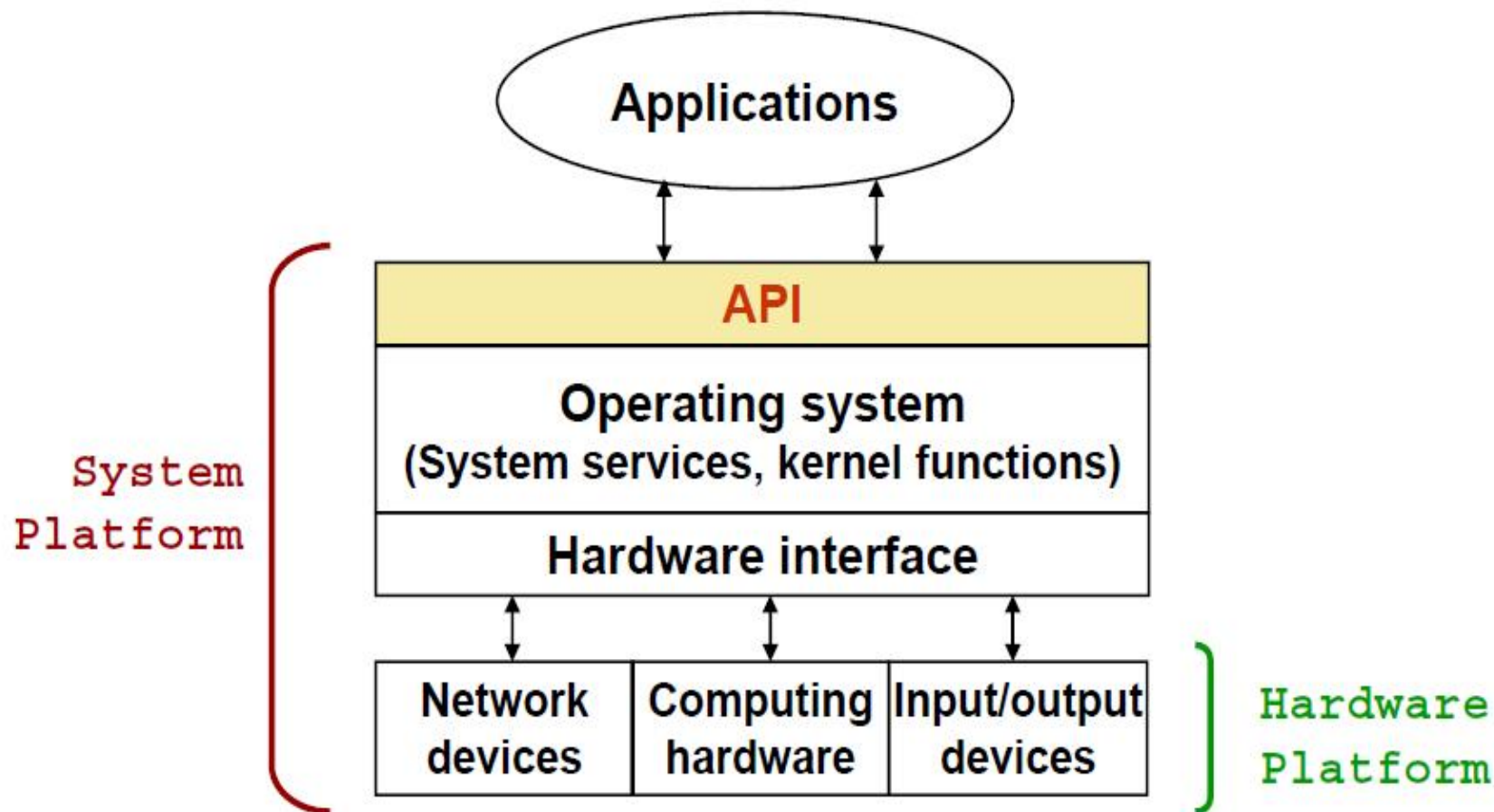


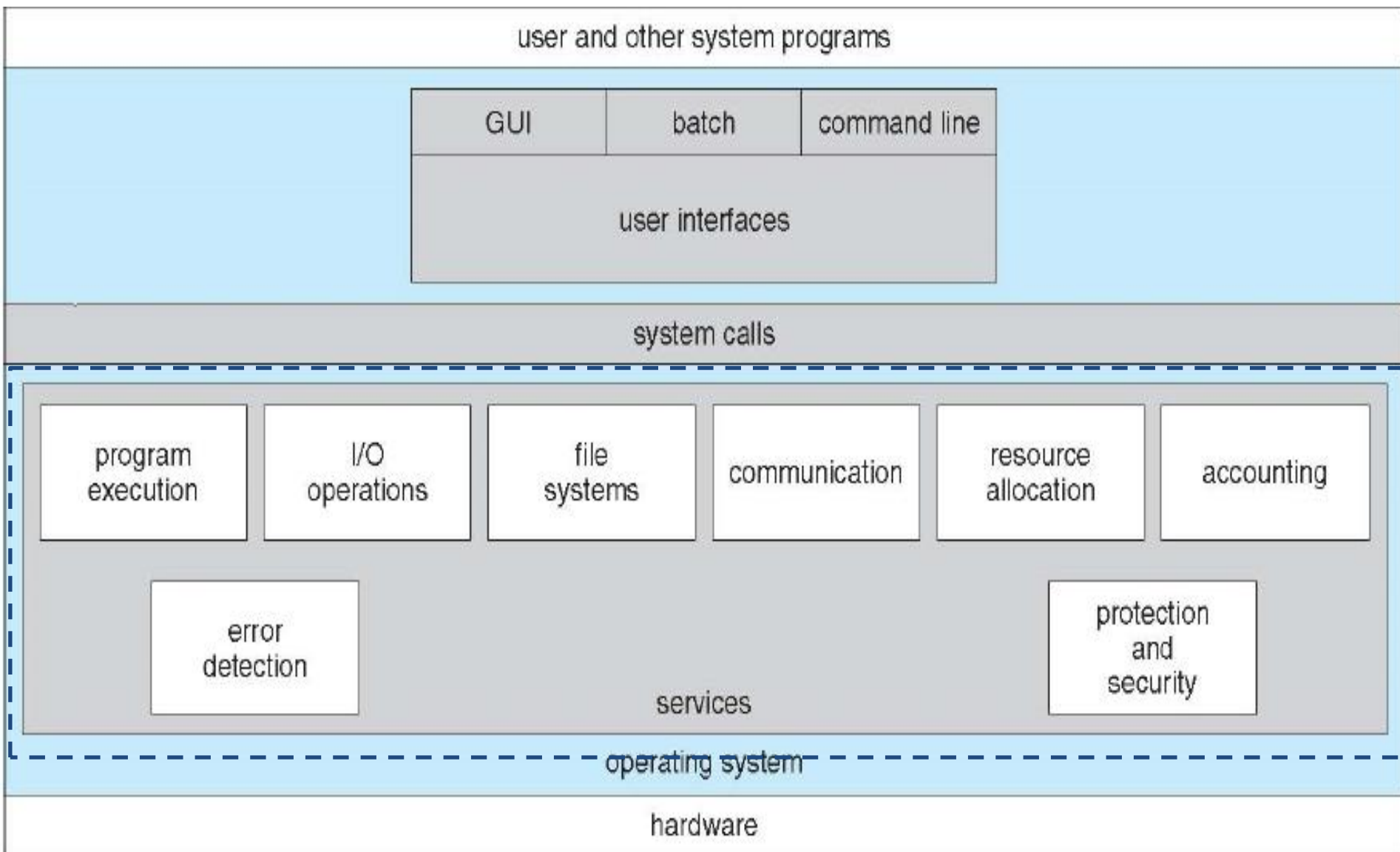
# 第二章 操作系统结构

1. 操作系统服务
2. 操作系统用户界面
3. 系统调用 (System Call)
4. 系统程序 (System Program)
5. 操作系统设计和实现
6. 操作系统的结构
7. 介绍虚拟机 (Virtual Machine)

## Simplified, Hierarchical View of a Computer System



# 第一节、操作系统服务



操作系统以**不同形式向程序和用户提供服务**  
系统服务提供对用户有用函数

### ● 基本服务

1. 用户界面：形式有命令行界面(CLI)、图形用户界面(GUI) 等
2. 程序执行：将程序载入内存并运行
3. I/O 操作：I/O可能涉及文件或设备
4. 文件系统操作：程序需要读写文件和目录
5. 通信：进程间可能需要交换信息
6. 错误检测：OS 需要知道可能出现的错误

- **增值服务：**通过共享计算机资源来提高效率：

7. 资源分配：多个作业并发运行时，系统为它们分配资源

8. 统计：需要记录哪些用户使用了多少和什么类型的资源

9. 保护和安全：用户可能需要控制信息的使用

- (1) 保护：确保所有对系统资源的访问受控

- (2) 安全：不受外界侵犯，延伸到外部I/O设备不受非法访问

## 第二节、操作系统用户界面

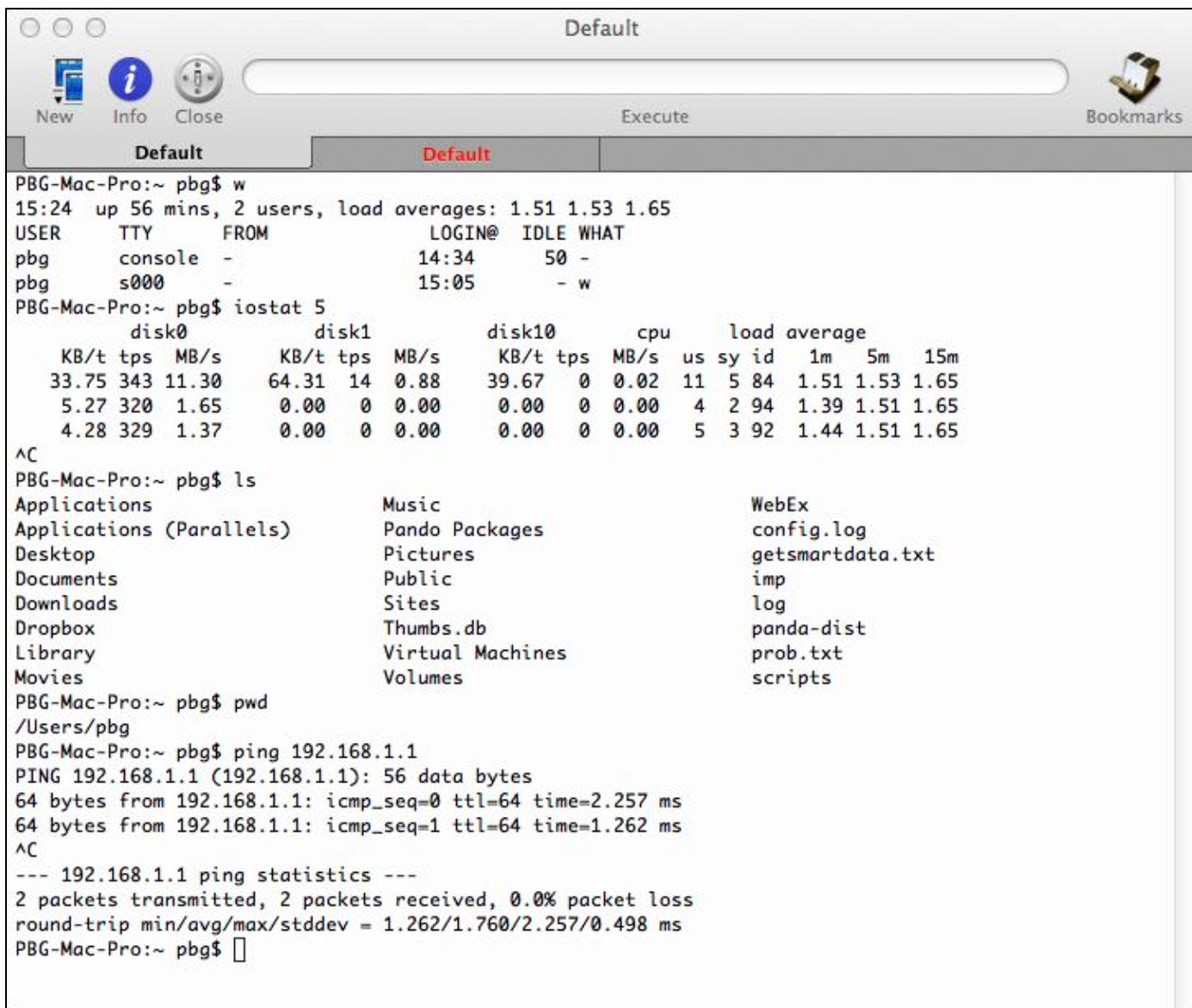


CLI (Command Line Interface) 允许用户直接输入操作系统完成的命令

有的在内核中实现，有的通过系统程序实现

有时有多种实现方式 – 外壳(shells)，主要作用是获取并执行用户指定的命令

- 有些命令是内置的，有些只是程序名



```

PBG-Mac-Pro:~ pbg$ w
15:24 up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER      TTY      FROM          LOGIN@  IDLE   WHAT
pbg       console -              14:34    50    -
pbg       s000    -              15:05    -    w
PBG-Mac-Pro:~ pbg$ iostat 5
            disk0      disk1      disk10      cpu      load average
      KB/t tps  MB/s    KB/t tps  MB/s    KB/t tps  MB/s  us sy id  1m  5m  15m
      33.75 343 11.30     64.31 14  0.88     39.67 0  0.02  11 5 84  1.51 1.53 1.65
       5.27 320  1.65       0.00 0  0.00       0.00 0  0.00   4 2 94  1.39 1.51 1.65
       4.28 329  1.37       0.00 0  0.00       0.00 0  0.00   5 3 92  1.44 1.51 1.65
^C
PBG-Mac-Pro:~ pbg$ ls
Applications          Music                  WebEx
Applications (Parallels)  Pando Packages       config.log
Desktop               Pictures              getsmartdata.txt
Documents             Public                imp
Downloads             Sites                 log
Dropbox               Thumbs.db             panda-dist
Library               Virtual Machines      prob.txt
Movies                Volumes               scripts
PBG-Mac-Pro:~ pbg$ pwd
/Users/pbg
PBG-Mac-Pro:~ pbg$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.257 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.262 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.262/1.760/2.257/0.498 ms
PBG-Mac-Pro:~ pbg$
  
```

```

C:\Windows\system32\cmd.exe

C:\Users\pfli>dir
驱动器 C 中的卷没有标签。
卷的序列号是 7088-C6D2

C:\Users\pfli 的目录

2015/07/09  13:44    <DIR>          .
2015/07/09  13:44    <DIR>          ..
2015/07/09  13:44    <DIR>          21E247D45E274BEAAA4D19A81203FE2A.TMP
2015/01/23  17:28    <DIR>          Contacts
2015/07/09  13:51    <DIR>          Desktop
2015/07/01  20:11    <DIR>          Documents
2015/07/12  09:47    <DIR>          Downloads
2015/05/17  09:35    <DIR>          Favorites
2015/01/23  17:28    <DIR>          Links
2015/01/23  17:28    <DIR>          Music
2015/01/23  17:28    <DIR>          Pictures
2015/01/23  17:28    <DIR>          Saved Games
2015/01/23  17:28    <DIR>          Searches
2015/01/23  17:28    <DIR>          Videos
      0 个文件              0 字节
     14 个目录 66,474,819,584 可用字节

C:\Users\pfli>
  
```

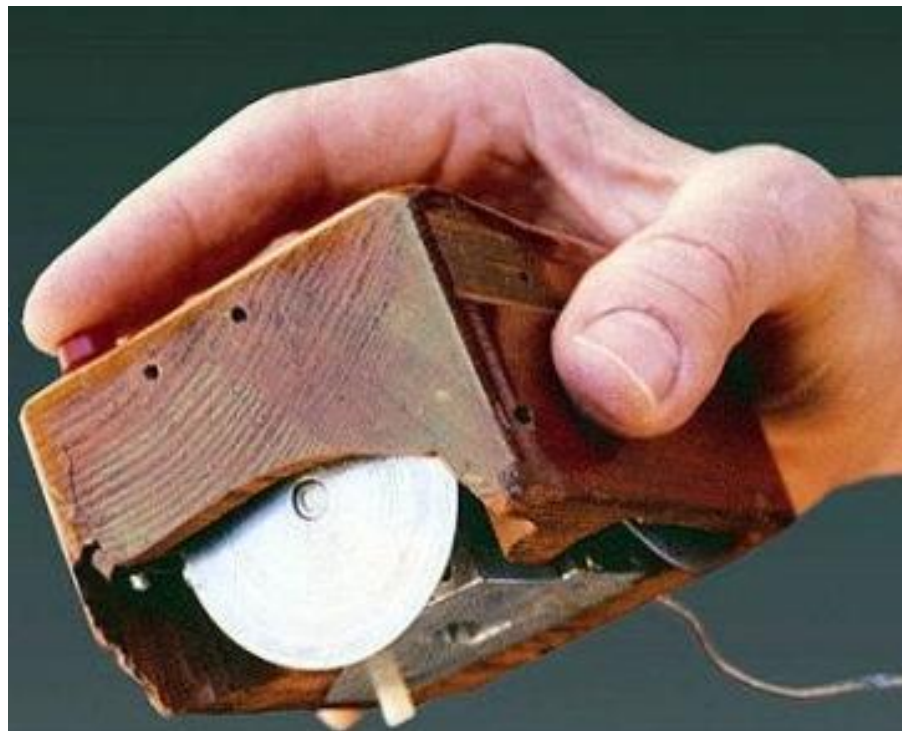
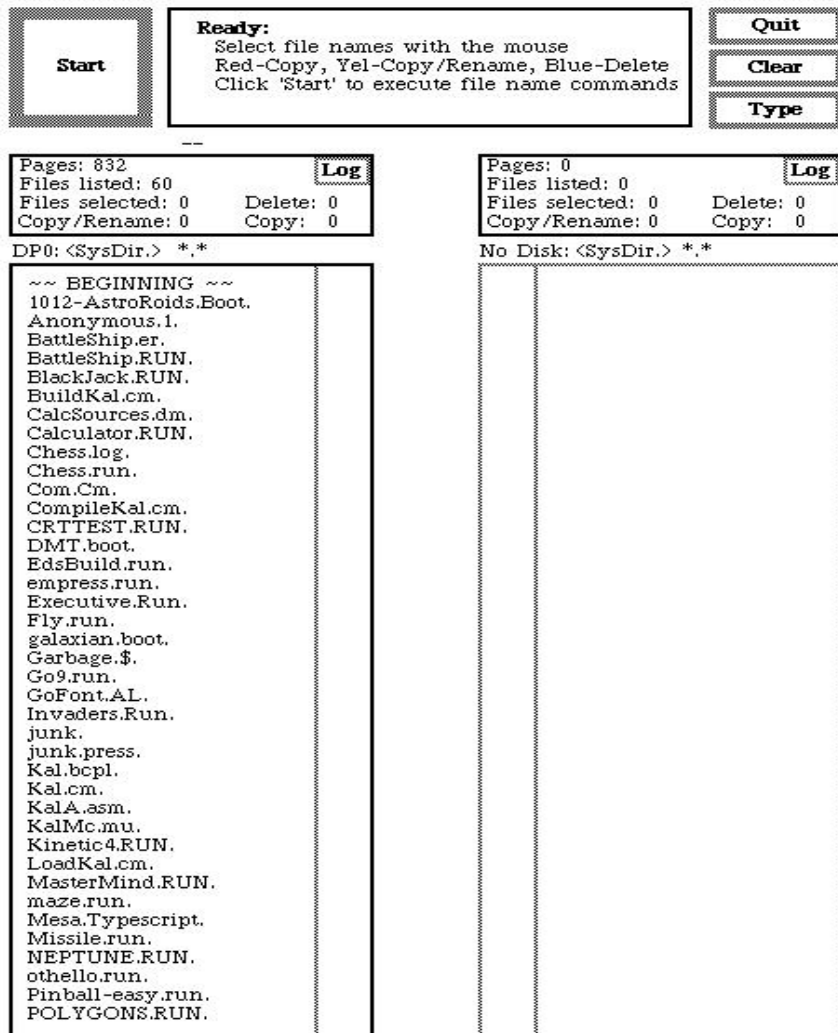
### ● 用户界面友好的桌面接口

- 通常使用鼠标、键盘和监视器
- 图标代表文件、程序、系统功能等
- 不同对象上鼠标按钮导致不同的动作
- GUI首次出现在 Xerox PARC

### ● 许多系统同时包含CLI和GUI界面

- Microsoft Windows使用带有命令行的图形界面
- Apple Mac OS X 采用“Aqua”界面，UNIX 使用命令行界面并有多shell可用
- Solaris 是 CLI 界面带有可选的图形界面 (Java 桌面, KDE)

# 第一个图形界面——Xerox Alto

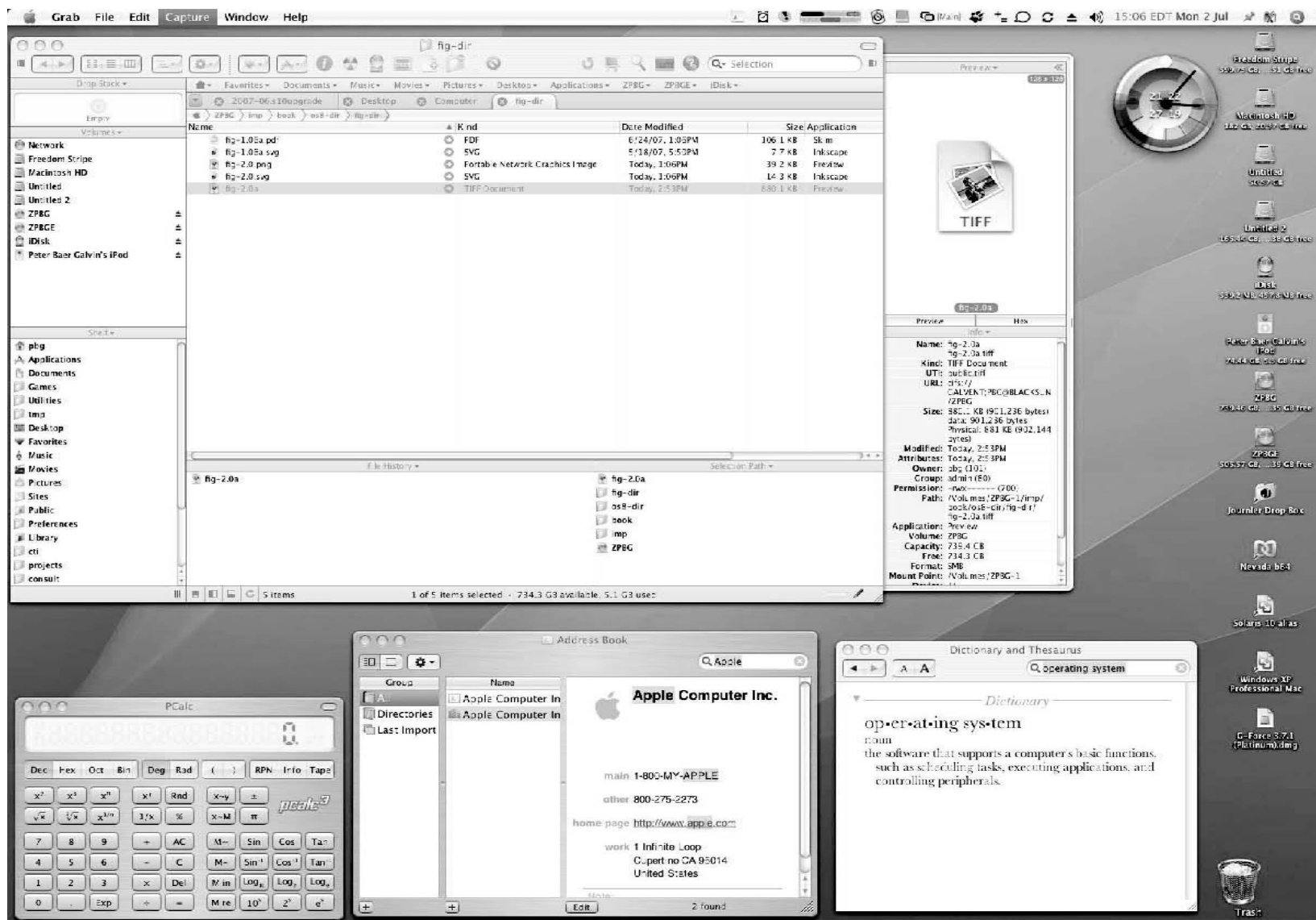




# ►► The Mac OS X GUI

## 第二章 操作系统结构

14





触摸屏交互

语音交互



- i-Air Touch 技术
- 语音操控
- 体感操控
- 大脑操控
- .....





# 第三节、系统调用 (System Call)

- 是操作系统服务的编程接口-**面向程序**
- 通常用高级语言编写 (C or C++)
- **程序通过应用程序接口(API)访问，而不是直接使用系统调用**
- 三种常用 APIs:
  - Windows 的Win32 API
  - POSIX系统 (包括几乎所有版本的UNIX, Linux, 和Mac OS X)的POSIX API
  - Java 虚拟机 (JVM) 的Java API

Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct System Call use, why?

- System Call: expensive, are system dependent
- API : easy、 portability

source file

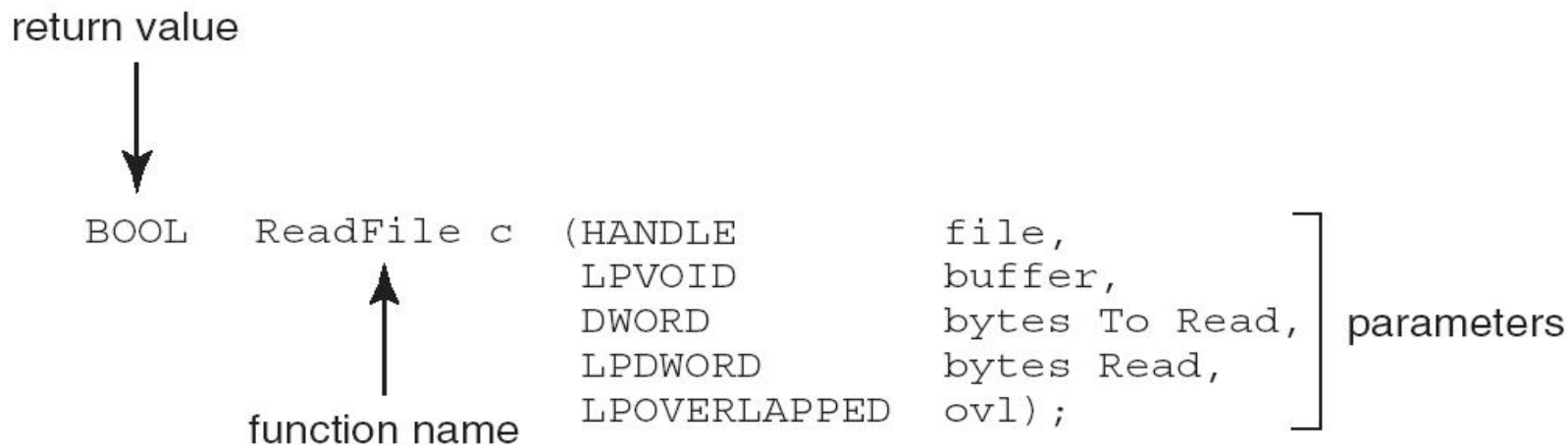
destination file

### Example System Call Sequence

- Acquire input file name
- Write prompt to screen
- Accept input
- Acquire output file name
- Write prompt to screen
- Accept input
- Open the input file
  - if file doesn't exist, abort
- Create output file
  - if file exists, abort
- Loop
  - Read from input file
  - Write to output file
- Until read fails
- Close output file
- Write completion message to screen
- Terminate normally

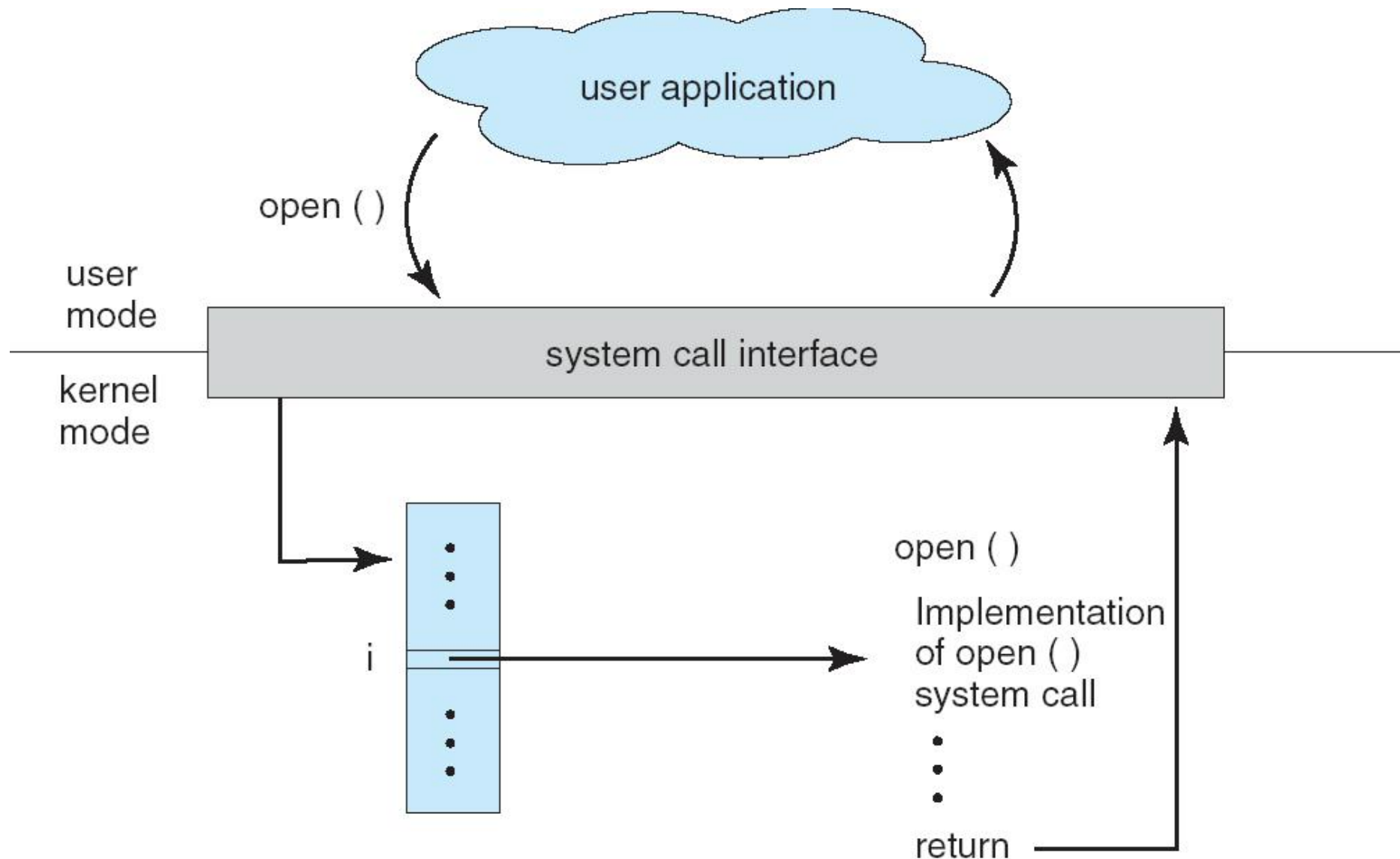
拷贝一个文件  
的内容到另一  
个文件的系统  
调用举例

- Win32 API中ReadFile()方法，从文件中读取内容

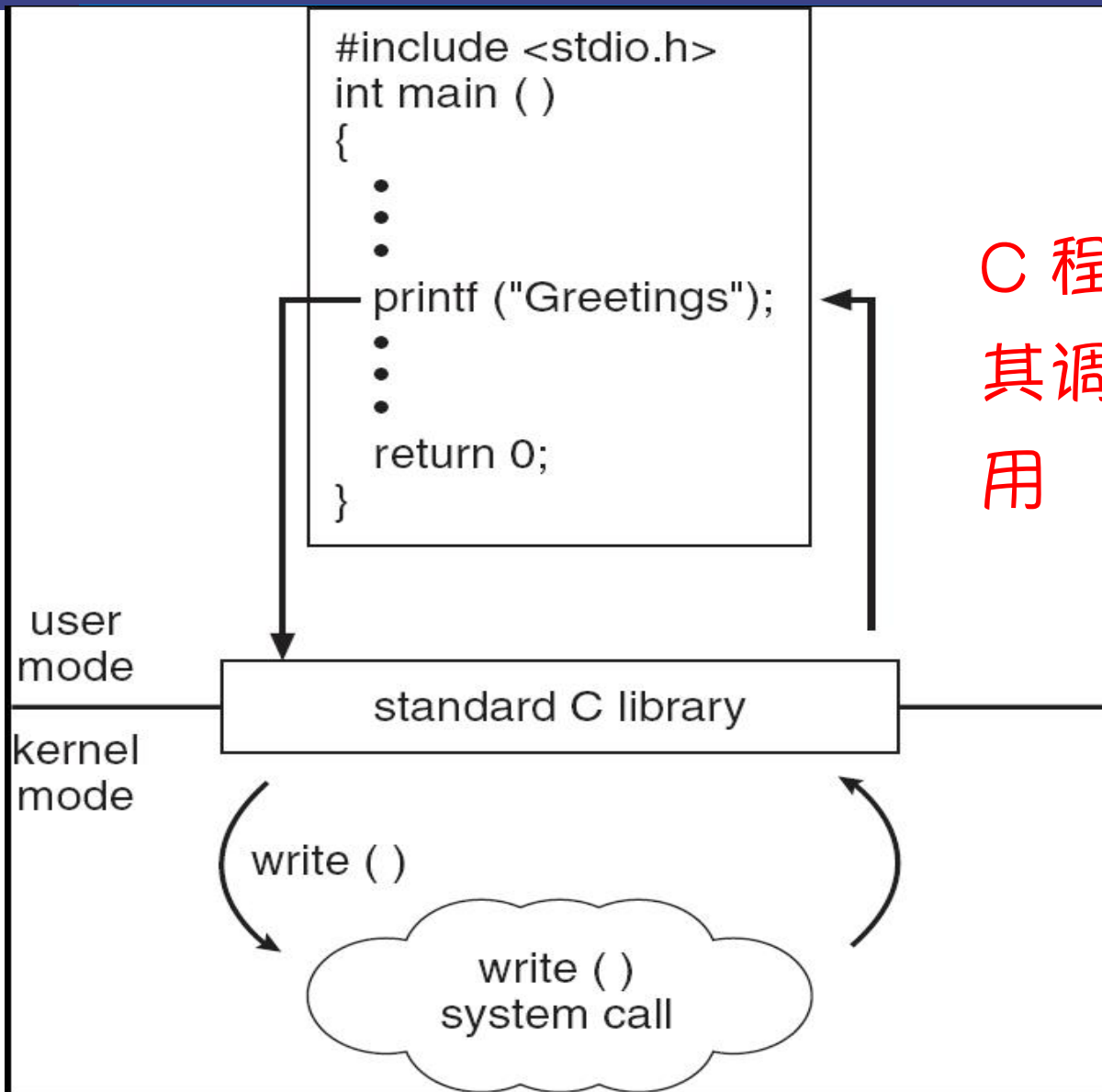


- ReadFile()函数的参数描述如下：

HANDLE file	—	所要读取的文件
LPVOID buffer	—	读进写出的数据缓冲
DWORD bytesToRead	—	将要读入缓冲区的字节数
LPDWORD bytesRead	—	上次读操作读的字节数
LPOVERLAPPED ovl	—	指示是否使用重叠 I/O



```
# ./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
```



C 程序库调用 `printf()`,  
其调用了 `write()` 系统调用



	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()

- 每个系统调用都有一个固有番号(System Call Number)
- 操作系统通过一张系统调用番号表来管理系统调用接口
  - 如，在Linux 平台上，系统调用函数以及番号一般定义在“syscalls.h”或“unistd.h”文件里
  - [https://lxr.missinglinkelectronics.com/linux/arch/x86/entry/syscalls/syscall\\_64.tbl](https://lxr.missinglinkelectronics.com/linux/arch/x86/entry/syscalls/syscall_64.tbl)
- 作为开发者来说，不需要关心系统调用是怎么实现的，只需要掌握它的使用规则就可以
- 一般系统调用被运行库(run-time support library)来管理，运行库提供API，开发者调用运行库提供的API即可

## How to get the linux kernel source files in Ubuntu?

1. Search linux kernel source version.

```
$ apt-cache search linux-source
```

2. Download linux kernel source files to the default directory- /usr/src

```
$ apt-get install linux-source-X.X.X
```

3. Find the file <unistd.h> using \$find command

```
$ find / | grep unistd.h
```

4. Confirm system call number in the file <unistd.h>

```
$vi /usr/include/x86_64-linux-gnu/asm/unistd_64.h
```

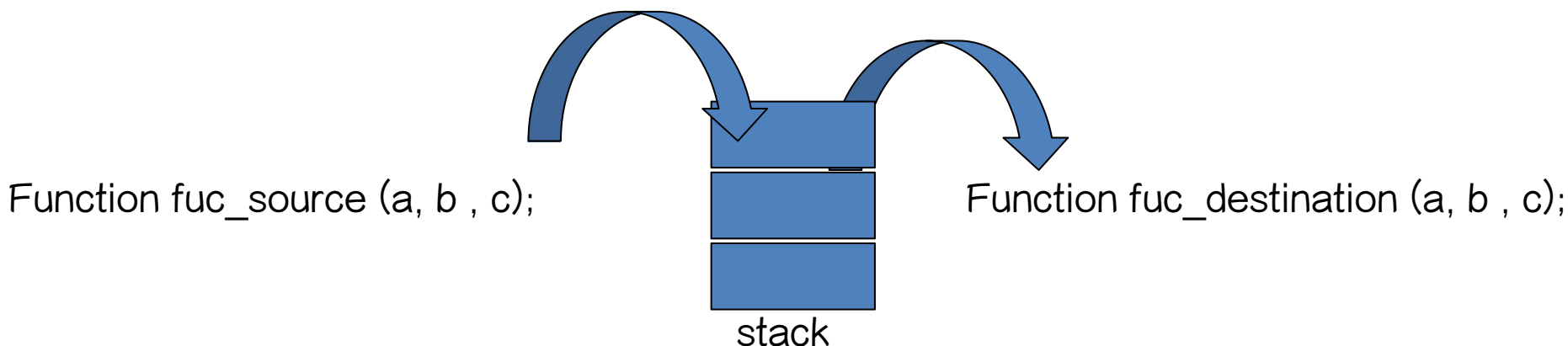
In linux kernel version 3.13.0-XXX(64-bit) has

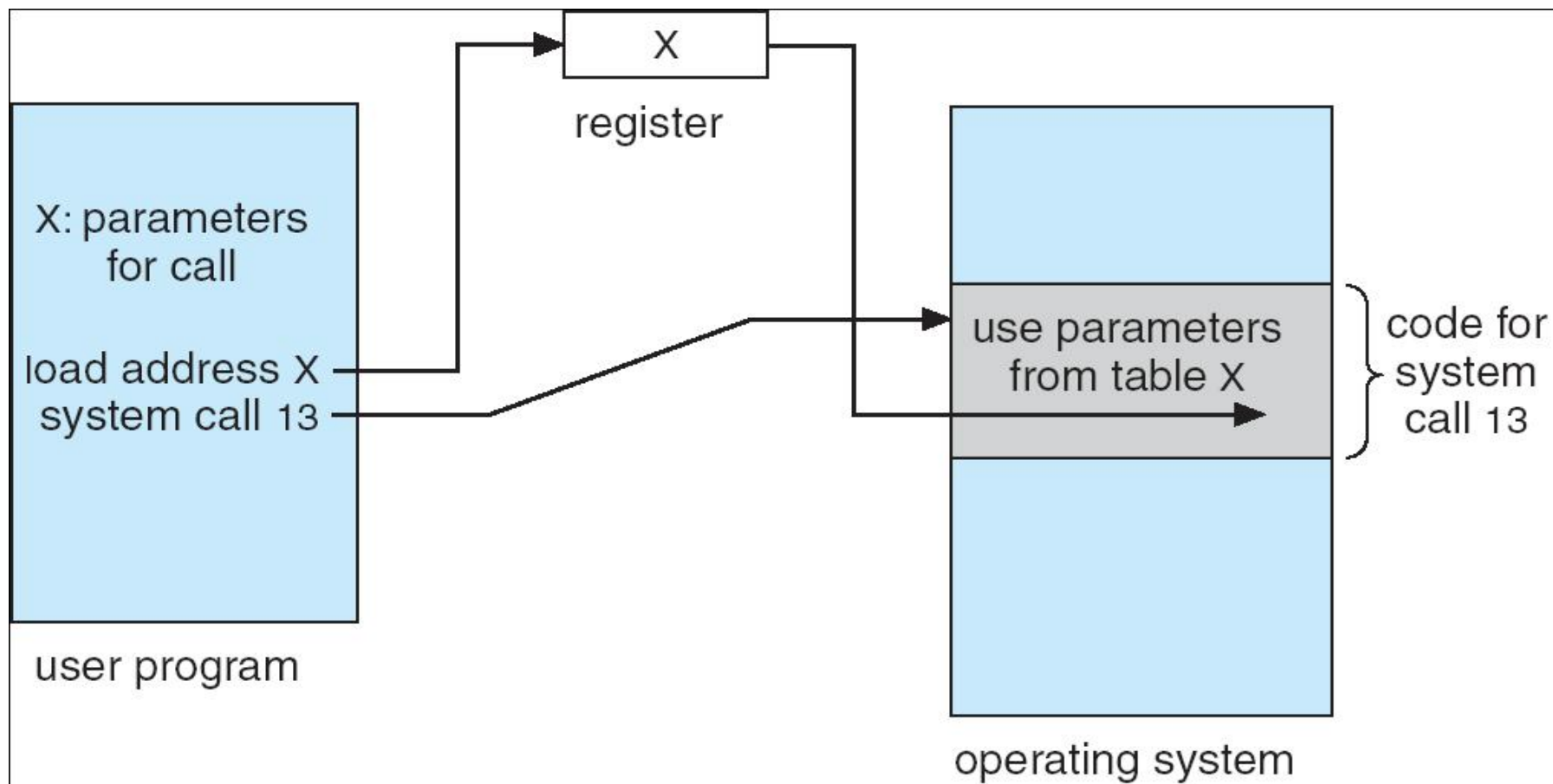
```
root@hbpark-VirtualBox: /usr/include
#ifndef _ASM_X87_UNISTD_64_H
#define _ASM_X86_UNISTD_64_H 1

#define __NR_read 0
#define __NR_write 1
#define __NR_open 2
#define __NR_close 3
#define __NR_stat 4
#define __NR_fstat 5
#define __NR_lstat 6
#define __NR_poll 7
#define __NR_lseek 8
#define __NR_mmap 9
#define __NR_mprotect 10
#define __NR_munmap 11
#define __NR_brk 12
#define __NR_rt_sigaction 13
#define __NR_rt_sigprocmask 14
#define __NR_rt_sigreturn 15
#define __NR_ioctl 16
#define __NR_pread64 17
#define __NR_pwrite64 18
#define __NR_readv 19
#define __NR_writev 20
```

向系统调用函数传递参数的方式，有以下三种

1. 寄存器
2. 块(block): 参数保存在内存中的一个块中，并把块地址用寄存器传递给系统调用函数
3. 栈(stack): 以栈的形式保存在内存中，用户程序向栈 push 参数，操作系统从栈中 pop 参数





1. System Call for Process Control
2. System Call for File Management
3. System Call for Device Management
4. System Call for Information Maintenance
5. System Call for Communication
6. System Call for Protection

### 1. System Call for Process Control

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



## 2. System Call for File Management

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

## 3. System Call for Device Management

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

## 4. System Call Information Maintenance

- get time or date, set time or date
- get system data, set system data
- get and set process, file, or device attributes

### 5. System Call for Communications

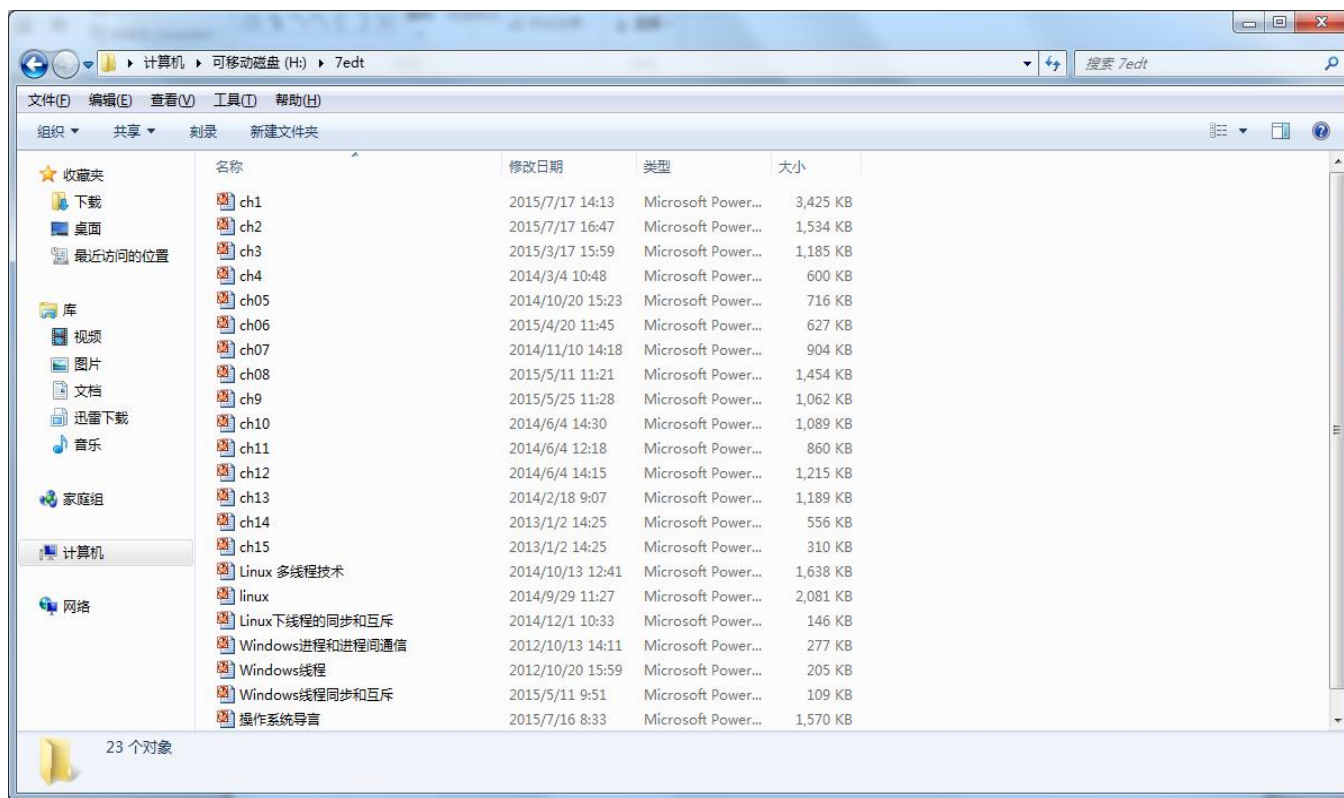
- create, delete communication connection
- send, receive messages if message passing model to host name or process name(From client to server)
- Shared-memory model create and gain access to memory regions
- transfer status information
- attach and detach remote devices

### 6. System Call for Protection

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

## 第四节、系统程序 (System Program)

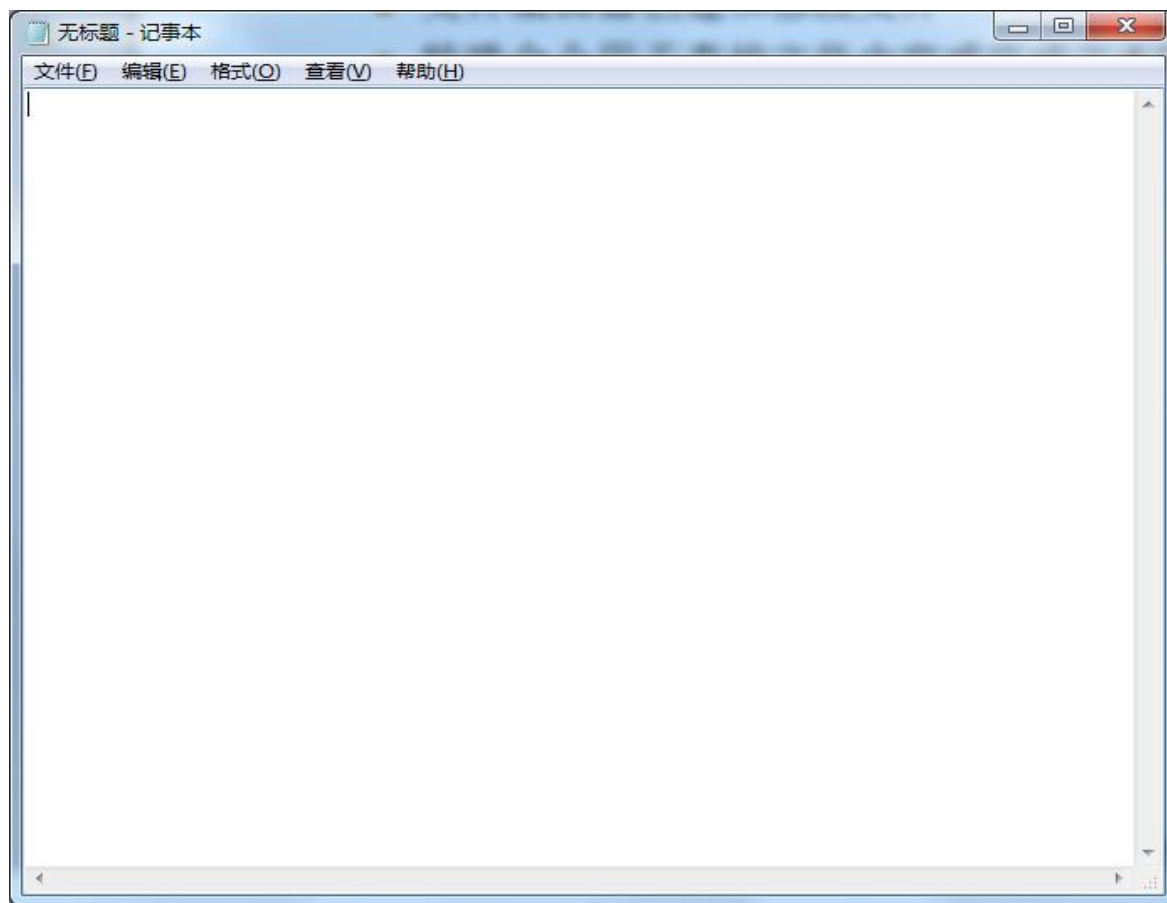
1. 提供一个方便的环境，以开发程序和执行程序
2. 系统程序不属于内核，但属于操作系统的一部分
3. 为用户使用操作系统服务，如文件管理器：创建、删除、复制、重命名、打印、转储、列出和操作文件和目录



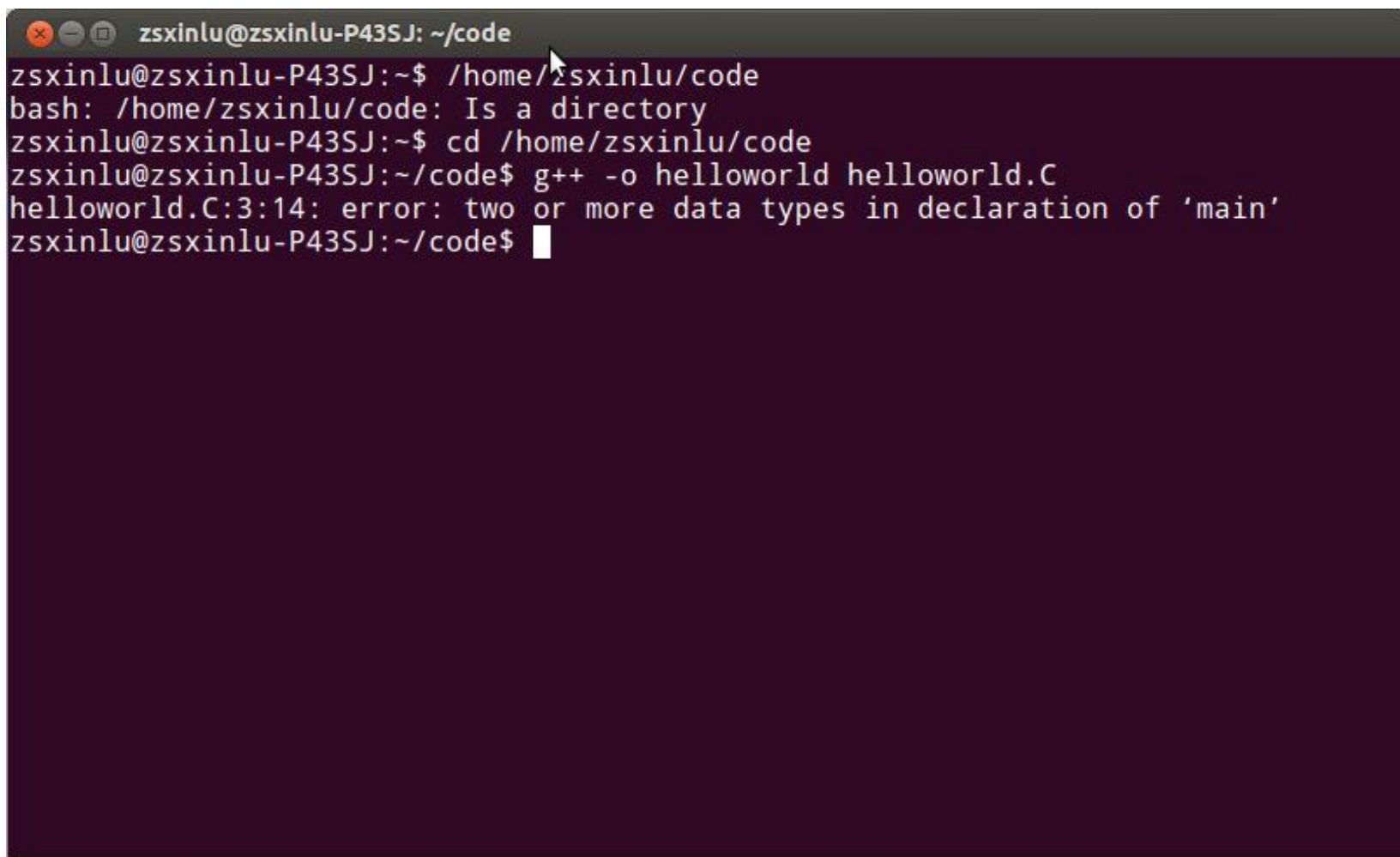
1. 状态信息：日期、时间、可用内存、磁盘空间和用户数等
2. 性能、登录和调试信息
3. 注册表：用于存储和检索配置信息



1. 文件编辑器创建和修改文件
2. 特殊命令用于查找文件内容或完成文本转换

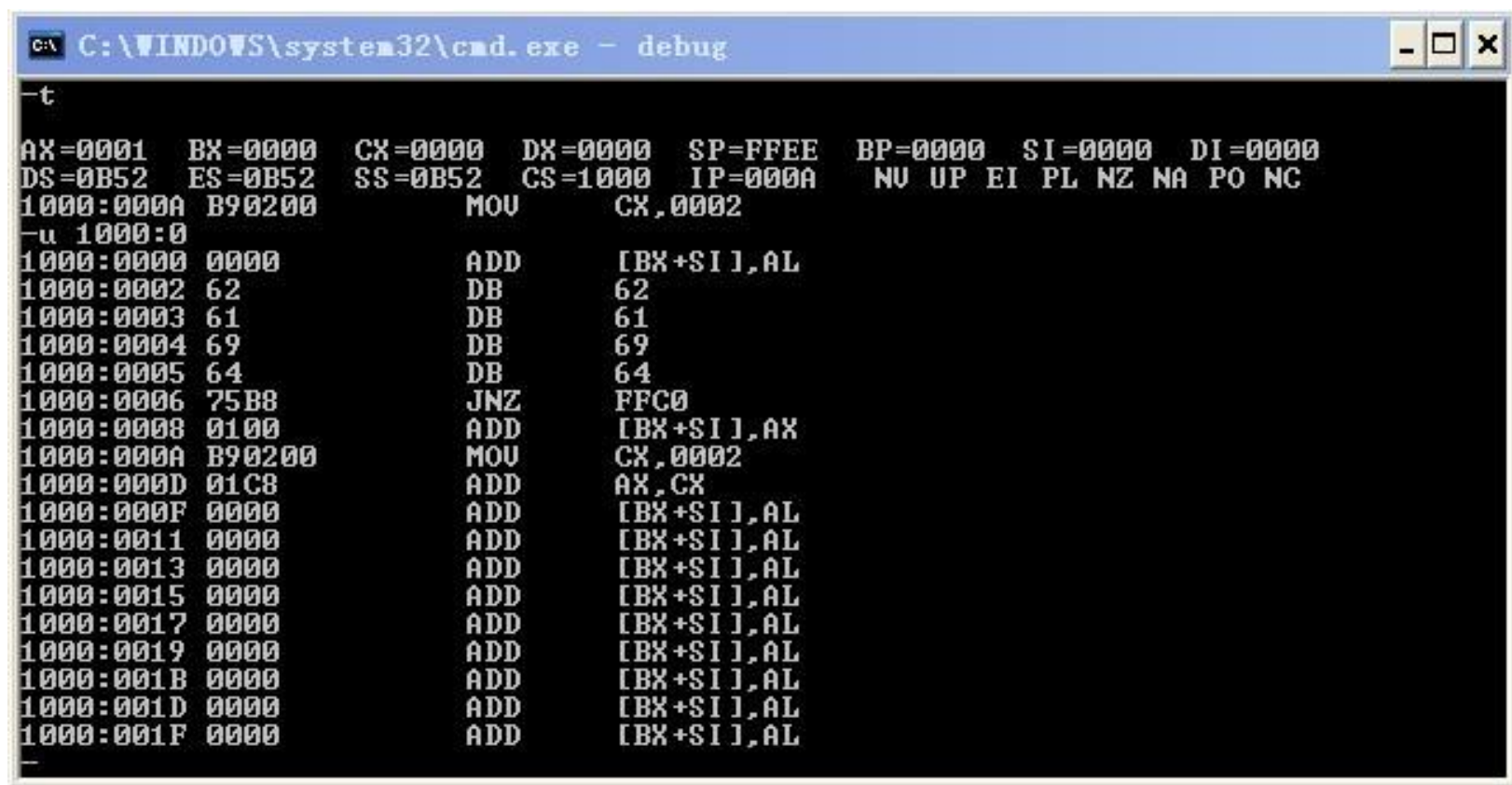


常用程序设计语言的编译程序、汇编程序、调试程序和解释程序



```
zsinlu@zsinlu-P43SJ: ~/code
zsinlu@zsinlu-P43SJ:~$ /home/zsinlu/code
bash: /home/zsinlu/code: Is a directory
zsinlu@zsinlu-P43SJ:~$ cd /home/zsinlu/code
zsinlu@zsinlu-P43SJ:~/code$ g++ -o helloworld helloworld.C
helloworld.C:3:14: error: two or more data types in declaration of 'main'
zsinlu@zsinlu-P43SJ:~/code$
```

绝对加载程序、重定位加载程序、链接编辑器和覆盖式加载程序，还有高级语言或机器语言的调试程序



```
C:\WINDOWS\system32\cmd.exe - debug

-t
AX=0001  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B52  ES=0B52  SS=0B52  CS=1000  IP=000A  NU UP EI PL NZ NA PO NC
1000:000A B90200      MOV     CX,0002
-u 1000:0
1000:0000 0000      ADD     [BX+SI],AL
1000:0002 62          DB      62
1000:0003 61          DB      61
1000:0004 69          DB      69
1000:0005 64          DB      64
1000:0006 75B8       JNZ     FFC0
1000:0008 0100      ADD     [BX+SI],AX
1000:000A B90200      MOV     CX,0002
1000:000D 01C8      ADD     AX,CX
1000:000F 0000      ADD     [BX+SI],AL
1000:0011 0000      ADD     [BX+SI],AL
1000:0013 0000      ADD     [BX+SI],AL
1000:0015 0000      ADD     [BX+SI],AL
1000:0017 0000      ADD     [BX+SI],AL
1000:0019 0000      ADD     [BX+SI],AL
1000:001B 0000      ADD     [BX+SI],AL
1000:001D 0000      ADD     [BX+SI],AL
1000:001F 0000      ADD     [BX+SI],AL
```



提供在进程、用户和计算机系统之间创建虚拟链接的机制



# Application Program vs System Program

- **Application Program** aims to produce software **which provides services to the user directly**
- **System Program** aims to produce software and software platform **which provide service to other software**

# 第五节、操作系统的设计和实现

- OS的设计和实现，没有完整的解决方案
- 不同类型操作系统的内部结构不同
- 从定义系统的目标和规格开始
- 系统设计受到硬件选择和系统类型的影响
- 设计目标：用户目标和系统目标
  1. 用户目标：系统应该方便和容易使用、易学、可靠、安全和快速
  2. 开发目标：系统应该容易设计、实现和维护，也应该灵活、可靠、高效且没有错误

- 区分的重要原则
  1. 策略(Policy): 做什么
  2. 机制(Mechanism): 怎么做
- 机制决定如何做, 策略决定做什么
- 策略与机制的区分对于灵活性来说很重要
- 策略可能会随时间或位置而有所改变

- 开发语言

- 早期：汇编语言
- 目前：高级语言（C, C++）

- 实际：混合多种语言

- 底层用汇编语言
- 主体用C语言
- 系统程序用C, C++, PERL, Python, shell scripts等

- 考量

- 汇编语言：运行高效，但编程耗时，不易移植
- 高级语言：运行效率差，但编程高效，易移植

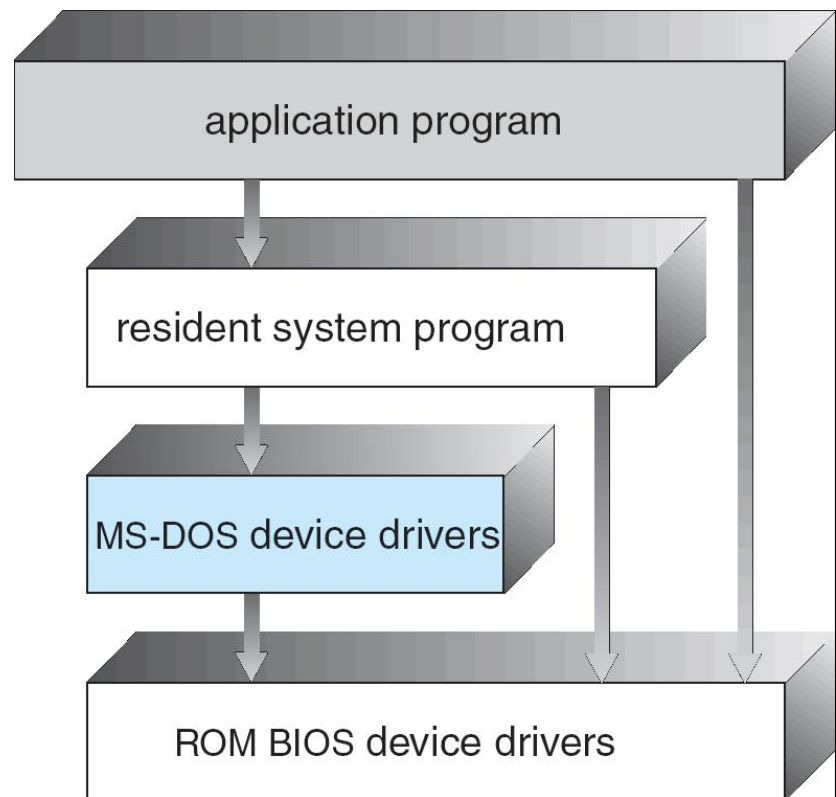
## 第六节、操作系统结构

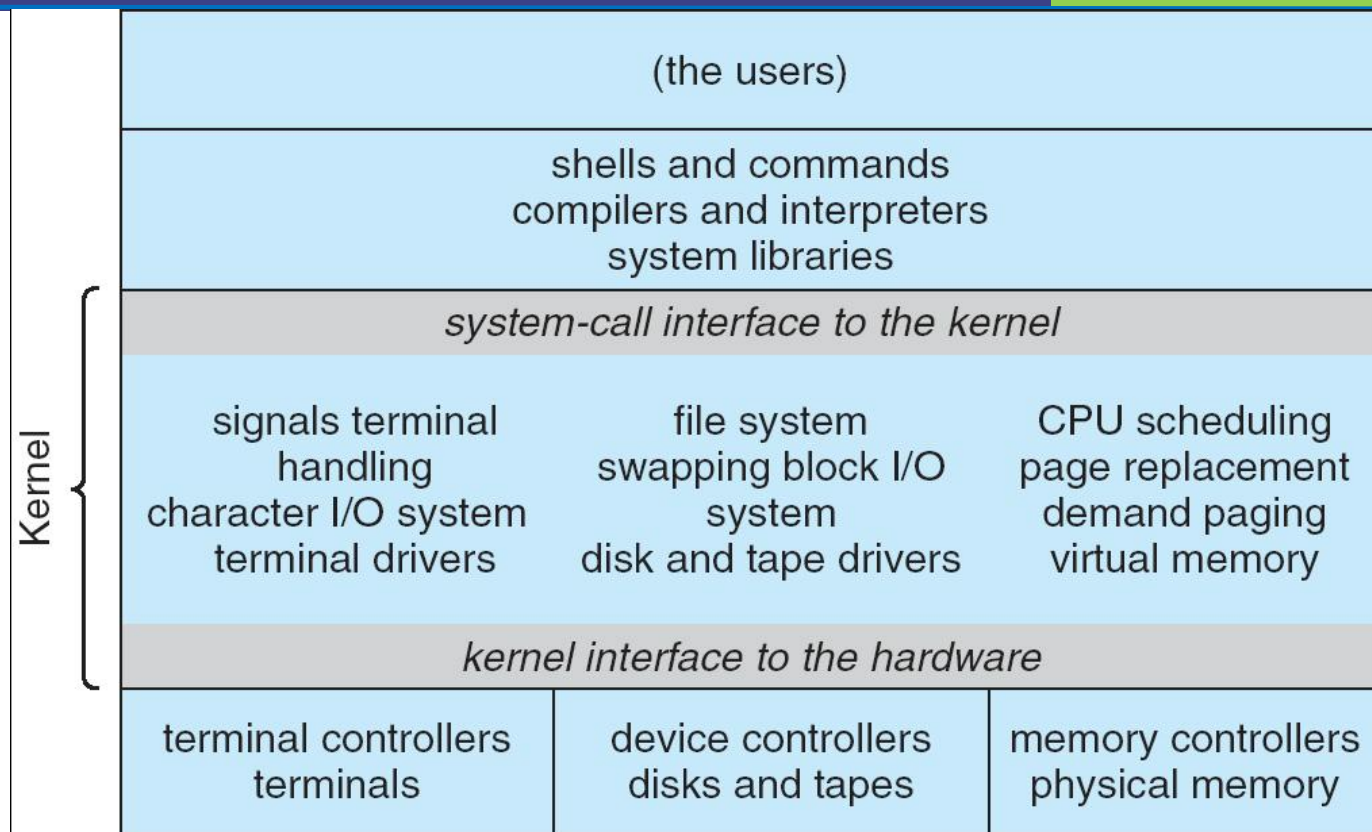




1. 简单结构 (Simple Structure)
2. 层次结构 (Layered)
3. 微内核 (Microkernel)
4. 模块结构 (Modules)
5. 混合结构 (Hybird)

1. 早期操作系统，规模小，简单，功能有限，以MS-DOS为例，以最小的空间提供最多的功能
2. 它没划分模块，尽管MS-DOS有某种结构，其接口和功能层没有划分清楚



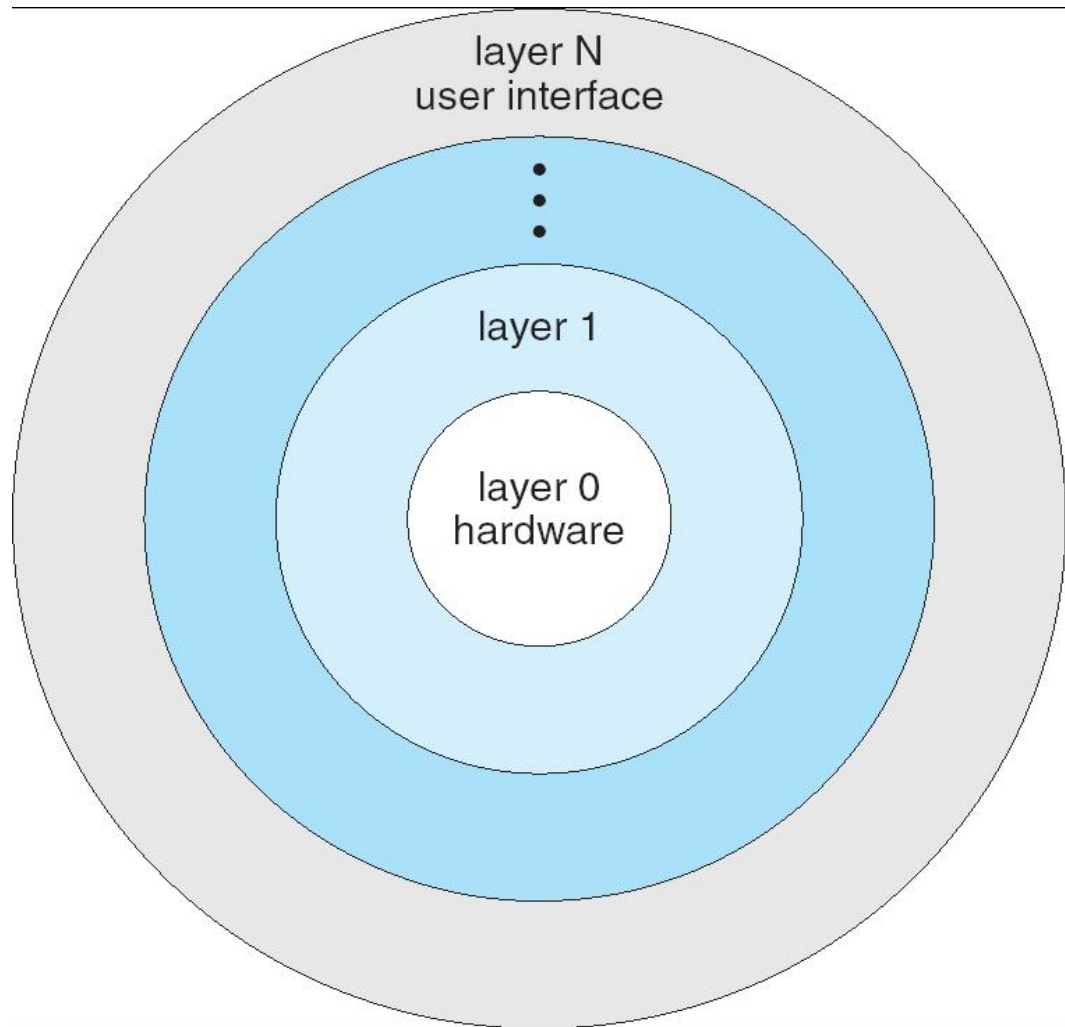


UNIX — 受到硬件功能的限制，原始的UNIX操作系统由两个独立部分组成：系统程序和内核。（**内核**：接口和驱动程序，硬件之上和系统调用接口下的所有部分，包括文件系统、CPU调度、内存管理和其他操作系统功能）

操作系统划分为若干层，  
在低层上构建高层。

底层(0层)为硬件；  
最高层(N层)为用户层；

每层只使用低层次的功能  
和服务

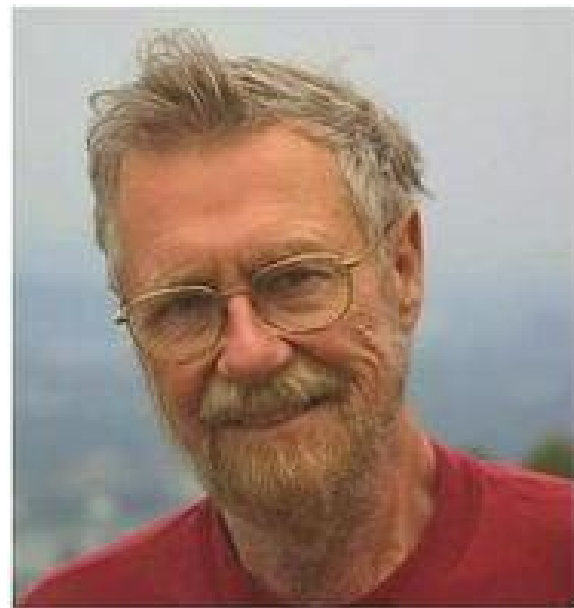


- 优点：简化了系统设计和实现，便于调试和升级维护
- 缺点：层定义，效率差
- 例子-THE
  - 1968年由 E. W. Dijkstra（艾兹格·W·迪科斯彻）开发中第一次提出了层次式结构设计方法
  - 运行在荷兰的 Electrologica X8（32K内存）上的一个简单批处理系统

### 艾兹格·W·迪科斯彻

(Edsger Wybe Dijkstra)

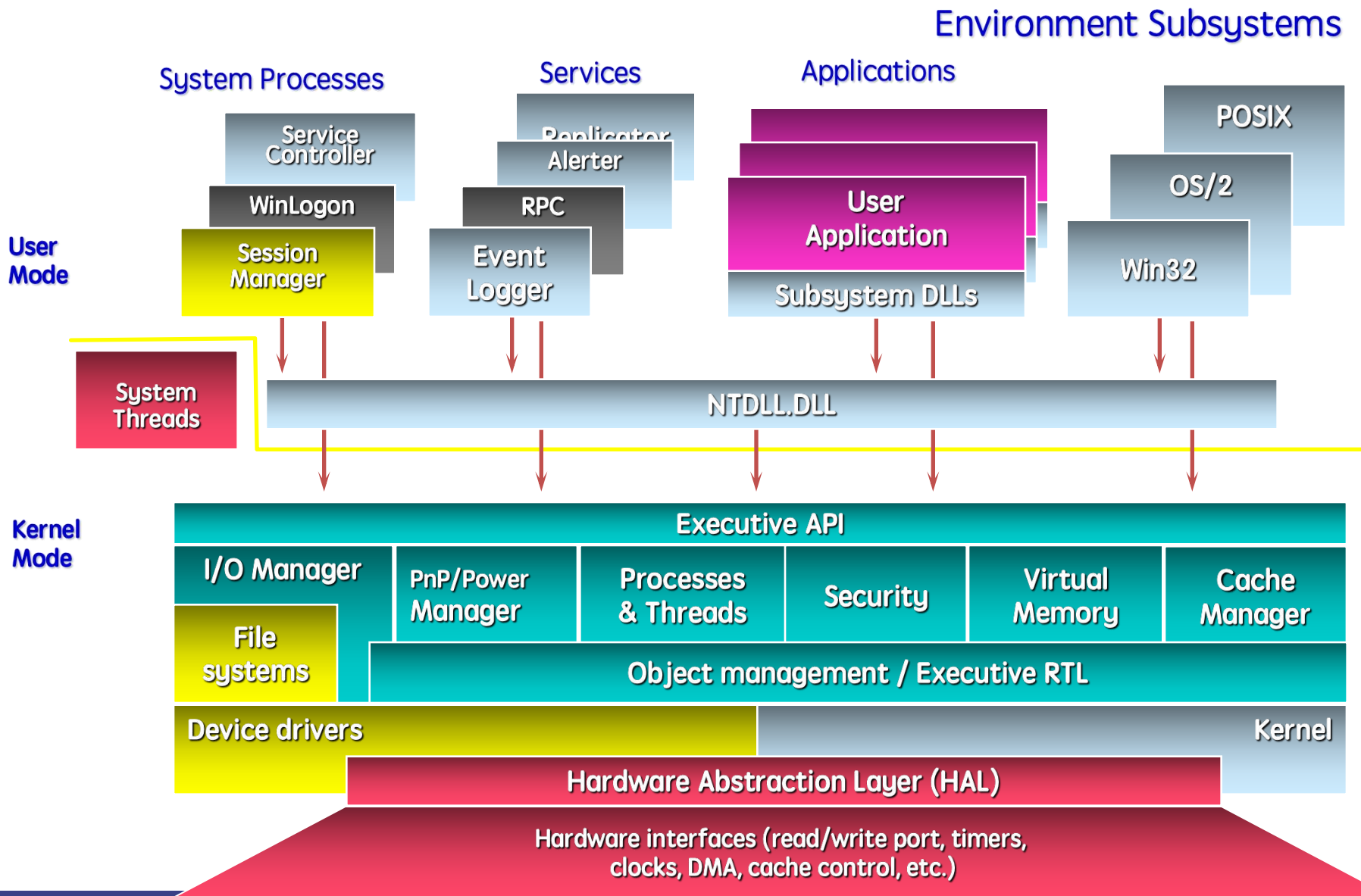
- 提出信号量和PV原语—第六章
- 解决了“哲学家进餐”问题—第六章
- 最短路径算法(SPF)和银行家算法的创造者-第七章
- 结构程序设计之父
- THE 操作系统的设计者和开发者—本章  
与D. E. Knuth并称为我们这个时代最伟大的计算机科学家

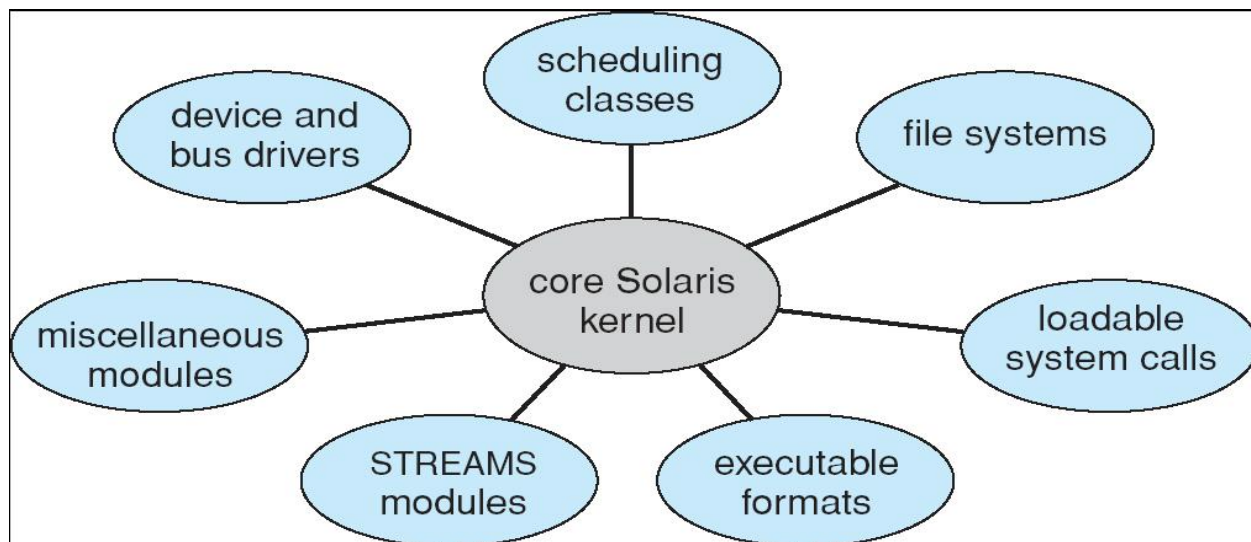


- 问题：内核越来越大，越来越难管理
- 内核微型化：从内核移出尽可能多功能到用户空间
- 发生在用户模块间的通信使用消息传递形式
- 优点：便于扩充微内核、便于移植操作系统到新架构系统上、更稳定（更少的代码运行在核心态）、更安全
- 缺点：用户空间和内核空间通信的系统开销增加

- 第一个微内核系统：CMU的Mach
  - <http://www.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html>
- Tru64 Unix使用Mach内核
- QNX-基于微内核的实时操作系统
- Windows NT, 2000, 2003等







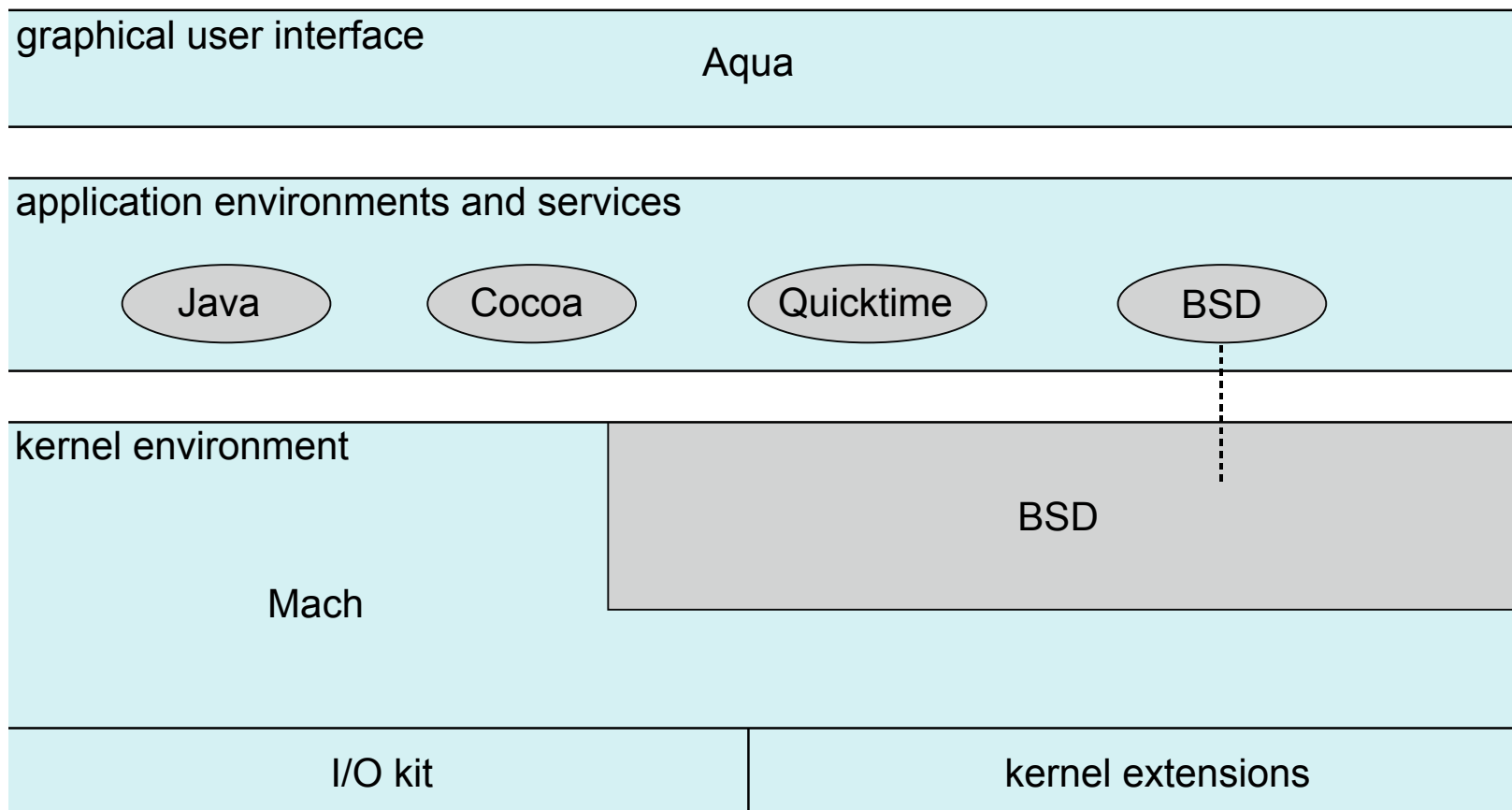
Solaris  
模块

大部分现代操作系统采用模块结构

- 使用面向对象方法，每个核心部件分开
- 每个组件在需要时被加载到内核
- 每个与其他组件的会话被称为接口

总体而言，类似于分层方法，但更灵活

许多现代操作系统不是采用一种单一结构，通过采用多种结构获取性能、安全、使用等方面的需求



基于Mac OS X, 增加部分功能性部件

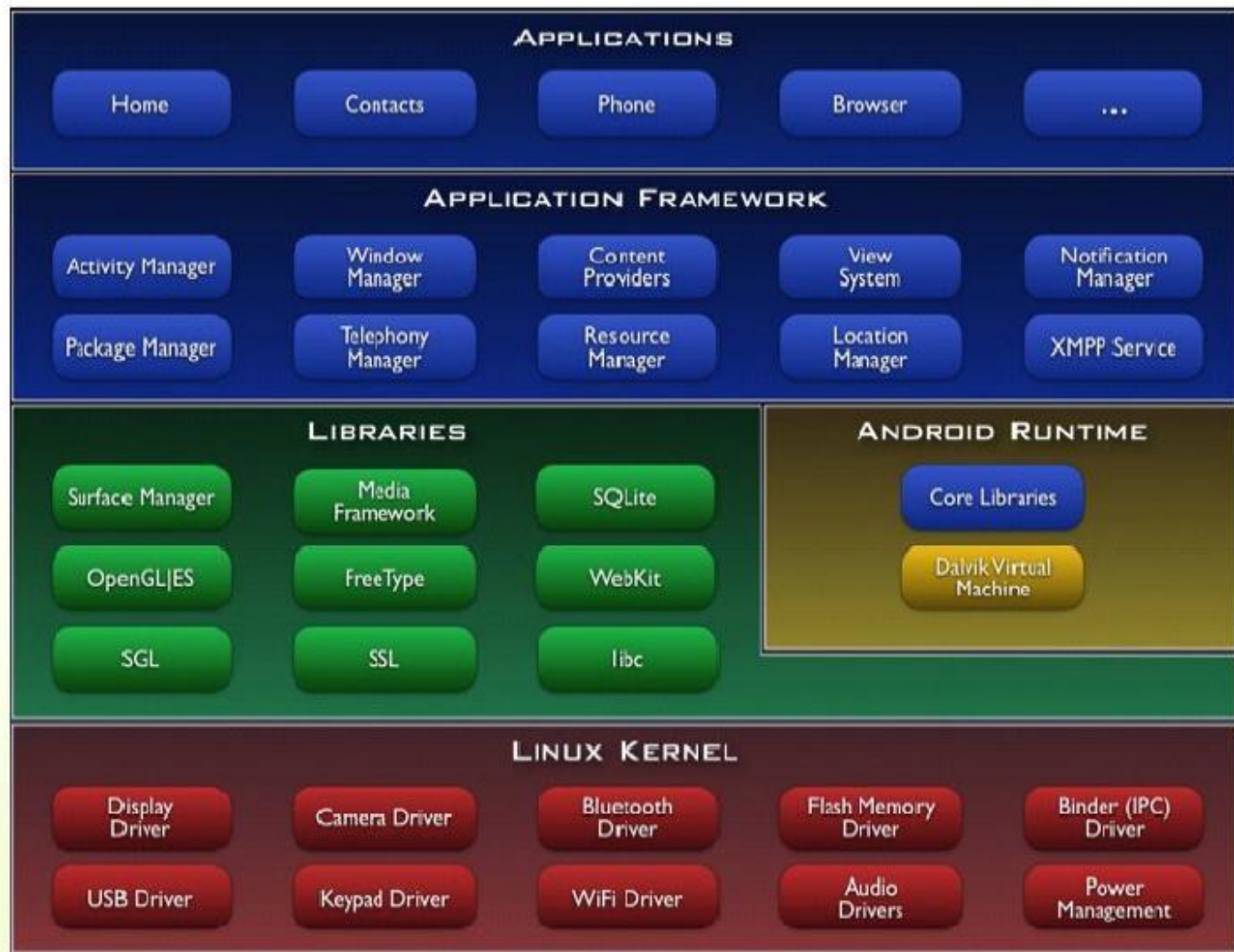
1. Cocoa Touch: 提供Objective-C API 用于开发apps
2. Media services : 图像、视频和声音从服务
3. Core services: 提供云计算、数据库等服务
4. Core OS: Mac OS X 内核

Cocoa Touch

Media Services

Core Services

Core OS

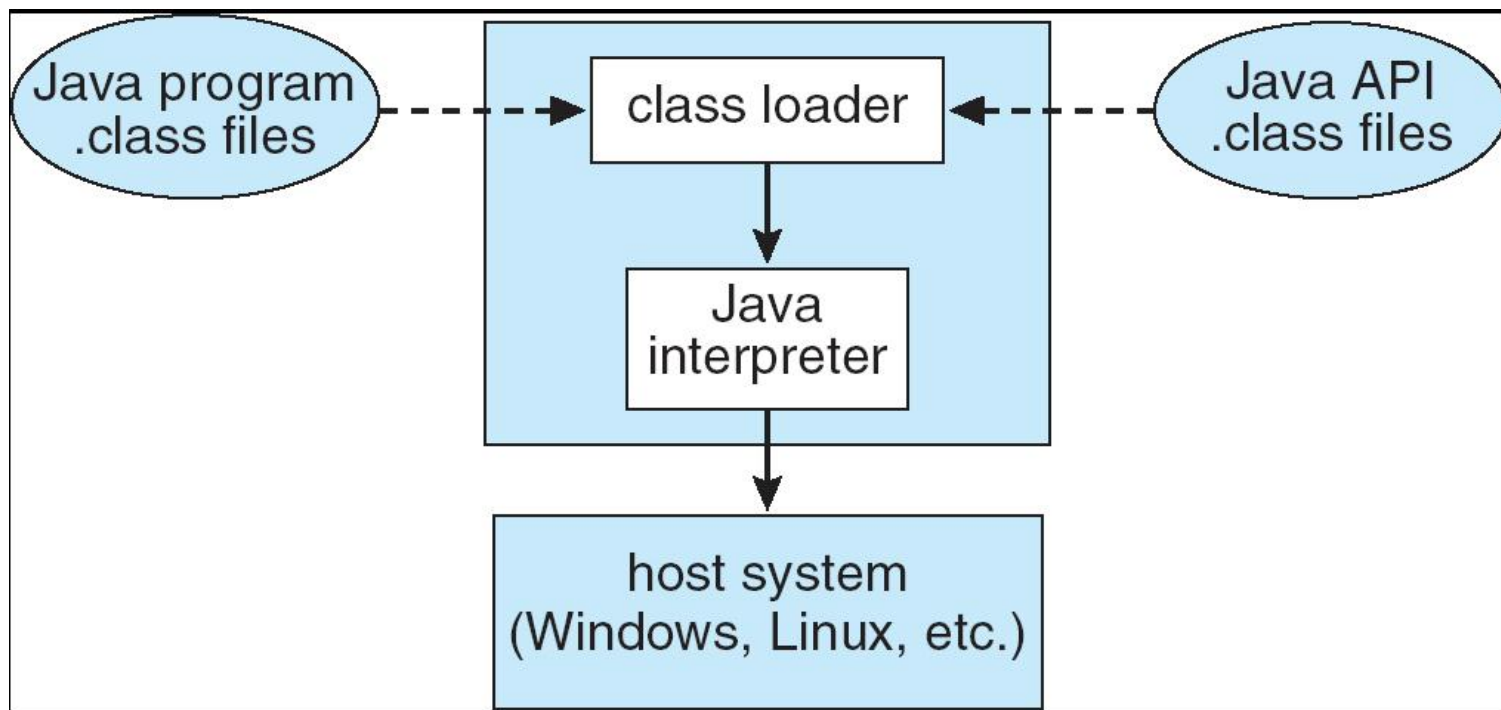


基于修改的 Linux 内核

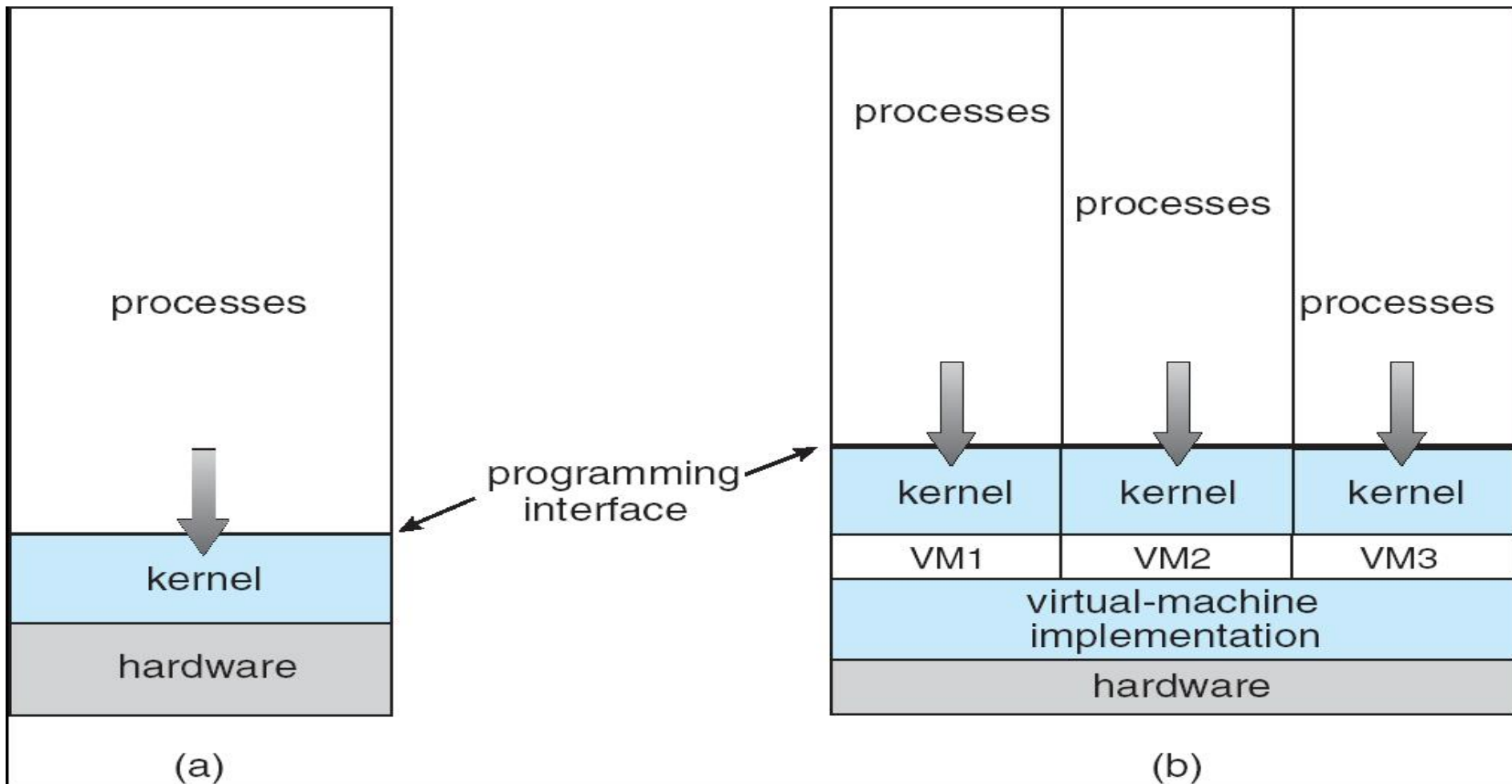
## 第七节、虚拟机 (VM)

- 通过软件模拟的具有完整硬件系统功能的、运行在一个完全隔离环境中的完整计算机系统
- 物理计算机的**资源被共享**，以创建虚拟机
- 虚拟机概念提供对系统资源的**完全保护**，因为每个虚拟机同其他虚拟机**隔离**
- 虚拟机是研发操作系统的完美载体
- 由于需要对物理机器进行精确复制，虚拟机实现困难

1. Java虚拟机有自己硬体架构，如处理器、堆栈、寄存器等，具有相应的指令系统
2. JVM屏蔽了与具体操作系统平台相关的信息，使得Java程序只需生成在Java虚拟机上运行的目标代码，就可以在多种平台上不加修改地运行

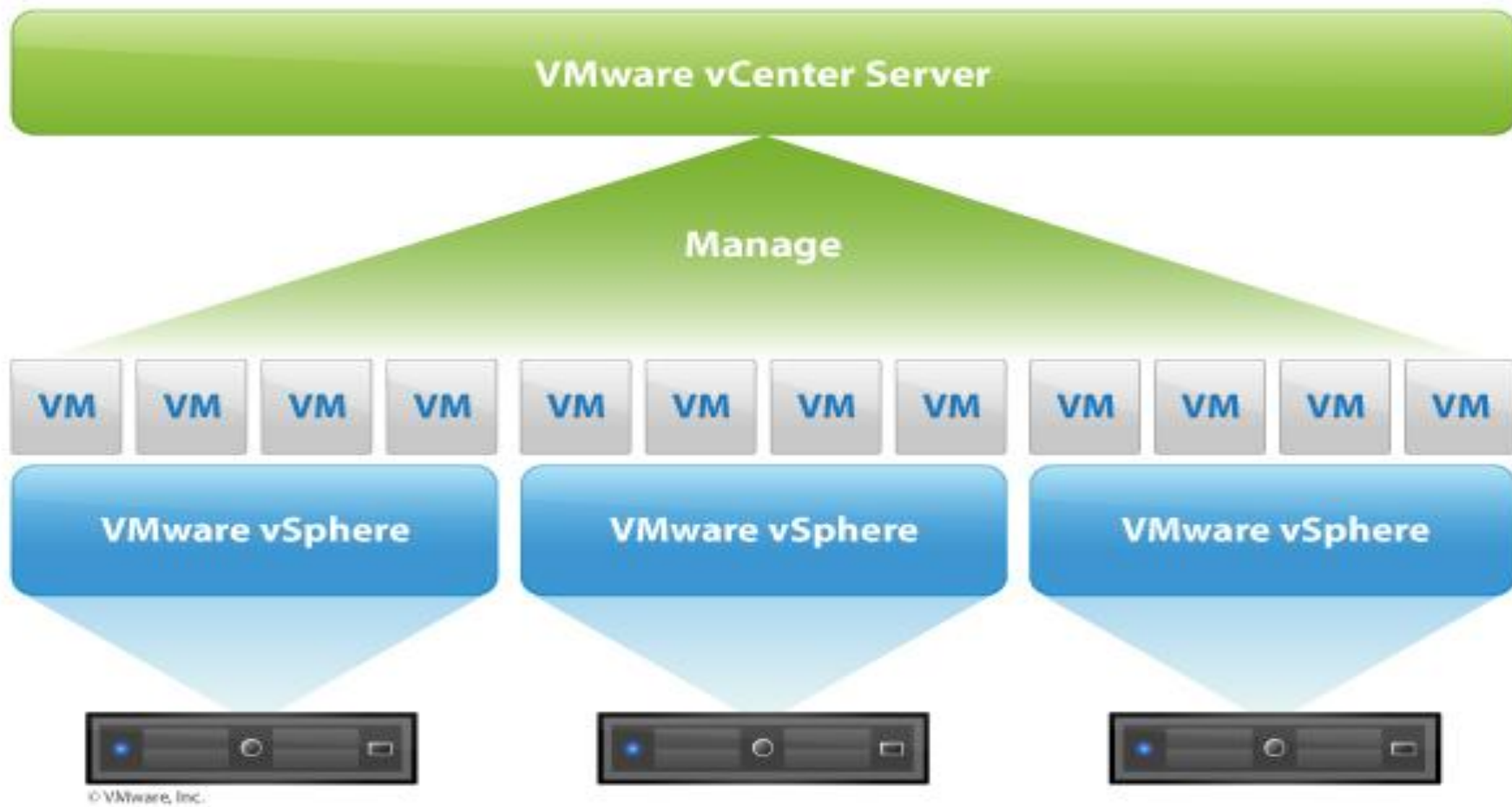


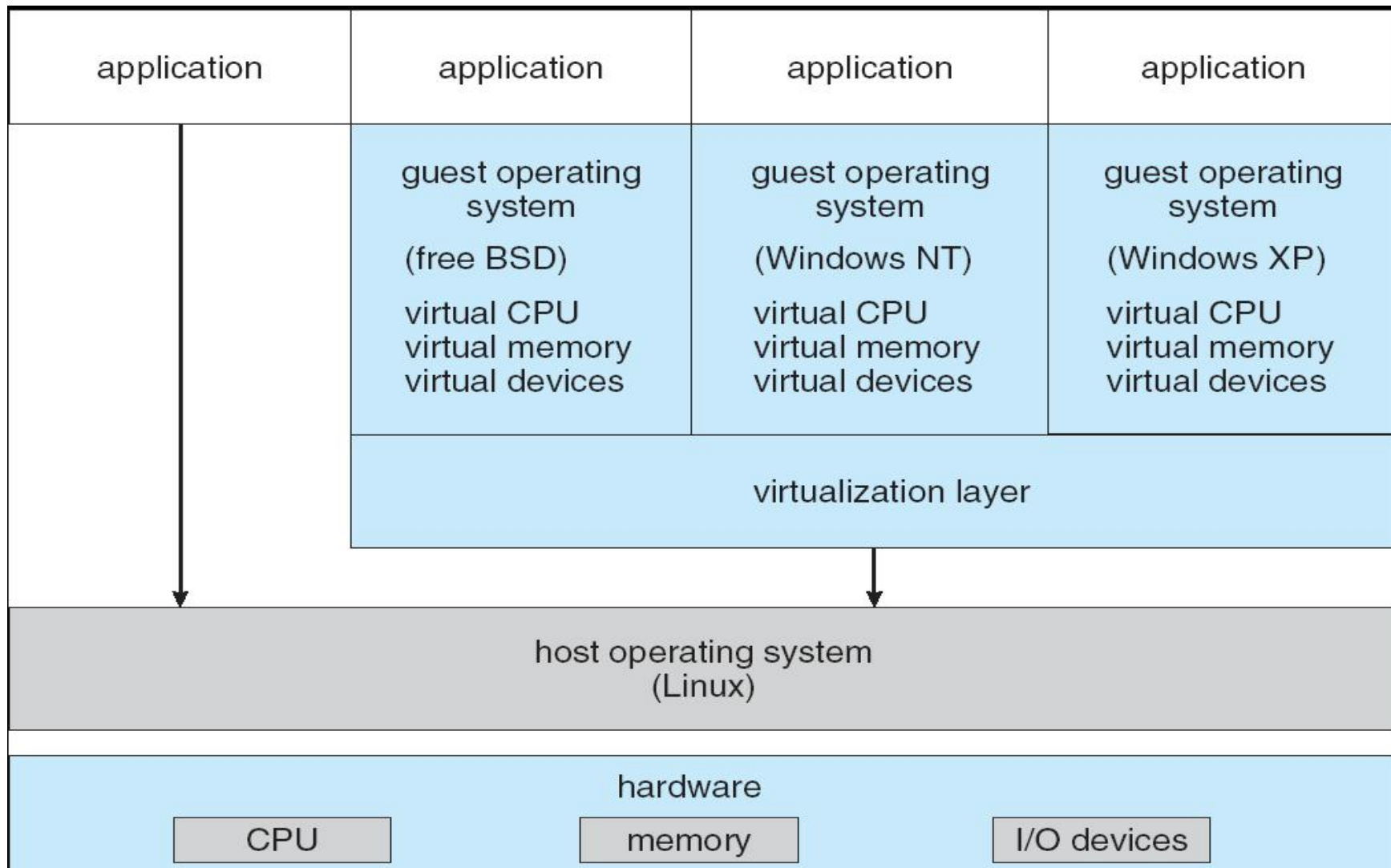


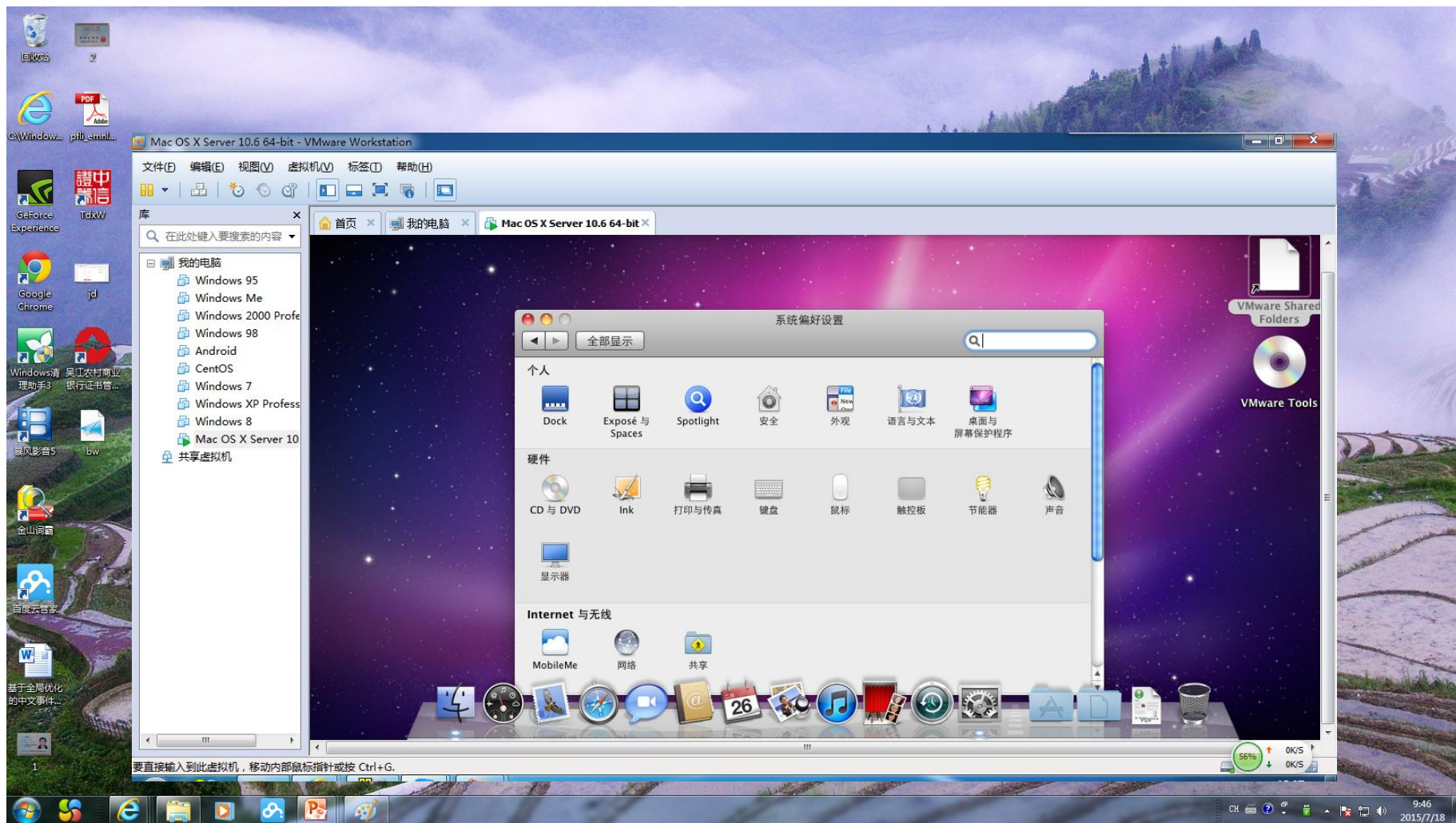


(a) Nonvirtual machine

(b) virtual machine







Thank You.