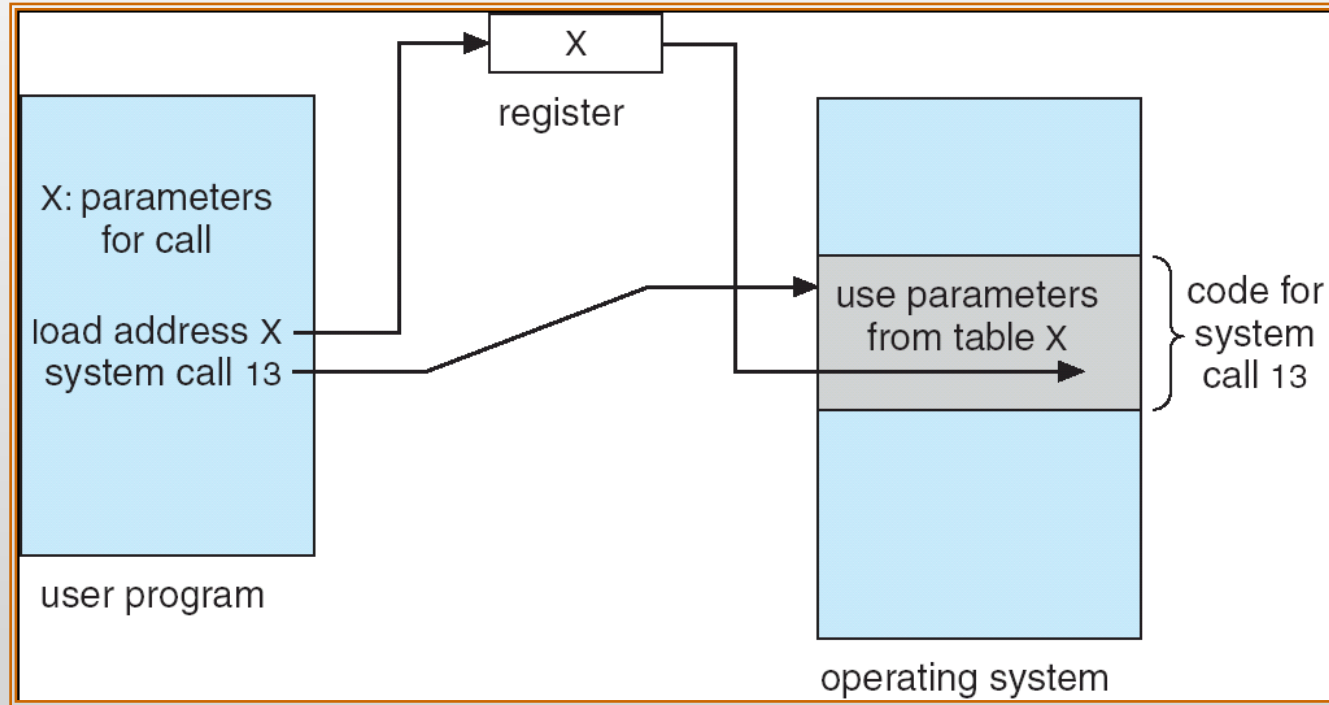
# Why use API?

- Simplicity
  - API is designed for applications
- Portability
  - API is an unified defined interface
- Efficiency
  - Not all functions require OS services or involve kernel

# System Calls: Passing Parameters

- **Three general methods** are used to pass parameters between a running program and the operating system.
  - Pass parameters in **registers**
  - Store the parameters in a **table in memory**, and the table address is passed as a parameter in a register
  - Push (store) the parameters onto the **stack** by the program, and pop off the stack by the operating system

# Parameter Passing via Table



# Review Slides (1)

- What are the two communication models provided by OS?
- What is the relationship between system calls, API and C library?
- Why use API rather than system calls?

# System Structure:

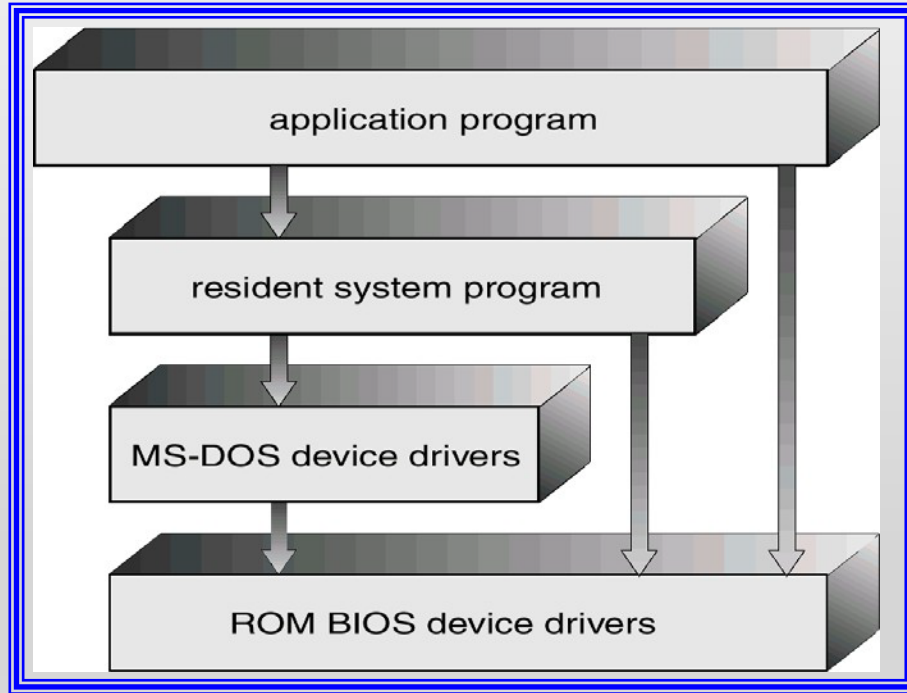
- Simple OS Architecture
- Layer OS Architecture
- Microkernel OS
- Modular OS Structure
- Virtual Machine
- Java Virtual Machine

# User goals and System goals

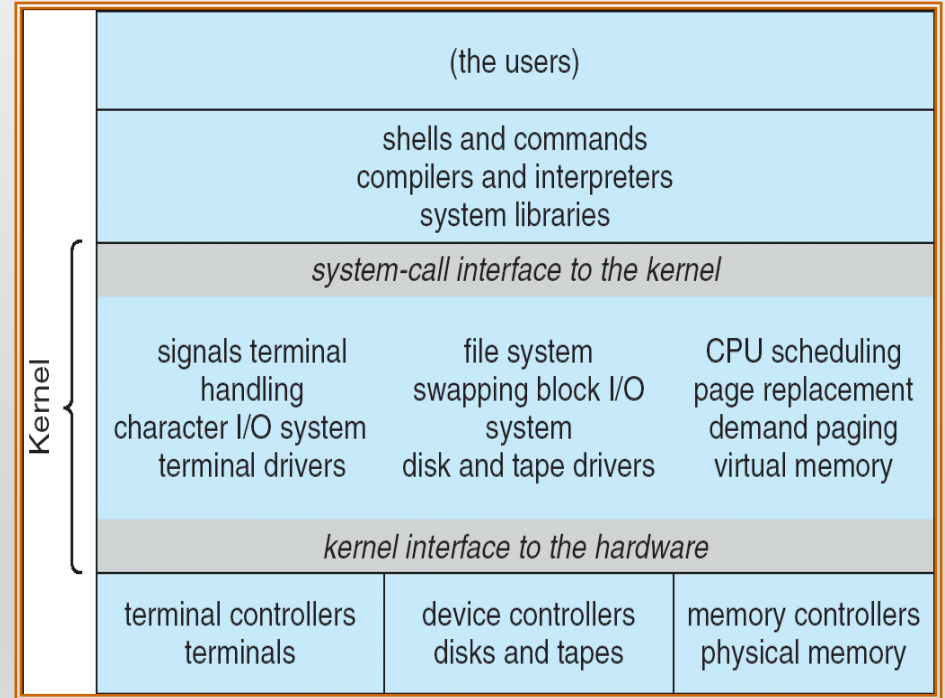
- User goals - operating system should be **easy to use** and **learn**, as well as **reliable, safe, and fast**
- System goals - operating system should be **easy to design, implement,** and **maintain**, as well as **reliable, error-free,** and **efficient**

# Simple OS Architecture

- Only one or two levels of code
- Drawbacks: **Un-safe, difficult to enhance**



MS-DOS

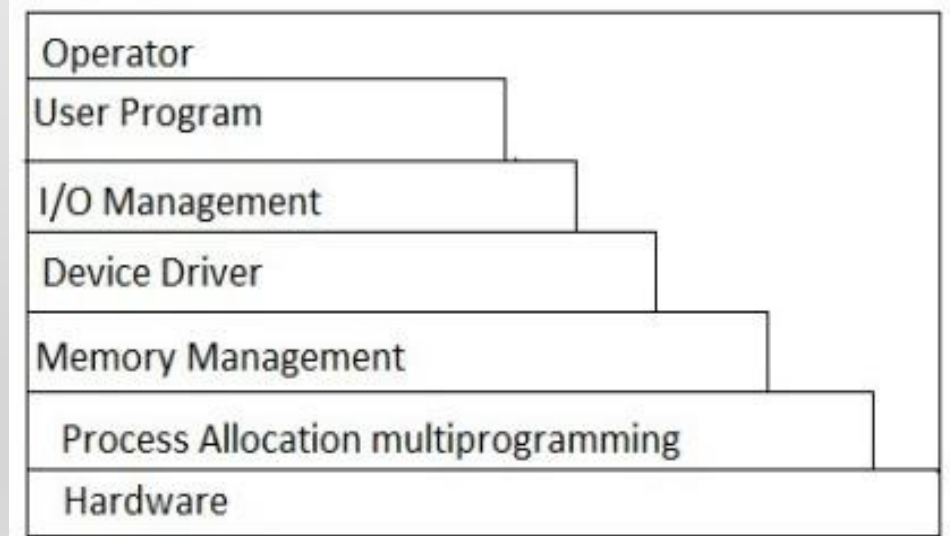
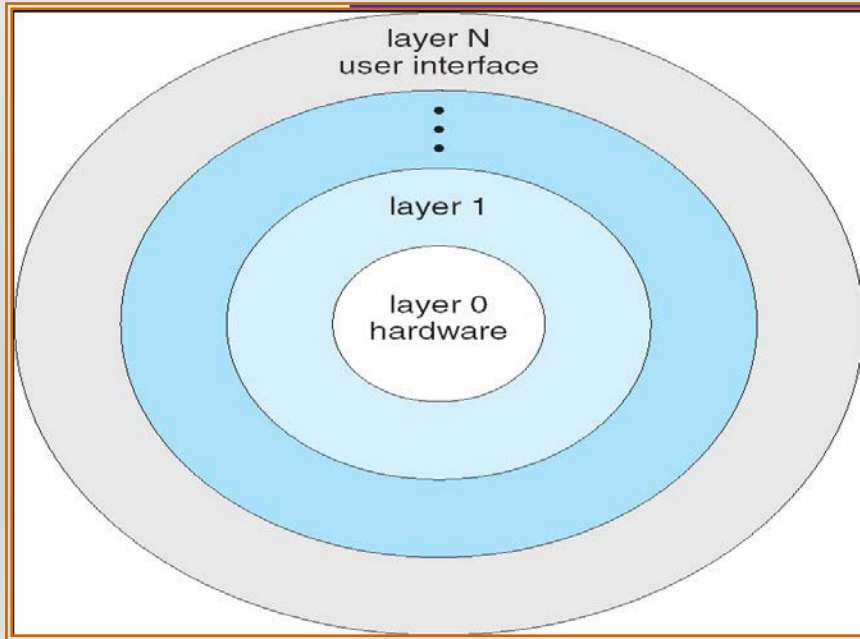


UNIX



# Layered OS Architecture

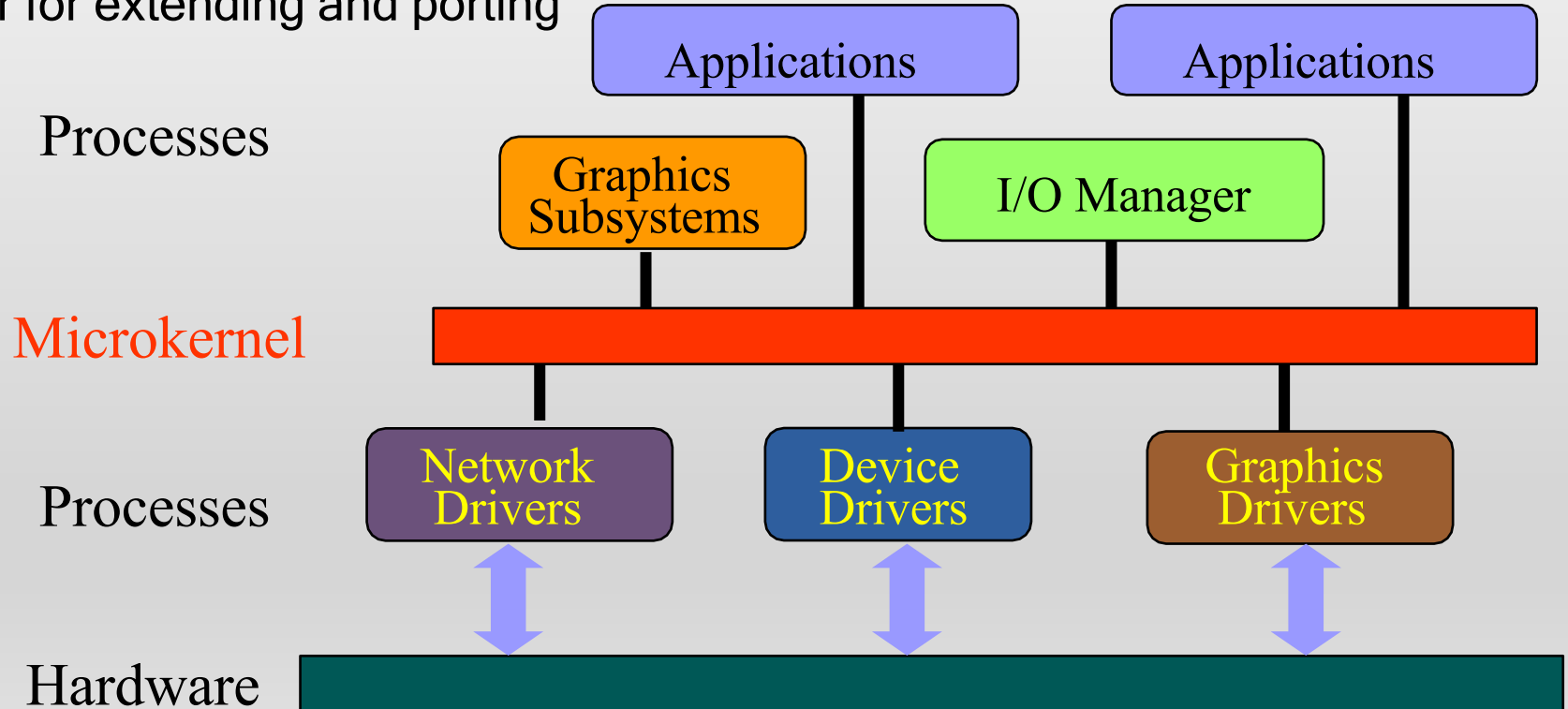
- Lower levels independent of upper levels
  - Nth layer can only access services provided by 0~(N-1)th layer
- Pros: Easier debugging/maintenance
- Cons: **Less efficient, difficult to define layers**



**fig:- layered Architecture**

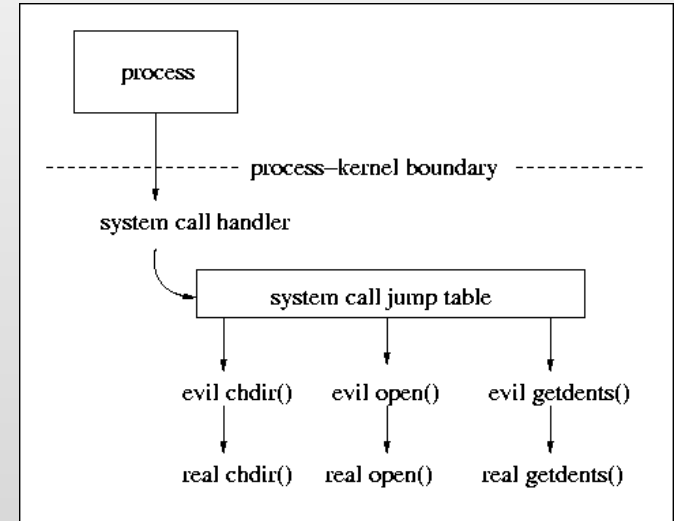
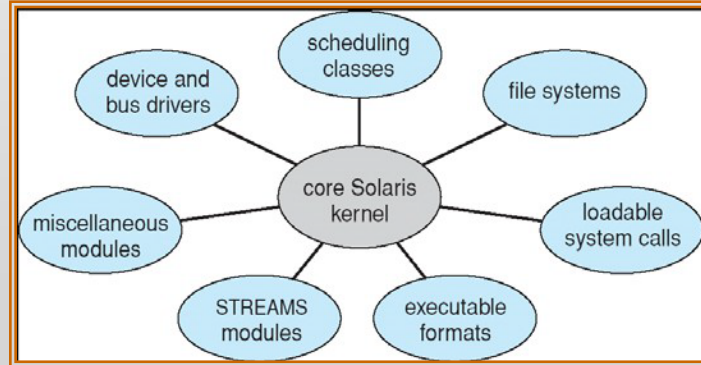
# Microkernel OS

- Moves as much from the kernel into “*user*” space
- Communication is provided by **message passing**
- Easier for extending and porting



# Modular OS Architecture

- Most modern OS 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
- Similar to layers but with more flexible
- E.g., Solaris

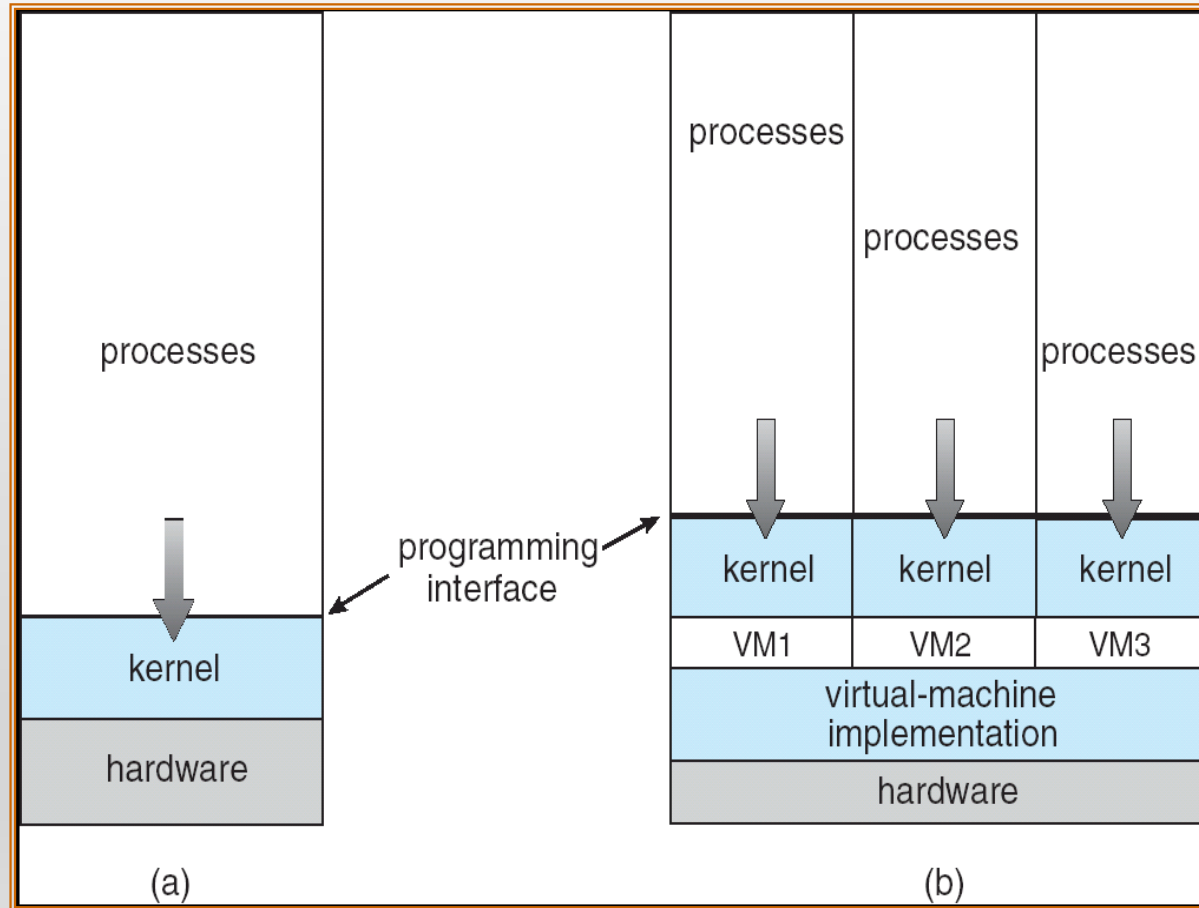


- How to write kernel module
  - [http://www.linuxchix.org/content/courses/kernel\\_hacking/lesson8](http://www.linuxchix.org/content/courses/kernel_hacking/lesson8)
  - [http://en.wikibooks.org/wiki/The\\_Linux\\_Kernel/Modules](http://en.wikibooks.org/wiki/The_Linux_Kernel/Modules)
  - [https://www.thc.org/papers/LKM\\_HACKING.html](https://www.thc.org/papers/LKM_HACKING.html)

# Virtual Machine

- A virtual machine takes the **layered** approach to its logical conclusion
  - It treats hardware and the **operating system** kernel as though they were all hardware
- A virtual machine provides an interface **identical to the underlying bare hardware**
  - Each process is provided with a (virtual) copy of the underlying computer
- Difficult to achieve due to “***critical instruction***”

# Virtual Machine

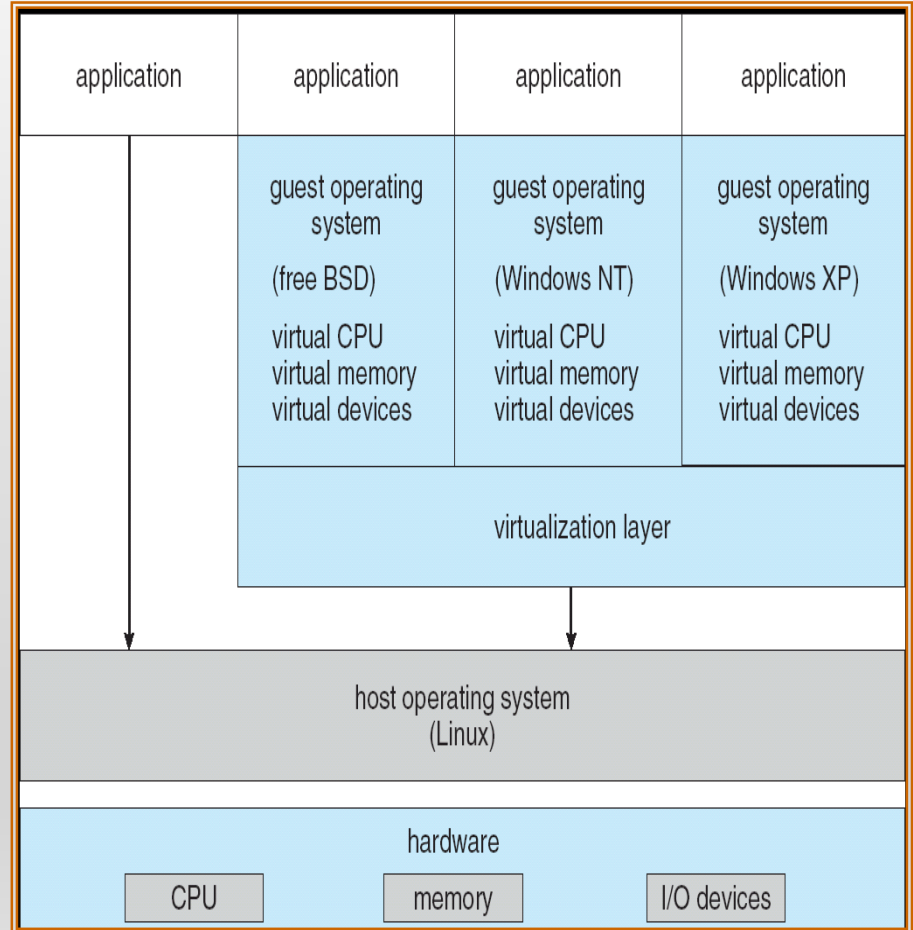


# Usage of Virtual Machine

- provides complete protection of system resources
- a means to solve system compatibility problems
- a perfect vehicle for operating-systems research and development
- A mean to increase resources utilization in **cloud computing**

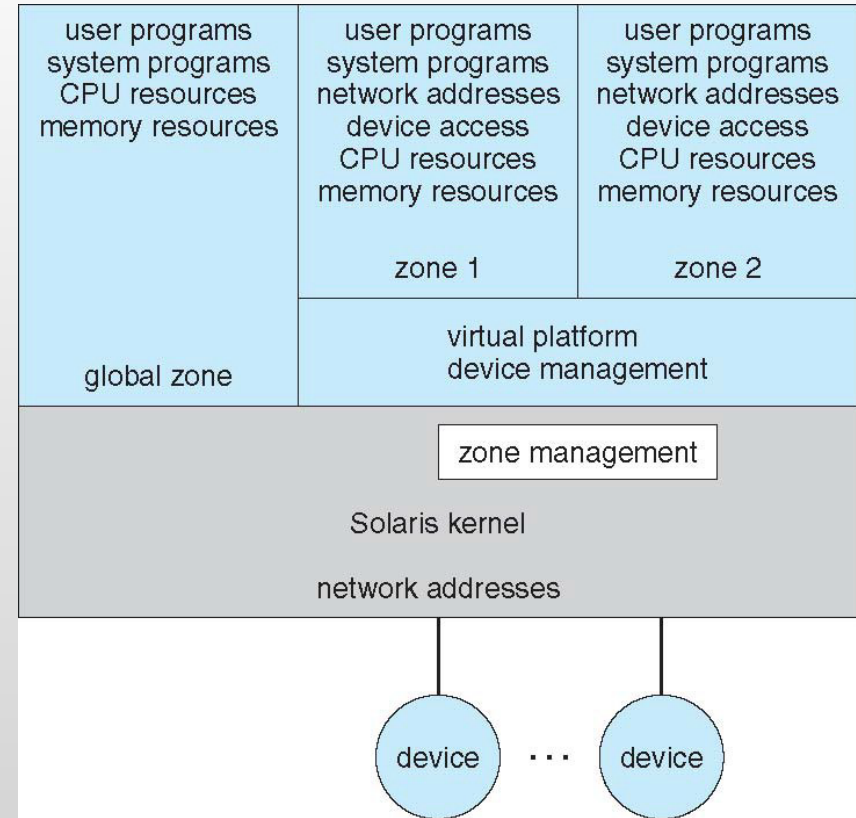
# Vmware (Full Virtualization)

- Run in **user** mode as an application on top of OS
- Virtual machine believe they are running on bare hardware but **in fact are running inside a user-level application**



# Para-virtualization: Xen

- Presents guest with system **similar but not identical** to the guest's preferred systems (**Guest must be modified**)
- **Hardware** rather than **OS and its devices are virtualized** (Only one kernel installed)
- Within a **container (zone)** processes thought they are the only processes on the system

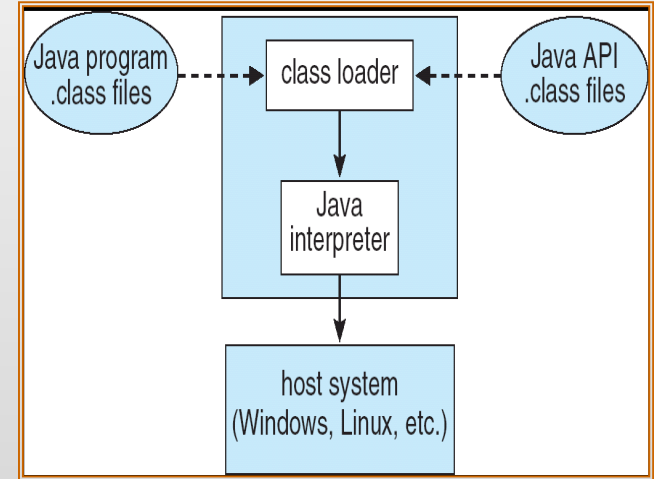


Solaris 10: creates a virtual layer between OS and the applications



# Java Virtual Machine

- Compiled Java programs are **platform-neutral bytecodes** executed by a **Java Virtual Machine (JVM)**
- JVM consists of
  - class loader
  - class verifier
  - runtime interpreter
- **Just-In-Time (JIT)** compilers increase performance



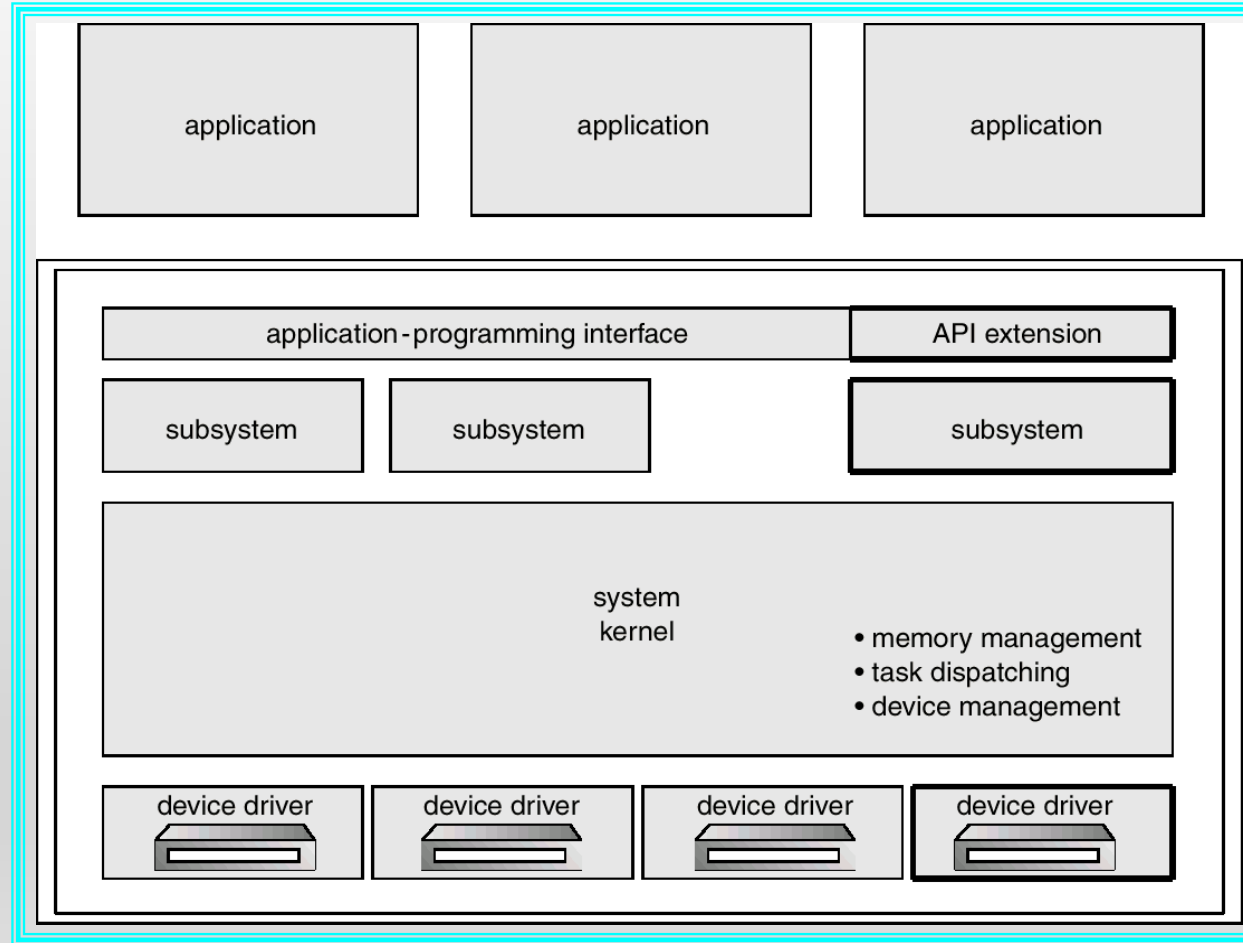
## Review Slides (2)

- What is the difference between the layer approach, the modular approach and microkernel?
- What are the advantages of using virtual machine?

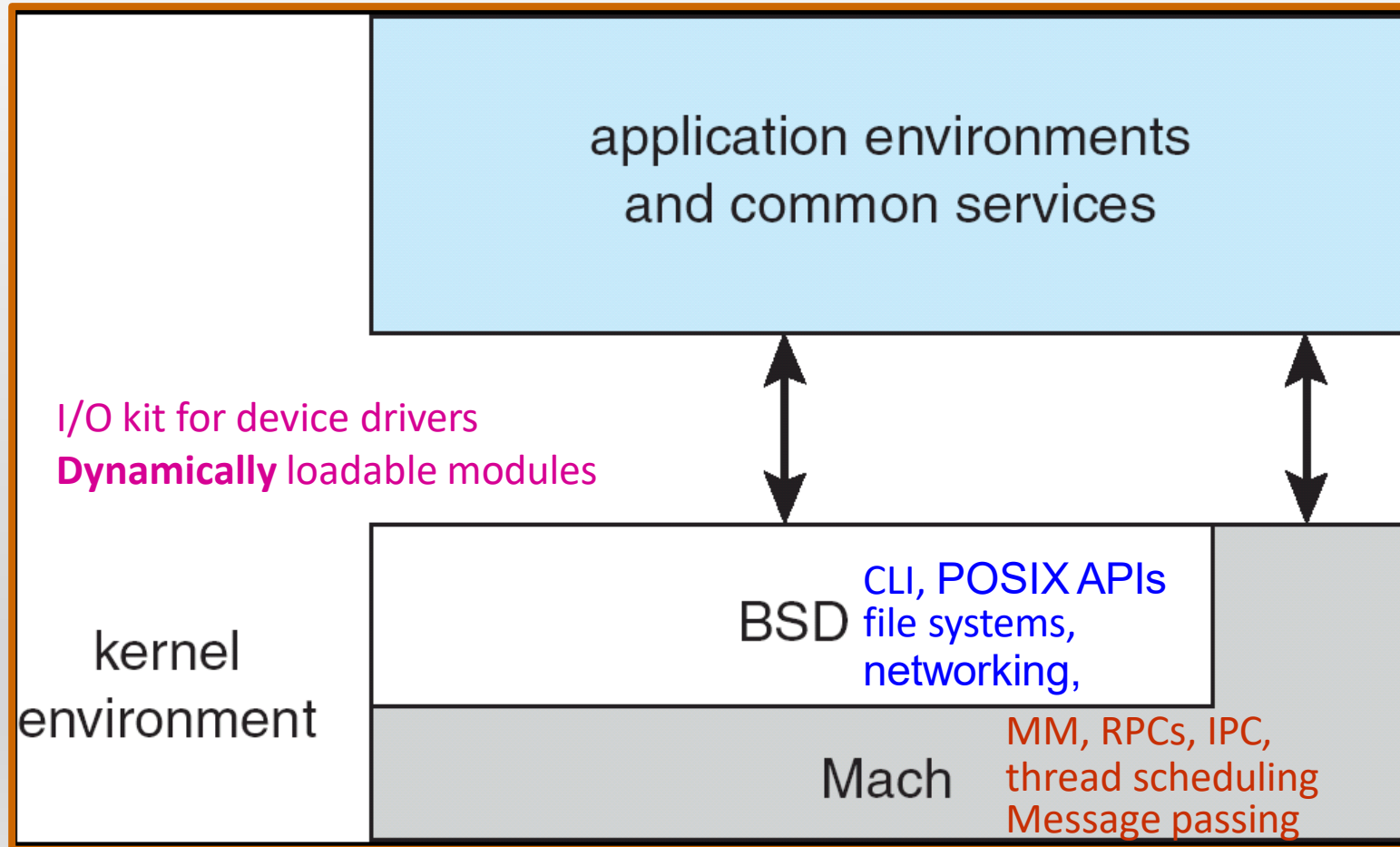
# Reading Material & HW

- Chap 2
- HW (Problem set)
  - 2.7, 2.10, 2.13
- Reference
  - *Understanding Full Virtualization, Paravirtualization, and Hardware Assist*
  - *[www.vmware.com/files/pdf/VMware\\_paravirtualization.pdf](http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf)*
  - *APIs, POSIX and the C Library*
  - *[http://book.chinaunix.net/special/ebook/Linux\\_Kernel\\_Development/0672327201/ch05lev1sec1.html](http://book.chinaunix.net/special/ebook/Linux_Kernel_Development/0672327201/ch05lev1sec1.html)*

# OS/2



# Mac OS X Structure hybrid structured



# Simulation

- Simulation: the host system has one system architecture and the guest system was **compiled for a different architecture**
- The programs (such as important programs that were compiled for the old system) could be run in an **emulator** that **translates** each of the outdated **system's instructions** into the current instruction set. (disadv.: 10 times slow usually)

*Q & A*

*Thank you for your attention*