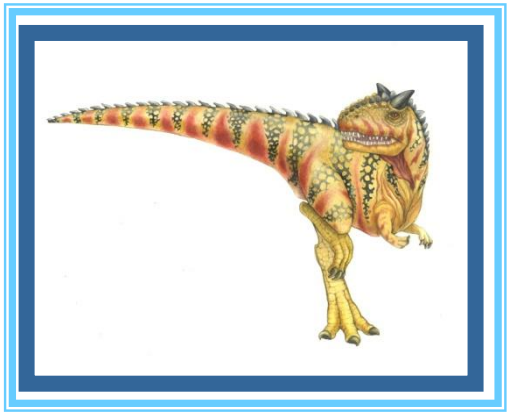
Chapter 2: Operating-System Structures



Chapter 2: Operating-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
- Operating System Generation
- System Boot

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 explain how operating systems are installed and customized and how they boot

2.1 Operating System Services

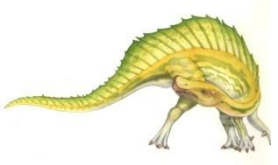
- One set of operating-system services provides functions that are helpful to the user:
  - User interface - Almost all operating systems have a user interface (UI)
    - Varies between Command-Line Interface (CLI), Graphics User Interface (GUI), Batch
  - Program execution - The system must be able 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 and delete them, search them, list file Information, permission management.

Operating System Services (Cont.)

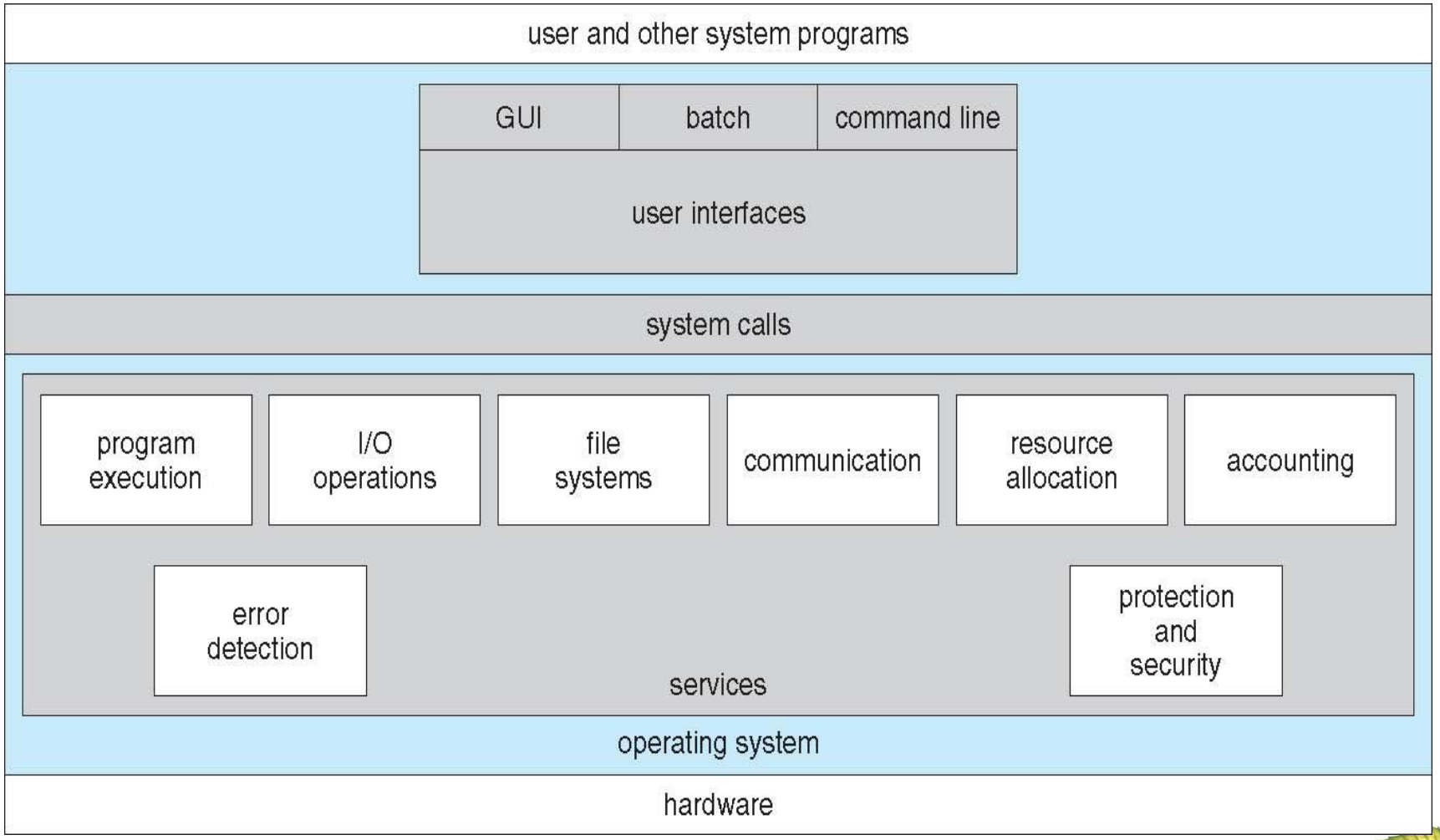
- One set of operating-system services provides functions that are helpful to the user (Cont):
  - Communications – Processes may exchange information, on the same computer or between computers over a network
    - Communications may be via shared memory or through message passing (packets moved by the 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

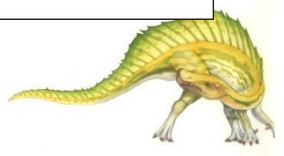
Operating System Services (Cont.)

- 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,mainmemory, 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

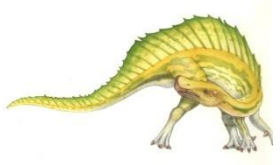




2.2 User Operating System Interface

- Operating System Interface
  - Command Interface （命令接口）
  - Program Interface （程序接口，system call）

- User interface，命令接口- Almost all operating systems have a user interface (UI)
  - Command-Line Interface (CLI)，命令行用户接口，文本界面
  - Graphics User Interface (GUI), 图形用户接口
  - More?



User Operating System 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
  - Linux、UNIX

Bourne Shell Command Interpreter

Terminal

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-vm)-(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-vm)-(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-vm)-(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      15Jun0718days      1         /usr/bin/ssh-agent -- /usr/bi
n/d
root     pts/3        15Jun07           18      4  w
root     pts/4        15Jun0718days                w
(root@pbg-nv64-vm)-(14/pts)-(16:07 02-Jul-2007)-(global)
-(/var/tmp/system-contents/scripts)#
```

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 at 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)
  - Linux

The Mac OS X GUI

Touchscreen(触摸屏) Interfaces

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

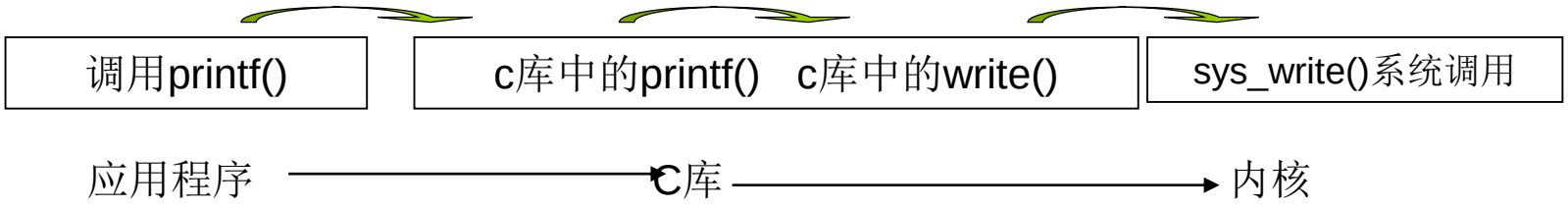
2.3 System Calls（系统调用）、程序接口

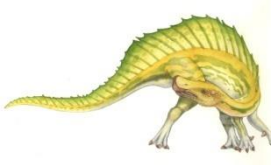
- System Calls： Programming interface to the services provided by the OS
- System calls are the programming interface between processes and the OS kernel.
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather 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)
- Why use APIs rather than system calls?

(Note that the system-call names used throughout this text are generic)

系统调用、API和C库

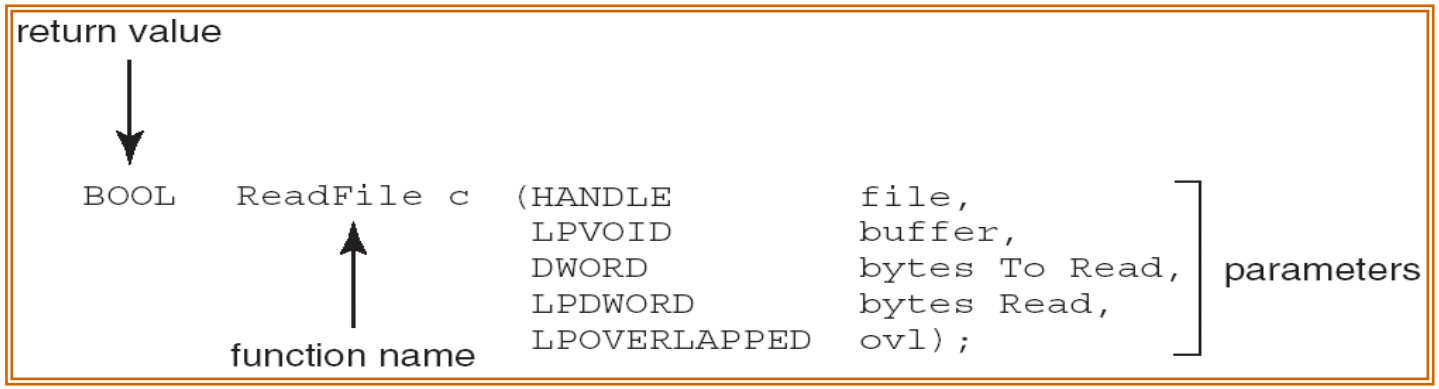
- 应用编程接口(API)其实是一组函数定义，这些函数说明了如何获得一个给定的服务；而系统调用是通过软中断向内核发出一个明确的请求，每个系统调用对应一个封装例程（wrapper routine，唯一目的就是发布系统调用）。一些API应用了封装例程。
  - API还包含各种编程接口，如：C库函数、OpenGL编程接口等
- 系统调用的实现是在内核完成的，而用户态的函数是在函数库中实现的





Example of Standard API

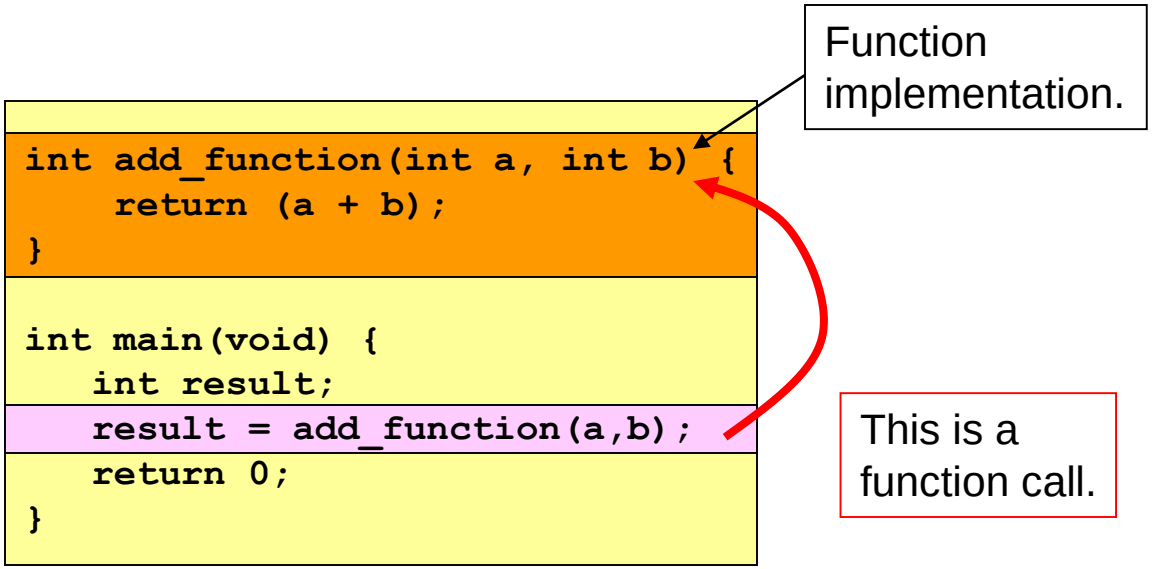
- Consider the ReadFile() function in the
- Win32 API—a function for reading from a file

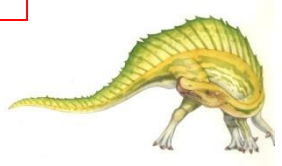


- A description of the parameters passed to ReadFile()
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read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

System Calls

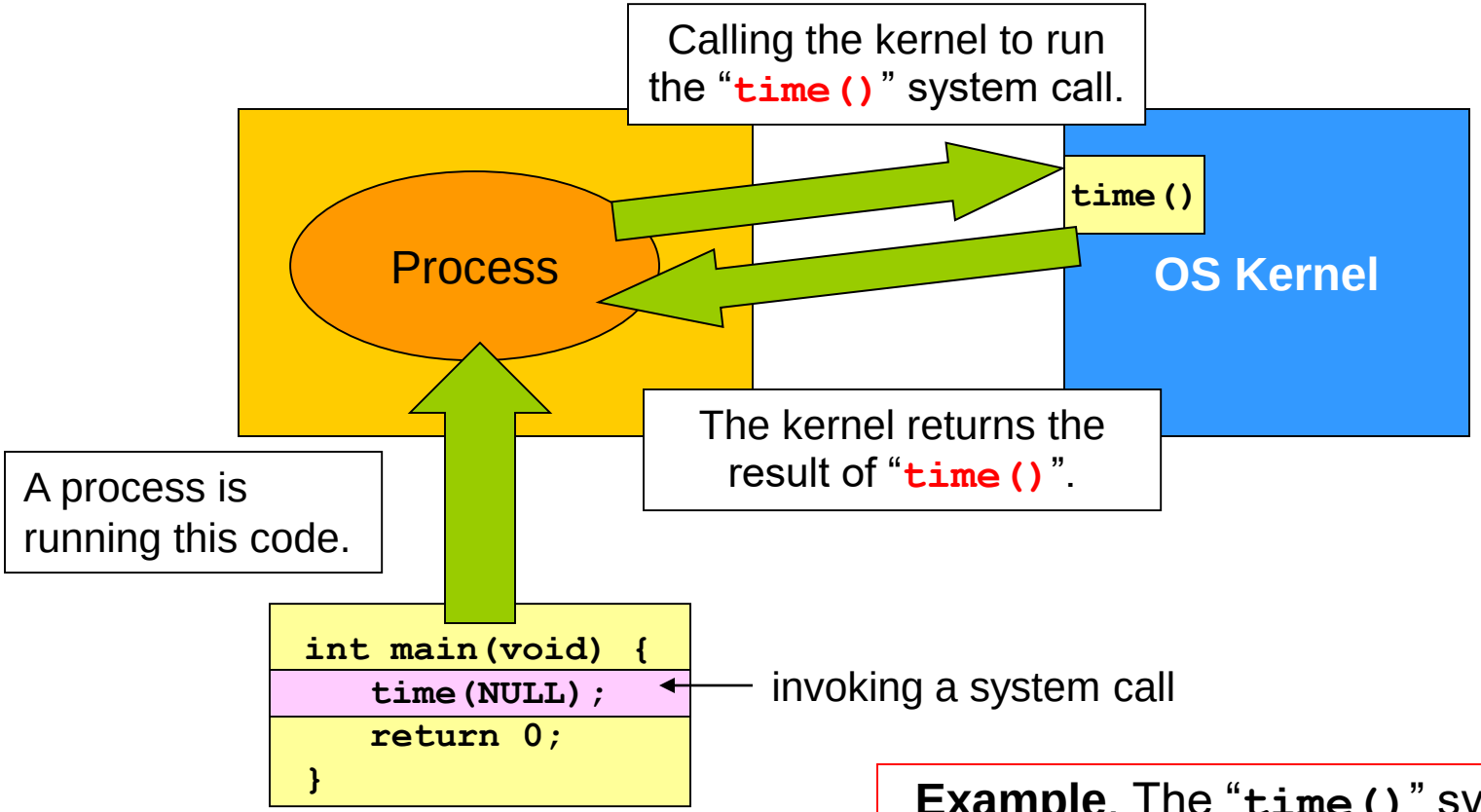
- What is a system call?
  - Informally, a system call is similar to a function call, but…
  - The function implementation is inside the OS, or we name it the OS kernel.





System Calls

- What is a system call?



Example. The "`time()`" system call.

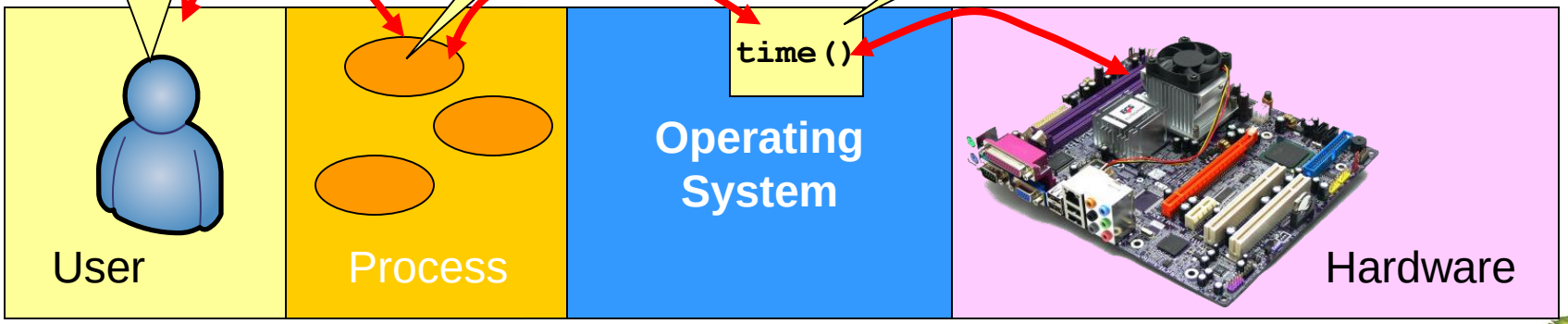
System Calls

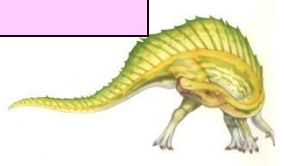
I want to know what time it is. So, I write and run this program.

```
int main(void) {
    time(NULL);
    return 0;
}
```

I come to serve! I'll call "**time()**" for you, master.

OK. I'll ask the **hardware clock** for the current time.

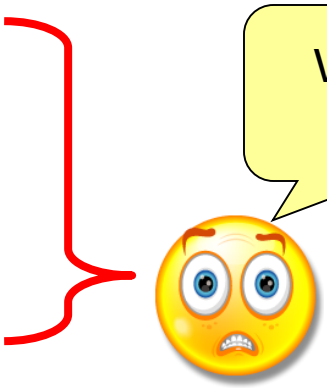


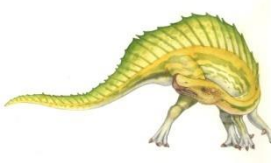


System Calls

- When do we know if a “function” is a system call?
  - Read the man page “`syscalls`” under Linux.
- Let’s guess: who are system calls?

Name	System Call?
`printf()`	No
`malloc()`	No
`fopen()`	No
`fclose()`	No





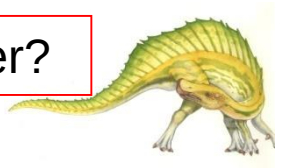
System Call vs Library Function Call

- If they are not system calls, then they are function calls!
- Take `fopen()` as an example.
  - `fopen()` invokes the system call `open()`.
  - So, why people invented `fopen()`?
  - Because `open()` is too primitive and is not programmer-friendly!

The following two calls have the same effect, but…

Function call	`fopen("hello.txt", "w");`
System call	`open("hello.txt", O_WRONLY | O_CREAT | O_TRUNC, 0666);`

As a programmer, which one do you prefer?

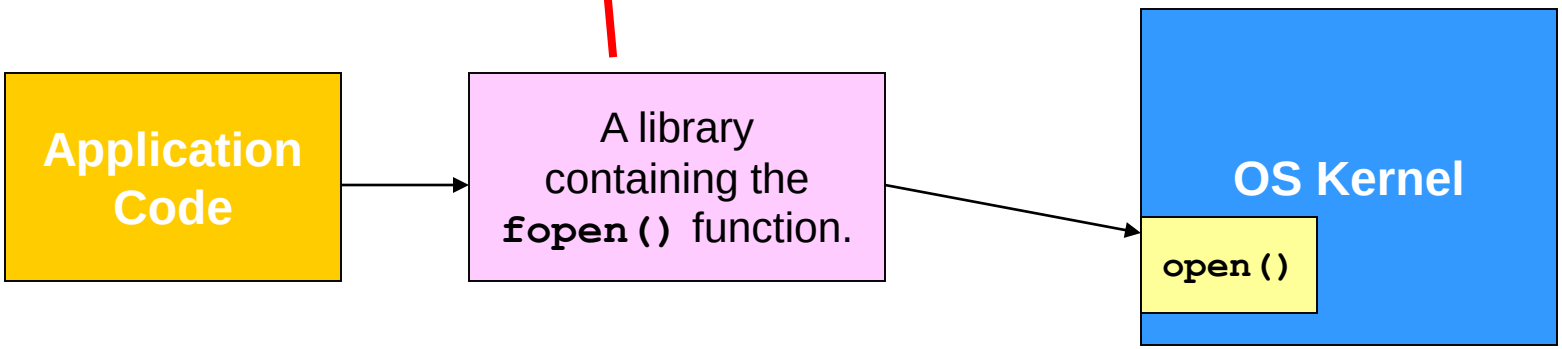


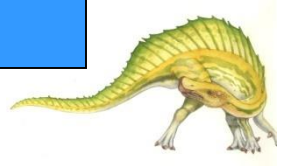
System Call vs Library Function Call

- Those functions are usually packed inside an object called the library file.

```
$ ls /lib/libc*
/lib/libc-2.9.so  ......
$ _
```

The C standard library file.

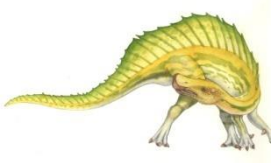




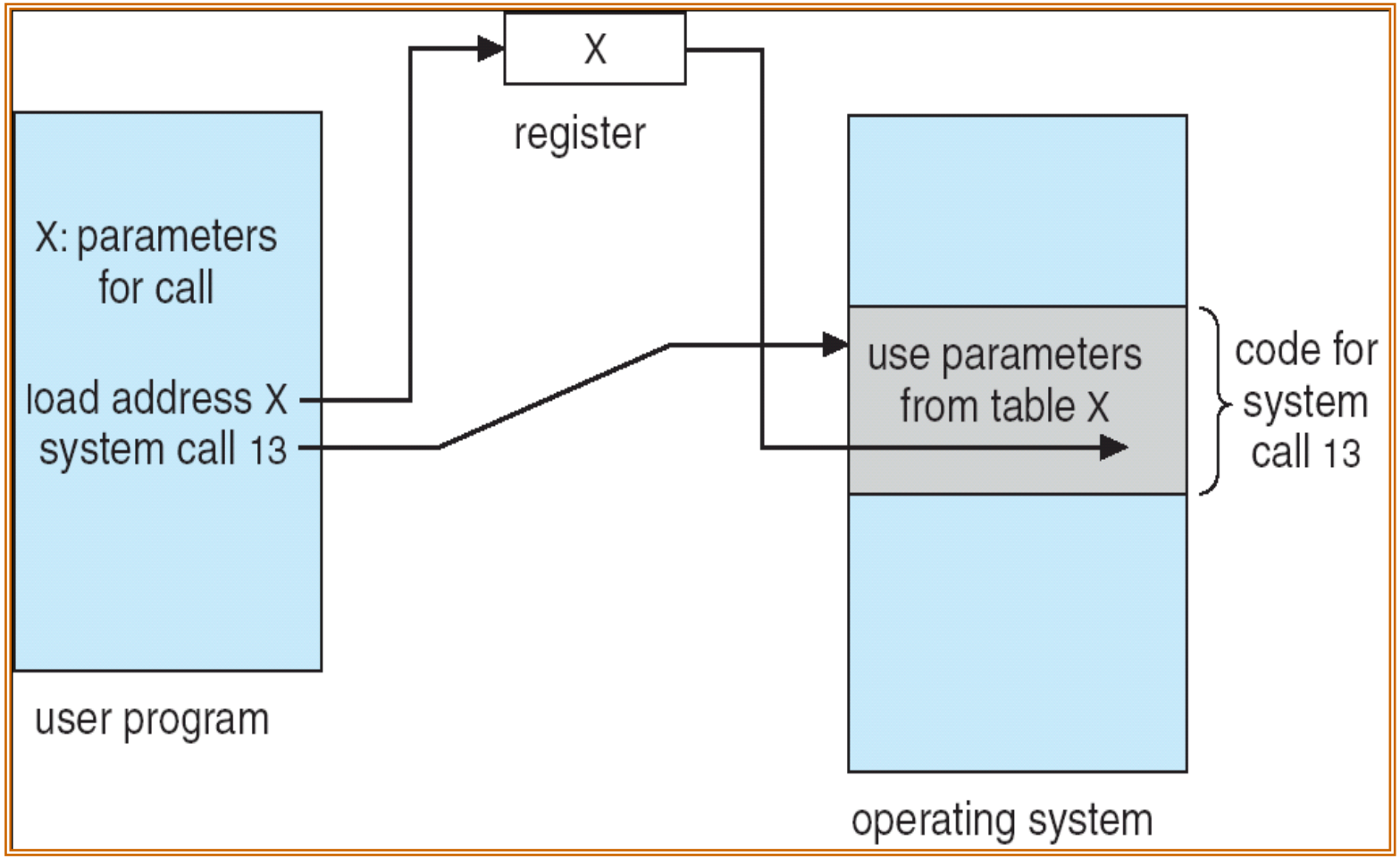
System Call Parameter Passing

三种传参方法：寄存器，内存，栈

- 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
  - Simplest: pass the parameters in *registers*
    - In some cases, may be more parameters than registers
  - 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
  - Parameters placed, or *pushed,* onto the *stack* by the program and *popped* off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed



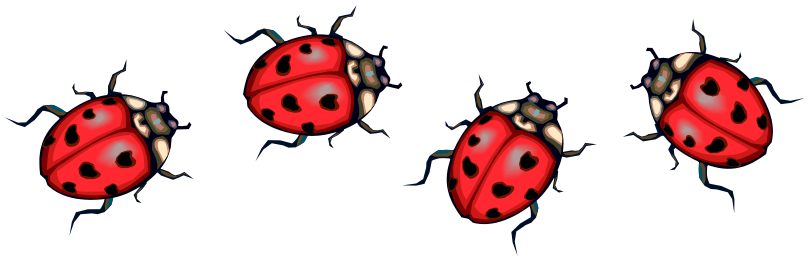
Parameter Passing via Table

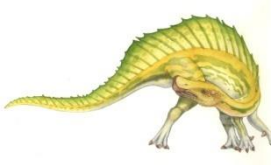


2.4 Types of System Calls

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

系统调用





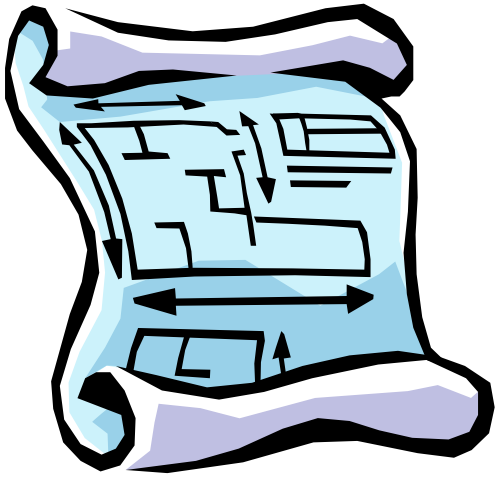
Examples of Windows and Unix/Linux System Calls

	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()

Win32 CreateProcess :example\newproc-win32.c

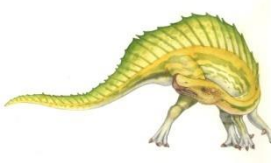
Solaris 10 dtrace Following System Call

```
# ./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
```



Linux的ptrace动态跟踪工具；

Linux的strace命令显示系统调用

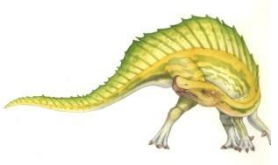


2.5 System Programs

- System programs provide a convenient environment for program development and execution. The can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs

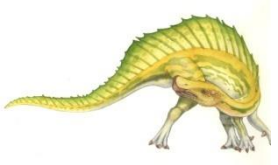
如：Linux/Unix各种命令（程序）

- 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

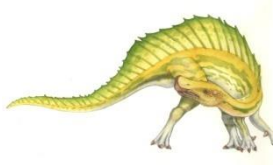


System Programs (cont'd)

- File modification
  - Text editors to create and modify files
  - 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

2.6 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



操作系统的设计问题

- 操作系统设计有着不同于一般应用系统设计的特征：
  - 复杂程度高
  - 研制周期长
  - 正确性难以保证

  Windows 2000开发的艰辛与规模
    - 最早Uinx是1400行代码；
    - Windows xp有4000万行代码；
    - fedroa core有2亿多行代码，Linux kernel 4.10有2千多万行代码。
- 解决途径：
  - 良好的操作系统结构
  - 先进的开发方法和工程化的管理方法（软件工程）
  - 高效的开发工具

Operating System Design and Implementation (Cont.)

- Important principle to separate

  Policy策略: What will be done?
  Mechanism机制: How to do it?

- Mechanisms determine how to do something, policies decide what will be done
  - The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later

操作系统的设计考虑

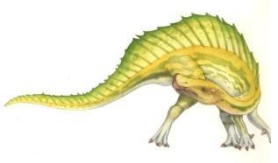
- 功能设计：操作系统应具备哪些功能
- 算法设计：选择和设计满足系统功能的算法和策略，并分析和估算其效能
- 结构设计：选择合适的操作系统结构
- 按照系统的功能和特性要求，选择合适的结构，使用相应的结构设计方法将系统逐步地分解、抽象和综合，使操作系统结构清晰、简单、可靠、易读、易修改，而且使用方便，适应性强

Implementation

- Much variation
  - Early OSes in assembly language
  - Then system programming languages like Algol, PL/1
  - Now C, C++
- Actually usually a mix of languages
  - Lowest levels in assembly
  - Main body in C
  - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
- More high-level language easier to port to other hardware
  - But slower
- Emulation can allow an OS to run on non-native hardware

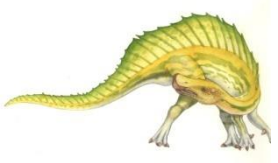
2.7 Operating-System Structure

- Simple Structure简单结构
- Layered Approach层次化结构
- Microkernel 微内核
- Monolithic Kernels Structure单/宏内核结构
- Modules 模块
- Virtual Machines 虚拟机

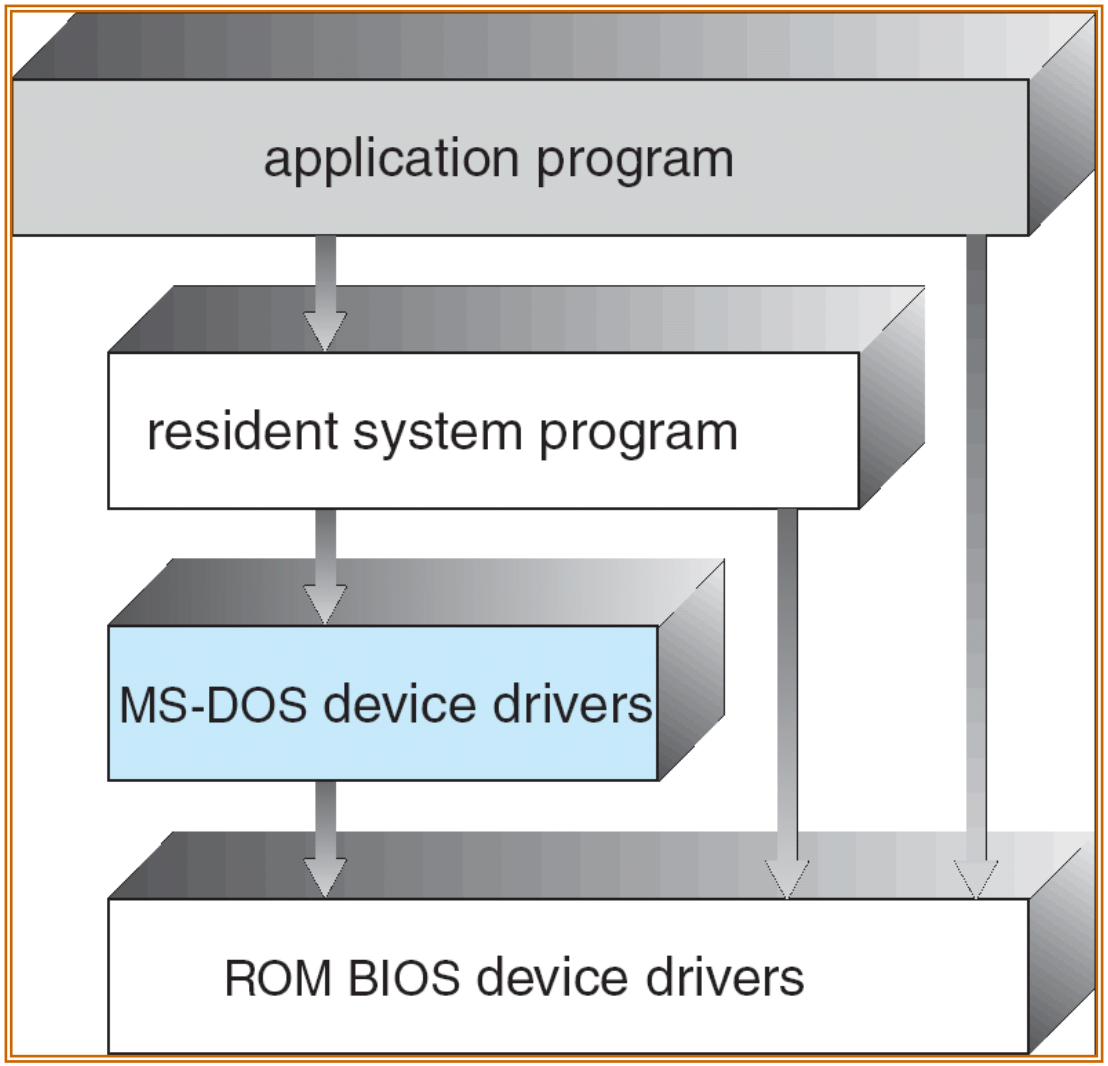


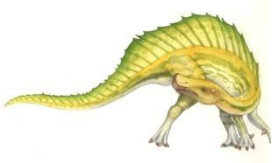
Simple Structure（简单结构）

- 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



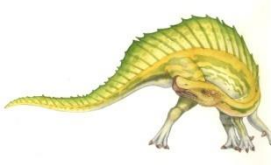
MS-DOS Structure



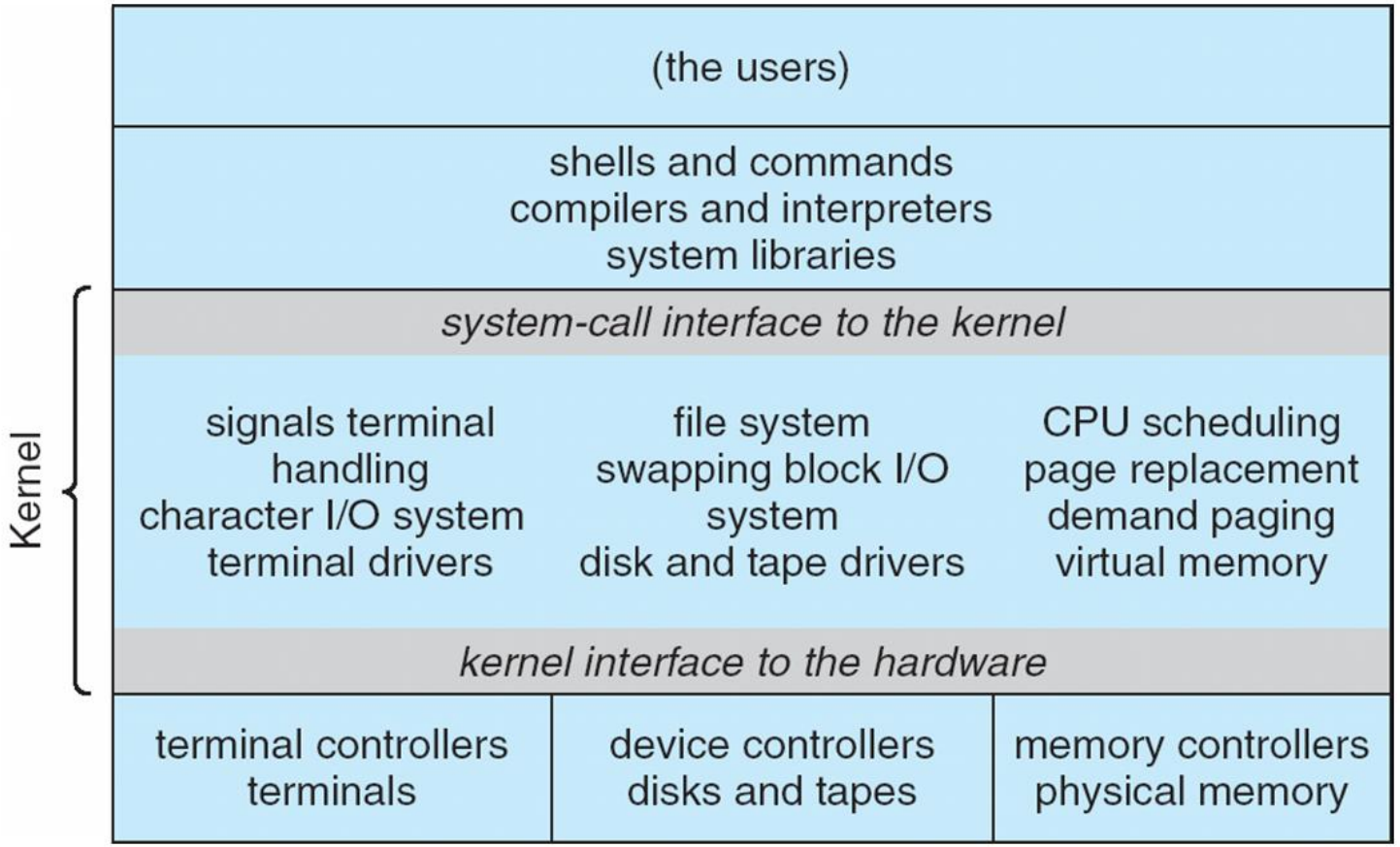


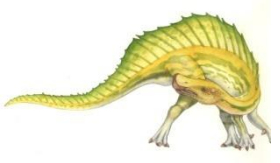
UNIX

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts
  - Systems programs
  - 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
- UNIX、Linux单内核结构



Traditional UNIX System Structure

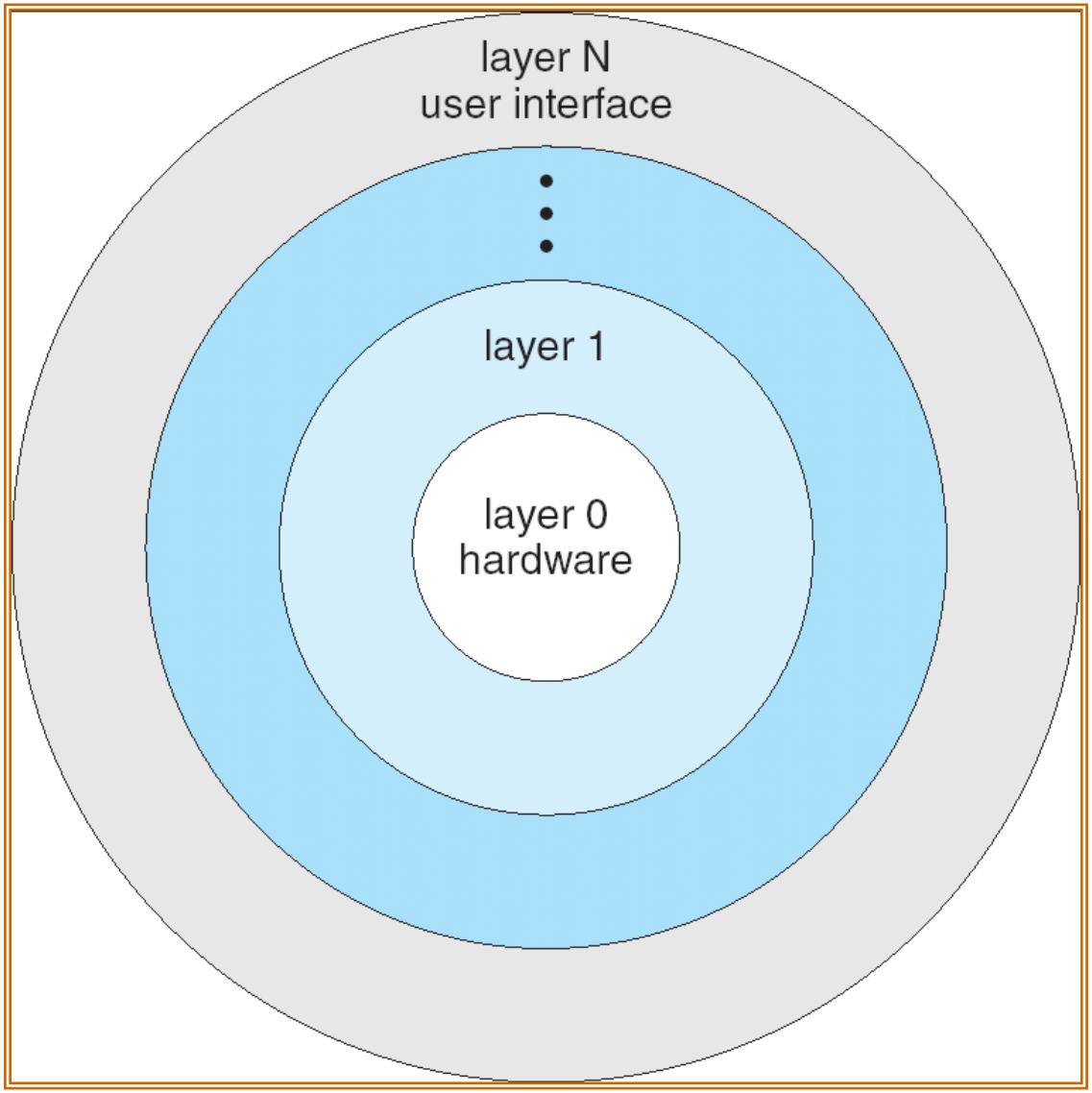


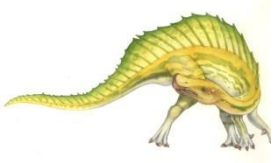


Layered Approach（层次结构）

- 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

Layered Operating System





Microkernel System Structure （微内核结构）

- Microkernels — only the most elementary functions are implemented directly in a central kernel— the microkernel. All other functions are delegated to autonomous processes that communicate with the central kernel via clearly defined communication interfaces
- 由下面两大部分组成 ：
  - “微”内核
  - 若干服务
- Moves as much from the kernel into “*user*” space
- Communication takes place between user modules using message passing

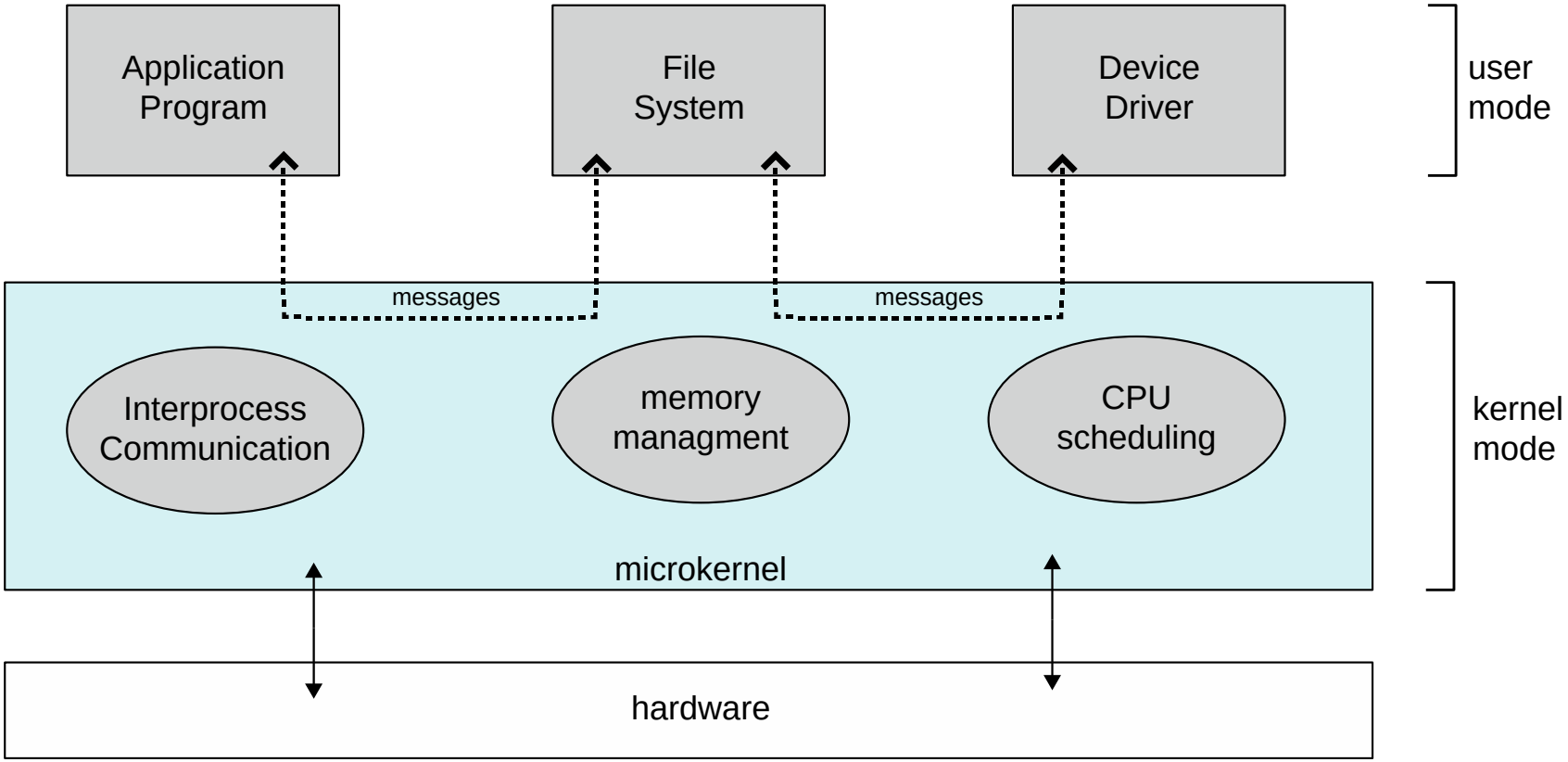
(1) **微内核**：这种范型中，只有最基本的功能直接由中央内核（即微内核）实现。所有其他的功能都委托给一些独立进程，这些进程通过明确定义的通信接口与中心内核通信。例如，独立进程可能负责实现各种文件系统、内存管理等。（当然，与系统本身的通信需要用到最基本的内存管理功能，这是由微内核实现的。但系统调用层次上的处理则由外部的服务器进程实现。）理论上，这是一种很完美的方法，因为系统的各个部分彼此都很清楚地划分开来，同时也迫使程序员使用“清洁的”程序设计技术。这种方法的其他好处包括：动态可扩展性和在运行时切换重要组件。但由于在各个组件之间支持复杂通信需要额外的CPU时间，所以尽管微内核在各种研究领域早已经成为活跃主题，但在实用性方面进展甚微。

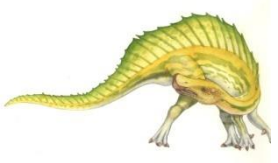
Microkernel System Structure （微内核结构）

- 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

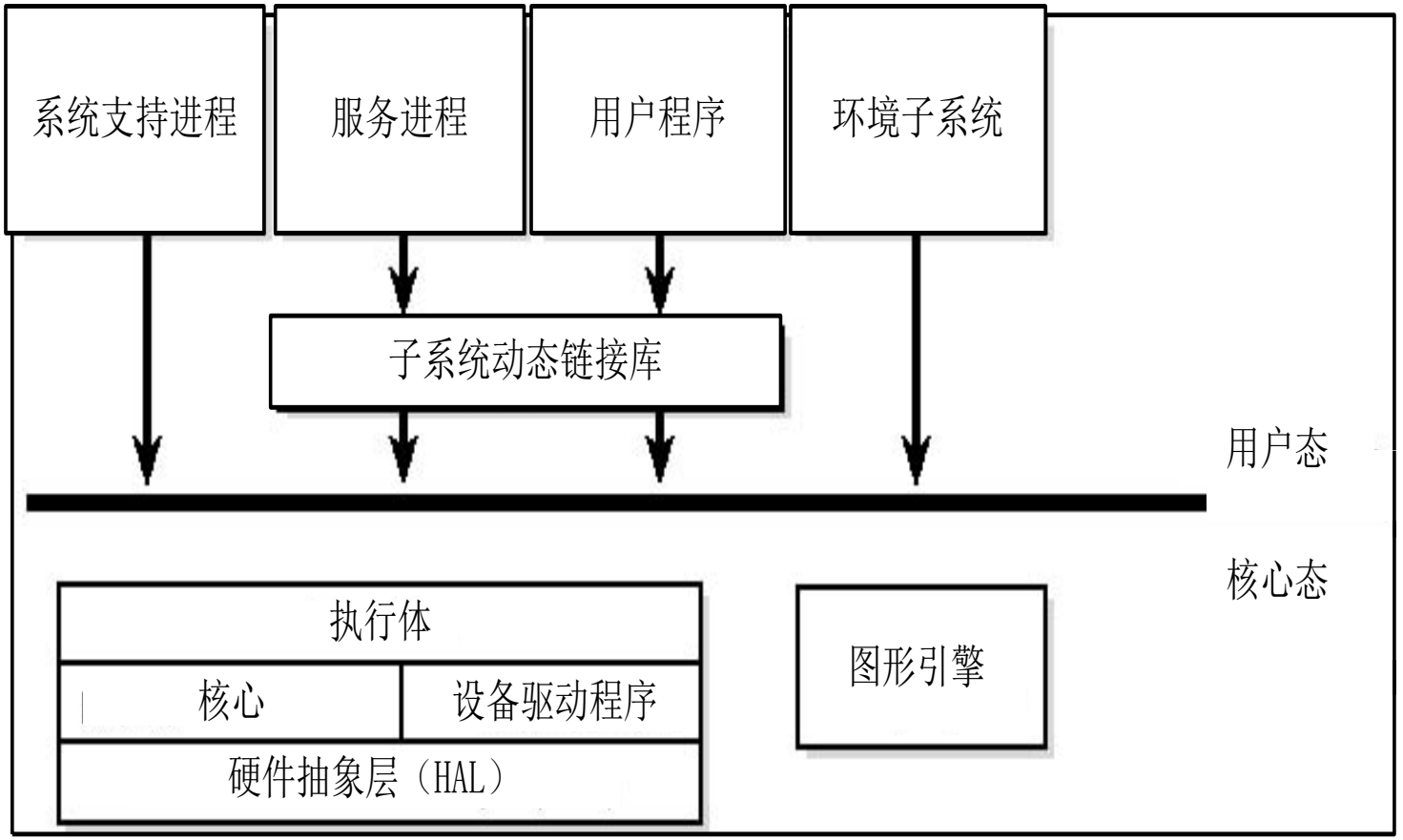
- Windows NT ... Windows 8、Windows 10、Mac OS
- L4 ： http://www.l4hq.org/

Microkernel System Structure

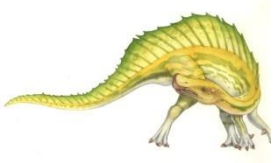




Windows Kernel



Windows NT 4.0 起，采用 microkernel 的架构

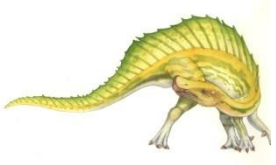


编程　宏内核

Monolithic Kernels Structure（单/宏内核结构）

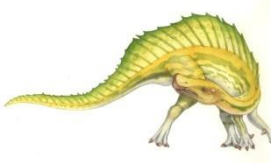
(2) 宏内核：与微内核相反，宏内核是构建系统内核的传统方法。在这种方法中，内核的全部代码，包括所有子系统（如内存管理、文件系统、设备驱动程序）都打包到一个文件中。内核中的每个函数都可以访问内核中所有其他部分。如果编程时不小心，很可能会导致源代码中出现复杂的嵌套。

- Monolithic Kernels： the entire code of the kernel — including all its subsystems such as memory management, filesystems, or device drivers — is packed into a single file. Each function has access to all other parts of the kernel;
- Earliest and most common OS architecture (UNIX,MS-DOS)
- Every component of the OS is contained in the Kernel
- Examples: OS/360, VMS and Linux



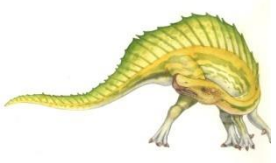
Monolithic Kernels

- Advantages:
  - highly efficient because of direct communication between components
- Disadvantages:
  - Difficult to isolate source of bugs and other errors
  - Hard to modify and maintain
  - Kernel gets bigger as the OS develops.



Modules（模块）

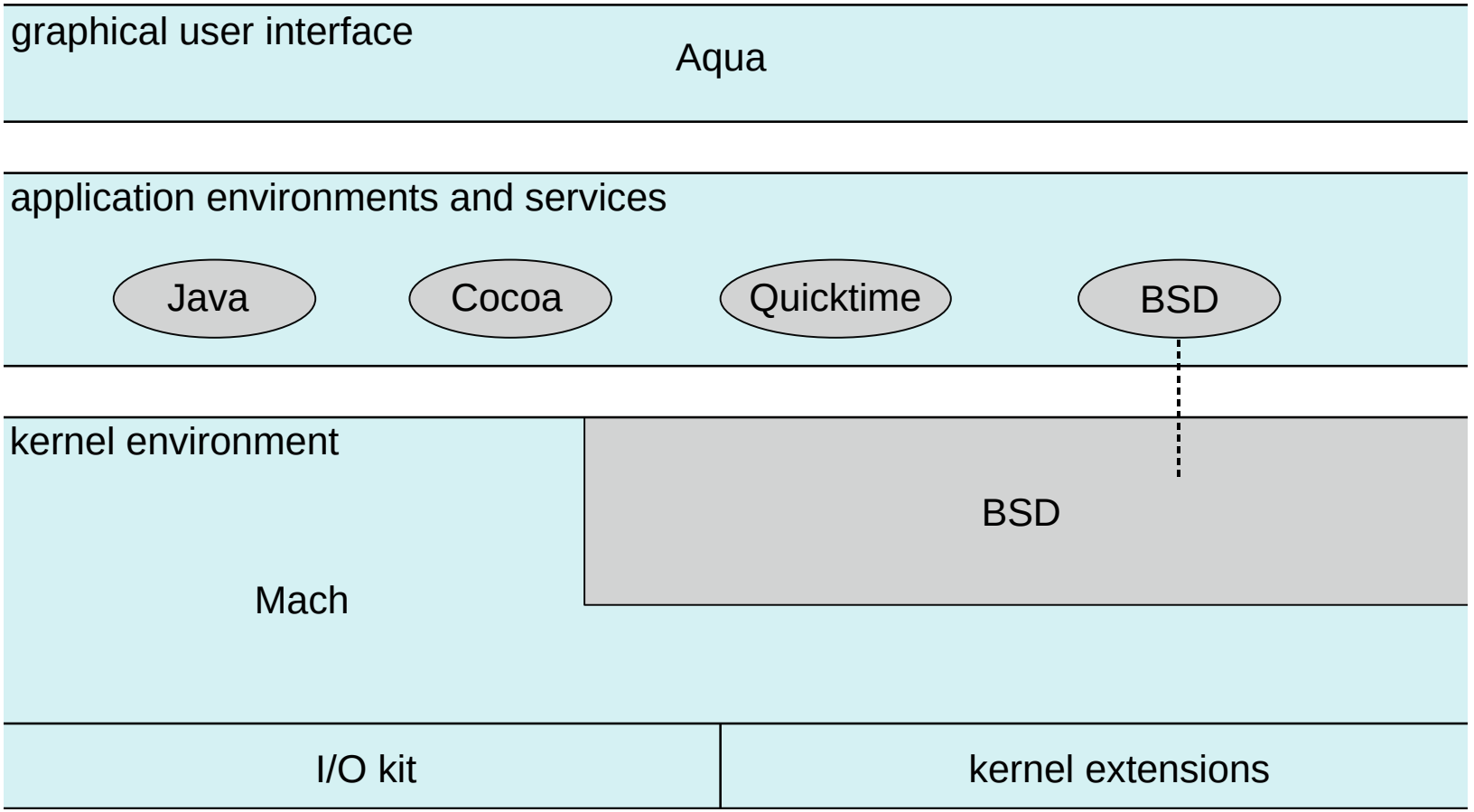
- Most modern operating systems implement kernel modules （内核模块）
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible
- Linux , Solaris, etc

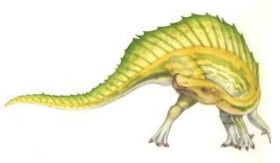


Hybrid Systems（混合系统）

- Most modern operating systems actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystem *personalities*
- Apple Mac OS X hybrid, layered, Aqua UI plus Cocoa programming environment
  - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called kernel extensions)

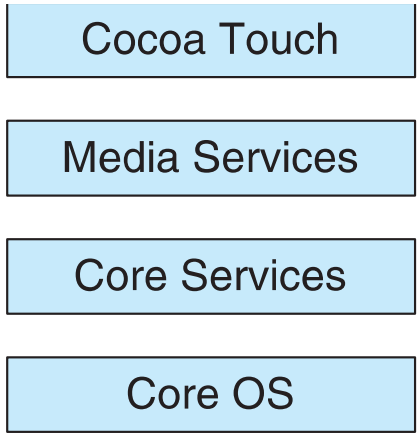
Mac OS X Structure





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



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 than runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

Android Architecture

Application Framework

Libraries

SQLite

openGL

surface manager

media framework

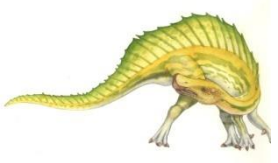
webkit

libc

Android runtime

Core Libraries

Dalvik virtual machine

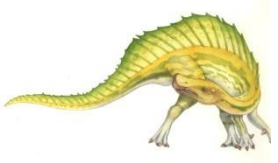


Virtual Machines

- 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 host creates the illusion that a process has its own processor and (virtual memory)
- Each guest provided with a (virtual) copy of underlying computer
- *virtual machine software*： VMWARE、 VirtualBox 、Microsoft Virtual PC
- **OpenStack云计算平台**

Virtual Machines (Cont.)

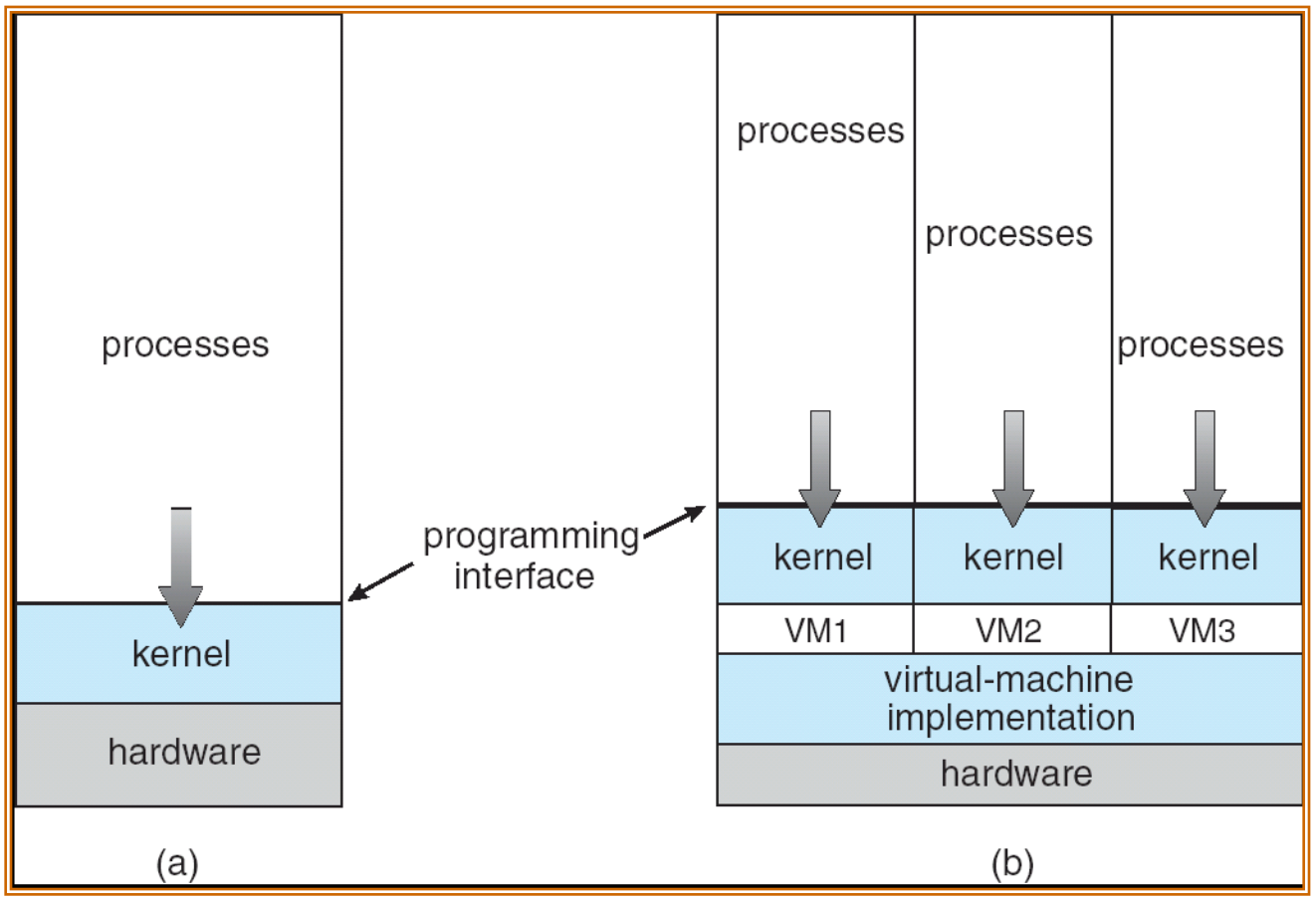
- The resources of the physical computer are shared to create the virtual machines
  - CPU scheduling can create the appearance that users have their own processor
  - Spooling and a file system can provide virtual card readers and virtual line printers
  - A normal user time-sharing terminal serves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



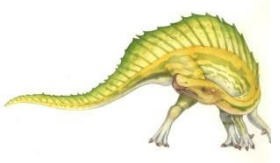
Virtual Machines (Cont.)

- 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

Virtual Machines (Cont.)

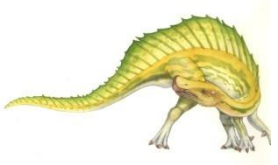


(a) Nonvirtual machine (b) virtual machine

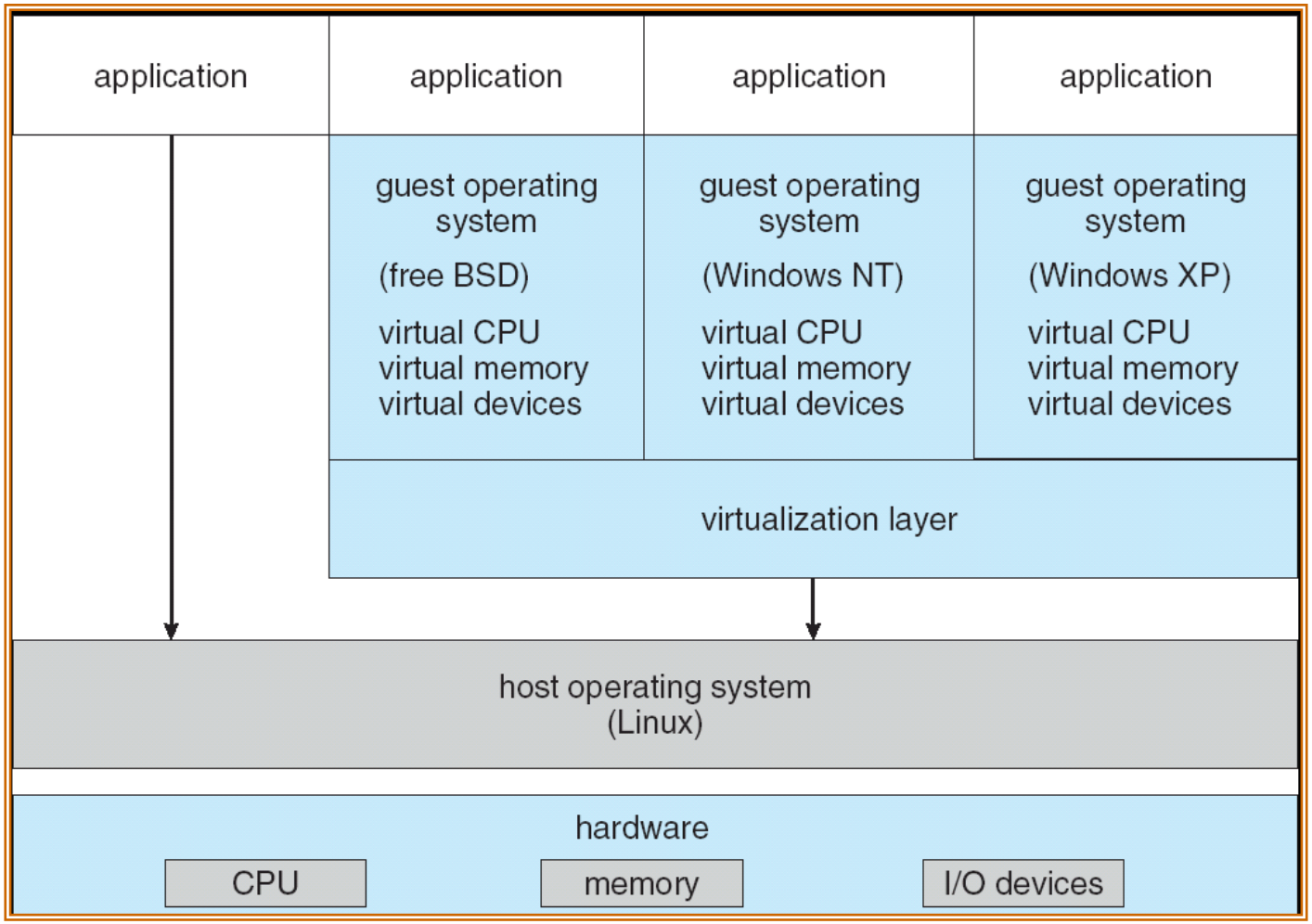


Para-virtualization（“半虚拟化”）

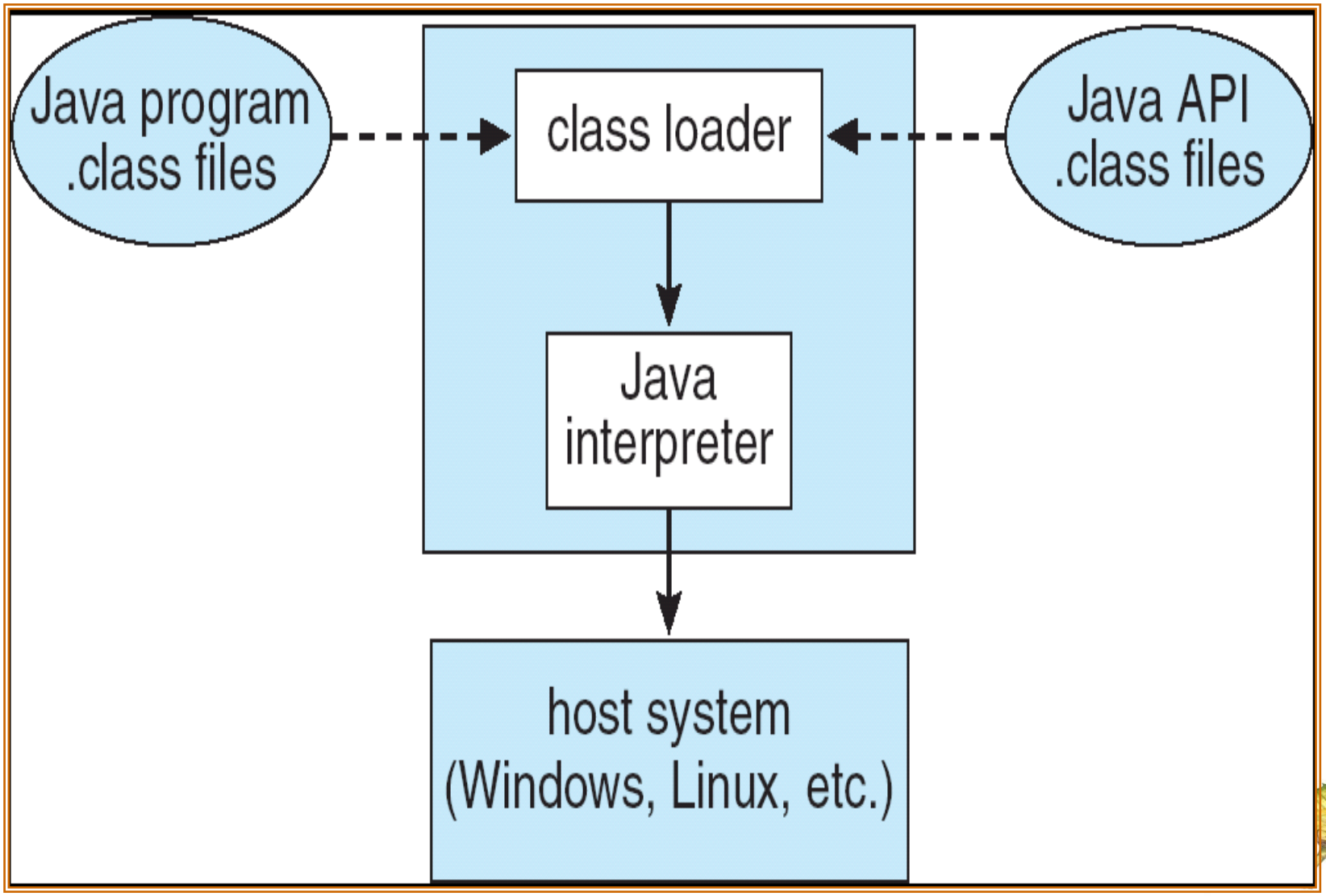
- Presents guest with system similar but not identical to hardware
- Guest must be modified to run on paravirtualized hardwareF
- Guest can be an OS, or in the case of Solaris 10 applications running in containers



VMware Architecture



The Java Virtual Machine



2.9 Operating System Generation

- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site
- SYSGEN program obtains information concerning the specific configuration of the hardware system
- *Booting* – starting a computer by loading the kernel
- *Bootstrap program* – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution

2.10 System Boot

- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – bootstrap loader, locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where boot block at fixed location loads bootstrap loader
  - When power initialized on system, execution starts at a fixed memory location
    - Firmware used to hold initial boot code

WINDOWS 启动

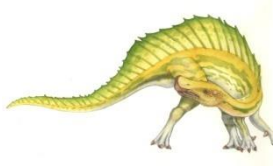
- ROM中POST（Power On Self-Test）代码
- BIOS/EFI（Extended Firmware Interfacte）
- MBR(Main Boot Record)
- 引导扇区(Boot sector)
- NTLDR/WinLoad
- NTOSKRNL/HAL/BOOTVID/KDCOM
- SMSS.EXE
- WinLogon.EXE

习题分析

- 习题分析及练习\ex_ch1-2.doc

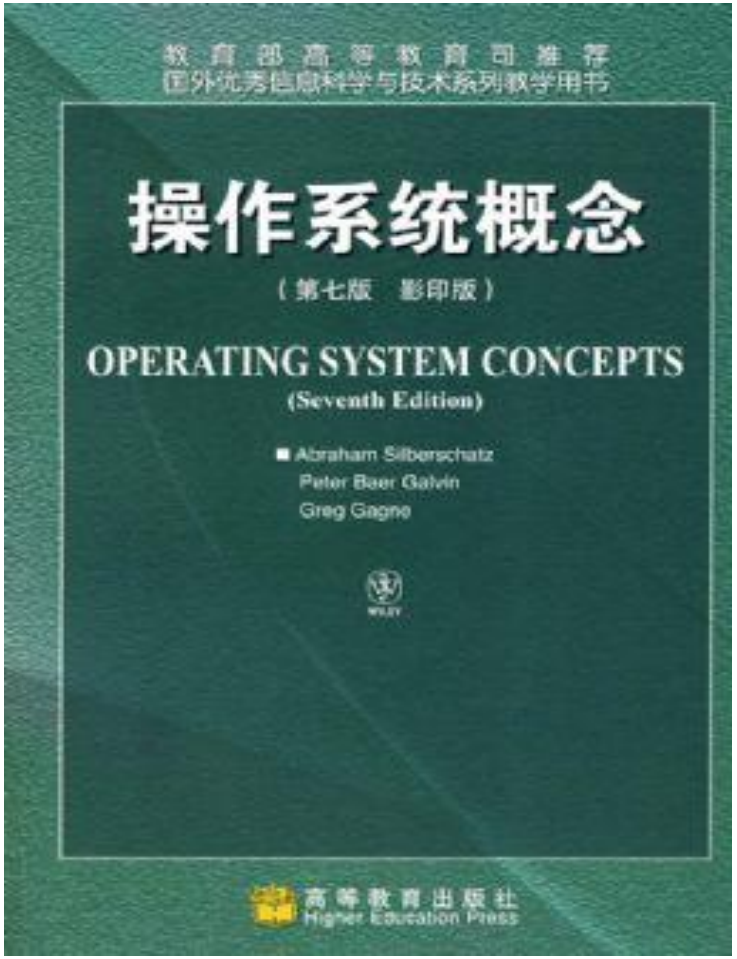
Homework

- 在“学在浙里”中，请按时完成



Reading Assignments

- Read for next week:
  - Chapters 3 of the text book:



End of Chapter 2