

Chapter 2: System Structures

Prof. Li-Pin Chang
National Chiao Tung University

Chapter 2: System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines

Objectives

- To describe the services an operating system provides to users, processes, and other systems
 - How OSs interacts with user programs (via **system calls**)
- To discuss the various ways of structuring an operating system
 - How OSs are structured

OPERATING SYSTEM SERVICES

Operating System Services

- One set of operating-system services provides **functions** that are helpful to the user:
 - **User interface** - Almost all operating systems have a user interface (UI)
 - Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch
 - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
 - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device.
 - **File-system manipulation** - The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

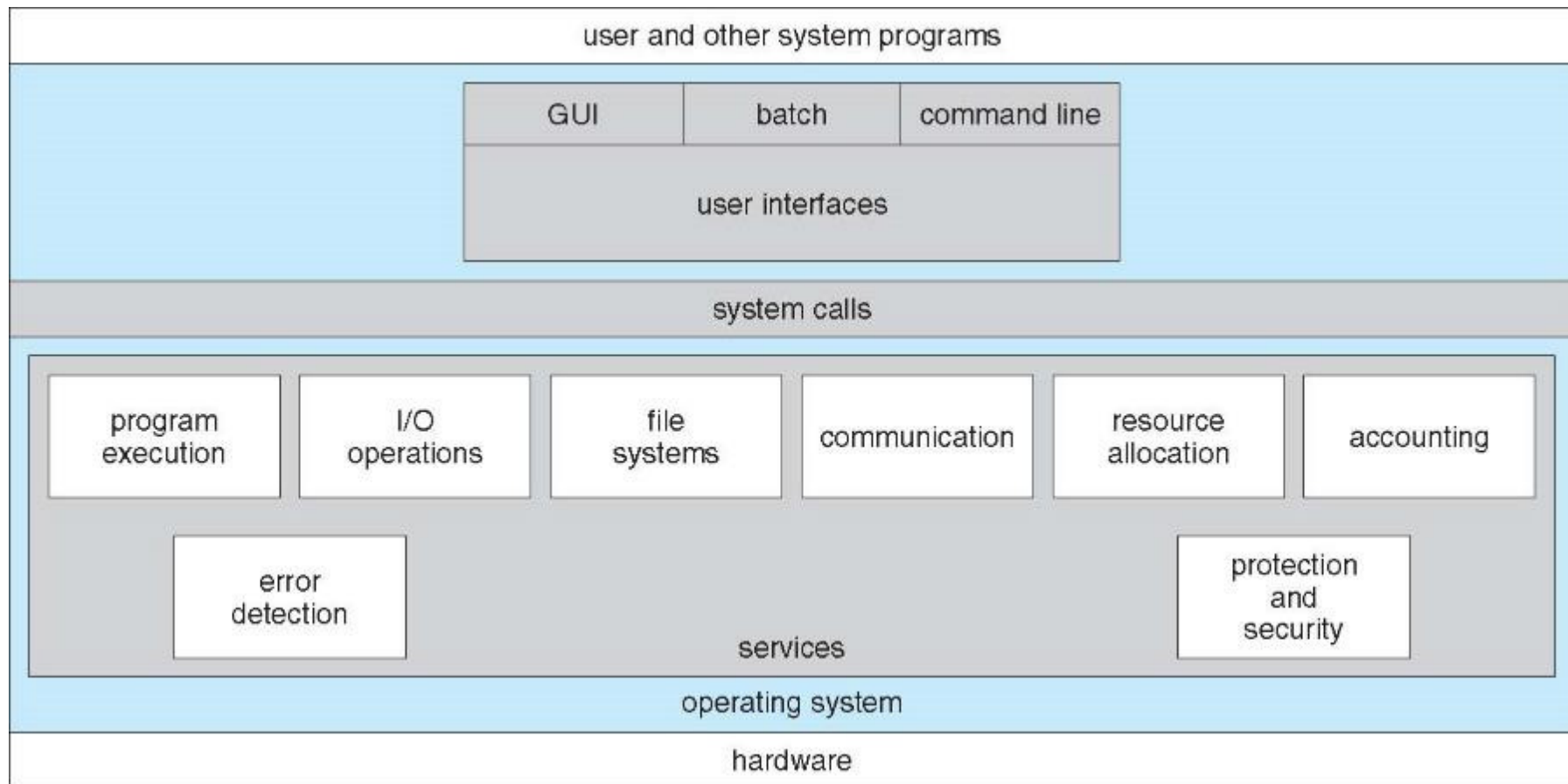
Operating System Services (Cont.)

- One set of operating-system services provides **functions** that are helpful to the user (Cont):
 - **Communications** – Processes may exchange information, on the same computer or between computers over a network
 - Communications may be via shared memory or through message passing (packets moved by the OS)
 - **Error detection** – OS needs to be constantly aware of possible errors
 - May occur in the CPU and memory hardware, in I/O devices, in user program
 - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
 - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

Operating System Services (Cont.)

- Another set of OS functions exists for ensuring the **efficient operation** of the system itself via resource sharing
 - Resource allocation - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
 - Many types of resources - Some (such as CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code.
 - Accounting - To keep track of which users use how much and what kinds of computer resources
 - Protection and security - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
 - Protection involves ensuring that all access to system resources is controlled
 - Security of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts
 - If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

A View of Operating System Services

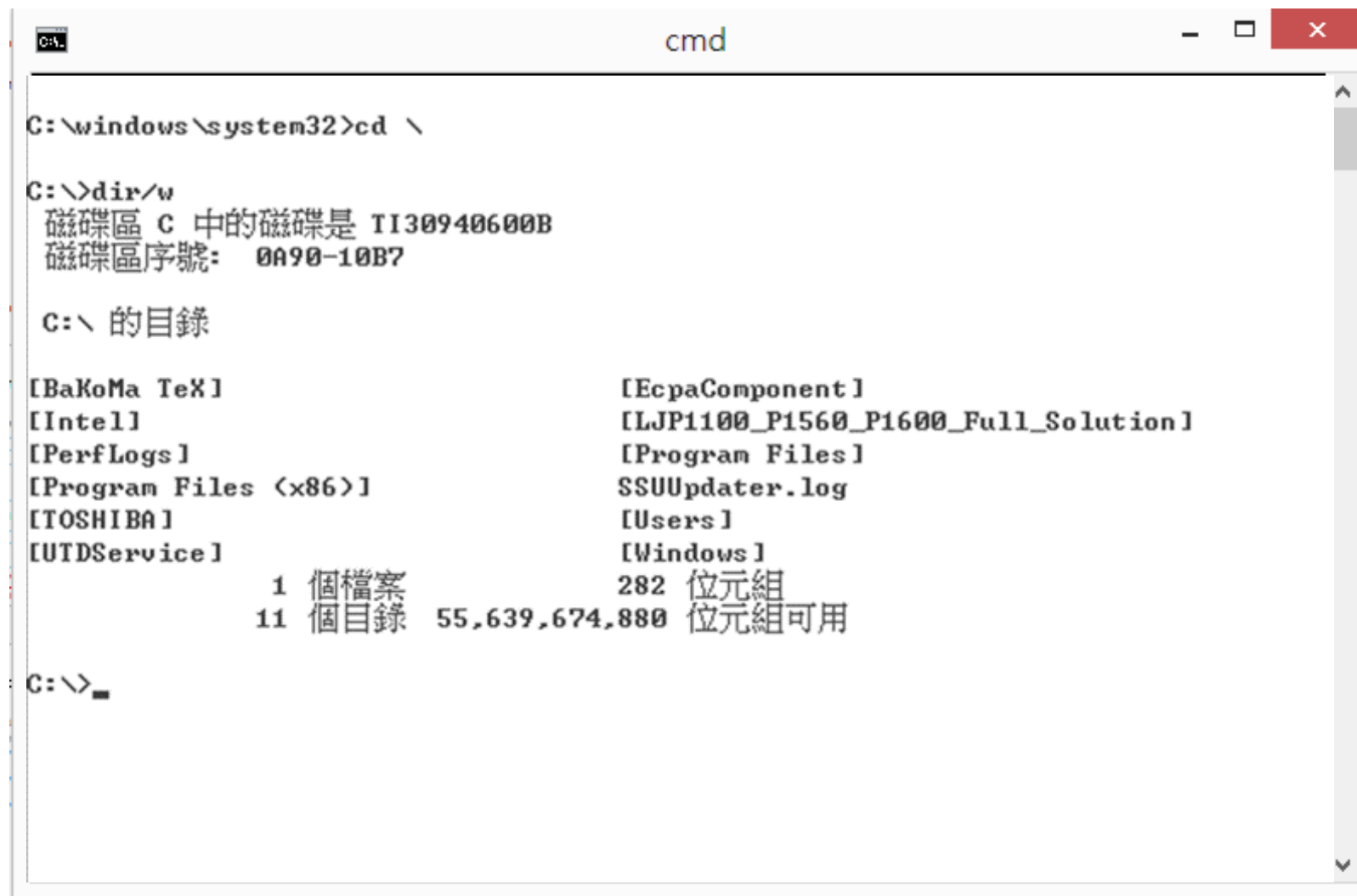


OPERATING-SYSTEM USER INTERFACE

Operating-System User Interface - CLI

- CLI allows direct command entry
 - Sometimes implemented in kernel, sometimes by systems program
 - Sometimes multiple flavors implemented – shells
 - Primarily fetches a command from user and executes it
 - Sometimes commands built-in, sometimes just names of programs
 - If the latter, adding new features doesn't require shell modification

Windows Shell Command Interpreter



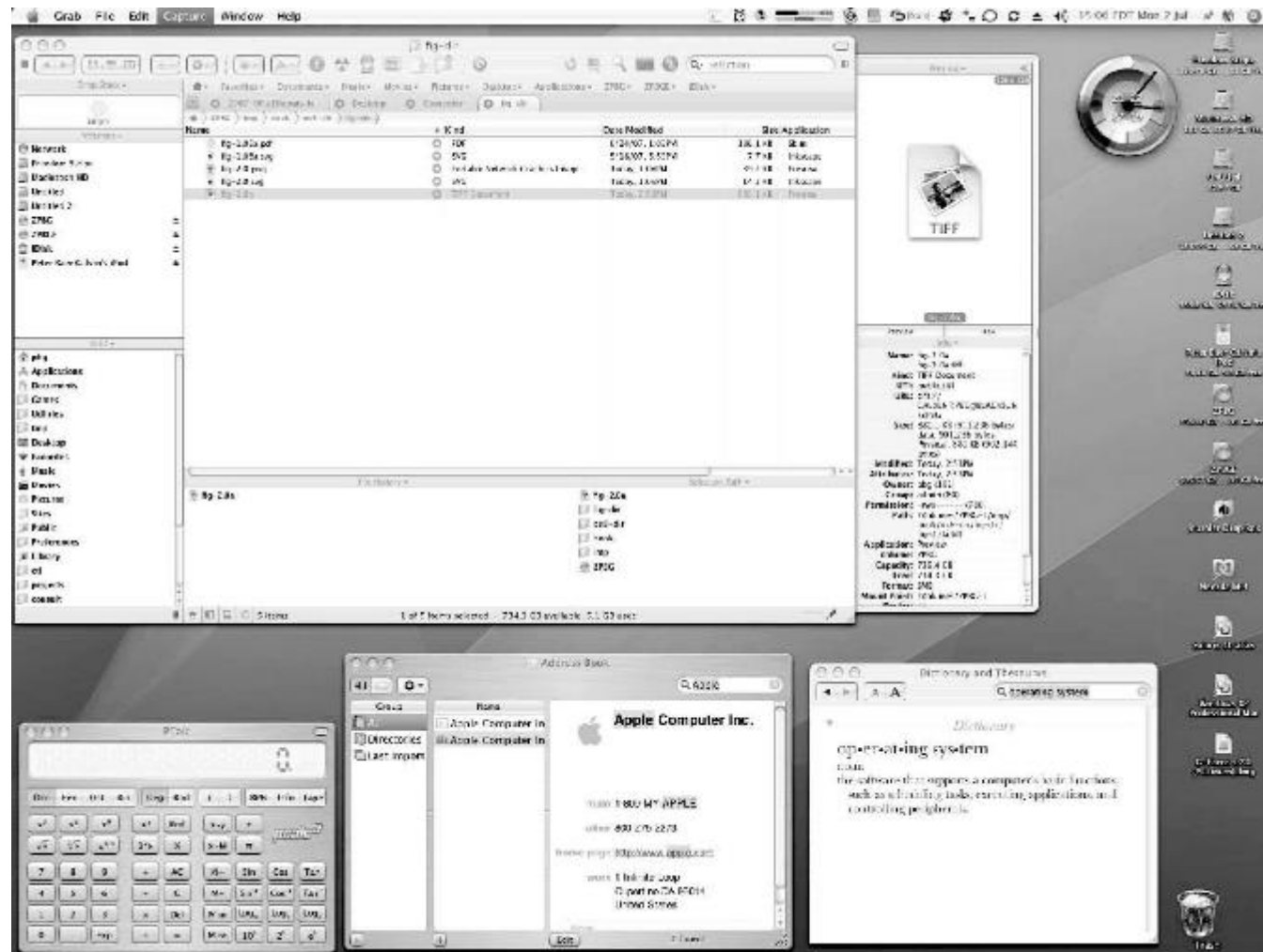
```
C:\windows\system32>cd \  
  
C:\>dir/w  
磁碟區 C 中的磁碟是 T130940600B  
磁碟區序號: 0A90-10B7  
  
C:\ 的目錄  
  
[BaKoMa TeX] [EcpaComponent]  
[Intel] [LJP1100_P1560_P1600_Full_Solution]  
[PerfLogs] [Program Files]  
[Program Files (x86)] SSUUpdater.log  
[TOSHIBA] [Users]  
[UTDService] [Windows]  
1 個檔案 282 位元組  
11 個目錄 55,639,674,880 位元組可用  
  
C:\>_
```

User Operating System Interface - GUI

- User-friendly desktop metaphor interface
 - Usually mouse, keyboard, and monitor
 - Icons represent files, programs, actions, etc
 - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a folder))
 - Invented by Xerox PARC
- Many systems now include both CLI and GUI interfaces
 - Microsoft Windows is GUI with CLI “command” shell
 - Apple Mac OS X as “Aqua” GUI interface with UNIX kernel underneath and shells available
 - Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)

PARC's accomplishments:
mouse, GUI, WYSIWYG editors,
postscript language, laser printers, ethernet, small talk

The Mac OS X GUI



Touchscreen Interfaces

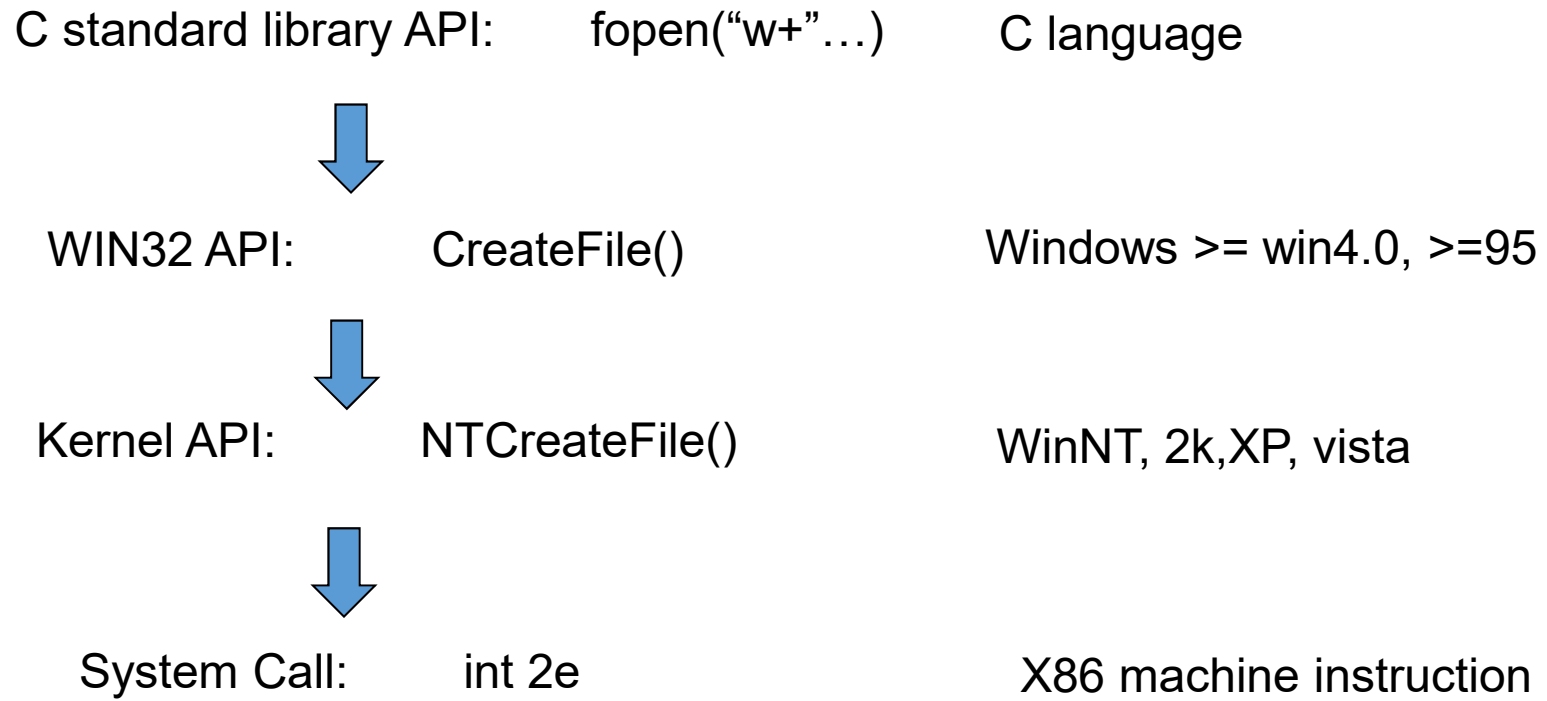
- Touchscreen devices require new interfaces
 - Mouse not possible or not desired
 - Actions and selection based on **gestures**
 - Virtual keyboard for text entry



SYSTEM CALLS

System Calls

- **Programming interface** to the services provided by the OS
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use
 - Typically written in a high-level language (C or C++)
 - Why use APIs rather than system calls?
 - **Portability and simplicity**
- 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)



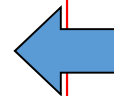
```
int printf ( const char * format, ... );
```



```
BOOL WINAPI WriteFile(  
    _In_          HANDLE hFile,  
    _In_          LPCVOID lpBuffer,  
    _In_          DWORD nNumberOfBytesToWrite,  
    _Out_opt_     LPDWORD lpNumberOfBytesWritten,  
    _Inout_opt_   LPOVERLAPPED lpOverlapped  
);
```



```
NTSTATUS NtWriteFile  
(  
    HANDLE          hFile,  
    HANDLE          hEvent,  
    PIO_APC_ROUTINE apc,  
    void*           apc_user,  
    PIO_STATUS_BLOCK io_status,  
    const void*      buffer,  
    ULONG           length,  
    PLARGE_INTEGER   offset,  
    PULONG           key  
)
```



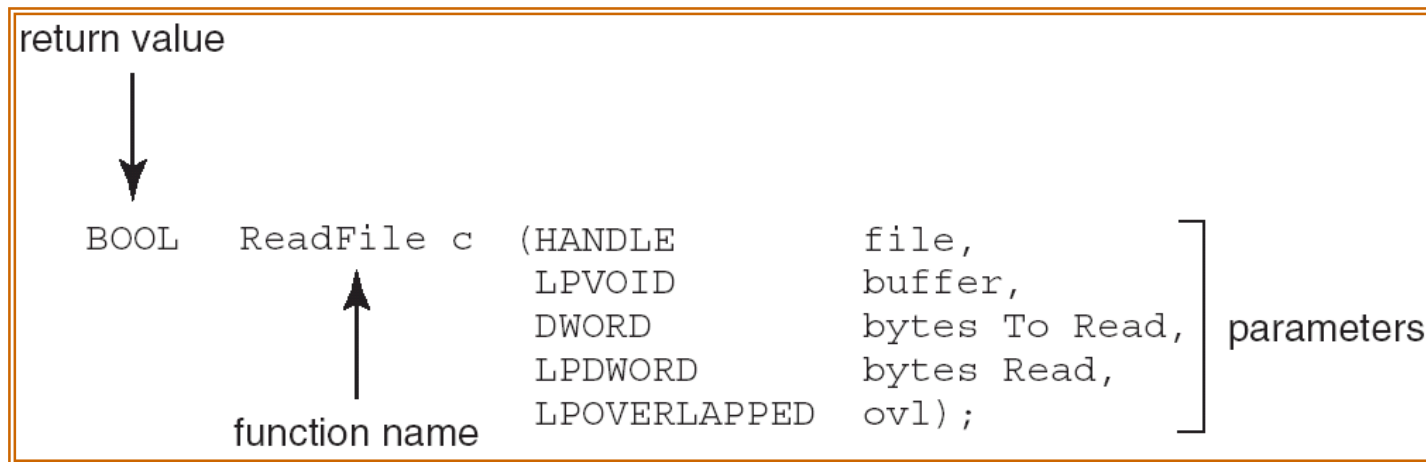
Err... Not disclosed by Microsoft...

May looks like:

```
mov eax, <service #>  
lea edx, <addr of 1st arg>  
int 2e
```

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

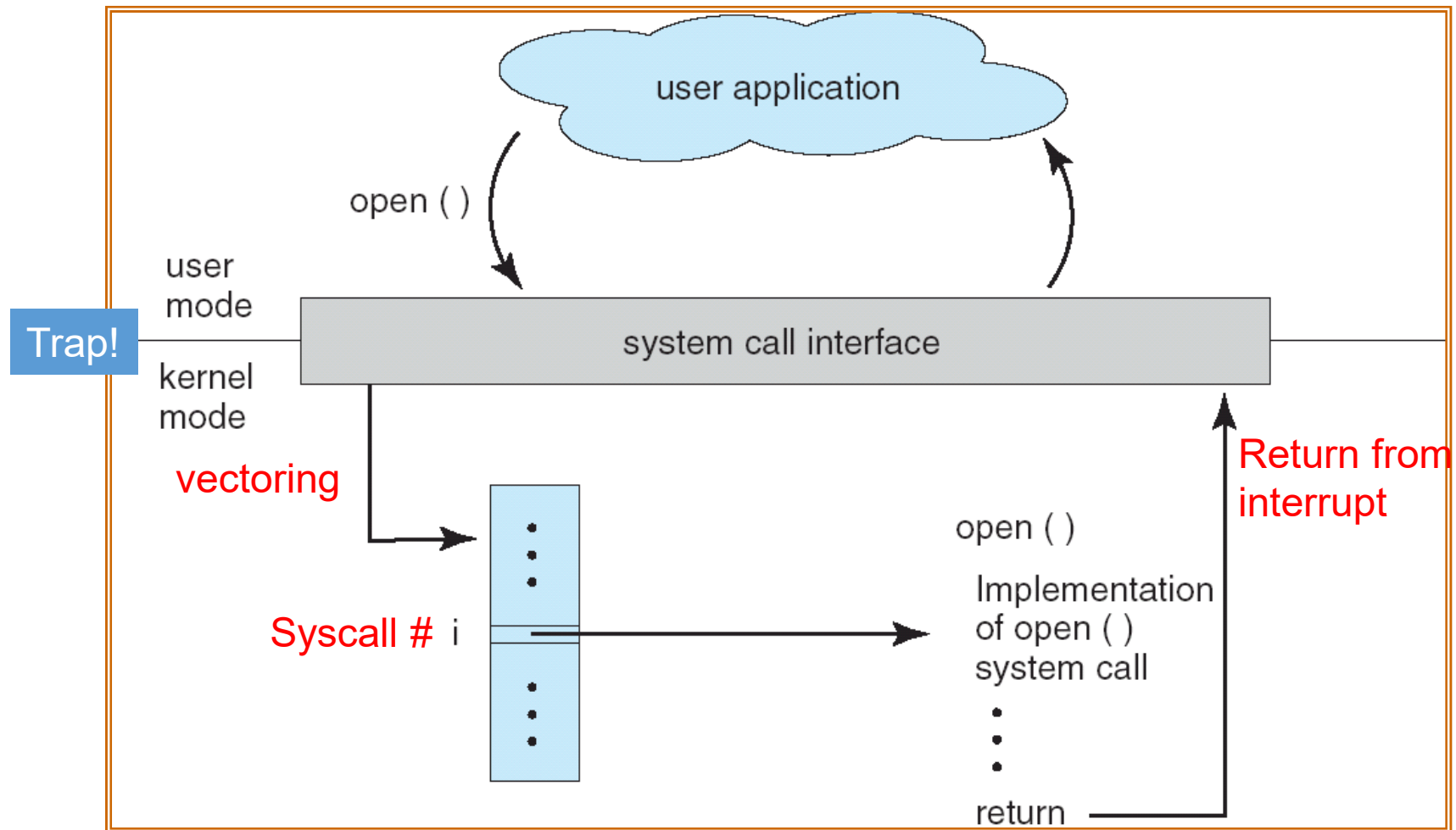
System Call Implementation

- Typically, **a number** associated with each system call
 - System-call interface maintains **a table** indexed according to these numbers
- The system call interface invokes 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
 - Just needs to obey API and understand what OS will do as a result call

Dual Mode Operations

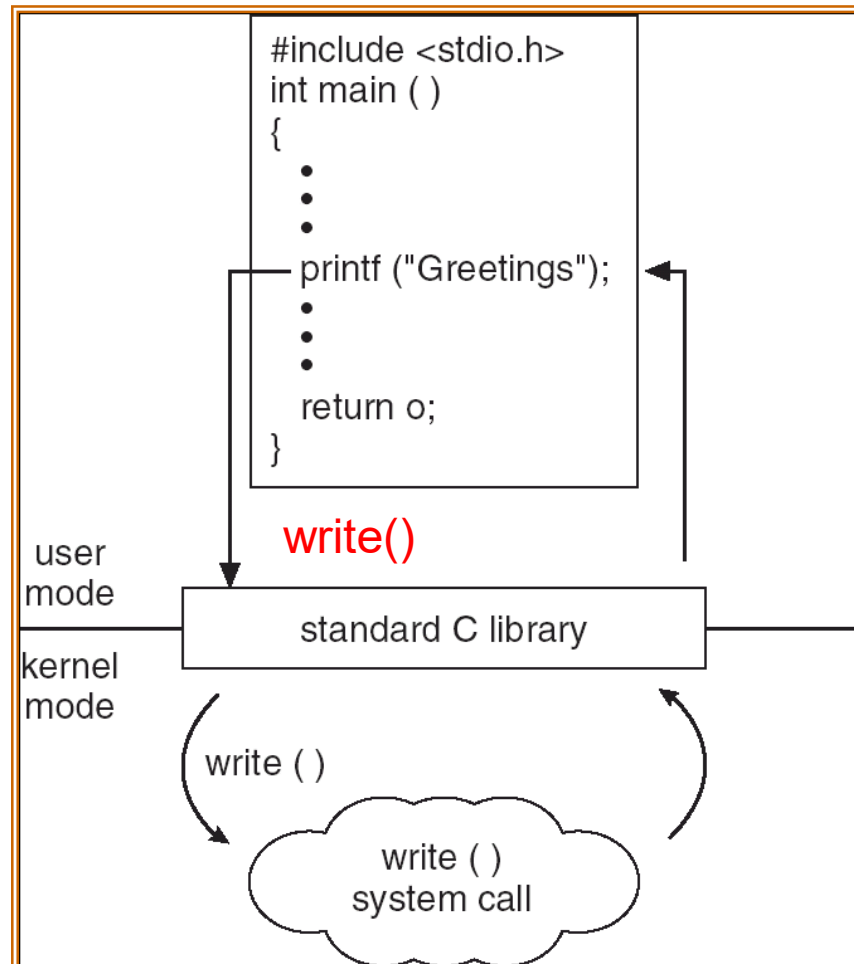
- Application calls into the kernel through **trap**
- **Dual-mode** operation allows OS to protect itself and other system components
 - User mode and kernel mode
 - Mode bit provided by hardware
- The purpose of dual-mode design
 - Provides ability to distinguish when system is running user code or kernel code
 - Some instructions designated as privileged, only executable in kernel mode

API – System Call – OS Relationship



Standard C Library Example

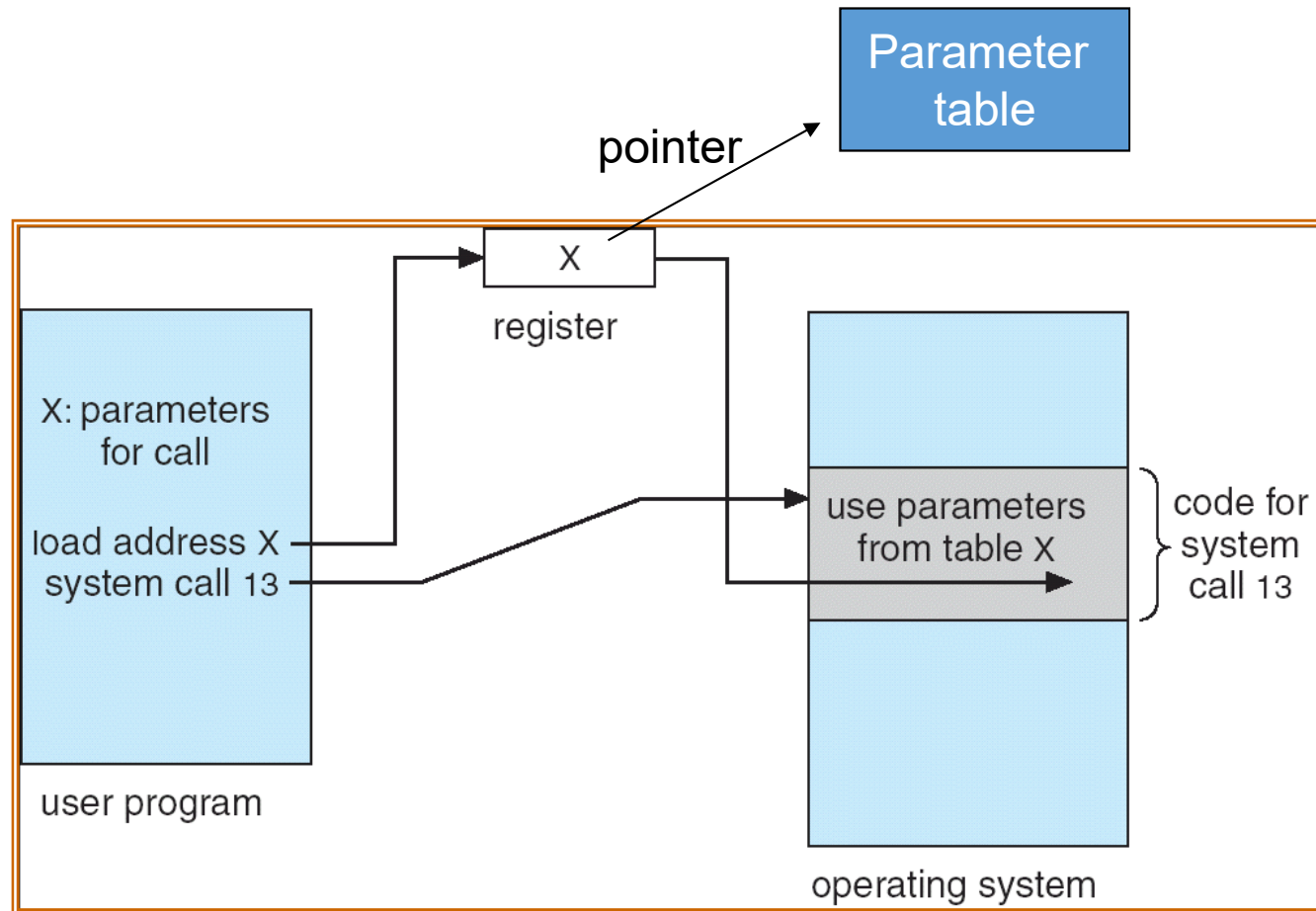
- C program invoking printf() library call, which calls write() system call



System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
 - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
 - Pass the parameters in **registers**
 - Parameters stored in a block, or **table**, in memory, and address of block passed as a parameter in a register
 - Parameters placed, or pushed, into the **stack** by the program and popped off the stack by the operating system

Parameter Passing via Table



System Call in Linux

```
section .data                                ;declare section
msg db  "Hello World! :)",0xa                ;our dear string
len equ $ - msg                             ;length of our dear string

section .text                                ; section declaration

    global _start                            ; exporting entry point
                                           ; to the ELF linker

_start:
; write Hello World string
    mov edx,len ;third arg: message length
    mov ecx,msg ;second arg: pointer to message to write
    mov ebx,1   ;first arg: file handle (stdout)
    mov eax,4   ;system call nr. (sys_write)
    int 0x80    ;call kernel (trigger a trap)
; and exit
    mov ebx,0   ;first syscall args: exit code
    mov eax,1   ;system call no. (sys_exit)
    int 0x80    ;call kernel
```

System Call in Windows

- Use the following fragment of assembly code to call the kernel
 - `mov eax, <service #>`
 - `lea edx, <addr of 1st arg>`
 - `int 2e`
- Return value is in EAX (if any)

TYPES OF SYSTEM CALLS

Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications

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() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Linux System Calls

List by system call number

[00](#) sys_setup [sys_ni_syscall]
[01](#) sys_exit
[02](#) sys_fork
[03](#) sys_read
[04](#) sys_write
[05](#) sys_open
[06](#) sys_close
[07](#) sys_waitpid
[08](#) sys_creat
[09](#) sys_link
[10](#) sys_unlink
[11](#) sys_execve
[12](#) sys_chdir
[13](#) sys_time
[14](#) sys_mknod
[15](#) sys_chmod
[16](#) sys_lchown
[17](#) sys_break [sys_ni_syscall]
[18](#) sys_oldstat [sys_stat]
[19](#) sys_lseek
[20](#) sys_getpid
[21](#) sys_mount

...

[70](#) sys_setreuid
[71](#) sys_setregid
[72](#) sys_sigsuspend
[73](#) sys_sigpending
[74](#) sys_sethostname
[75](#) sys_setrlimit
[76](#) sys_getrlimit
[77](#) sys_getrusage
[78](#) sys_gettimeofday
[79](#) sys_settimeofday
[80](#) sys_getgroups
[81](#) sys_setgroups
[82](#) sys_select [old_select]
[83](#) sys_symlink
[84](#) sys_oldlstat [sys_lstat]
[85](#) sys_readlink
[86](#) sys_uselib
[87](#) sys_swapon
[88](#) sys_reboot
[89](#) sys_readdir [old_readdir]
[90](#) sys_mmap [old_mmap]
[91](#) sys_munmap

...

[140](#) sys__llseek [sys_lseek]
[141](#) sys_getdents
[142](#) sys__newselect [sys_select]
[143](#) sys_flock
[144](#) sys_msync
[145](#) sys_readv
[146](#) sys_writev
[147](#) sys_getsid
[148](#) sys_fdatasync
[149](#) sys__sysctl [sys_sysctl]
[150](#) sys_mlock
[151](#) sys_munlock
[152](#) sys_mlockall
[153](#) sys_munlockall
[154](#) sys_sched_setparam
[155](#) sys_sched_getparam
[156](#) sys_sched_setscheduler
[157](#) sys_sched_getscheduler
[158](#) sys_sched_yield
[159](#) sys_sched_get_priority_max
[160](#) sys_sched_get_priority_min
[161](#) sys_sched_rr_get_interval

...

SYSTEM PROGRAMS (SYSTEM SOFTWARE)

System Programs

- System programs provide a convenient environment for program development and execution. They can be divided into:
 - File manipulation (cp, mv...)
 - Status information (ls...)
 - File modification (vi...)
 - Programming language support (cc, ld, ar...)
 - Program loading and execution
 - Communications (telnet...)
- GNU binutils
- Most users' view of the operation system is defined by system programs, not the actual system calls

System Programs

- Provide a convenient environment for program development and execution
 - Some of them are simply user interfaces to system calls; others are considerably more complex
- File management - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- Status information
 - Some ask the system for info - date, time, amount of available memory, disk space, number of users
 - Others provide detailed performance, logging, and debugging information
 - Typically, these programs format and print the output to the terminal or other output devices
 - Some systems implement a registry - used to store and retrieve configuration information

System Programs (cont'd)

- File modification
 - **Text editors** to create and modify files (vi)
 - Special commands to search contents of files or perform transformations of the text
- Programming-language support - **Compilers, assemblers, debuggers** and interpreters sometimes provided
- Program loading and execution- **Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders**, debugging systems for higher-level and machine language
- Communications - Provide the mechanism for creating virtual connections among processes, users, and computer systems
 - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another
 - **ftp, telnet, ssh, etc**

OPERATING SYSTEM DESIGN AND IMPLEMENTATION

Operating System Design and Implementation

- Design and Implementation of OS not “solvable”, but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- Start by defining **goals** and **specifications**
- Affected by choice of hardware, type of system
- User goals and System goals
 - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
 - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
- **Different goals for different types of systems**
 - Real-time OS: time predictability, low latency, reliability
 - Mainframe OS: throughput, fairness

Operating System Design and Implementation (Cont.)

- Important principle to separate
 - **Mechanism**: How to do it?
 - **Policy**: What will be done?
- Mechanisms determine how to do something, policies decide what will be done next
- The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later
- Use disk I/O as an example:
 - Mechanism: How to read and write from disk?
 - Policy: Which disk I/O operation should be performed first?

Which one(s) of the following are policies; which are mechanisms?

- a) process suspend/resume
- b) allocating the smallest among the memory blocks which are larger than the requested size
- c) marking a disk block as allocated
- d) servicing the disk I/O request which is closest to the disk head

Simple	→ MSDOS
Monolith	→ UNIX
Microkernel	→ Mach
Layered	→ Err....
Virtual machine	→ Cloud

OPERATING-SYSTEM STRUCTURE

OS架構重要的是應用在哪裡
在學理上是否先進並沒有很重要

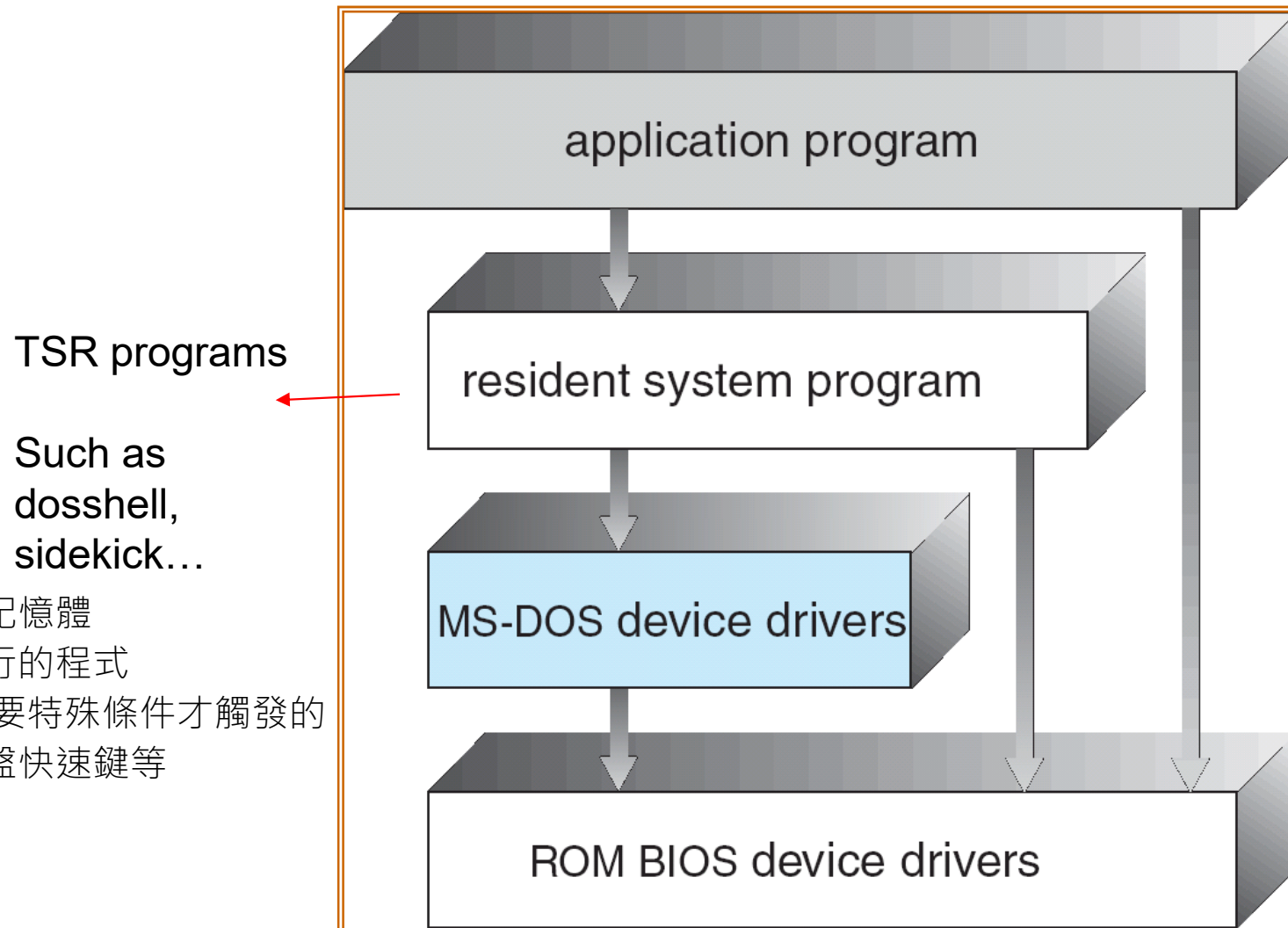
Simple Structure 基本上就是沒什麼硬性規定

- 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
- No protection. Applications can directly access any memory addresses and control hardware
- MS-DOS is left no choice because the CPU 8086 offers no hardware protection for user-kernel separation

沒有user mode、kernel mode的分別，只要應用程式出問題(亂複寫記憶體etc.)整個系統就會crash(要重開機)。

但廣泛運用在嵌入式系統，因為佔空間小，速度快而嵌入式系統載進去就不會更動，程式又是可信任(已經測試好)，沒有容易crash的問題。

MS-DOS Layer Structure



很受歡迎

UNIX--monolithic

架構上蠻落後的，但效能好大家喜歡。

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts:

1. Systems programs

- binutils, cc, as, ld, etc

2. The kernel

- Consists of everything below the system-call interface and above the physical hardware
- Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

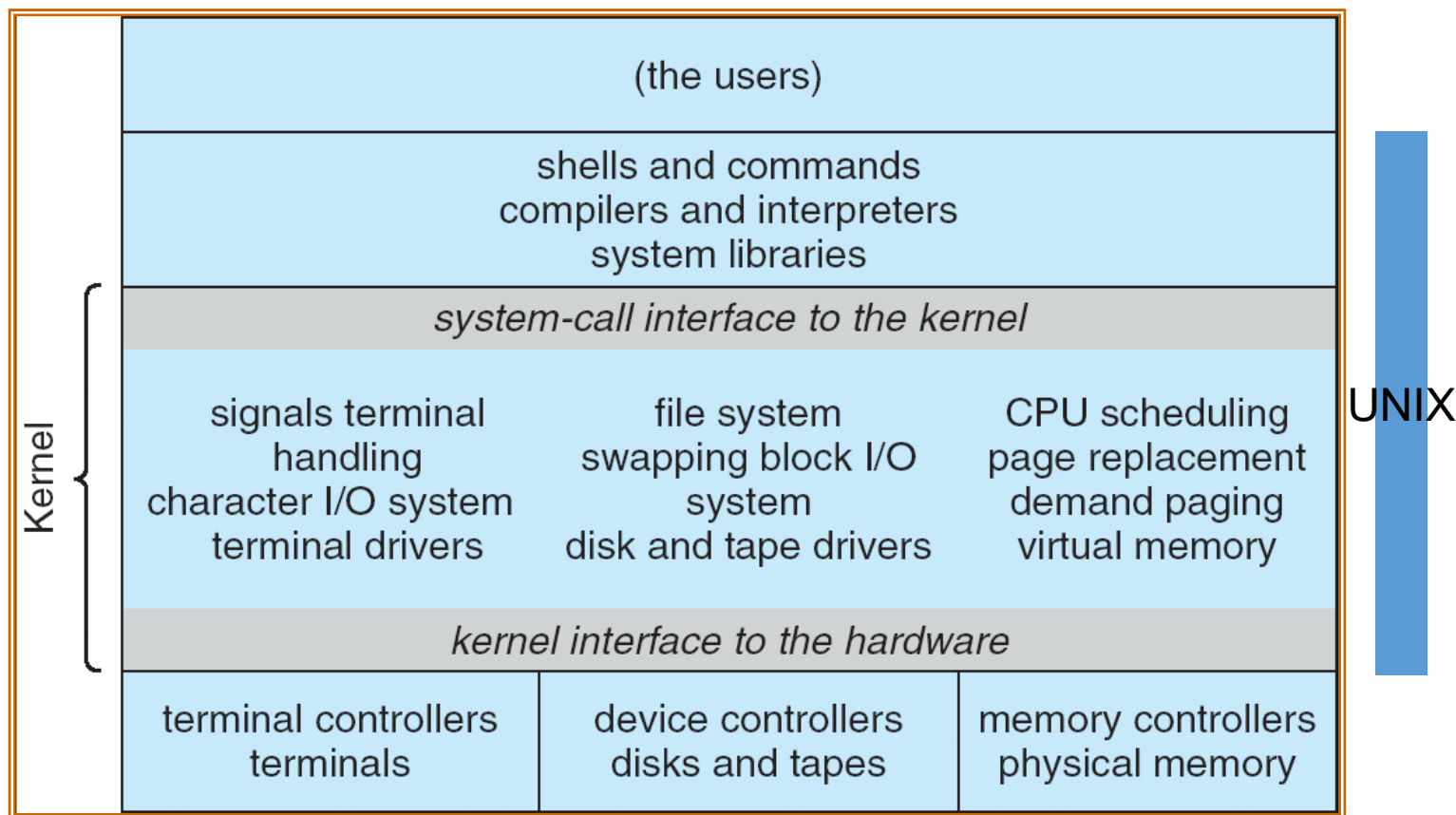
“LINUX is a monolithic style system. This is a giant step back into the 1970s ”

龐大

-- Andrew Tanenbaum₄₃

UNIX System Structure

僅有user mode、kernel mode這樣基本的保護

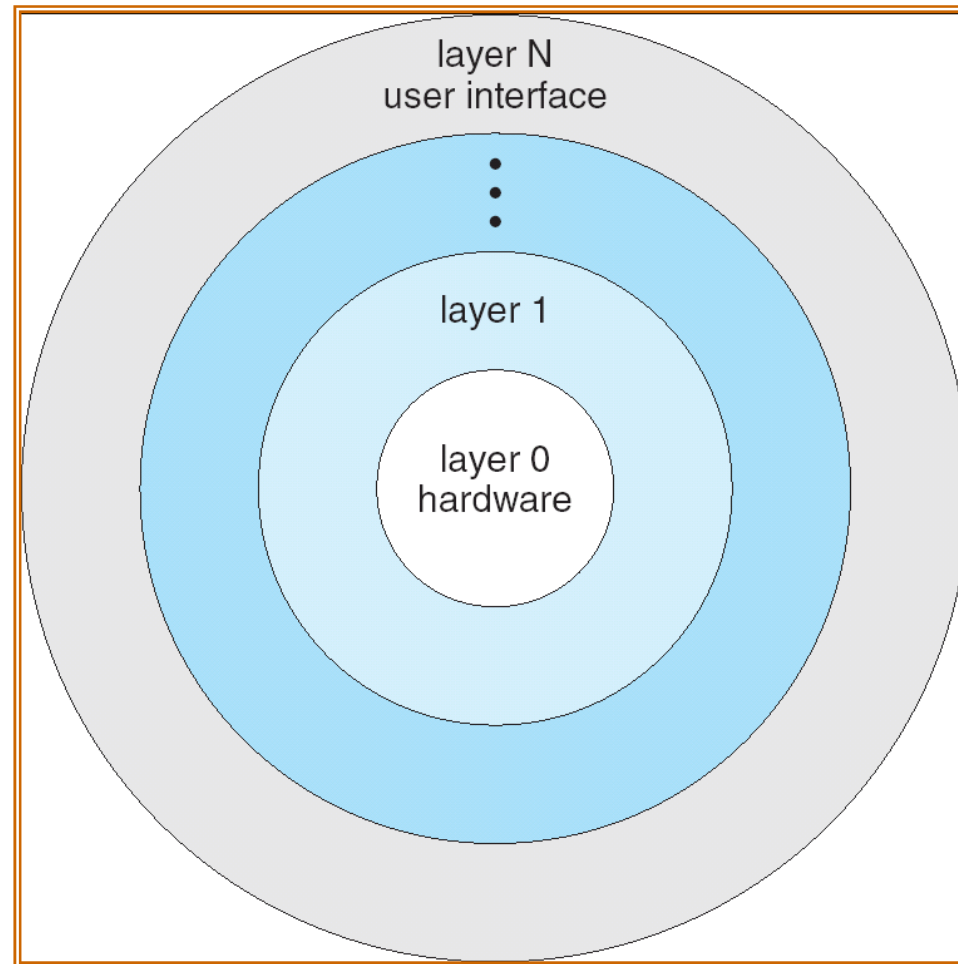


UN*X is, of course, a huge monolith operating system

Layered Approach (Conceptual) 目前沒有os實現過

- 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
- Pros: easy to debug and to scale
 - Debugging layer i needs to know nothing about layer i-1
 - Well structured, easy to scale
- Cons: **very hard to define layers** 有循環相依性，難以決定哪一個是上層
 - For example, a driver of storage devices may need to allocate memory
 - However, memory management may needs to use storage as virtual memory
 - Performance is also a concern

Layered Operating System



每一個layer只需要知道下一層怎麼做就好(有點像網路分層)，而不像UNIX把kernel包一起，一旦做kernel coding就要把整個kernel弄懂(極端痛苦)

Layer i should be invoked by only layer $i+1$

Modules

先編譯好的driver之類(已經放在kernel裡)
偵測到硬體後能直接載入
可以像App一樣的卸除和載入

- 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

Common interfaces of a Linux kernel module

Initialize

Clean up

Read (char)

Write (char)

Read (block)

Write (block)

Case Study: Linux Module

```
#include <linux/kernel.h> /* header file for structure pr_info */
#include <linux/init.h>
#include <linux/module.h> /* header file for all modules */
#include <linux/version.h>

MODULE_DESCRIPTION("Hello world !!");
MODULE_AUTHOR("John Doe");
MODULE_LICENSE("GPL");

static int __init hello_init(void)
{
    pr_info("Hello, world\n");
    pr_info("The process is \"%s\" (pid %i)\n", current->comm, current->pid);
    return 0;
}

static void __exit hello_exit(void)
{
    printk(KERN_INFO "Goodbye\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

Source:

<http://blog.wu-boy.com/2010/06/linux-kernel-driver-%E6%92%B0%E5%AF%AB%E7%B0%A1%E5%96%AE-hello-world-module-part-1/>

Case Study: Linux Module

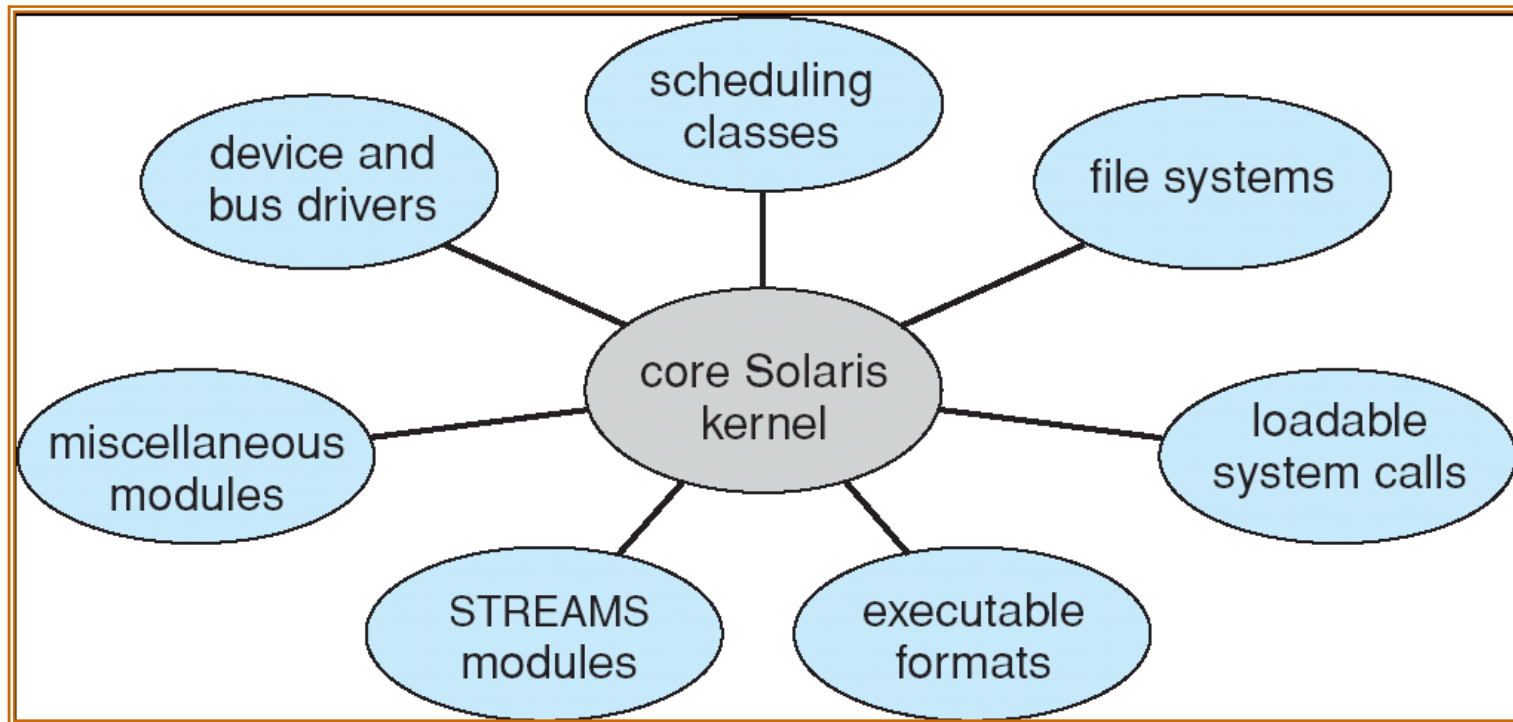
- Insert & init the module
 - insmod ./hello.ko
- Remove & clean up the module
 - rmmod ./hello.ko

```
Jun 21 11:50:00 cvs5 kernel: [8381603.818051] The process is "insmod" (pid 30560)
Jun 21 11:50:08 cvs5 kernel: [8381612.335386] Goodbye
Jun 21 12:07:13 cvs5 rsyslogd: [origin software="rsyslogd" swVersion="4.2.0" x-pid='
.com"] rsyslogd was HUPed, type 'lightweight'.
Jun 21 12:07:13 cvs5 rsyslogd: [origin software="rsyslogd" swVersion="4.2.0" x-pid='
.com"] rsyslogd was HUPed, type 'lightweight'.
Jun 21 14:55:23 cvs5 kernel: [8392723.612597] Hello, world
Jun 21 14:55:23 cvs5 kernel: [8392723.612601] The process is "insmod" (pid 10072)
Jun 21 14:55:37 cvs5 kernel: [8392737.604360] Goodbye
Jun 21 15:05:18 cvs5 kernel: [8393318.795982] Hello, world
Jun 21 15:05:18 cvs5 kernel: [8393318.795985] The process is "insmod" (pid 13127)
Jun 21 15:05:25 cvs5 kernel: [8393325.537903] Goodbye
```

Source:

<http://blog.wu-boy.com/2010/06/linux-kernel-driver-%E6%92%B0%E5%AF%AB%E7%B0%A1%E5%96%AE-hello-world-module-part-1/>

Solaris Modular Approach



- More flexible than a layered kernel, a module calls any one another
- More efficient than a microkernel, as frequent mode switch is not needed
- More structured than a monolithic kernel

統一的

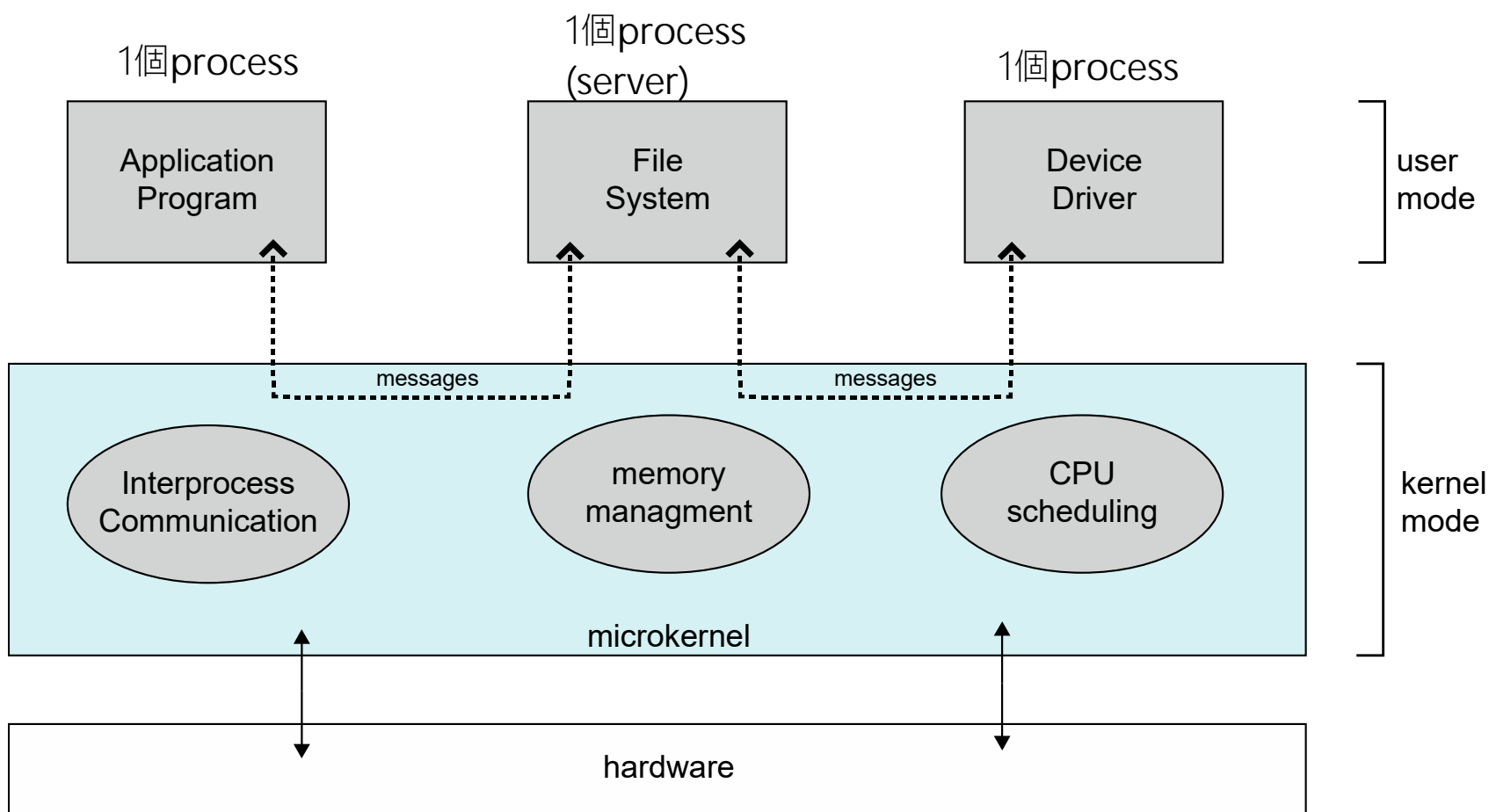
Microkernel System Structure

因為UNIX的kernel太多東西了
此架構希望能放user mode就放

- Moves as much from the kernel into “user” space
- Communication takes place between user modules using message passing
- Benefits: 以檔案系統為例，只需要寫一個user program，且抽換很方便。
 - Easier to extend a microkernel (by adding user-mode modules)
 - Easier to port the operating system to new architectures 只需要換micro kernel(較小,幾K)就好 ex.Intel→AMD
 - More reliable and secure (less code is running in kernel mode)
- Detriments: kernel中會crash的東西更少，且kernel提供給外界的功能更少(漏洞更少)
 - Performance overhead of user space to kernel space communication and user-kernel mode switches
 - What happened to Windows NT 3.5? 每次模式切換都是一個中斷(在CPU很昂貴)，而且資料傳遞只能用copy的(也很慢)，導致效能很差

Microkernel System Structure

出發點很好，但效能很差

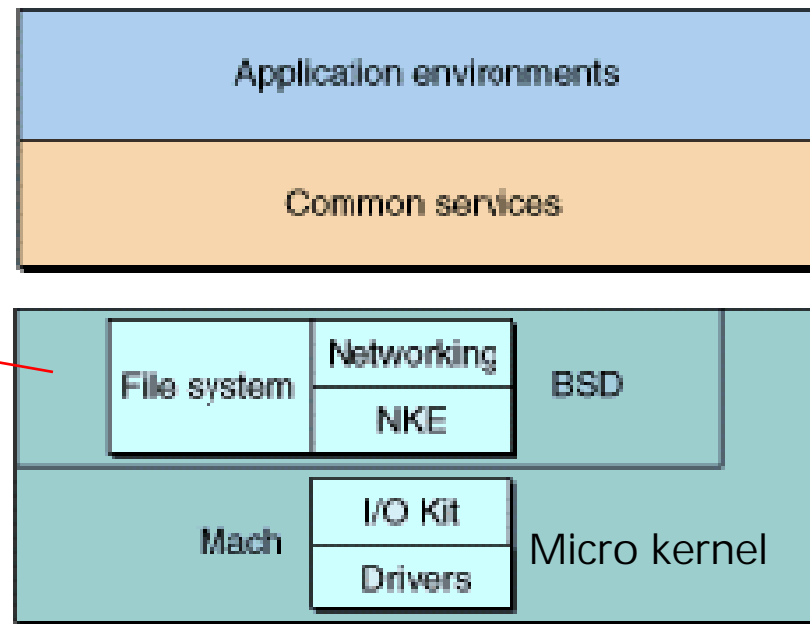


Mac OS X Structure

- Mach (μ -kernel): memory management, RPC, IPC, message passing, thread scheduling
- BSD: networking, file systems, POSIX APIs

system call的API是posix
，所有UNIX的程式都能
在Mac OS跑

Kernel
environment



火熱進行中XD

Google Fuschia Micro Kernel架構

- A mysterious new operating system developed by Google, based on the Zircon microkernel



可能用在IoT Device上?

因為IoT Device硬體很多樣，且效能不是重點

Quiz

What are the design objectives of the micro-kernel approach?

1. Scalability
2. Robustness
3. Efficiency
4. Security

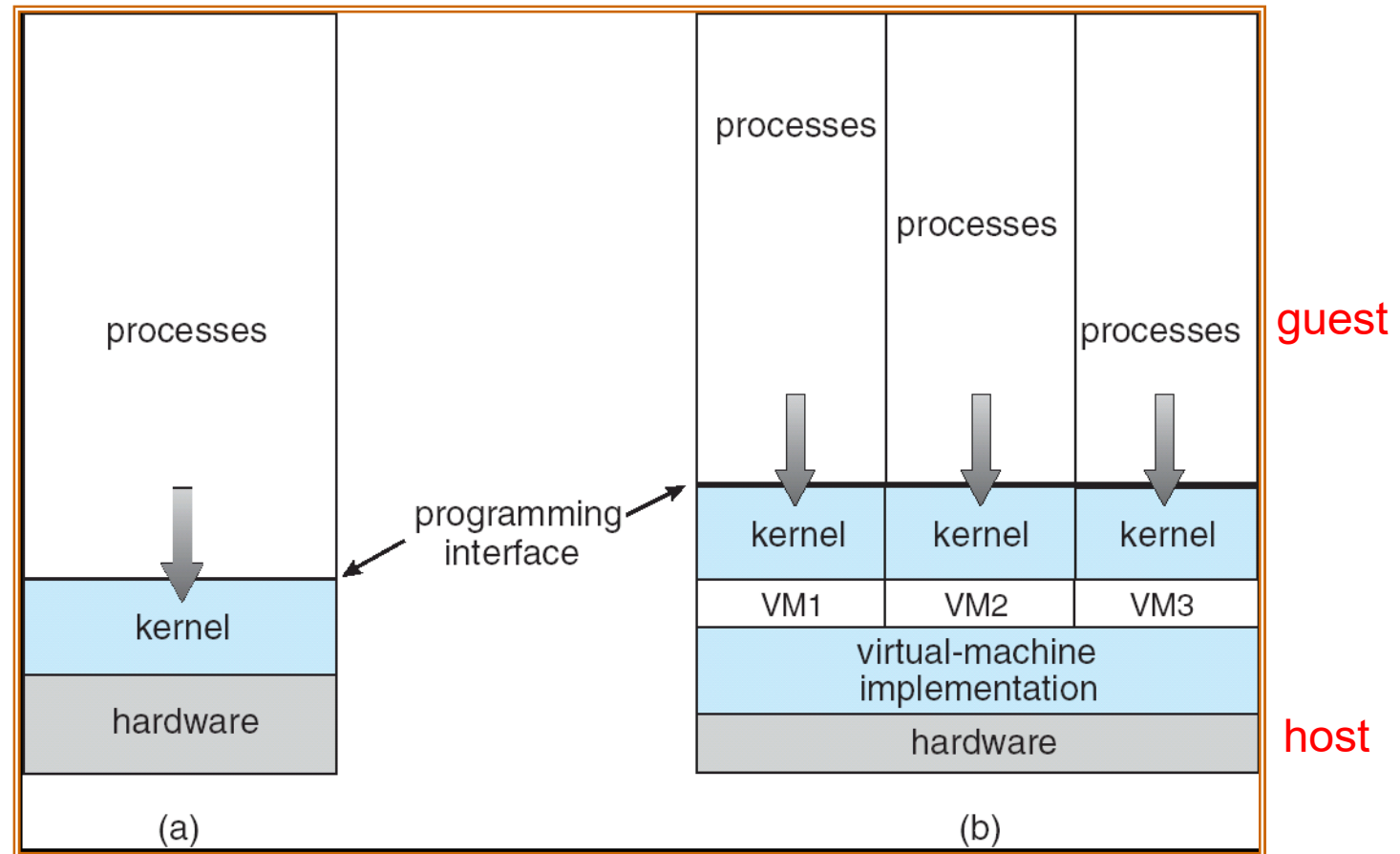
Virtual Machines

- 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
- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory
- Virtual machine is not a new concept. It has been developed in 197x. VM again become popular because
 - It becomes hard to maintain outdated servers
 - Cloud computing (service virtualization)

Virtual Machines (Cont.)

- The resources of the physical computer are shared to create the virtual machines
 - CPU scheduling can create the appearance that users have their own processor
 - Spooling and a file system can provide virtual card readers and virtual line printers
 - A normal user time-sharing terminal serves as the virtual machine operator's console

Virtual Machines (Cont.)

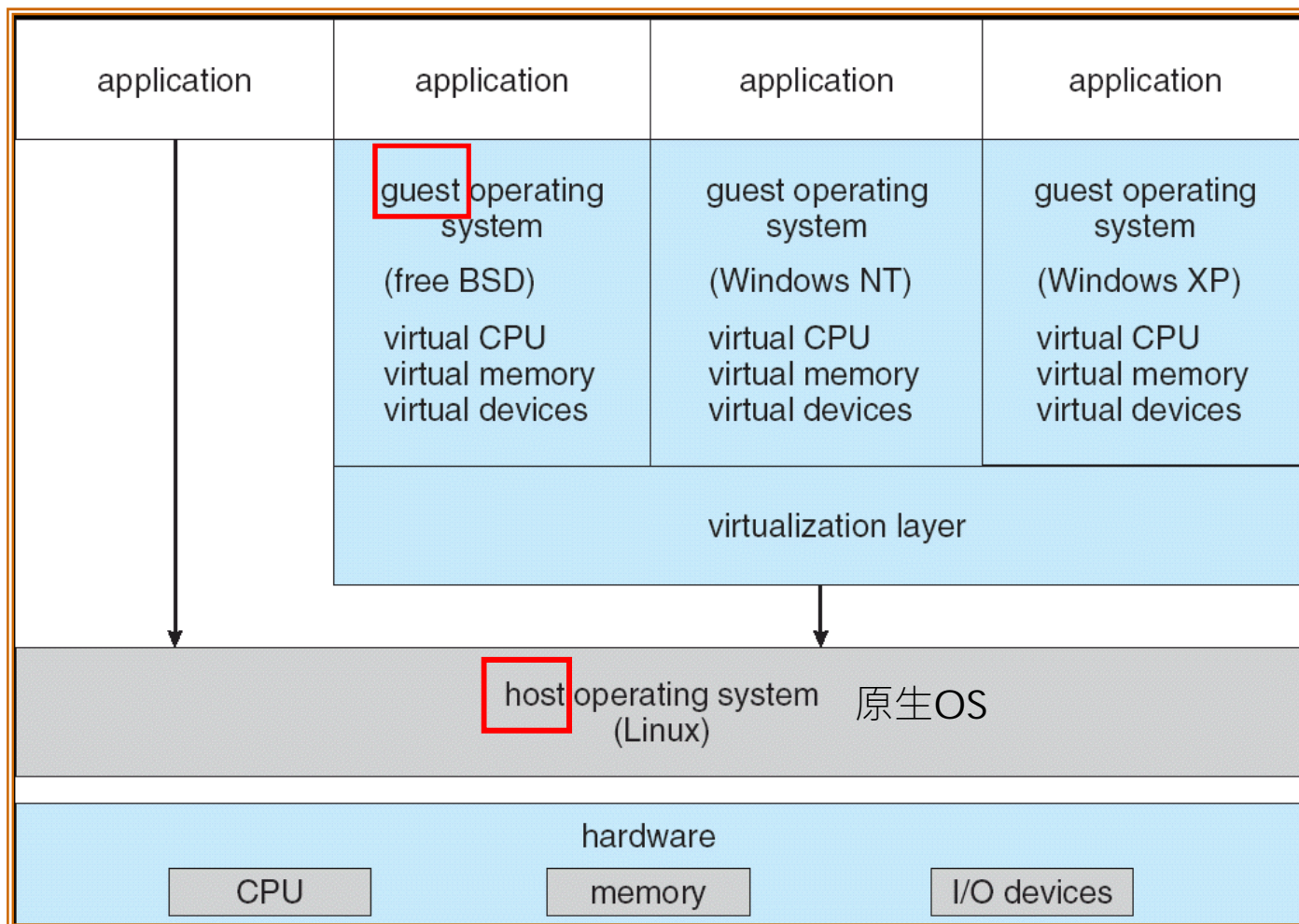


(a) bare-metal (b) virtual machines

Virtual Machines (Cont.)

- The virtual-machine concept provides **complete protection of system resources since each virtual machine is isolated from all other virtual machines**. This isolation, however, permits no direct sharing of resources.
- A virtual-machine system is a **perfect vehicle for operating-systems research and development**. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
- The virtual machine concept is **difficult** to implement due to the effort required to provide an exact duplicate to the underlying machine
如果使用相同ISA(直接跑binary code,不用轉換)還好，但碰到其他硬體(I/O etc.)效能就很差。

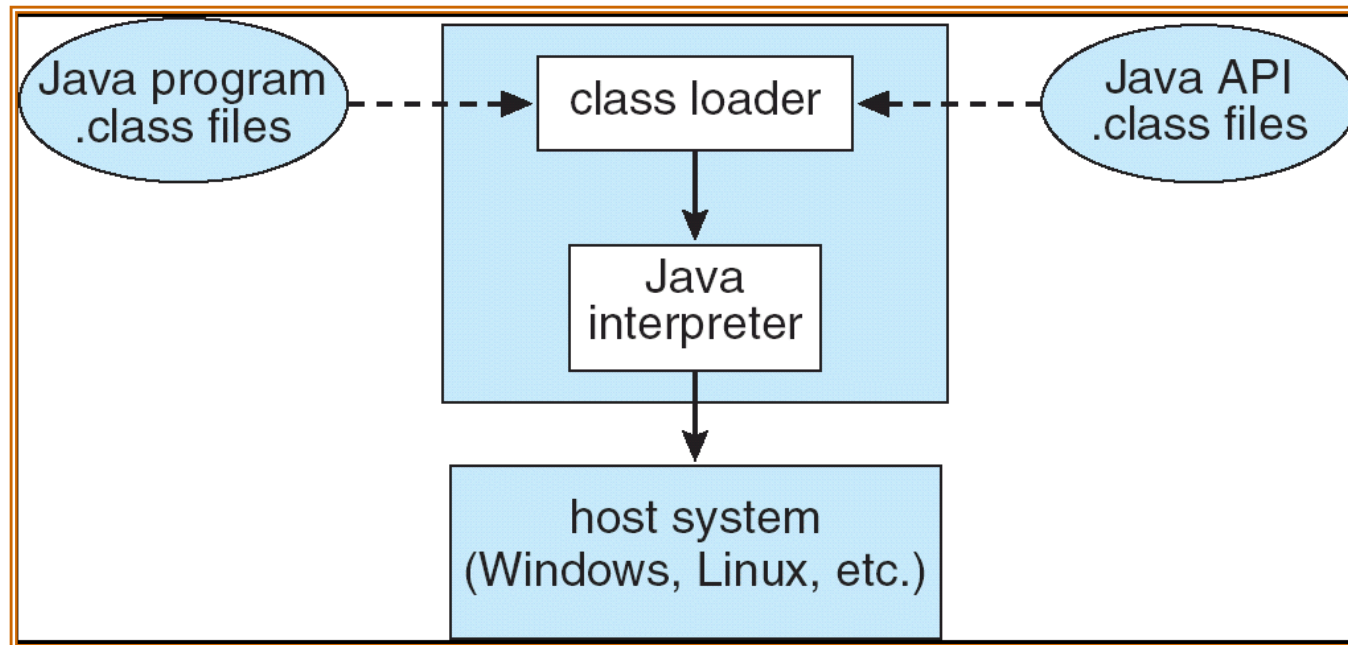
VMware Architecture



VM看到的硬體都
虛擬出來的

Native execution

The Java Virtual Machine



Java programs: the source code

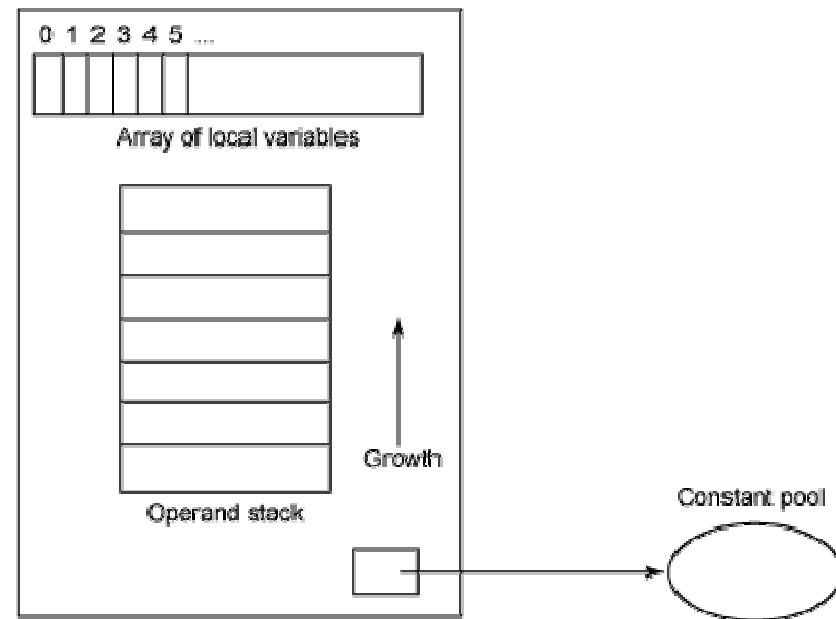
Byte code: the compiled binary for JVM

JVM or java runtime: a hardware-independent virtual machine

*Non-native execution

Java Bytecode

```
iload_1  
iload_2  
iadd  
istore_3
```



A virtual machine?



Quiz

The virtual machine approach is suitable to which one(s) of the following scenarios?

1. OS development
2. Cloud computing
3. Performance-critical gaming
4. Writing an application for heterogeneous hardware platforms

Summary: Virtual Machines

- Virtualizes hardware
- Pros
 - Guest operating systems run without modifications
 - Perfect resource partition and fault isolation
- Cons
 - Inefficient mapping between emulated hardware and the underlying hardware
 - Hard to implement hardware virtualization

Summary: Microkernel

- Provide “a minimal set of kernel primitives”
- Pros
 - Efficiently translation from kernel services to hardware operations
 - Stable, extendible, secure, portable
- Cons
 - Guest OS needs to be modified to run over a microkernel
 - Frequent mode switches
 - High message-passing overhead

Review

- Simple structure
 - Pros: simple, cons: poorly structured
- Monolith
 - Pros: efficient, cons: not scalable
- Layered
 - Pros: more structured, cons: difficult to comply with
- Microkernel
 - Pros: robust and scalable, cons: inefficient
- Virtual machine
 - Pros: perfect resource isolation, cons: possibly inefficient mapping from VM to host hardware

End of Chapter 2