


# Contents

1. Introduction
-  2. System Structures
3. Process Concept
4. Multithreaded Programming
5. Process Scheduling
6. Synchronization
7. Deadlocks
8. Memory-Management Strategies
9. Virtual-Memory Management
10. File System
11. Mass-Storage Structures
12. I/O Systems
13. Protection, Security, Distributed Systems 1

# Chapter 2

## System Structures

# Operating-System Structures

- Goals: Provide a way to understand an operating systems
  - Services
  - Interface
  - System Components
- The type of system desired is the basis for choices among various algorithms and strategies!

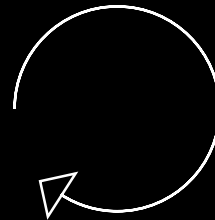
# Operation-System Services

- Goal:
  - Provide an environment for the execution of programs.
  - Services are provided to programs and their users.
- User Interface (UI)
  - Command Line Interface, Batch Interface, Graphical User Interface (GUI), etc.
  - Interface between the user and the operating system

# Operation-System Services

- Friendly UI's
  - Command-line-based interfaces or menu-based window-and-menu interface
- e.g., UNIX shell and command.com in MS-DOS

User-friendly?



Get the next command  
Execute the command

- Program Execution
  - Loading, running, terminating, etc

# Operation-System Services

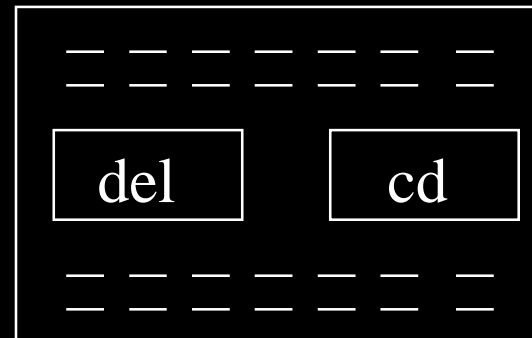
- I/O Operations
  - General/special operations for devices:
    - Efficiency & protection
- File-System Manipulation
  - Read, write, create, delete, etc.
  - Files and Directories
  - Permission Management
- Communications
  - Intra-processor or inter-processor communication – shared memory or message passing

# Operation-System Services

- Error Detection
    - Possible errors from CPU, memory, devices, user programs → Ensure correct & consistent computing
  - *Resource Allocation*
    - *Utilization & efficiency*
  - *Accounting*
    - *Statistics or Accounting*
  - *Protection & Security*
- user convenience or *system efficiency!*

# User OS Interface – Command Interpreter

- Two approaches:
  - Contain codes to execute commands
    - Fast but the interpreter tends to be big!
    - Painful in revision!





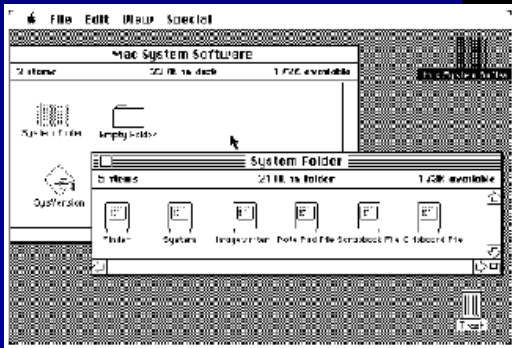
# User OS Interface – Command Interpreter

- Implement commands as system programs → Search exec files which corresponds to commands (UNIX)
  - Issues
    - a. Parameter Passing
      - Potential Hazard: virtual memory
    - b. Being Slow
    - c. Inconsistent Interpretation of Parameters

現在大多數  
command在  
command  
interpreter外面，少  
部分在裡面

# User OS Interface – GUI

- Components
  - Screen, Icons, Folders, Pointer, etc.
- History
  - Xerox PARC research facility (1970's)
  - Mouse – 1968
  - Mac OS – 1980's
  - Windows 1.0 ~ 8



# User OS Interface – GUI

- Unix & Linux
  - Common Desktop Environment (CDE), X-Windows, K Desktop Environment (KDE), GNOME
- Trend
  - Mixture of GUI and command-line interfaces
  - Multimedia, Intelligence, etc.

# System Calls

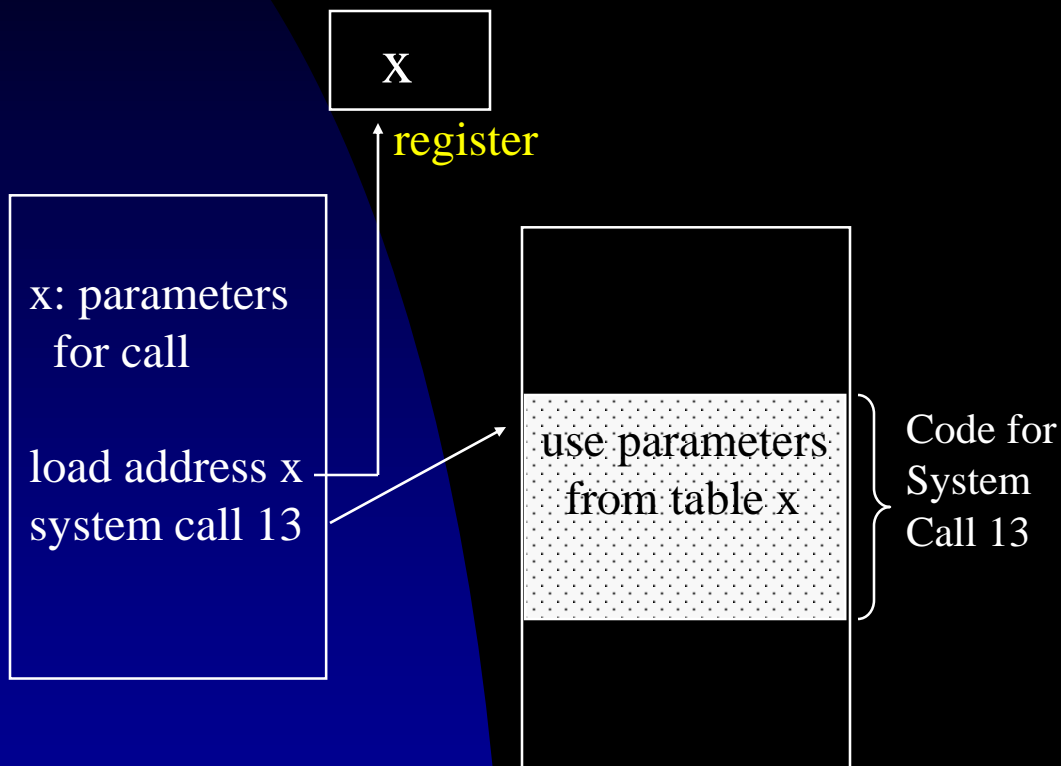
- System calls
  - Interface between processes & OS
- How to make system calls?
  - Assembly-language instructions or subroutine/functions calls in high-level language such as C or Perl?
    - Generation of in-line instructions or a call to a special run-time routine.
- Example: read and copy of a file!
  - Library Calls vs System Calls

# System Calls

提供最小的service set

- Application Programming Interface (API)
  - Examples: Win 32 API for Windows, Unix like POSIX API for POSIX-based Systems, Java API for Java virtual machines
  - Benefits (API vs System Calls)
    - Portability
    - Ease of Use & Better Functionality

# System Calls



- How a system call occurs?
  - Types and information
- Parameter Passing
  - Registers
  - Registers pointing to blocks
    - Linux register指向記憶體
  - Stacks

# System Calls

- Process Control
- File Management
- Device Management
- Information Maintenance
- Communications

# System Calls

- Process & Job Control
  - End (normal exit) or abort (abnormal)
    - Error level or no
    - Interactive, batch, GUI-supported systems
  - Load and execute
    - How to return control?
      - e.g., shell load & execute commands
  - Creation and/or termination of processes
    - Multiprogramming?

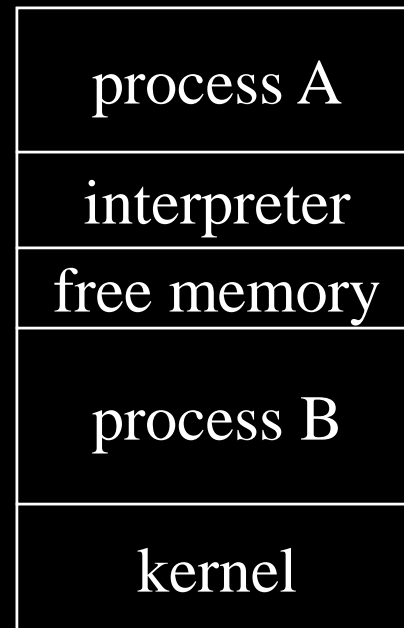
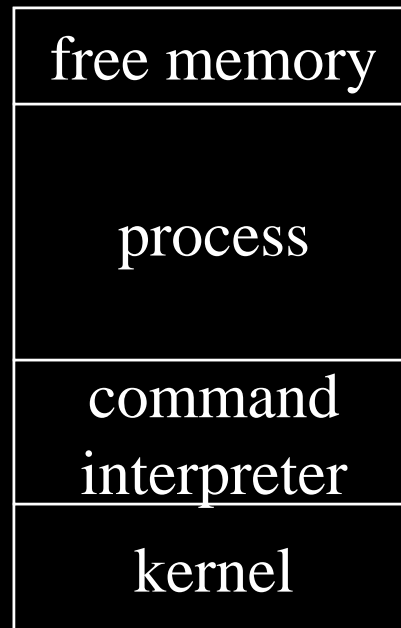


# System Calls

- Process & Job Control (continued)
  - Process Control
    - Get or set attributes of processes
  - Wait for a specified amount of time or an event
    - Signal event
  - Memory dumping, profiling, tracing, memory allocation & de-allocation

# System Calls

- Examples: MS-DOS & UNIX



# System Calls

- File Management
  - Create and delete
  - Open and close
  - Read, write, and reposition (e.g., rewinding)
    - lseek
  - Get or set attributes of files
  - Operations for directories

# System Calls

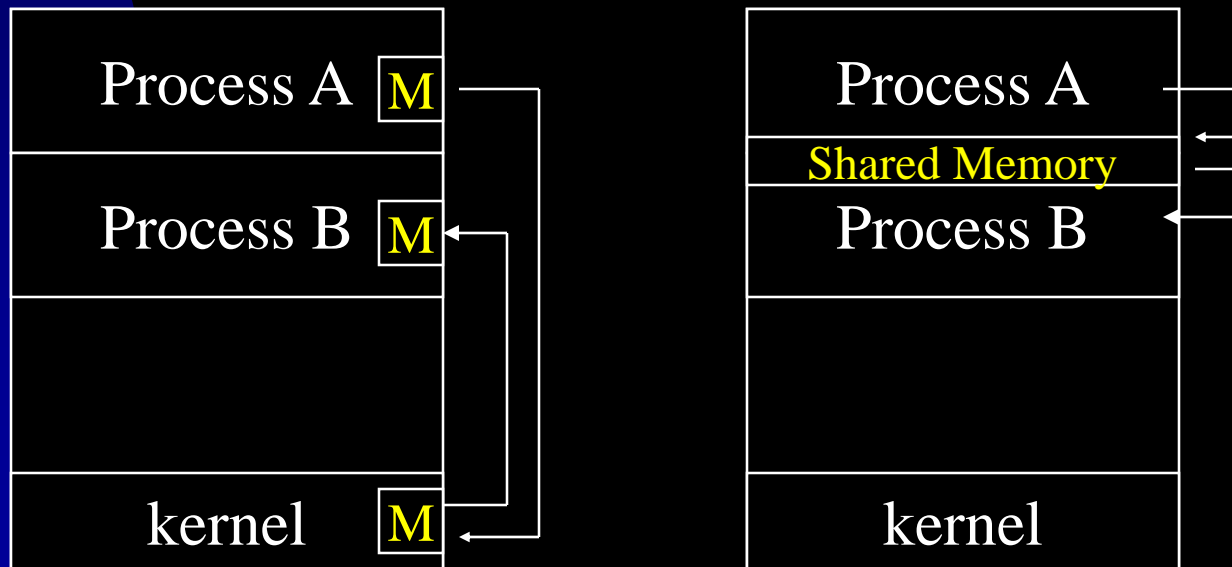
- Device management
  - Physical or virtual devices, e.g., files.
  - Request or release
    - Open and close of special files
    - Files are abstract or virtual devices.
  - Read, write, and reposition (e.g., rewinding)
  - Get or set file attributes
  - Logically attach or detach devices

# System Calls

- Information maintenance
  - Get or set date or time
  - Get or set system data, such as the amount of free memory
- Communication socket
  - Message Passing
    - Open, close, accept connections
      - Host ID or process ID
    - Send and receive messages
    - Transfer status information
  - Shared Memory
    - Memory mapping & process synchronization
- Protection

# System Calls

- Shared Memory
  - Max Speed & Comm Convenience
- Message Passing
  - No Access Conflict & Easy Implementation



# System Programs

- Goal:
  - Provide a convenient environment for program development and execution
- Types
  - File Management, e.g., rm.
  - Status information, e.g., date.
  - File Modifications, e.g., editors.
  - Program Loading and Executions, e.g., loader.
  - Programming Language Supports and background services, e.g., compilers.
  - Communications, e.g., telnet.

# System Design & Implementation

- Design Goals & Specifications:
  - User Goals, e.g., ease of use
  - System Goals, e.g., reliable
- Rule 1: Separation of Policy & Mechanism
  - Policy : What will be done?
  - Mechanism : How to do things?
  - Example: timer construct and time slice
- Two extreme cases:

Microkernel-based OS ←··········→ Macintosh OS



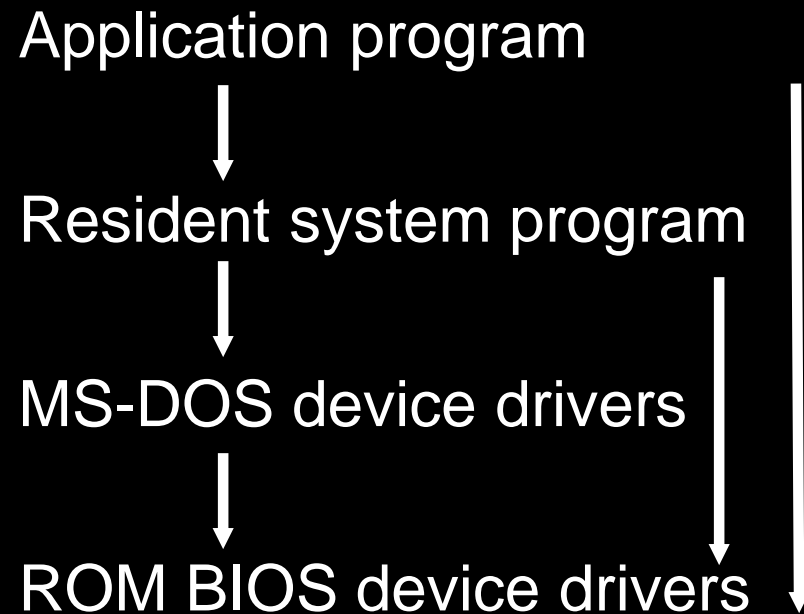
# System Design & Implementation

- OS Implementation in High-Level Languages
  - E.g., UNIX, OS/2, MS NT, etc.
  - Advantages:
    - Being easy to understand & debug
    - Being written fast, more compact, and portable
  - Disadvantages:
    - Less efficient but more storage for code

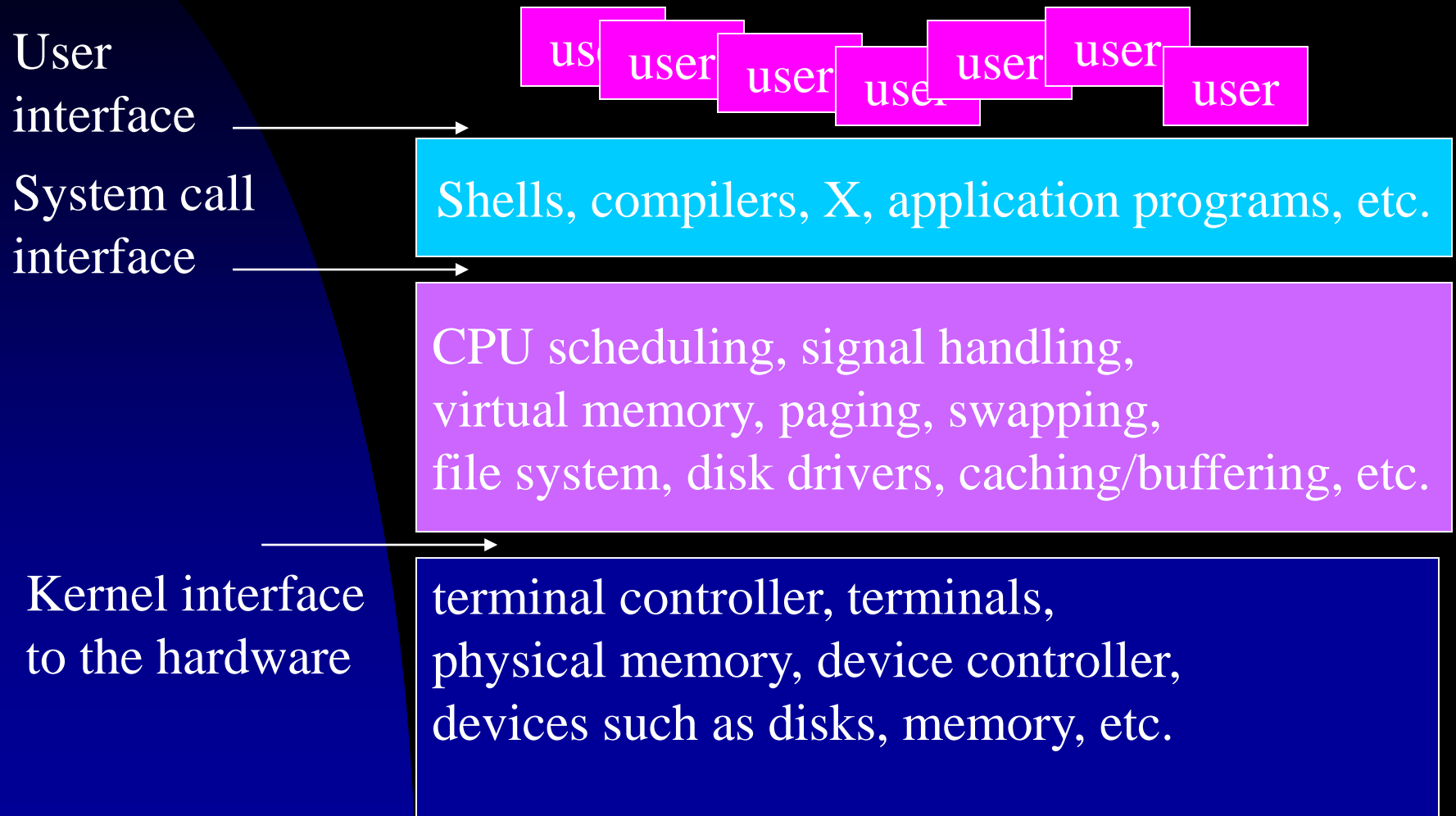
\* Tracing for bottleneck identification, exploring of excellent algorithms, etc.

# OS Structure – MS-DOS

- MS-DOS Layer Structure

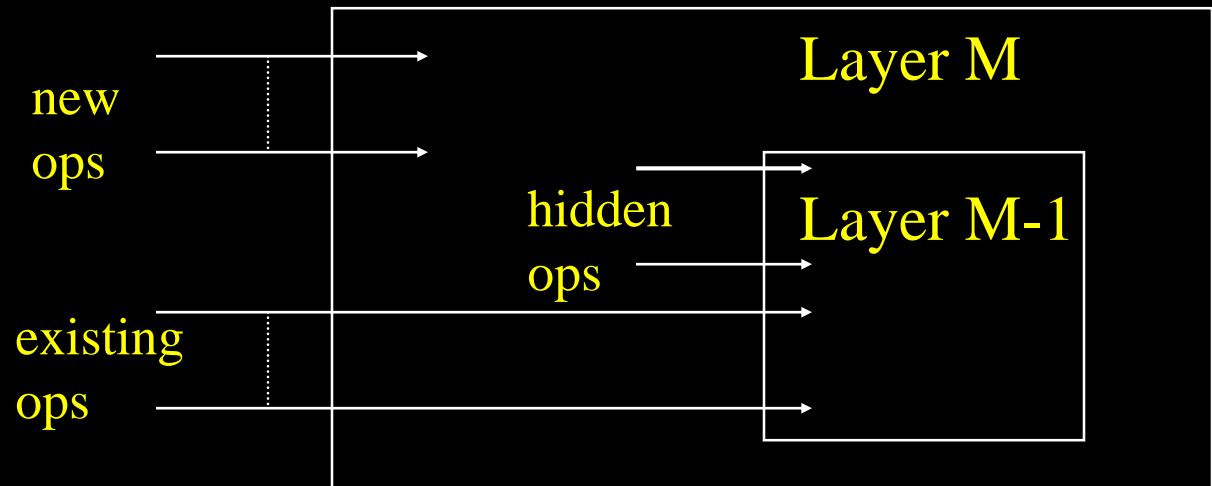


# OS Structure – UNIX



# OS Structure

- A Layered Approach – A Myth



Advantage: Modularity ~ Debugging & Verification

若第M-1層確定正確  
只需檢查M層

Difficulty: Appropriate layer definitions, less efficiency due to overheads !

# OS Structure

- A Layer Definition Example:

L5 User programs

L4 I/O buffering

L3 Operator-console device driver

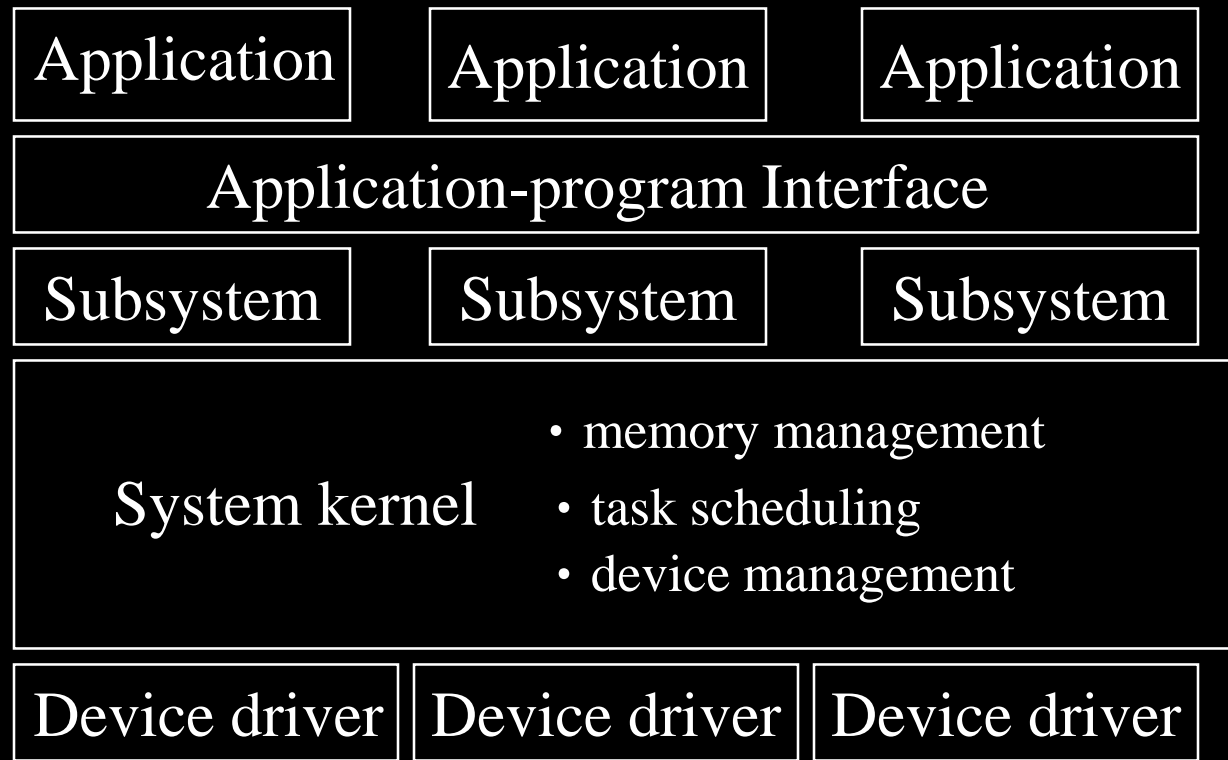
L2 Memory management

L1 CPU scheduling

L0 Hardware

# OS Structure – OS/2

## ■ OS/2 Layer Structure



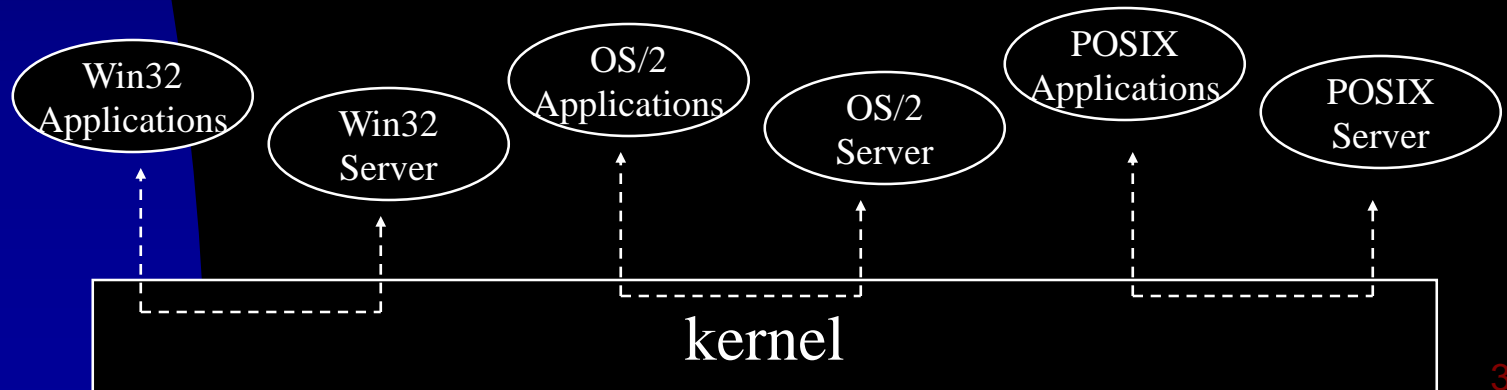
\* Some layers of NT were from user space to kernel space in NT4.0 30

# OS Structure – Microkernels

- The concept of microkernels was proposed in CMU in mid 1980s (Mach).
  - Moving all nonessential components from the kernel to the user or system programs!
  - No consensus on services in kernel
    - Mostly on process and memory management and communication
- Benefits:
  - Ease of OS service extensions → portability, reliability, security  
kernel 小較好維護

# OS Structure – Microkernels

- Examples
  - Microkernels: True64UNIX (Mach kernel), MacOS X (Mach kernel), QNX (msg passing, proc scheduling, HW interrupts, low-level networking)
  - Hybrid structures: Windows NT





# OS Structure – Modules

相較於layer approach,modules之前互相平行可互相調用

- A Modular Kernel
  - A Set of Core Components
  - Dynamic Loadable Modules
    - E.g., Solaris: Scheduling Classes, File Systems, Loadable System Calls, Executable Formats, STREAMS Modules, Miscellaneous, Device and Bus Drivers
  - Characteristics:
    - Layer-Like – Modules
    - Microkernel-Like – the Primary Module

# OS Structure – Hybrid Systems

- Definition: A combination of different structures
- Example 1: Mac OS X
  - Application Environments and Common Services
  - BSD: Command Line Interface, Support for Networking and File Systems, an Implementation of POSIX APIs.
  - Mach: Memory Management, Support for Remote Procedure Calls, Interprocess Communication Facilities
  - The Kernel Environment: I/O Kit for the Development of Device Drivers and Dynamically Loadable Modules.

Aqua Graphical  
User Interface

App. Environ. &  
Common Services

BSD

Mach

Kernel Environment

# OS Structure – Hybrid Systems

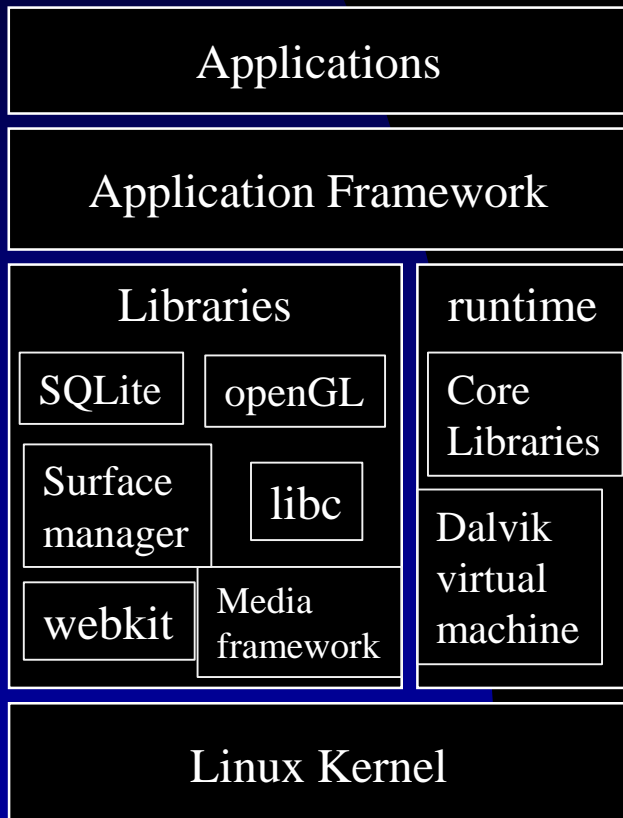
- Example 2: iOS



- It is structured on Mac OSX, where Cocoa Touch, Media Services, and Core Services provide an API to provide frameworks for application development, services for graphics and A/V, and many features, such as databases and cloud computing.

# OS Structure – Hybrid Systems

- Example 3: Android
  - Designed by the Open Handset Alliance and primarily led by Google.
  - Android API is developed for Java program development to run on Dalvik.

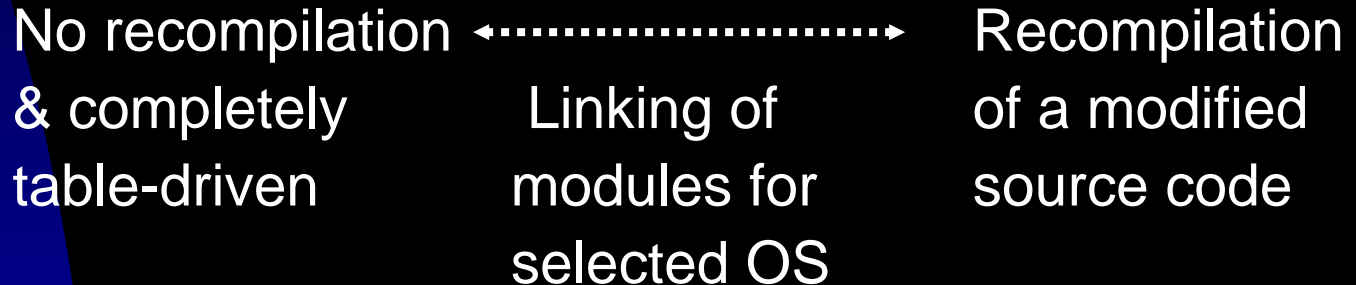


# Operating System Debugging

- Debugging
  - An activity in finding and fixing errors or bugs, including performance problem, that exist in hardware or software.
- Terminologies
  - Performance Tuning – A Procedure that Seeks to Improve Performance by Removing Bottlenecks.
  - Core Dump – A Capture of the Memory of a Process or OS
  - Crash – A Kernel Failure

# System Generation

- SYSGEN (System Generation)
  - Ask and probe for information concerning the specific configuration of a hardware system
    - CPU, memory, device, OS options, etc.




- Issues
  - Size, Generality, Ease of Modification

# System Boot

- Booting
  - The procedure of starting a computer by loading the kernel.
  - The bootstrap program or the bootstrap loader
    - Firmware being ROM or EEPROM resident
    - Boot/system disk with a boot block

# Contents

1. Introduction
2. System Structures
-  3. Process Concept
4. Multithreaded Programming
5. Process Scheduling
6. Synchronization
7. Deadlocks
8. Memory-Management Strategies
9. Virtual-Memory Management
10. File System
11. Mass-Storage Structures
12. I/O Systems
13. Protection, Security, Distributed Systems 40



# Chapter 3

## Process Concept

# Q & A

# Projects – Nachos 4.0

- Not Another Completely Heuristic Operating System
- Written by Tom Anderson and his students at UC Berkeley  
<http://www.cs.washington.edu/homes/tom/nachos/>
- It simulates an MIPS architecture on host systems  
(Unix/Linux/Windows/MacOS X)
- User programs need a cross-compiler (target MIPS)
- Nachos appears as a single threaded process to the host operating system.

# Multimedia Systems

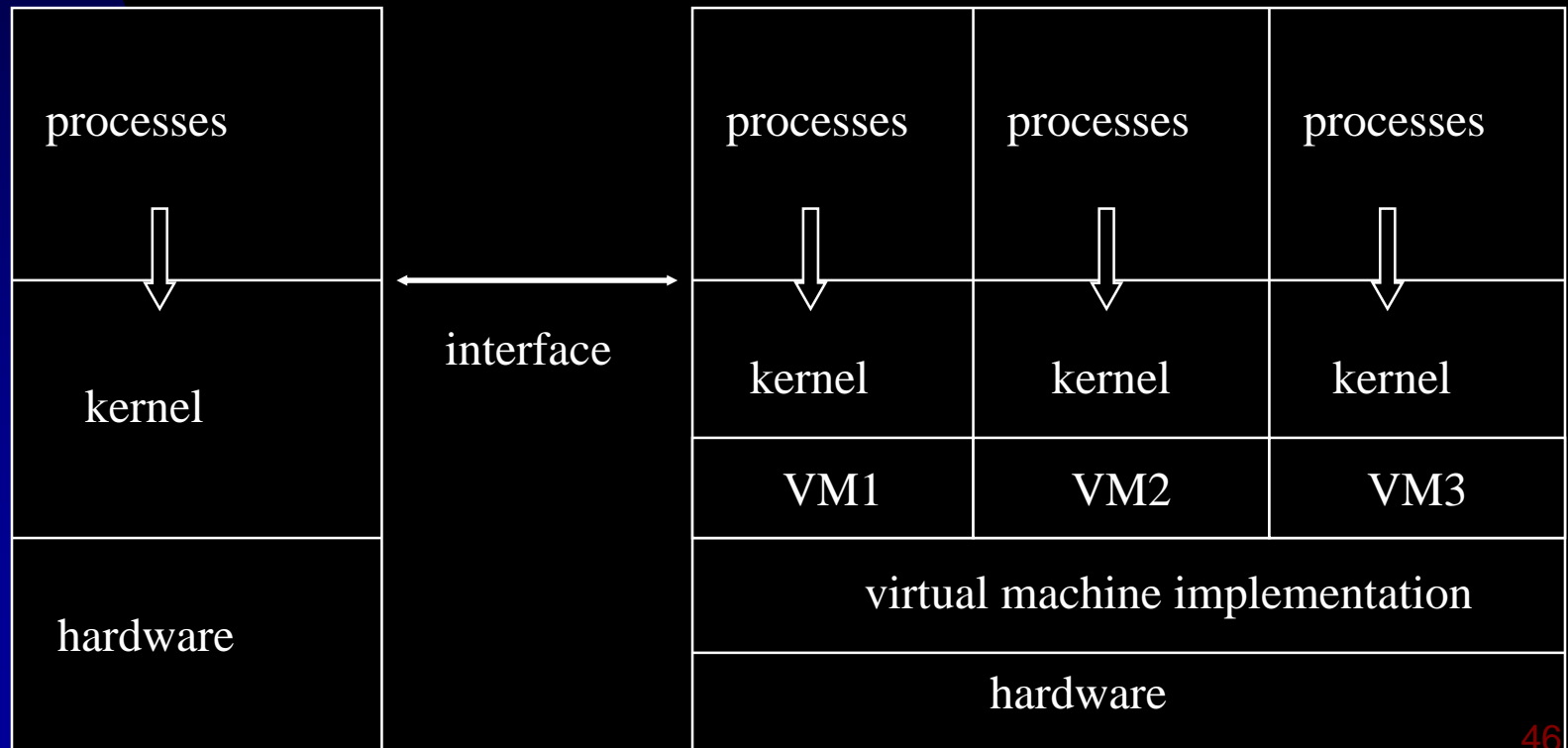
- Multimedia Data
  - Audio and video files and conventional files
  - Multimedia data must be delivered according to certain time restrictions (e.g., 30 frames per second)
- Variety on Platforms
  - Desktop personal computers, Personal Digital Assistant (PDA), cellular telephones, etc.

# Handheld Systems

- Handheld Systems
  - E.g., Personal Digital Assistant (PDA) and cellular phones.
- New Challenges – convenience vs portability
  - Limited Size and Weight
  - Small Memory Size (e.g., 512KB ~ 128MB)
    - No Virtual Memory
  - Slow Processor
    - Battery Power
  - Small Display Screen
    - Web-clipping

# Virtual Machine

- Virtual Machines: provide an interface that is identical to the underlying bare hardware

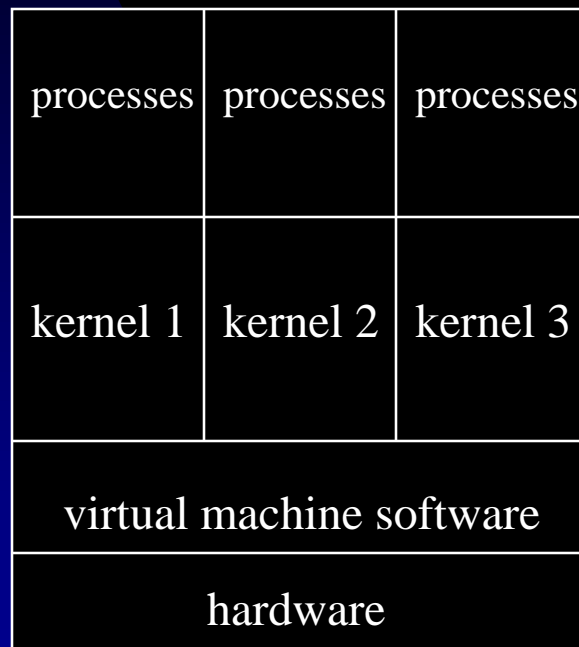


# Virtual Machine

- Implementation Issues:
  - Emulation of Physical Devices
    - E.g., Disk Systems
      - An IBM minidisk approach
  - User/Monitor Modes
    - (Physical) Monitor Mode
      - Virtual machine software
    - (Physical) User Mode
      - Virtual monitor mode & Virtual user mode

# Virtual Machine

virtual  
user  
mode  
virtual  
monitor  
mode  
monitor  
mode



P1/VM1 system call

Trap

Service for the system call

Finish  
service

Restart VM1

Set program counter  
& register contents,  
& then restart VM1

Simulate the  
effect of the I/O  
instruction

time



# Virtual Machine

- Disadvantages:
  - Slow!
    - Execute most instructions directly on the hardware
  - No direct sharing of resources
    - Physical devices and communications

\* I/O could be slow (interpreted) or fast (spooling)

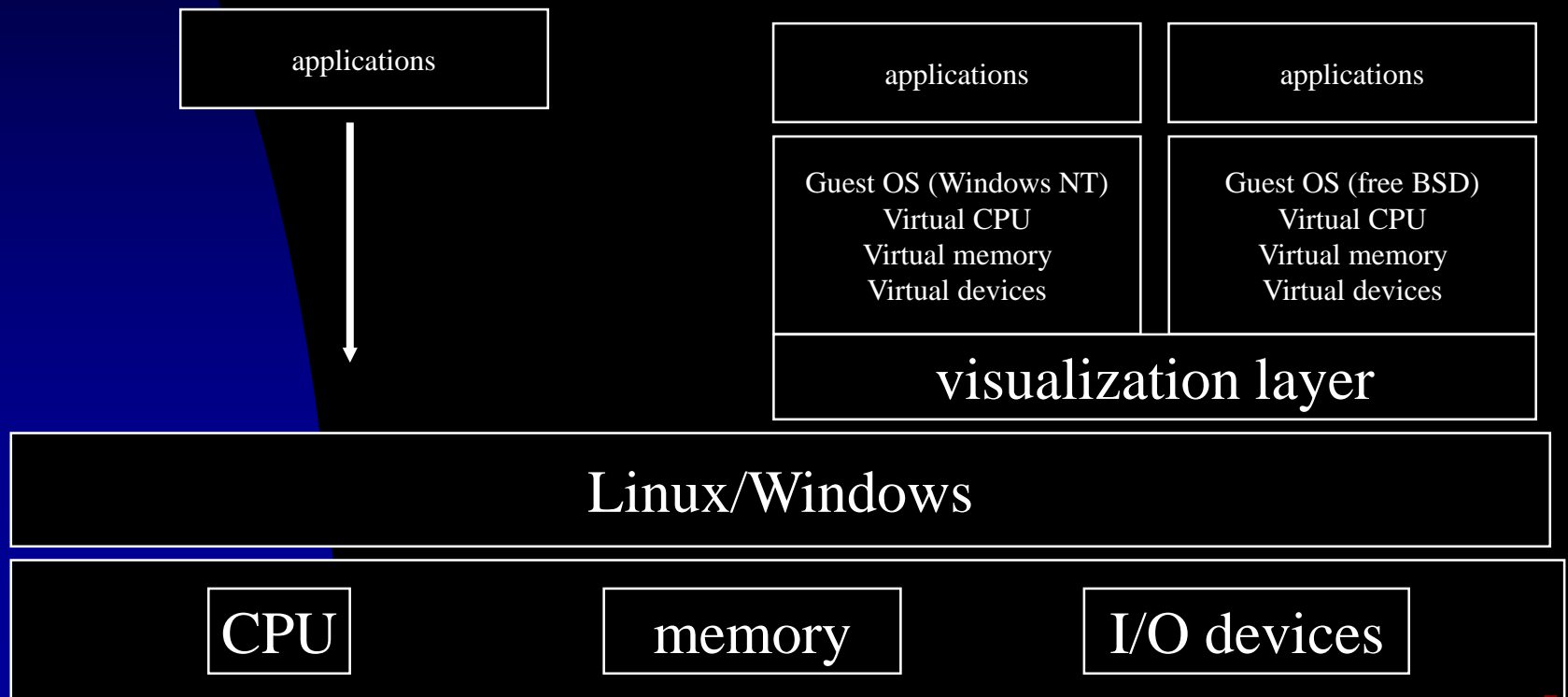
# Virtual Machine

- Advantages:
  - Complete Protection – Complete Isolation !
  - OS Research & Development
    - System Development Time
  - Extensions to Multiple Personalities, such as Mach (software emulation)
    - Emulations of Machines and OS's, e.g., Windows over Linux
  - System Consolidation

\* Simulation: Programs of a guest system are run on an emulator that translate each of the guest system instructions into the native instruction set of the host system.

# Virtual Machine – VMware

- VMware – The visualization layer abstracts the physical hardware into isolated virtual machines running as guest operating systems.



# Virtual Machine – Para-Virtualization

- Definition: A variation on virtualization that presents a guest/operating system that is similar but not identical to the underlying hardware.
  - Efficiency in Resource Utilization
  - Simplified Implementation
- Example: Container or Zone of Solaris 10
  - A Virtual Layer Between a Host OS and Applications
    - The OS and devices are virtualized.

# Virtual Machine – Java

java .class files



class loader



verifier



java interpreter



host system

- Sun Microsystems in late 1995
  - Java Language and API Library
  - Java Virtual Machine (JVM)
    - Class loader (for bytecode .class files)
    - Class verifier
    - Java interpreter
      - An interpreter, a just-in-time (JIT) compiler, hardware

# Virtual Machine – Java

java .class files



class loader



verifier



java interpreter



host system

- JVM
  - Garbage collection
    - Reclaim unused objects
  - Implementation being specific for different systems
    - Programs are architecture neutral and portable

# Operating System Examples – Linux Ver. 2.5+

Numeric  
Priority

Time  
Quantum

0  
.  
.  
99  
100  
.  
.  
.  
140

Real  
Time  
Tasks

Other  
Tasks

200ms

.  
. .  
. .  
. .  
. .  
. .  
. .  
. .

10ms

- Scheduling Algorithm
  - $O(1)$
  - SMP, load balancing, and processor affinity
  - Fairness and support for interactive tasks
  - Priorities
    - Real-time: 0..99
    - Nice: 100..140

# Operating System Examples – Linux Ver. 2.5+

- Each processor has a runqueue
  - An active array and an expired array
  - Switching of the two arrays when all processes in the active array have their quantum expired.
- Priority-Driven Scheduling
  - Fixed Priority – Real-Time
  - Dynamic Priority – nice  $\pm x$ , for  $x \leq 5$ 
    - Interactive tasks are favored.
    - The dynamic priority of a task is recalculated when its quantum is expired.