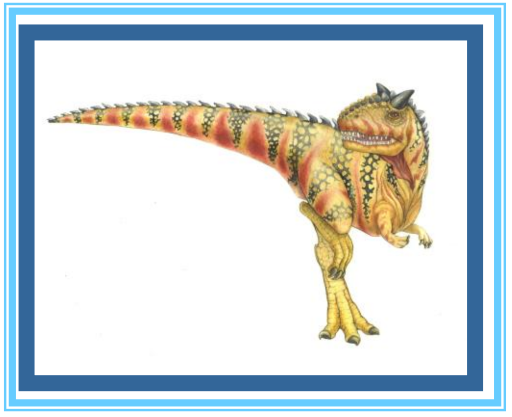
Chapter 2: Operating-System Structures



Chapter 2: Operating-System Structures

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

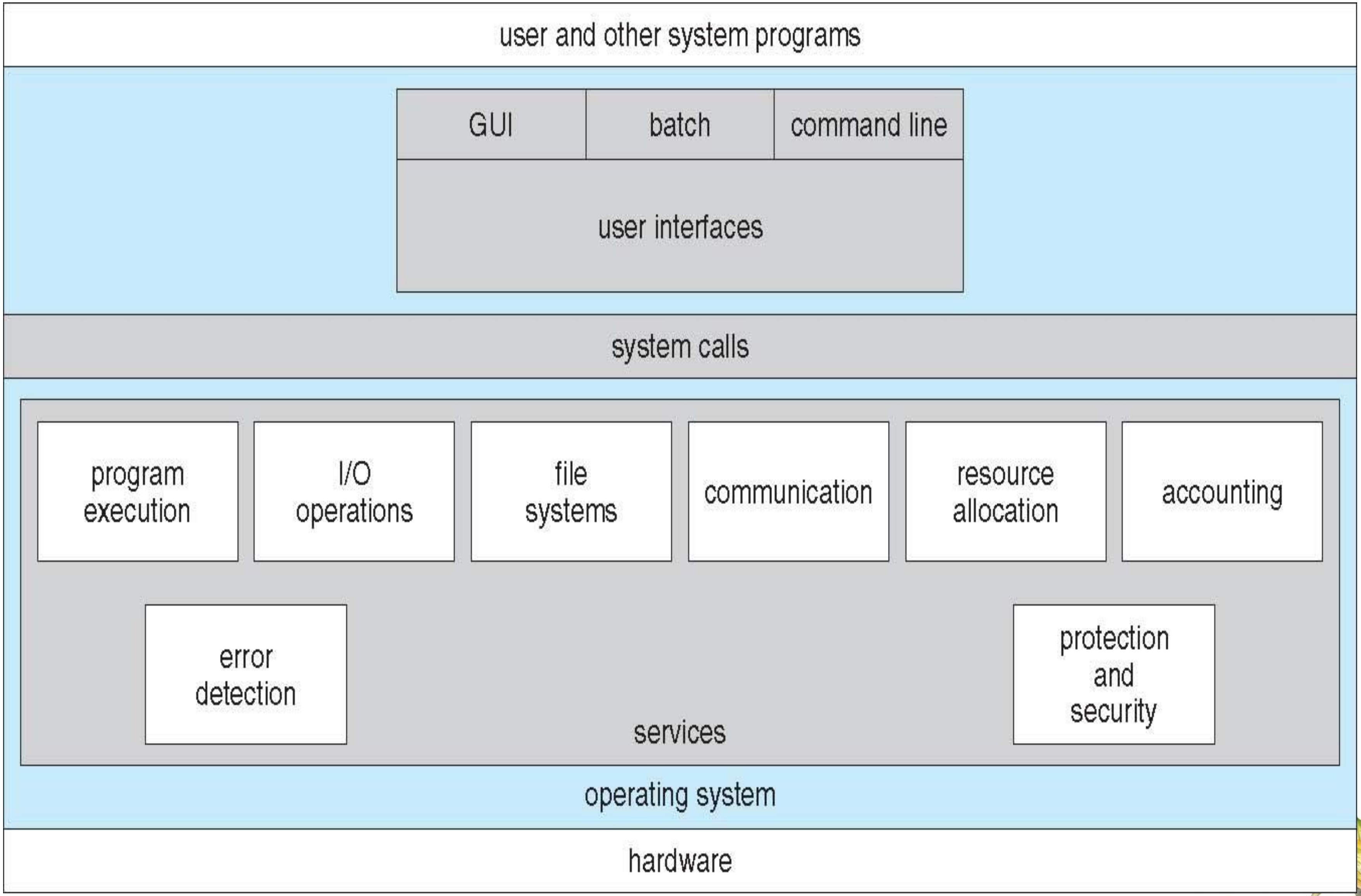
Objectives

- 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;

Operating System Services

- We can view OS from several points:
  - The services that the OS provides;
  - The interface provided to users and programmers;
  - Its components and interconnections;

A View of Operating System Services



Services to the user

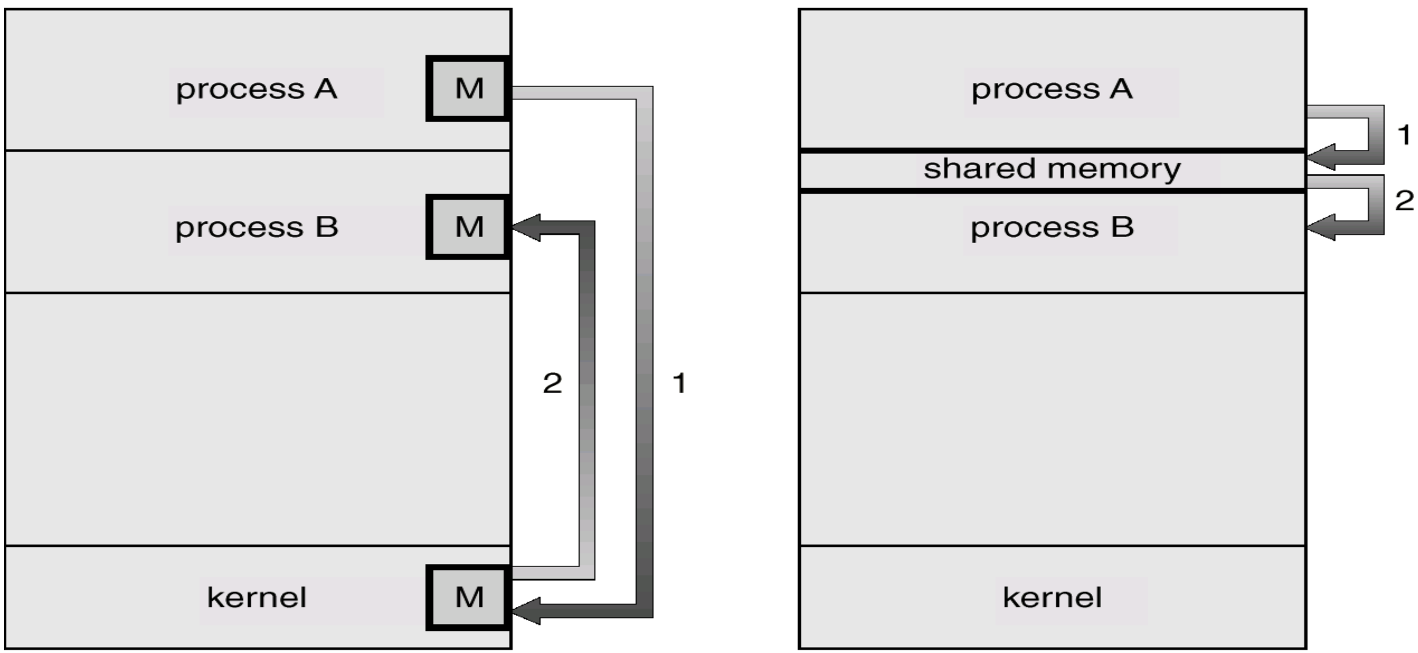
- User interface - Almost all operating systems have a user interface (UI)
  - Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch, System call,……
- Program execution
  - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)

Services to the user (Cont)

- I/O operations
  - A running program may require I/O, which may involve a file or an I/O device
- File-system manipulation
  - Obviously, programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

Services 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)



(a) (b)

Services to the user (Cont)

- 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

Services for the System Itself

- **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

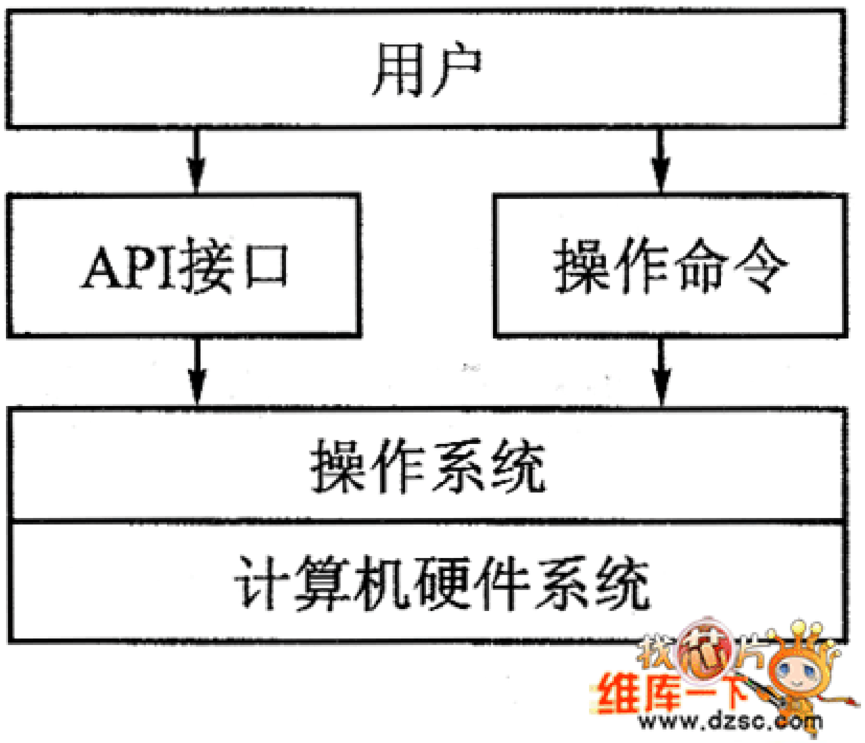
Services for the System Itself (Cont)

- **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.

User Operating System Interface

- Command interface
  - Command Line Interface (CLI) or command interpreter, Batch
  - GUI interface
  - ……
- Program interface
  - System call



User Operating System Interface - CLI

- Command Line Interface (CLI) or command interpreter 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

User Operating System Interface - GUI

- User-friendly desktop 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

User Operating System Interface - GUI

- 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)

Bourne Shell Command Interpreter

Terminal

File Edit View Terminal Tabs Help

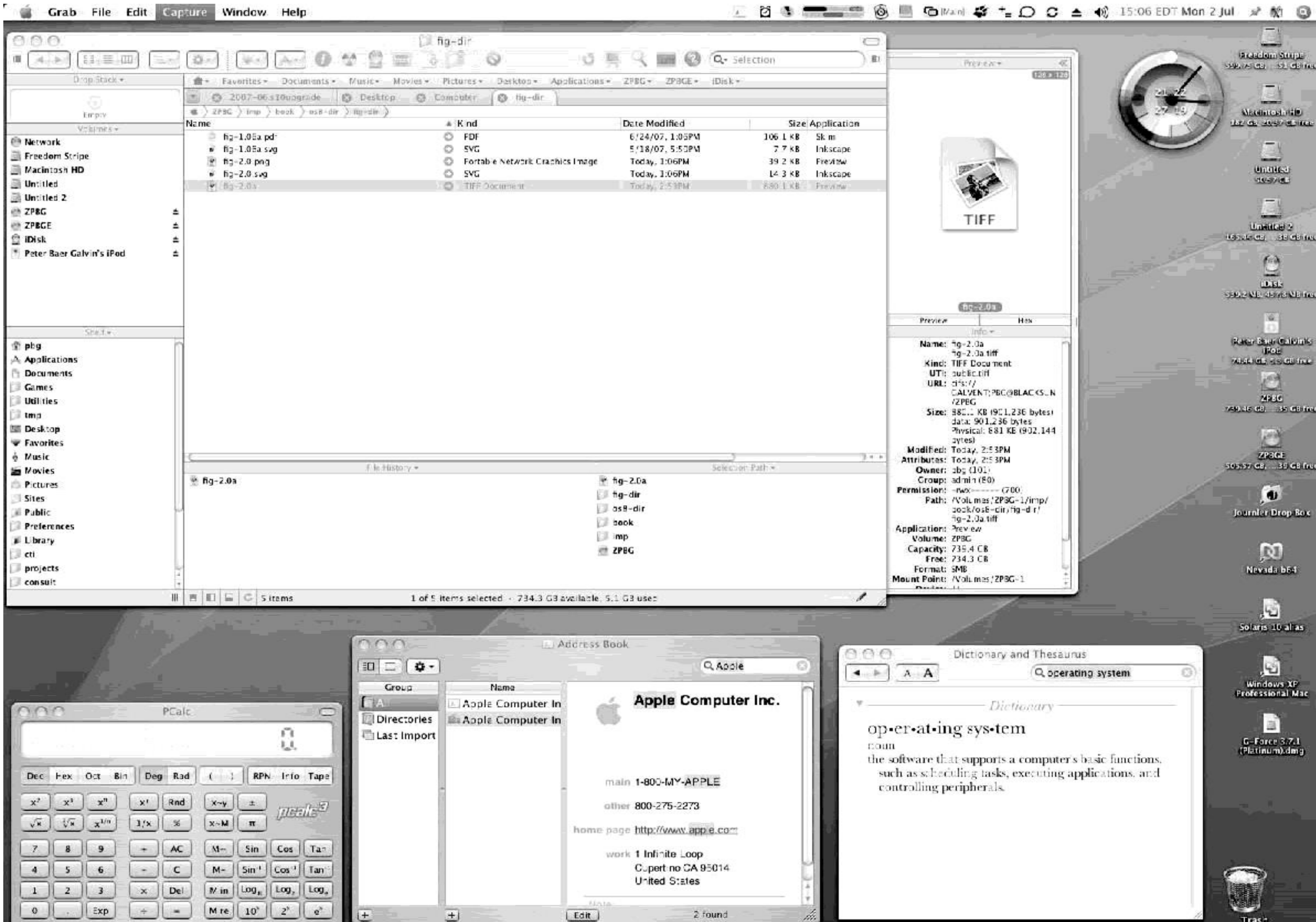
```
fd0        0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
sd0        0.0    0.2    0.0    0.2  0.0  0.0    0.4   0   0
sd1        0.0    0.0    0.0    0.0  0.0  0.0    0.0   0   0
                 extended device statistics
device     r/s    w/s   kr/s   kw/s wait actv  svc_t  %w  %b
fd0        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)#
```

Bourne Shell (Version 7 Unix shell,1977)

Touchscreen Interfaces

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

The Mac OS X GUI

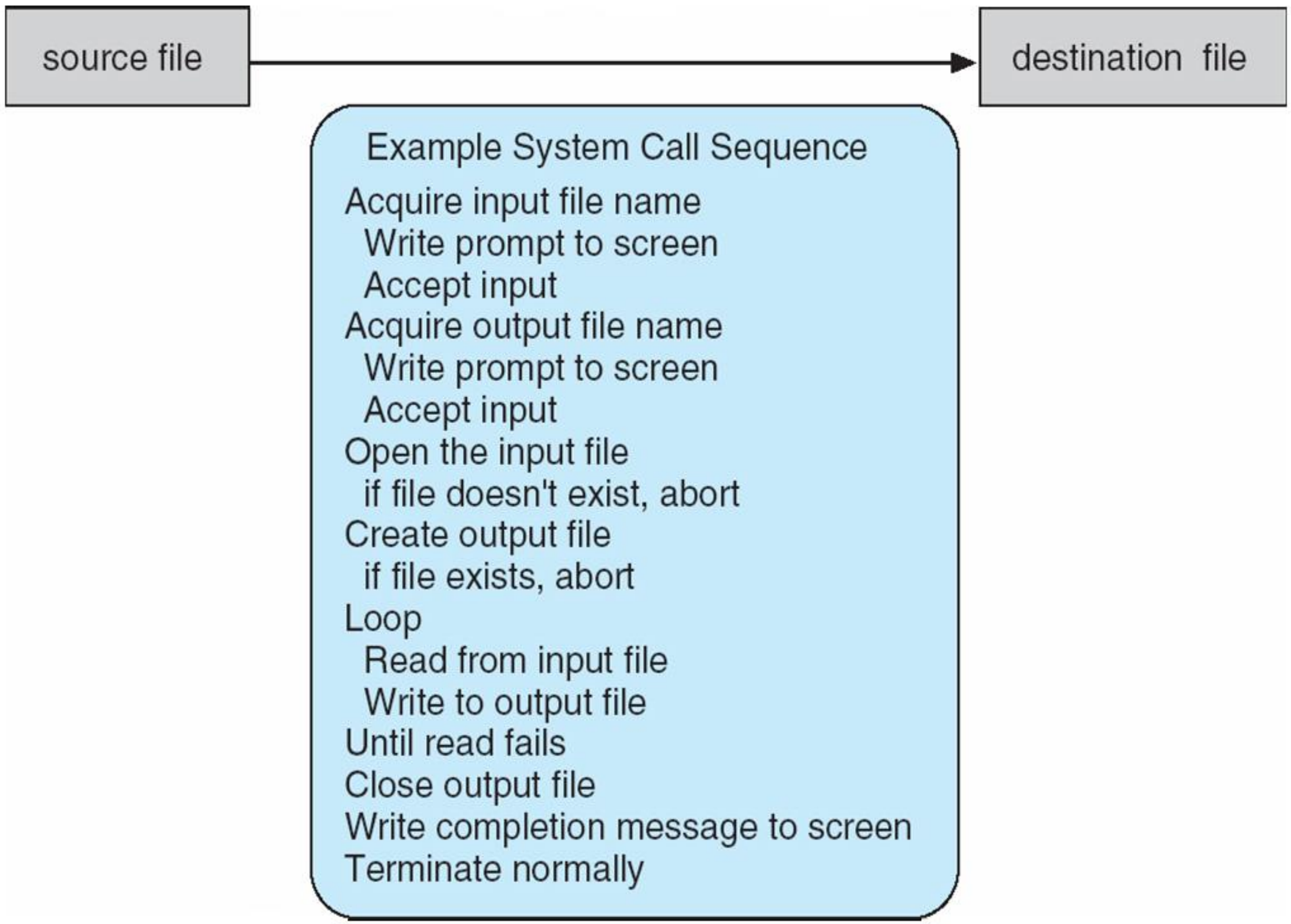


System Calls

- Programming interface to the services provided by the OS;
- 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
  - Win32 API for Windows;
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X);
  - Java API for the Java virtual machine (JVM).

Example of System Calls

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



Example of Standard API

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

return value

```
BOOL   ReadFile c  (HANDLE        file,
                    LPVOID        buffer,
                    DWORD         bytes To Read,
                    LPDWORD       bytes Read,
                    LPOVERLAPPED  ovl);
```

function name

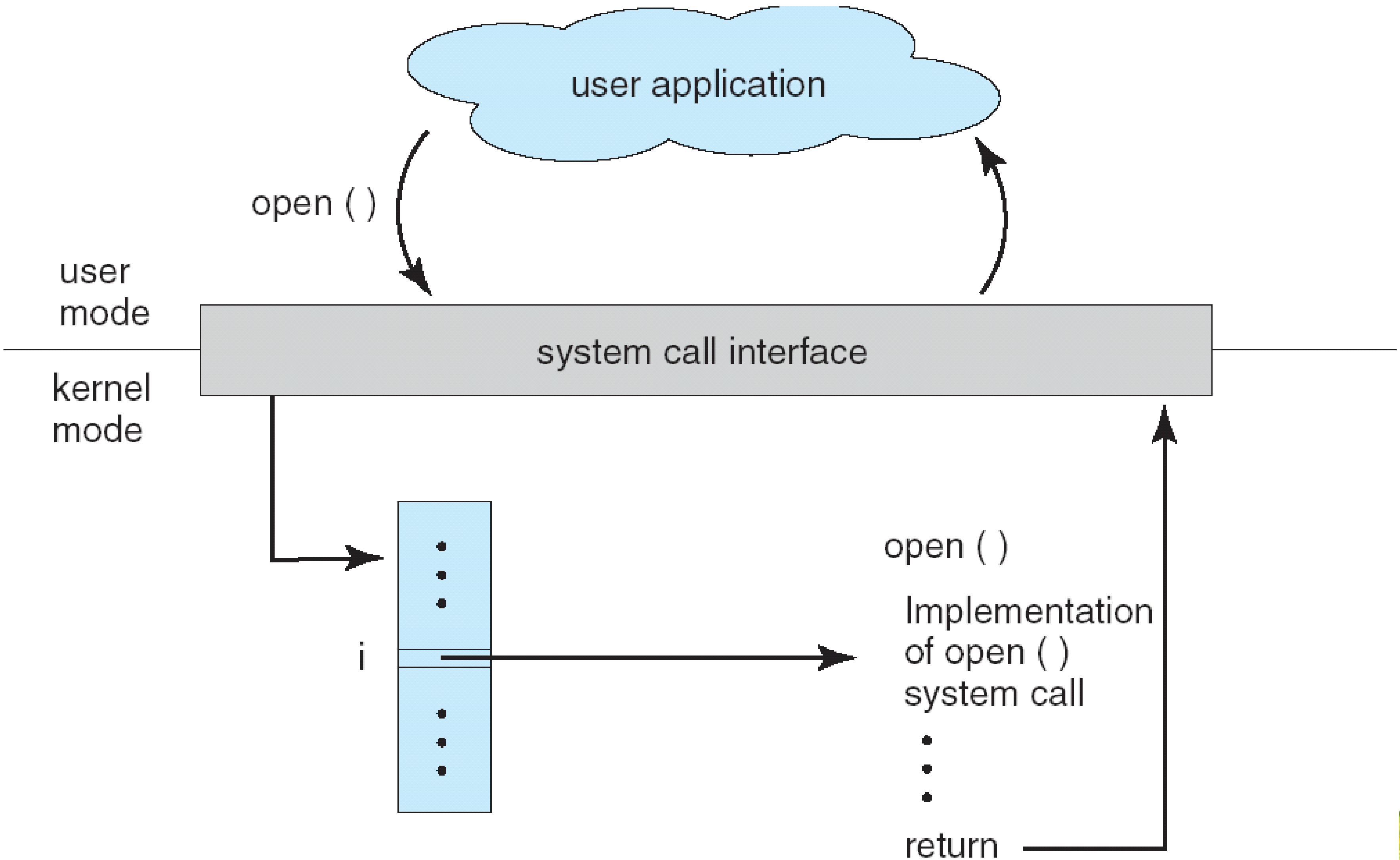
parameters

- 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

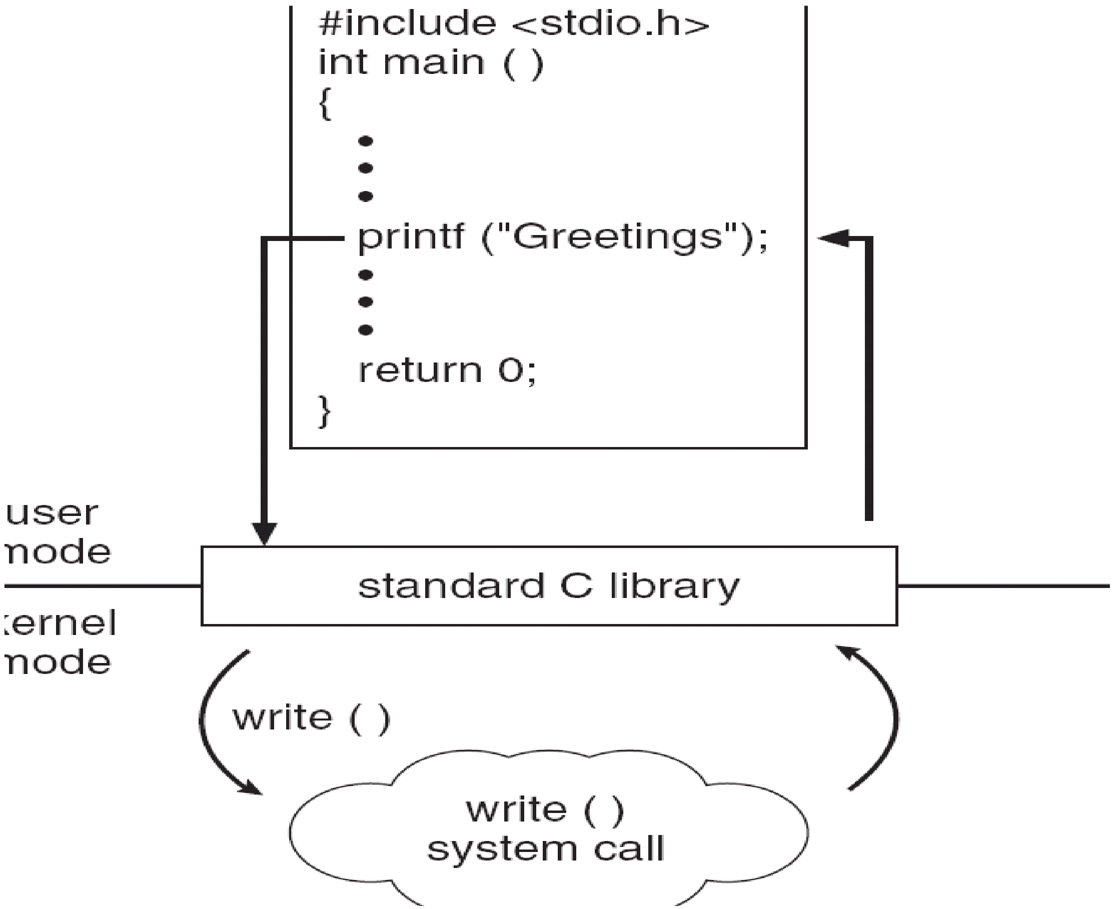
- Typically, a number associated with each system call:
  - System-call interface maintains a table indexed according to these numbers(like interrupt).
- When the user process executes the system call, the system call interface generates a software trap.
- The CPU switches to kernel mode and the system call routine is executed
- The caller need know nothing about how the system call is implemented

API – System Call – OS Relationship



Standard C Library Example

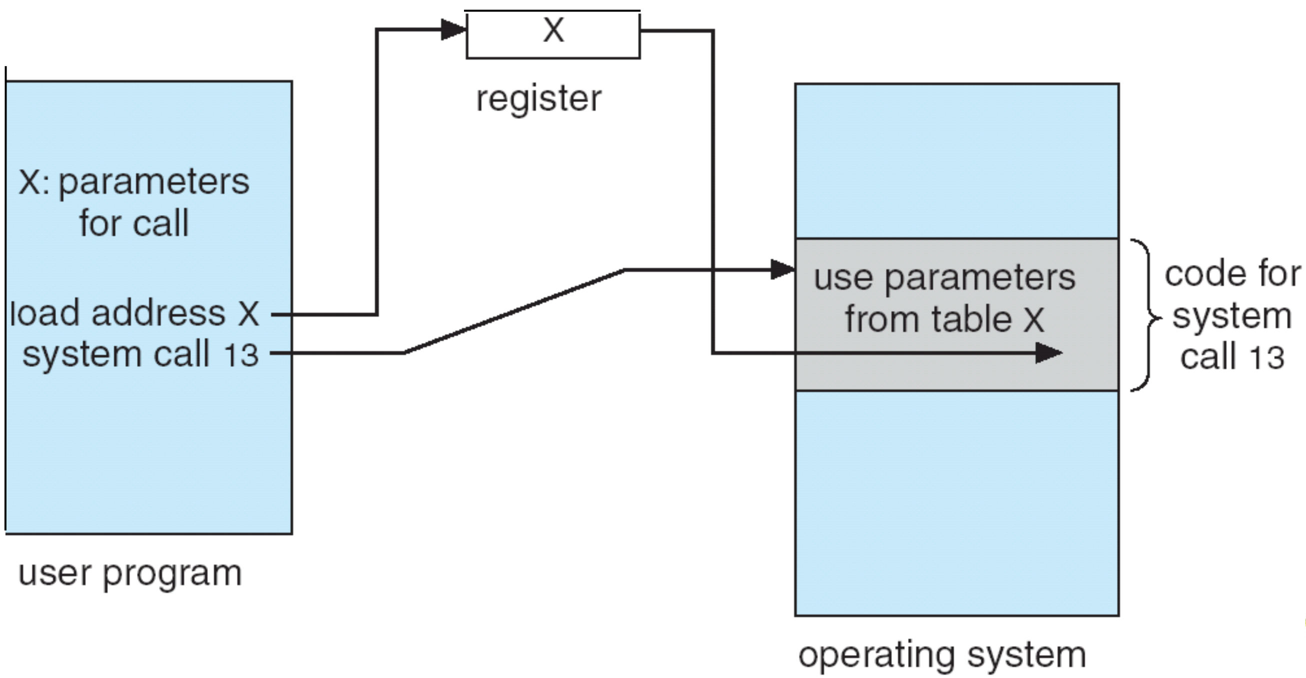
C program invoking printf() library call, which calls write() system call



System Call Parameter Passing

Three general methods used to pass parameters to the OS

- Simplest: 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(Linux and Solaris)
- Parameters placed, or *pushed,* onto the *stack* by the program and *popped* off the stack by the operating system



Types of System Calls

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

Types of System Calls

- Process control
  - create process, terminate process
  - end, abort
  - load, execute
  - 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

Types of System Calls

- File management
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes
- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes

Types of System Calls (Cont.)

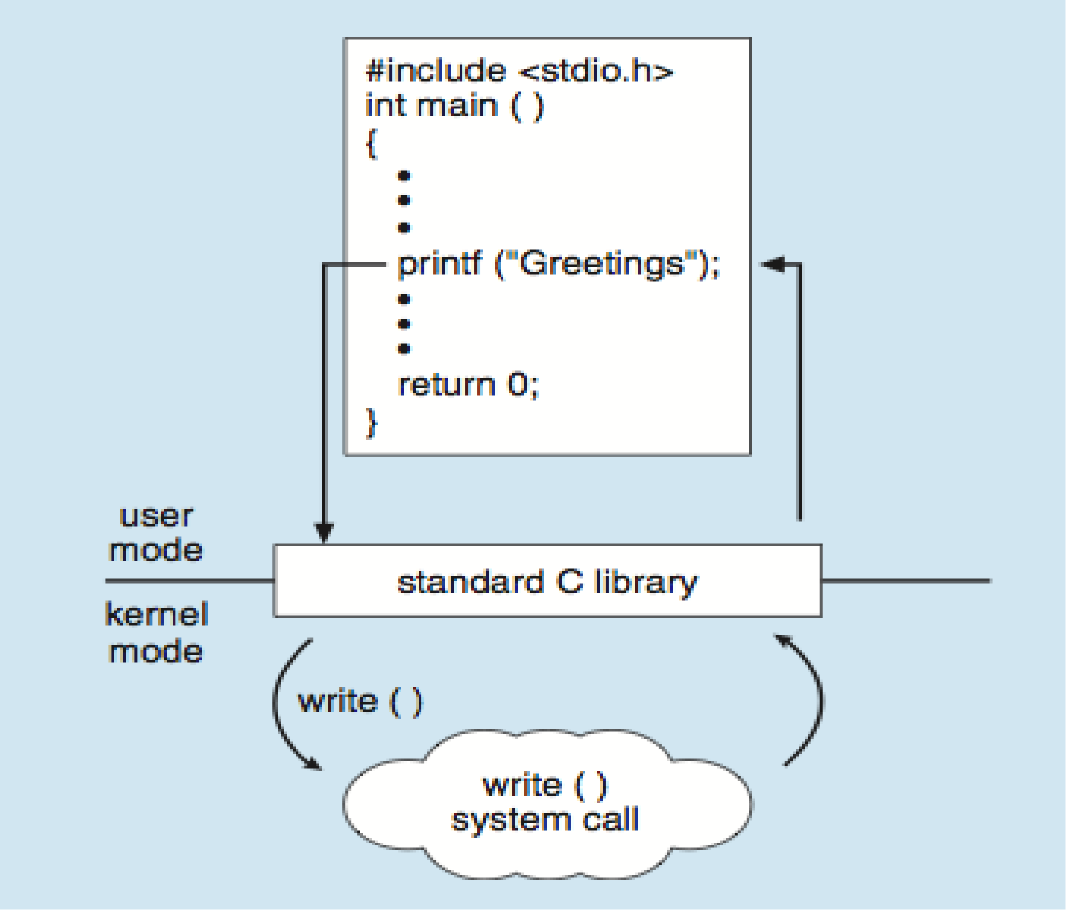
- Communications
  - create, delete communication connection
  - send, receive messages if **message passing model** to **host name** or **process name**
    - From **client** to **server**
  - **Shared-memory model** create and gain access to memory regions
  - transfer status information
  - attach and detach remote devices
- Protection
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

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() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Standard C Library Example

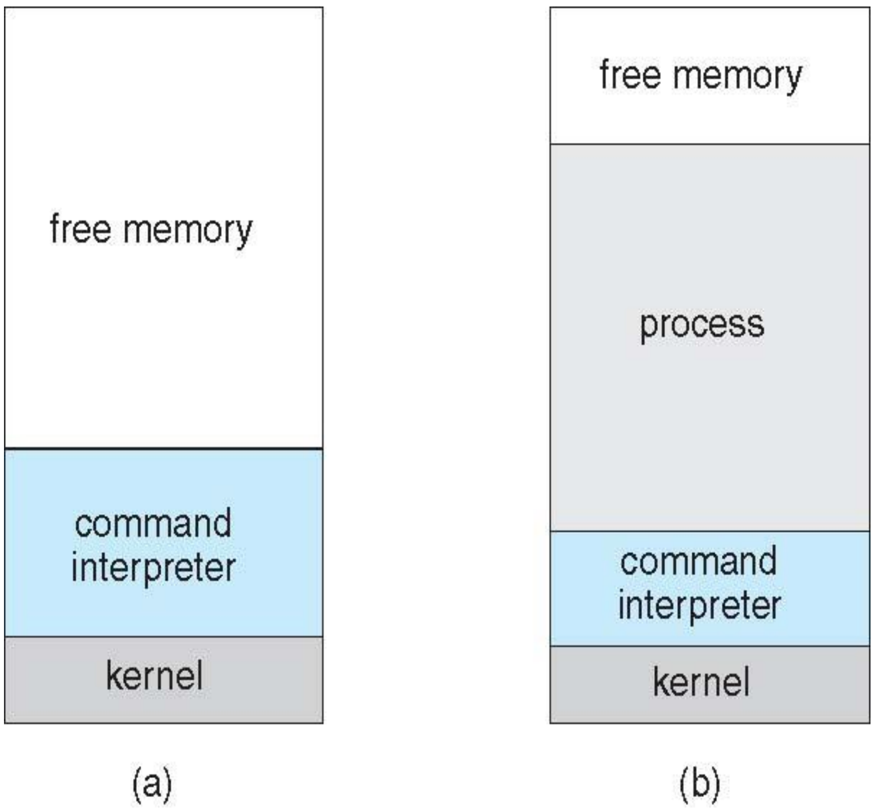
C program invoking printf() library call, which calls write() system call



问题:API与system call的区别?

Example: MS-DOS

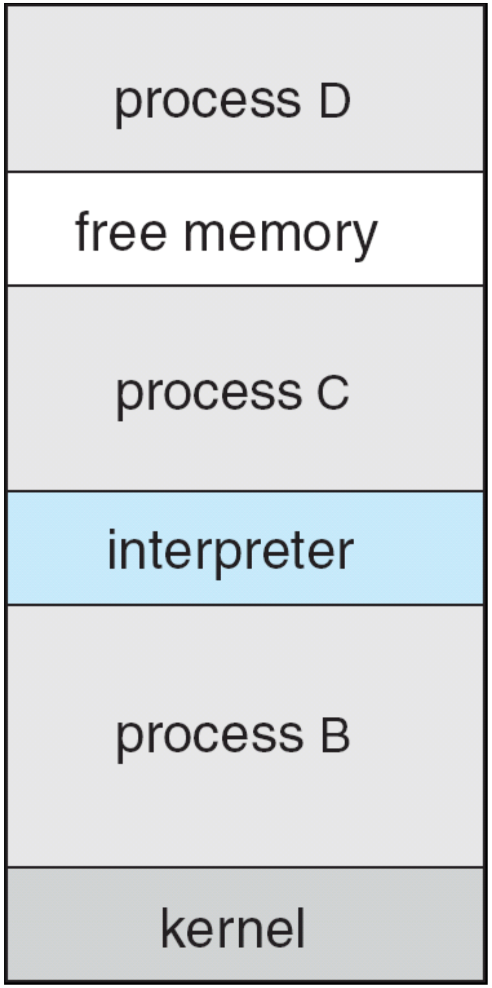
- Single-tasking
- Shell invoked when system booted
- Simple method to run program
  - No process created
- Single memory space
- Loads program into memory, overwriting all but the kernel
- Program exit -> shell reloaded



At system startup — running a program

Example: FreeBSD

- Unix variant
- Multitasking
- User login -> invoke user’s choice of shell
- Shell executes fork() system call to create process
  - Executes exec() to load program into process
  - Shell waits for process to terminate or continues with user commands
- Process exits with:
  - code = 0 – no error
  - code > 0 – error code



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
- 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.)

- **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

System Programs (Cont.)

- **Background Services**
  - Launch at boot time
    - Some for system startup, then terminate
    - Some from system boot to shutdown
  - Provide facilities like disk checking, process scheduling, error logging, printing
  - Run in user context not kernel context
  - Known as **services**, **subsystems**, **daemons**

- **Application programs**
  - Don’t pertain to system
  - Run by users
  - Not typically considered part of OS
  - Launched by command line, mouse click, finger poke

Operating System Design and Implementation

- OS is an complex system, Internal structure of different Operating Systems can vary widely
- Affected by choice of hardware, type of system, Start by defining goals and specifications
- *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

Operating System Design and Implementation (Cont)

- Important principle to separate:

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

- It allows maximum flexibility if policy decisions are to be changed later (ie. Sys. Prog. vs. Sys. Call, editer, copy etc;以文件复制为例:可以通过基本的调用组合实现，也可以通过一组调用组合，显然前者更加灵活。

- Specifying and designing an OS is highly creative task of **software engineering**

Implementation

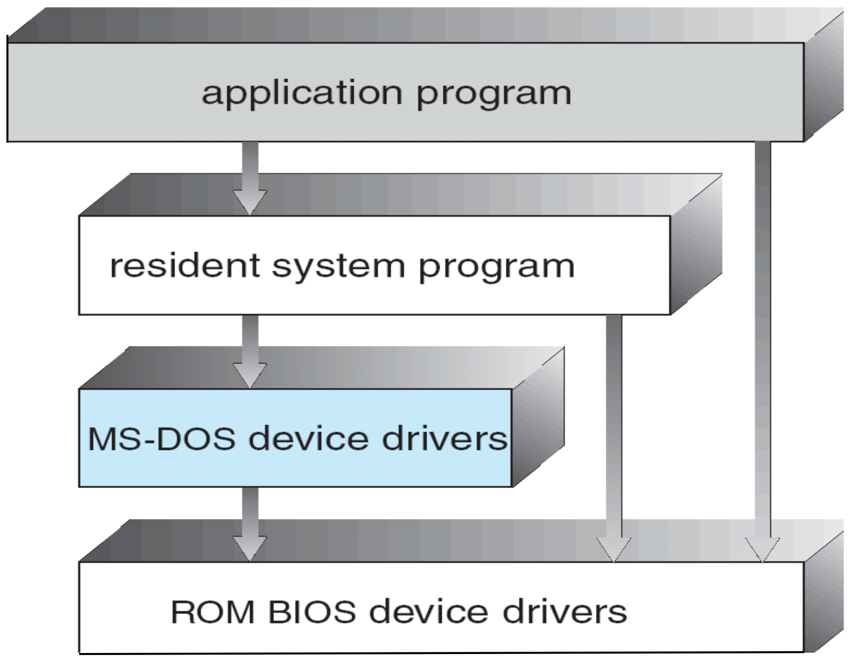
- Much variation
  - Early OSes in assembly language
  - Then system programming languages like Algol, PL/1
  - Now C, C++
- Actually usually a mix of languages
  - Lowest levels in assembly
  - Main body in C
  - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
- More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

Operating System Structure

- General-purpose OS is very large program
- Various ways to structure ones
  - Simple structure – MS-DOS
  - More complex -- UNIX
  - Layered – an abstrcation
  - Microkernel -Mach

Simple Structure

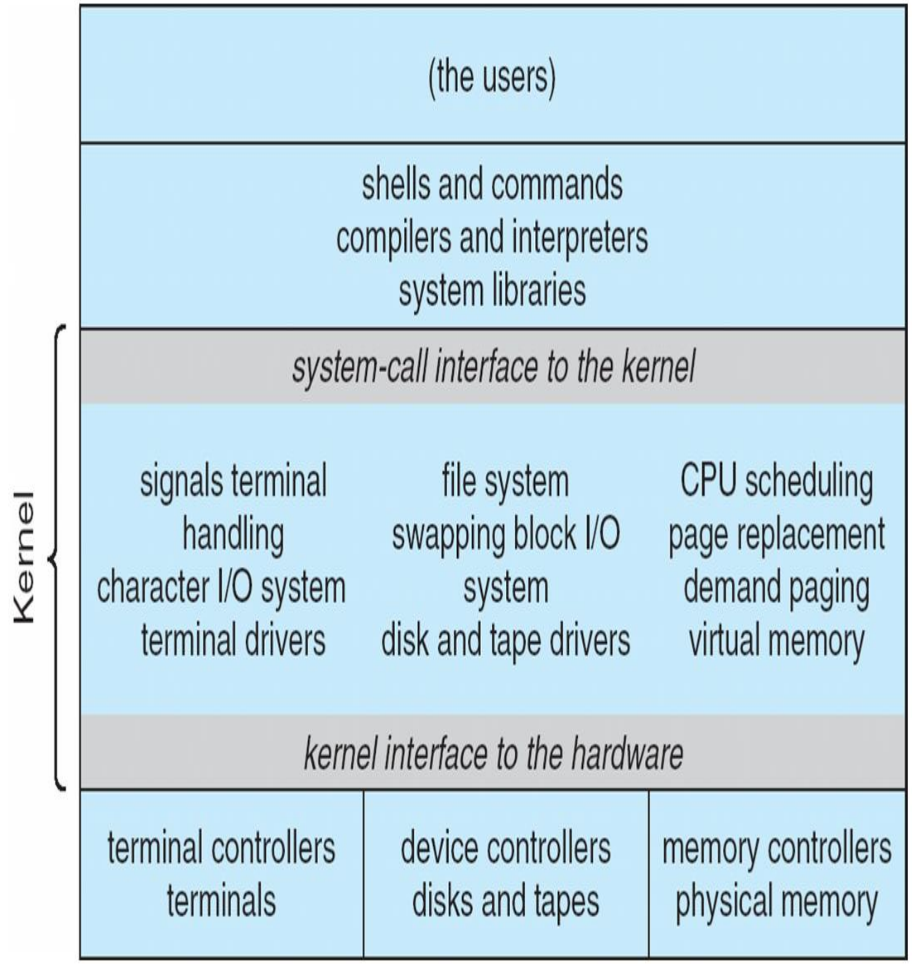
- MS-DOS – written to provide the most functionality in the least space
  - Not divided into modules, Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated



Direct access

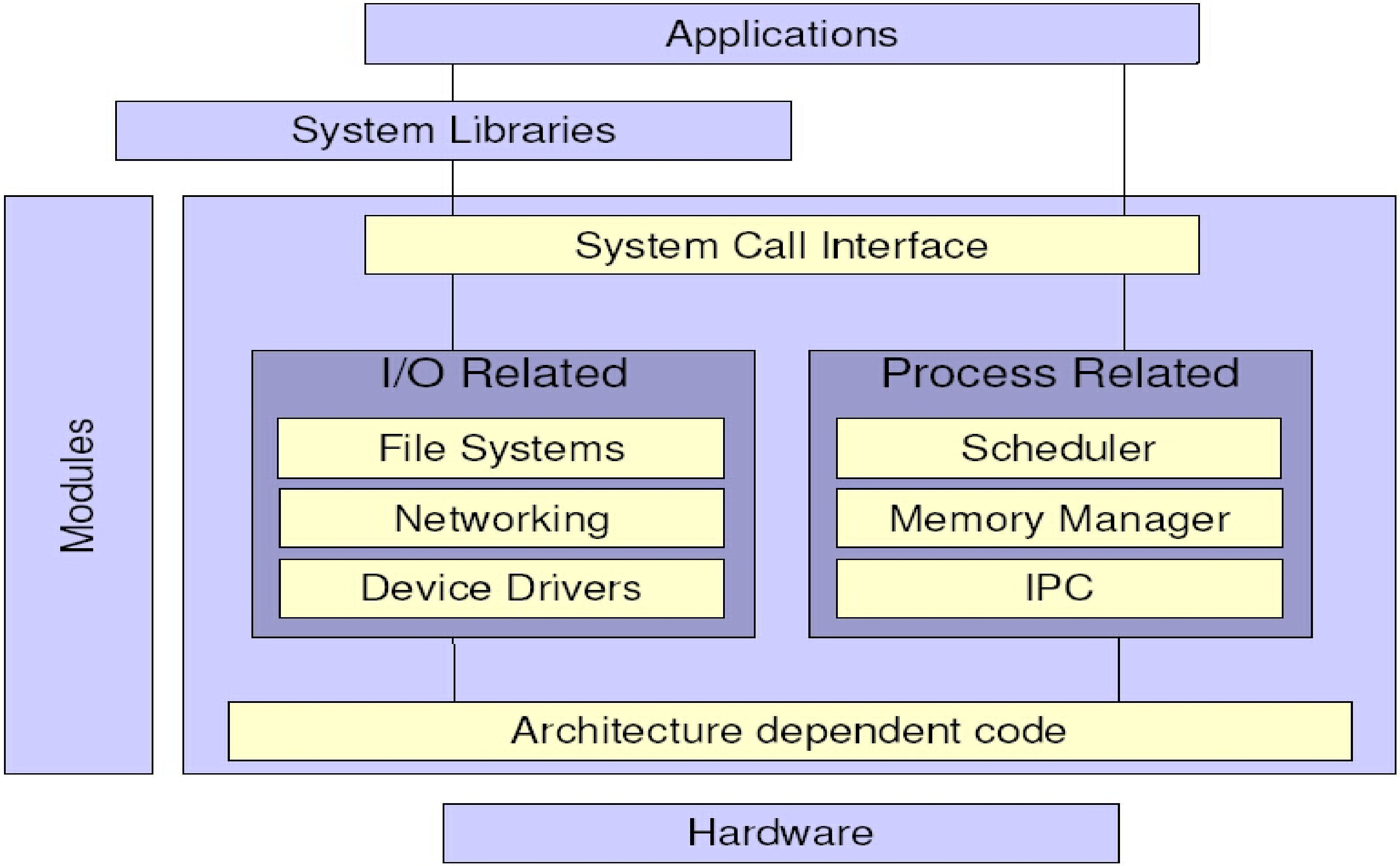
Non Simple Structure -- UNIX

- 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



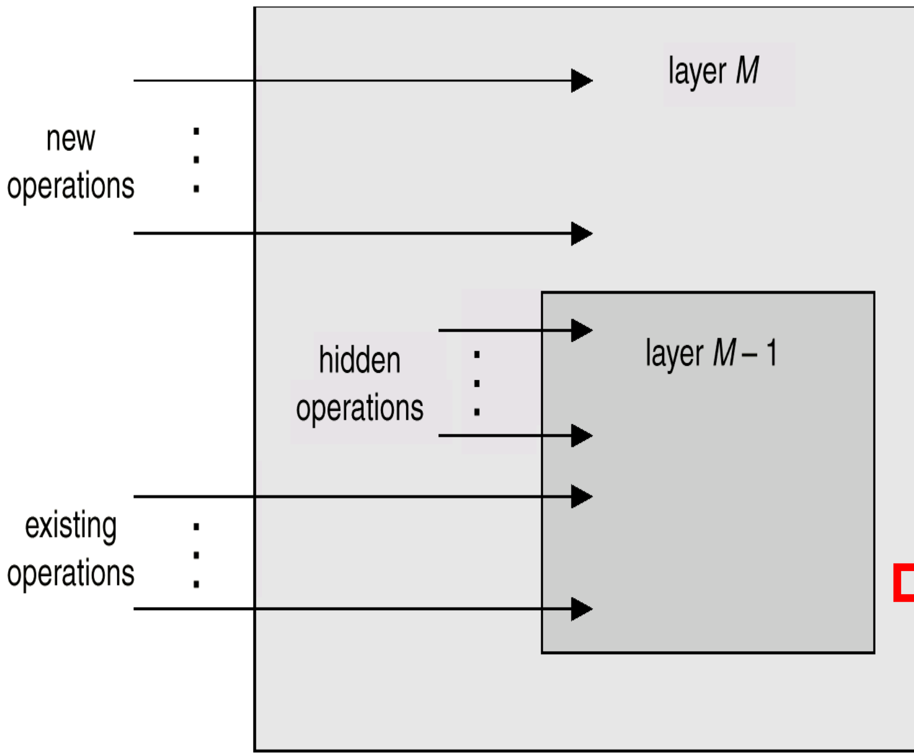
Linux Structure

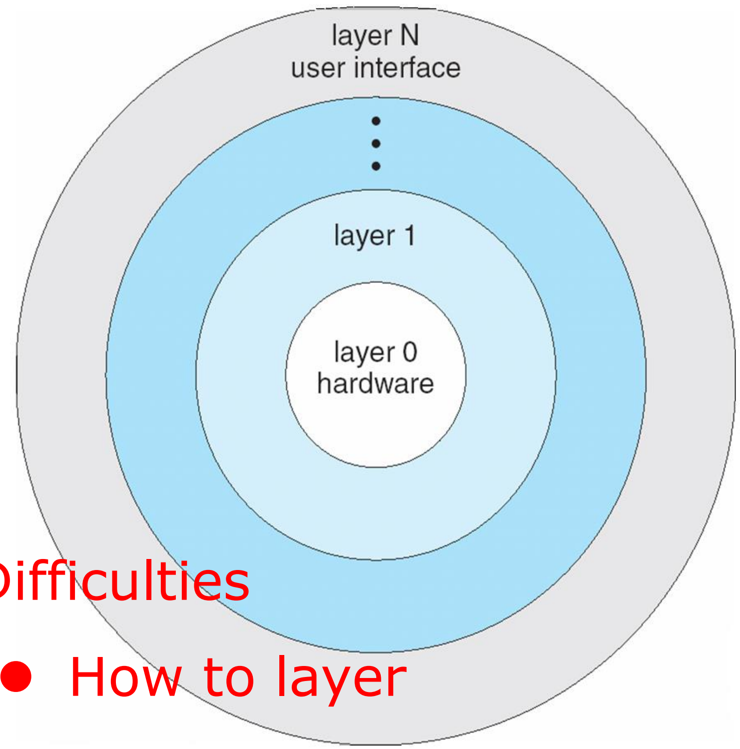
Similar 2-layered structure as Unix.



Layered Approach

- The operating system is divided into a number of layers
  - The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- Each layer is a module, layers are selected such that each uses functions (operations) and services of only lower-level layers





- Difficulties
  - How to layer
  - How many layers

Layered Structure of the THE OS

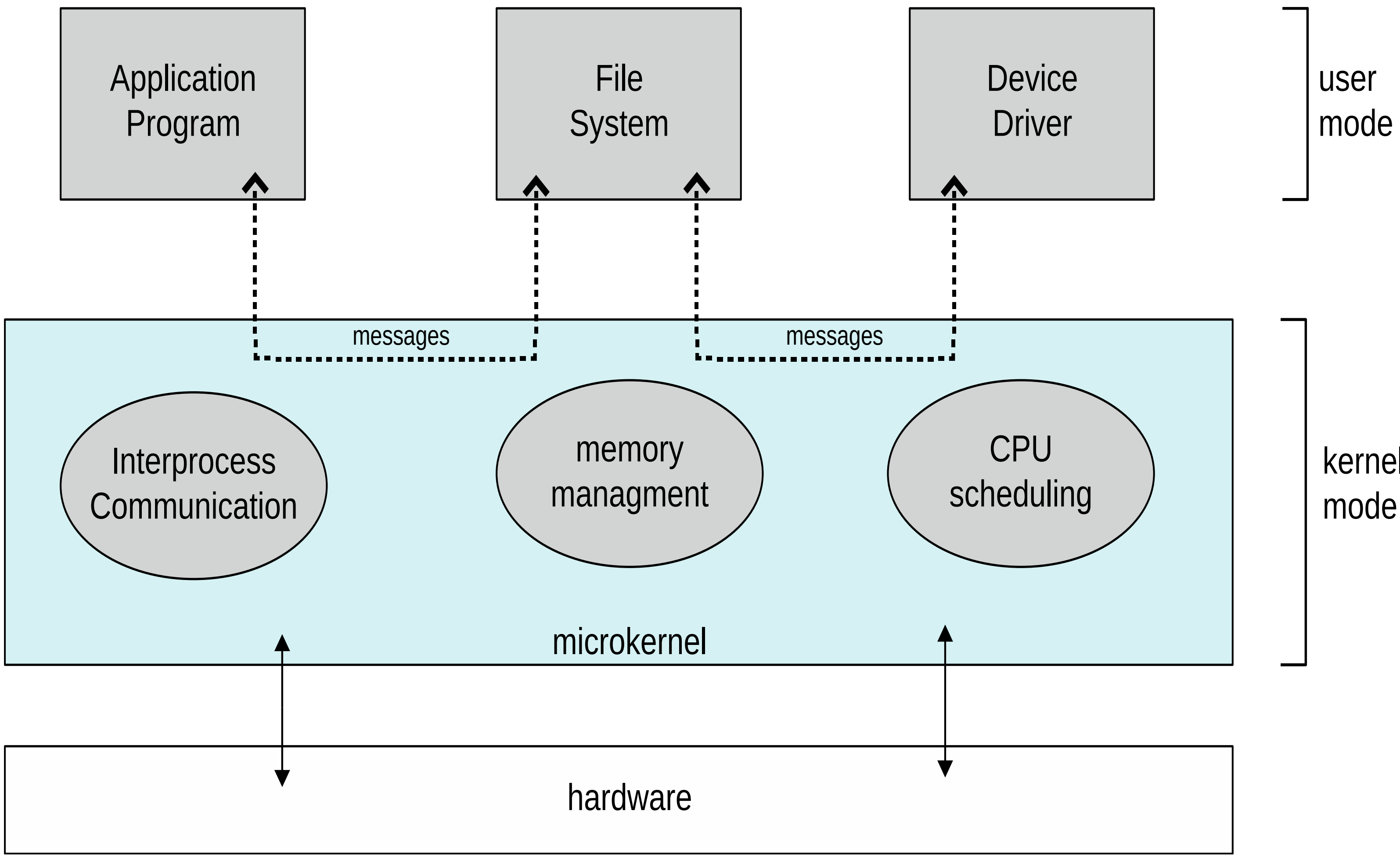
- A layered design was first used in THE operating system. Its six layers are as follows: THE操作系统首先使用层次化设计。

Layer 5: user programs
Layer 4: buffering for input and output
Layer 3: operator-console device driver
Layer 2: memory management
Layer 1: CPU scheduling

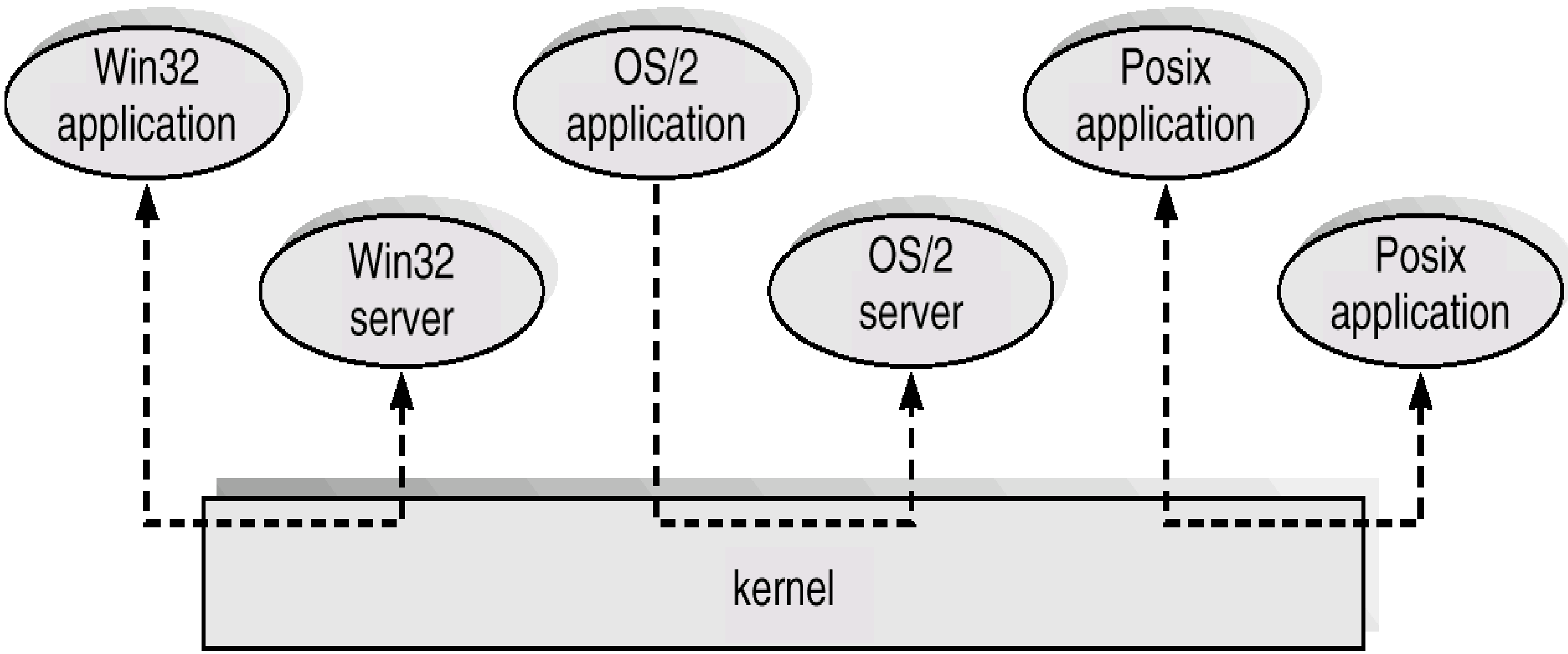
Microkernel System Structure

- Moves as much from the kernel into user space
- **Mach** example of **microkernel**
  - Mac OS X kernel (**Darwin**) partly based on Mach
- Communication takes place between user modules using **message passing**
- 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

Microkernel System Structure



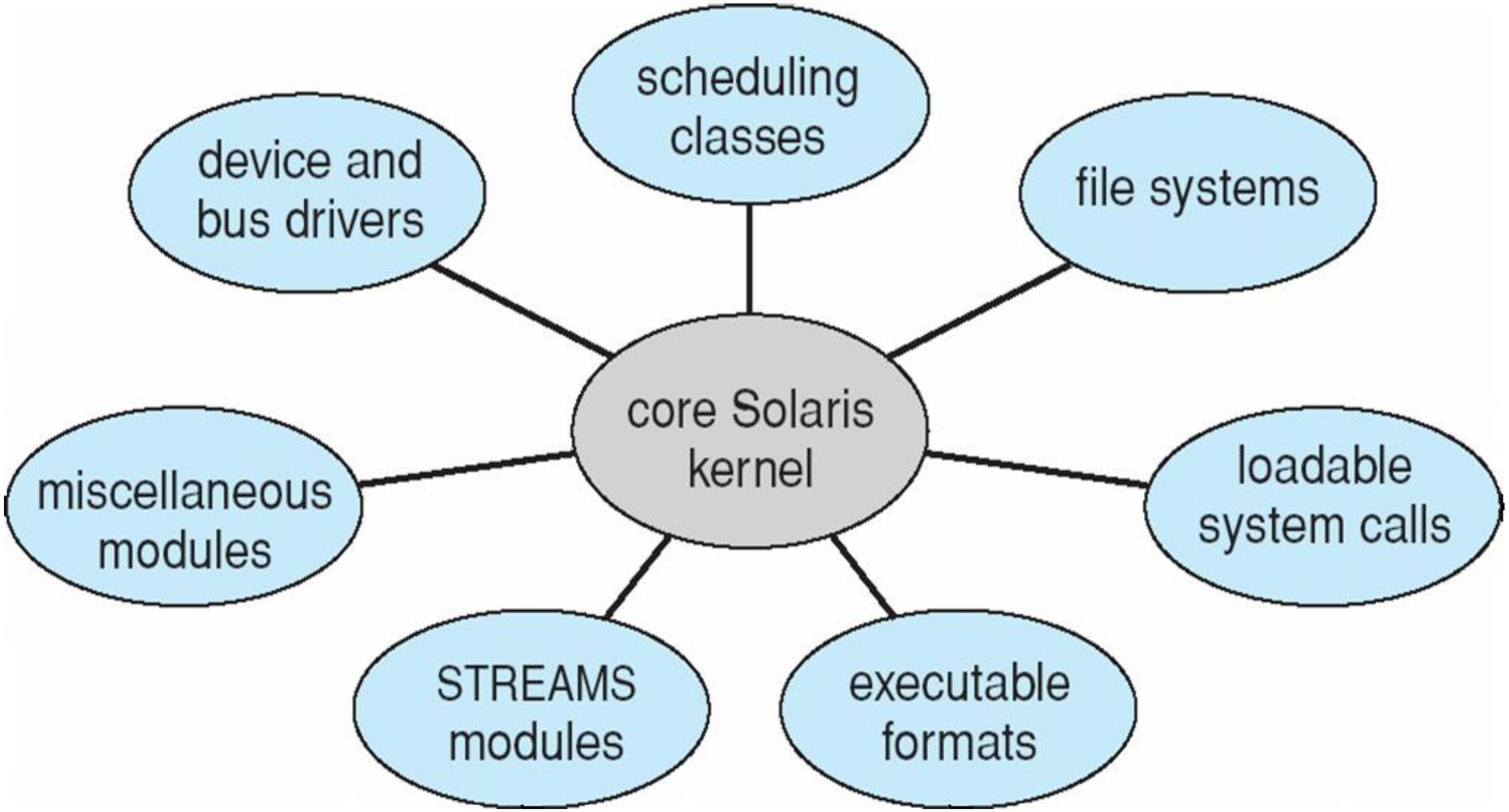
Windows NT Client-Server Structure



Modules

- 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
- Overall, similar to layers but with more flexible
  - Linux, Solaris, etc

Solaris Modular Approach

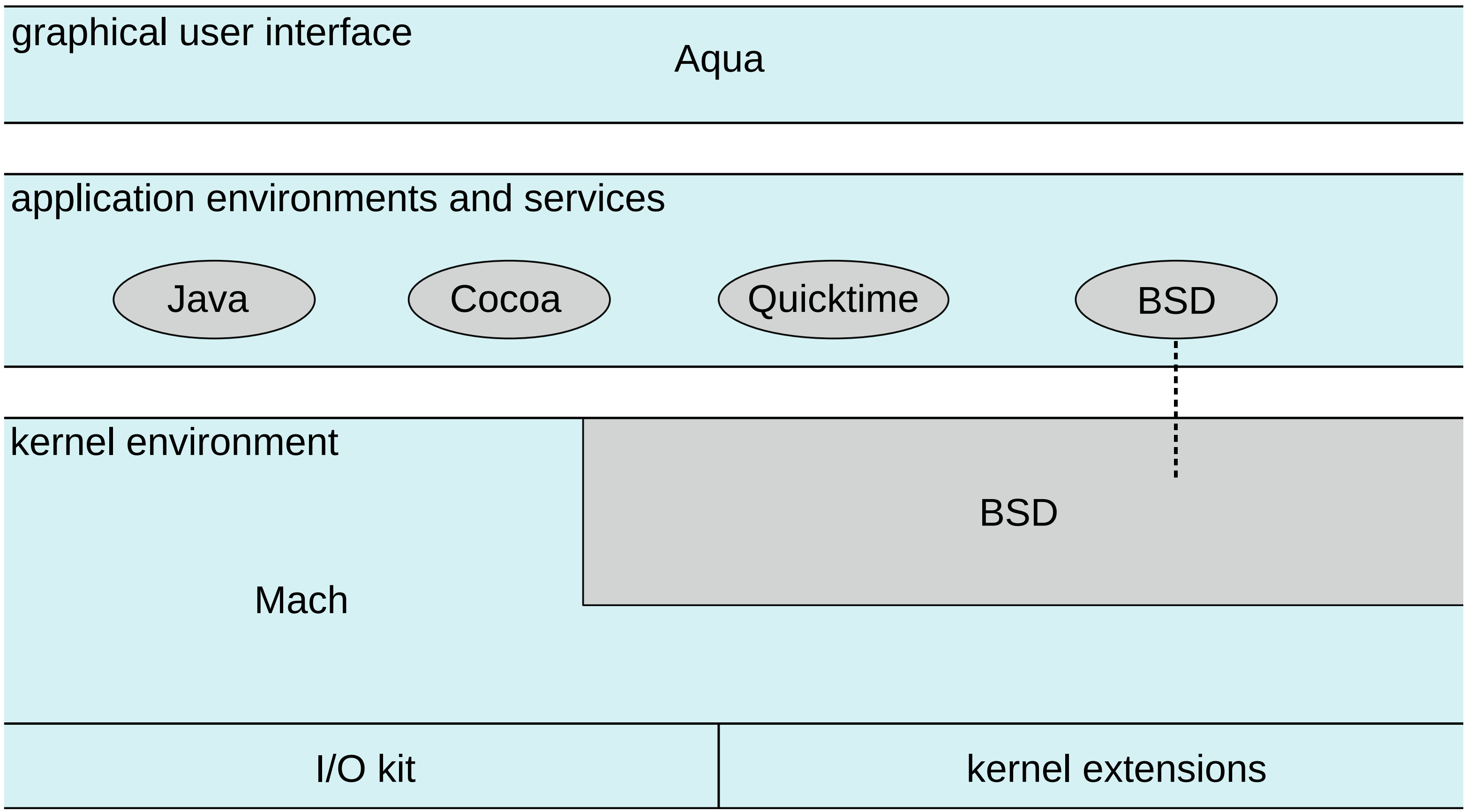


Hybrid Systems

- Most modern operating systems are 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（iOS运行环境）, Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called **kernel extensions**)

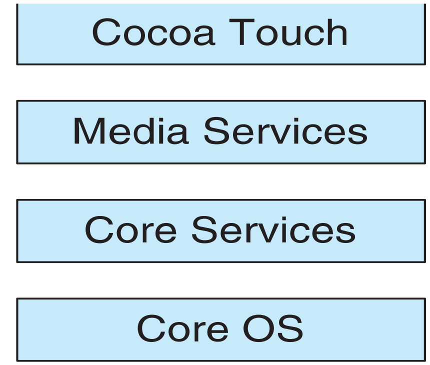
思考问题：monolithic（单内核）与微内核的区别？

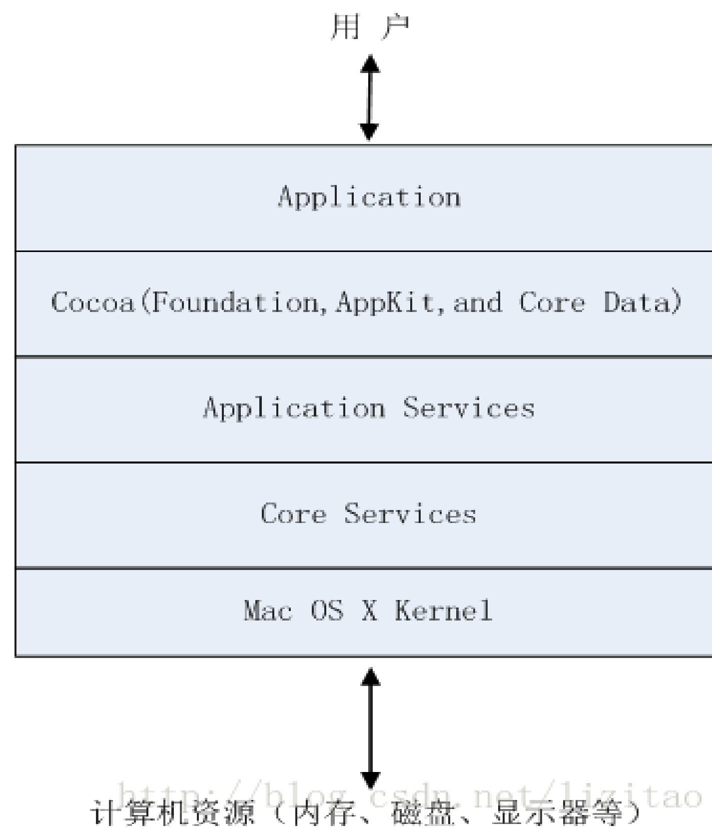
Mac OS X Structure



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

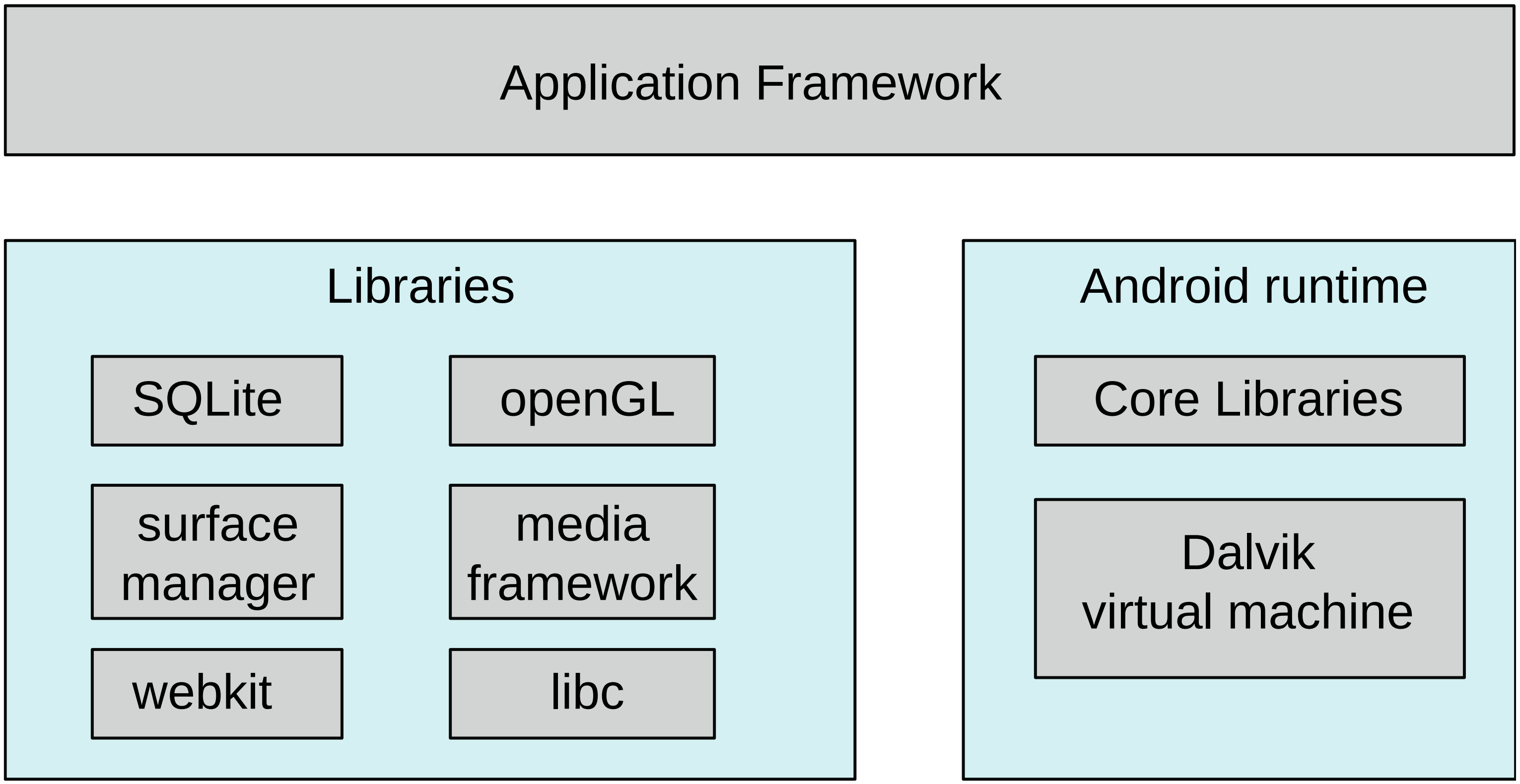




Android

- 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（Google的Virtual Machine——Dalvik Virtual Machine（DVM））
  - 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



Virtual Machines History and Benefits

Very old idea

- IBM in 1960's used term for virtual machines (e.g. CP-40)
- First appeared commercially in IBM mainframes in 1972
- 21st century revival (Xen, VMWare etc.)

Formal Requirements for Virtualizable Third Generation Architectures

Gerald J. Popek
University of California, Los Angeles
and
Robert P. Goldberg
Honeywell Information Systems and
Harvard University

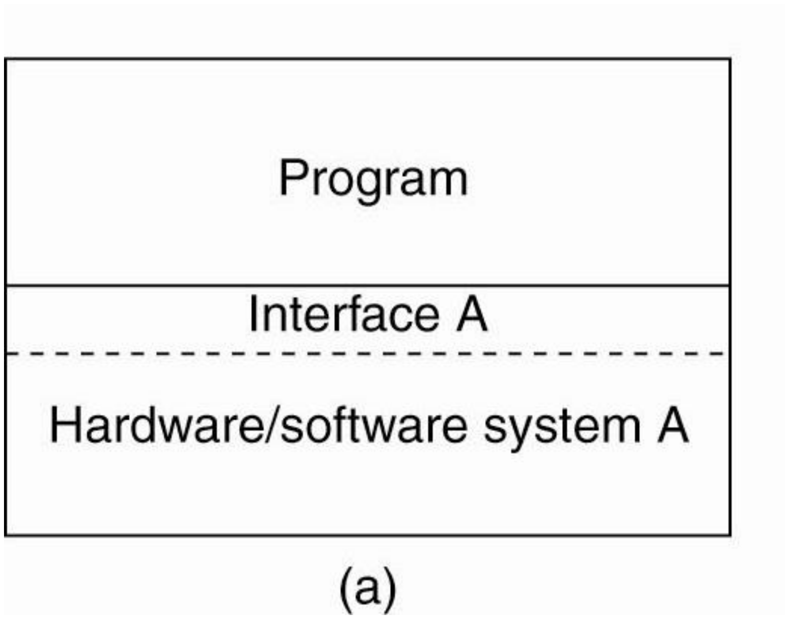
Virtual machine systems have been implemented on a limited number of third generation computer systems, e.g. CP-67 on the IBM 360/67. From previous empirical studies, it is known that certain third generation computer systems, e.g. the DEC PDP-10, cannot support a virtual machine system. In this paper, model of a third-generation-like computer system is developed. Formal techniques are used to derive precise sufficient conditions to test whether such an architecture can support virtual machines.

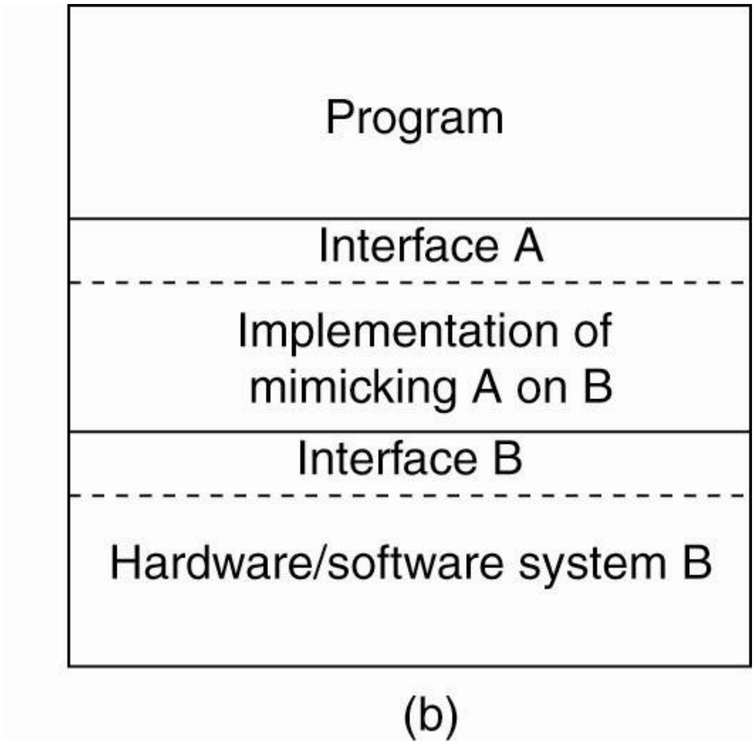
Communications of the ACM, vol 17, no 7, 1974, pp.412-421

Virtual Machines

- A virtual machine takes the layered approach to its logical conclusion.
- A virtual machine provides an interface *identical(同一的)*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

The Role of Virtualization

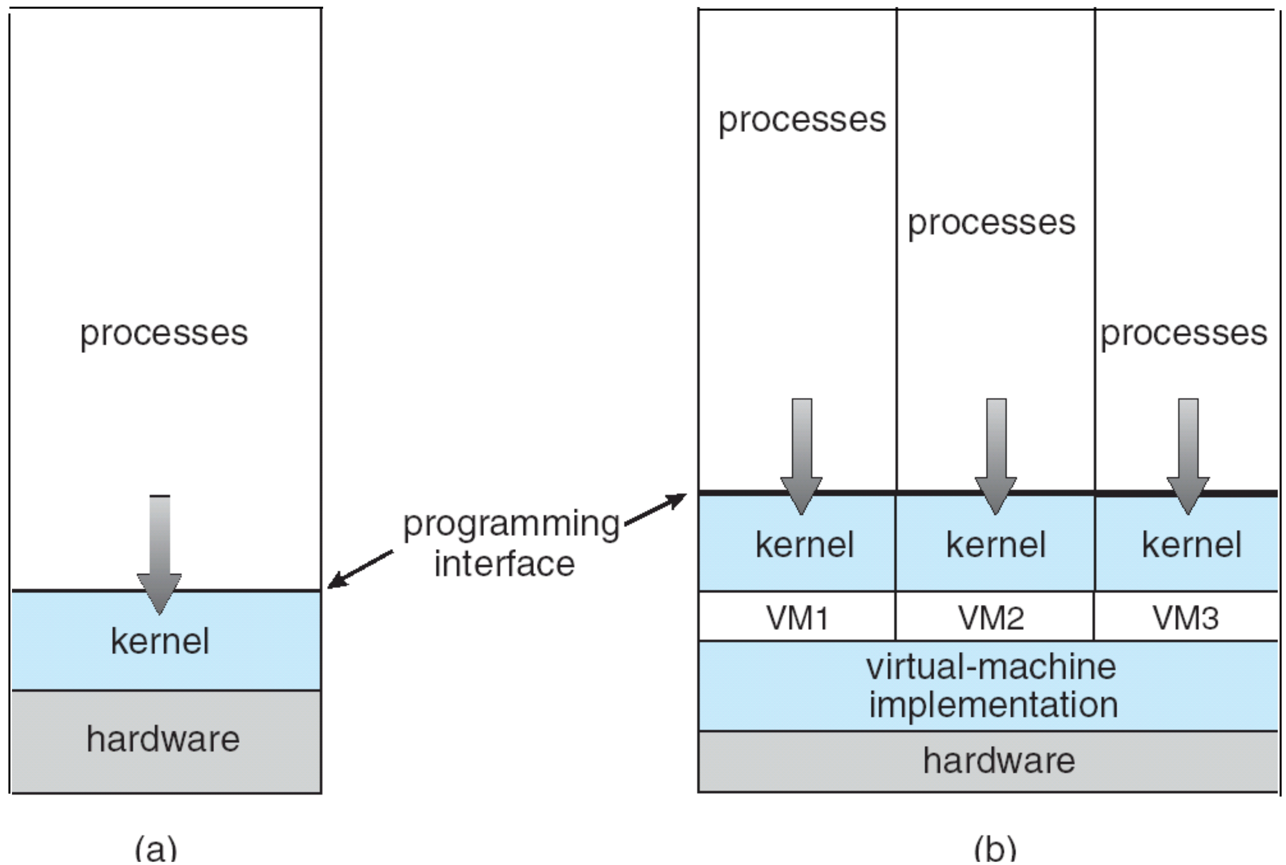




(a) General organization between a program, interface, and system.

(b) General organization of virtualizing system A on top of system B.

Virtual Machines (Cont)

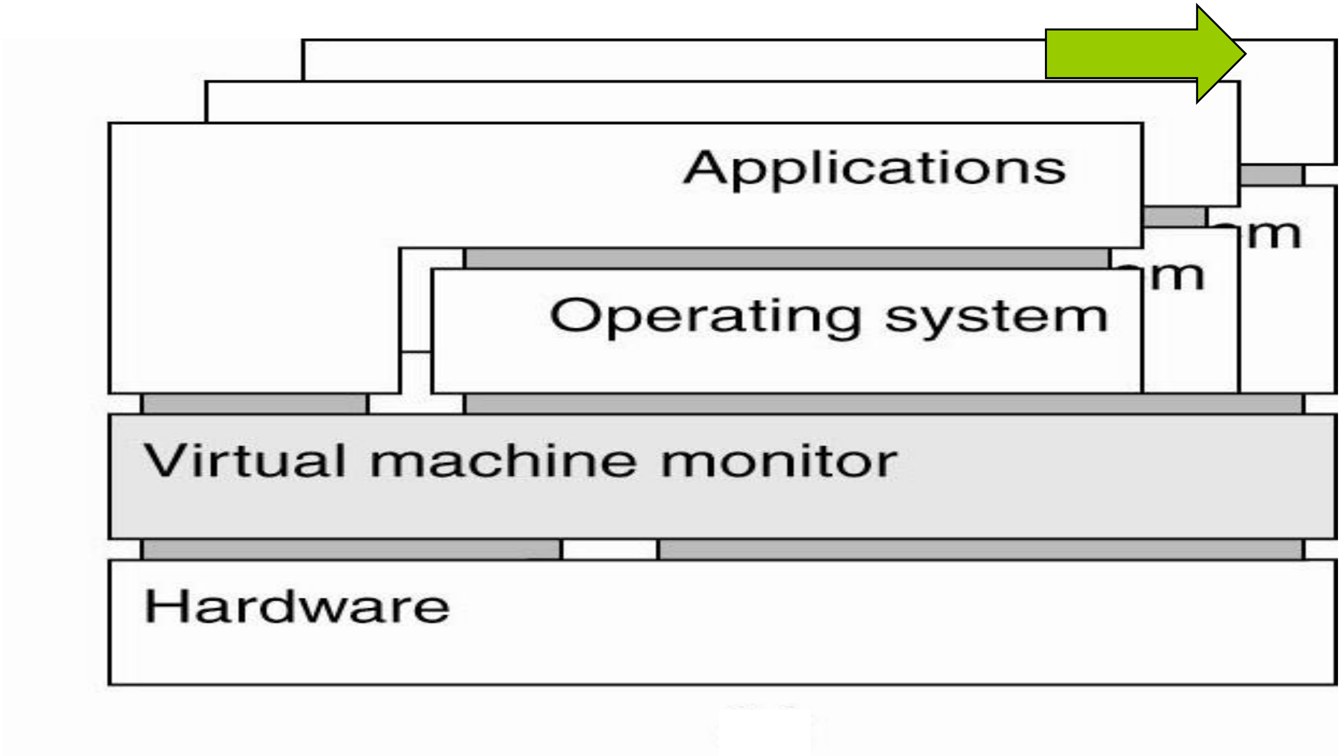


(a) Nonvirtual machine (b) virtual machine

Definitions

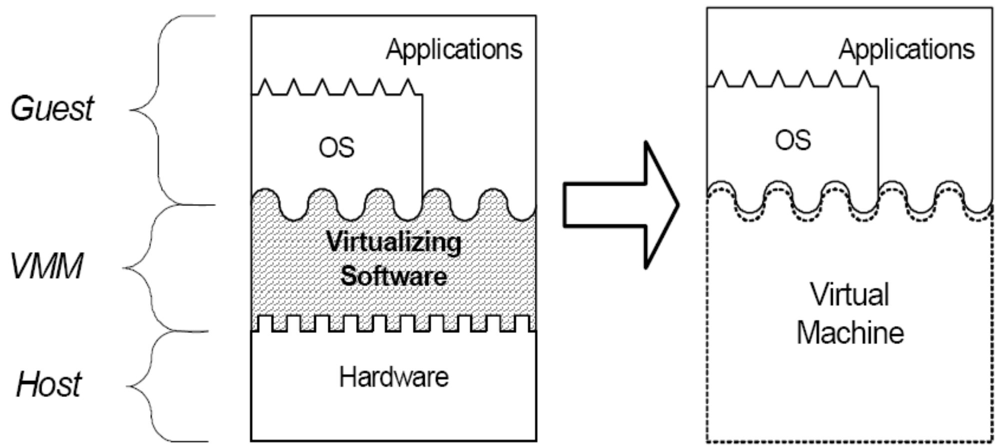
Virtual Machine Monitor (VMM)

A virtualization system that partitions a single physical “machine” into multiple virtual machines.

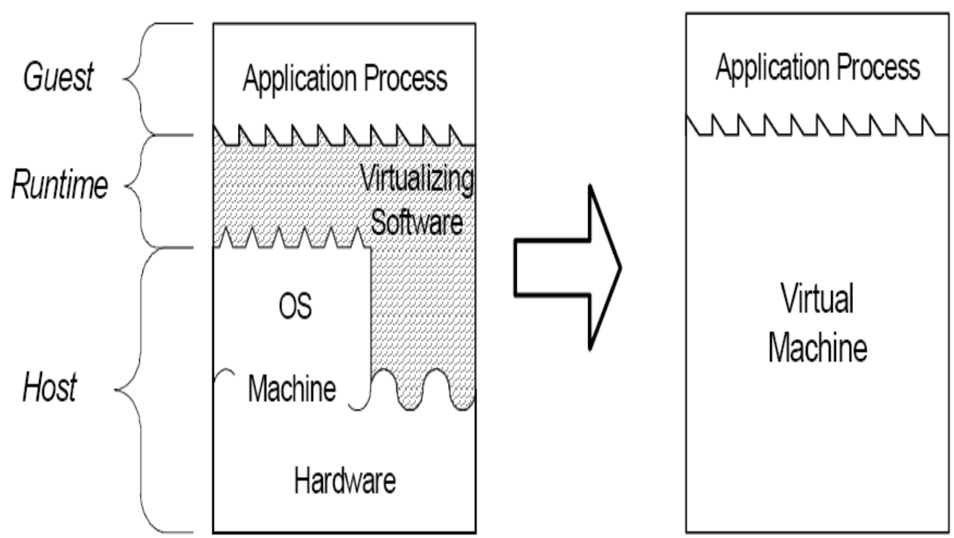


VMM Types

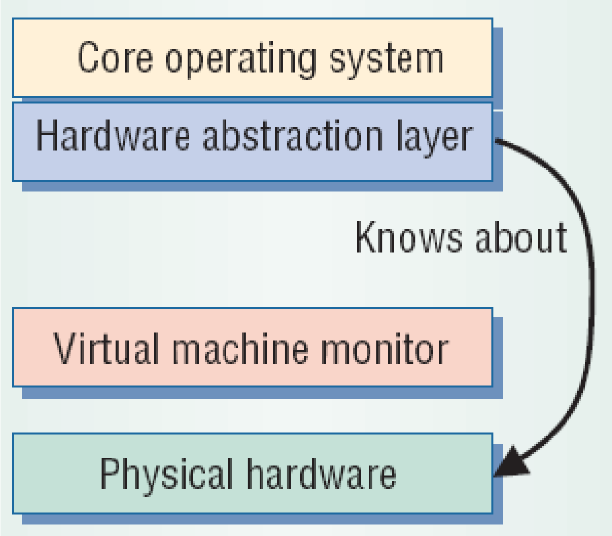
- System



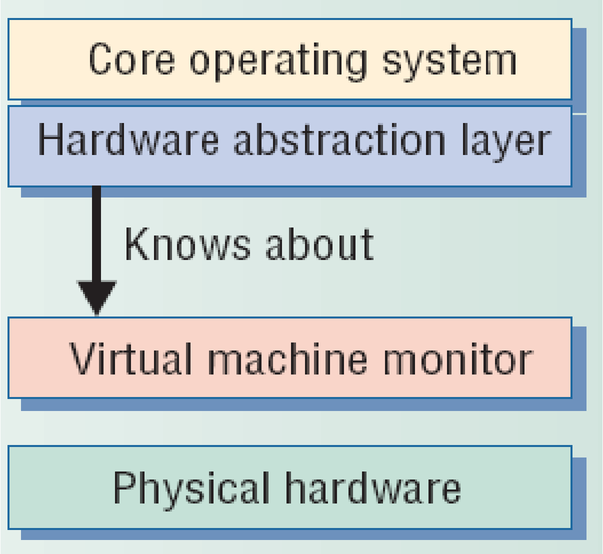
- Process



Para virtualization vs Full virtualization

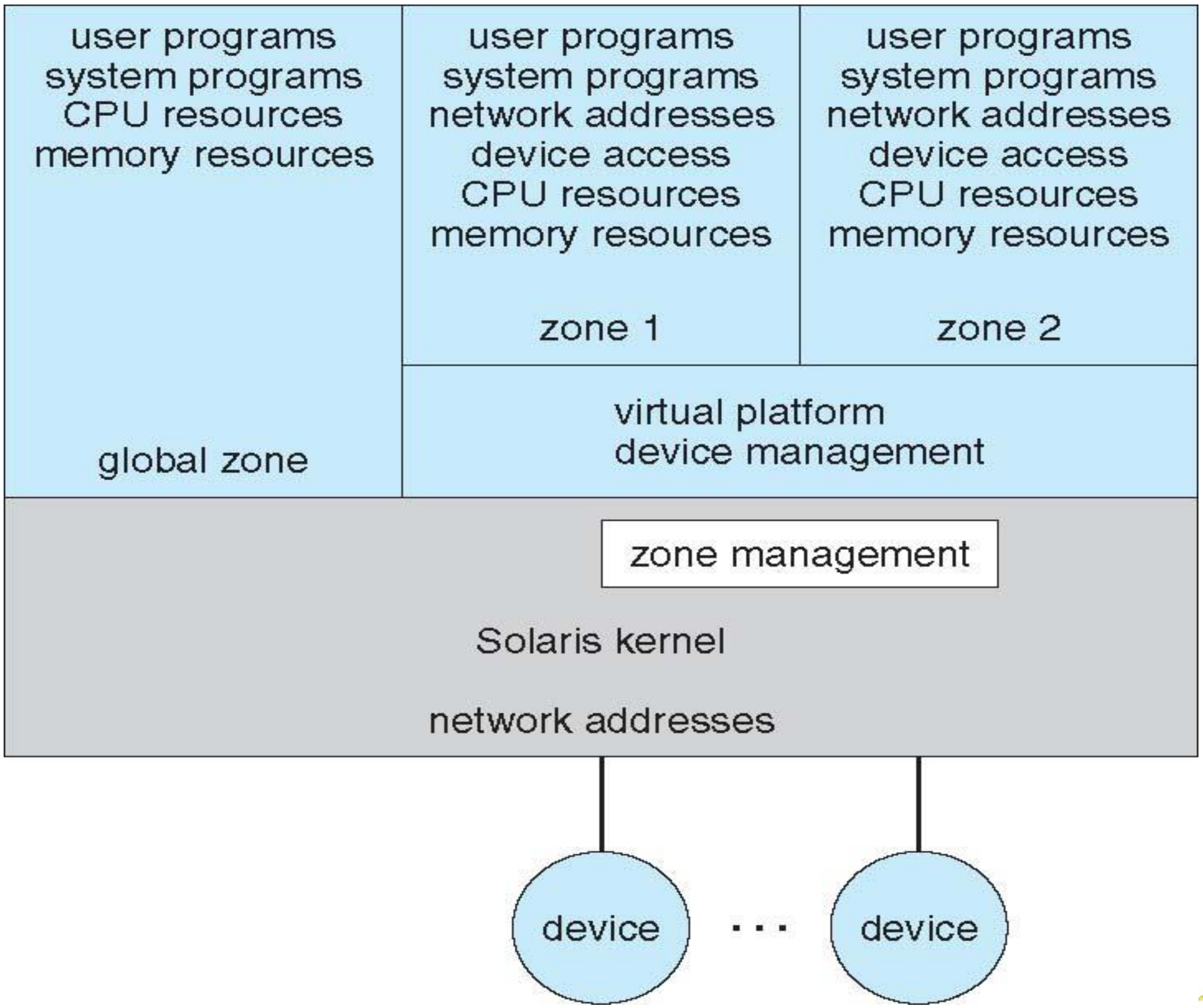


Full virtualization

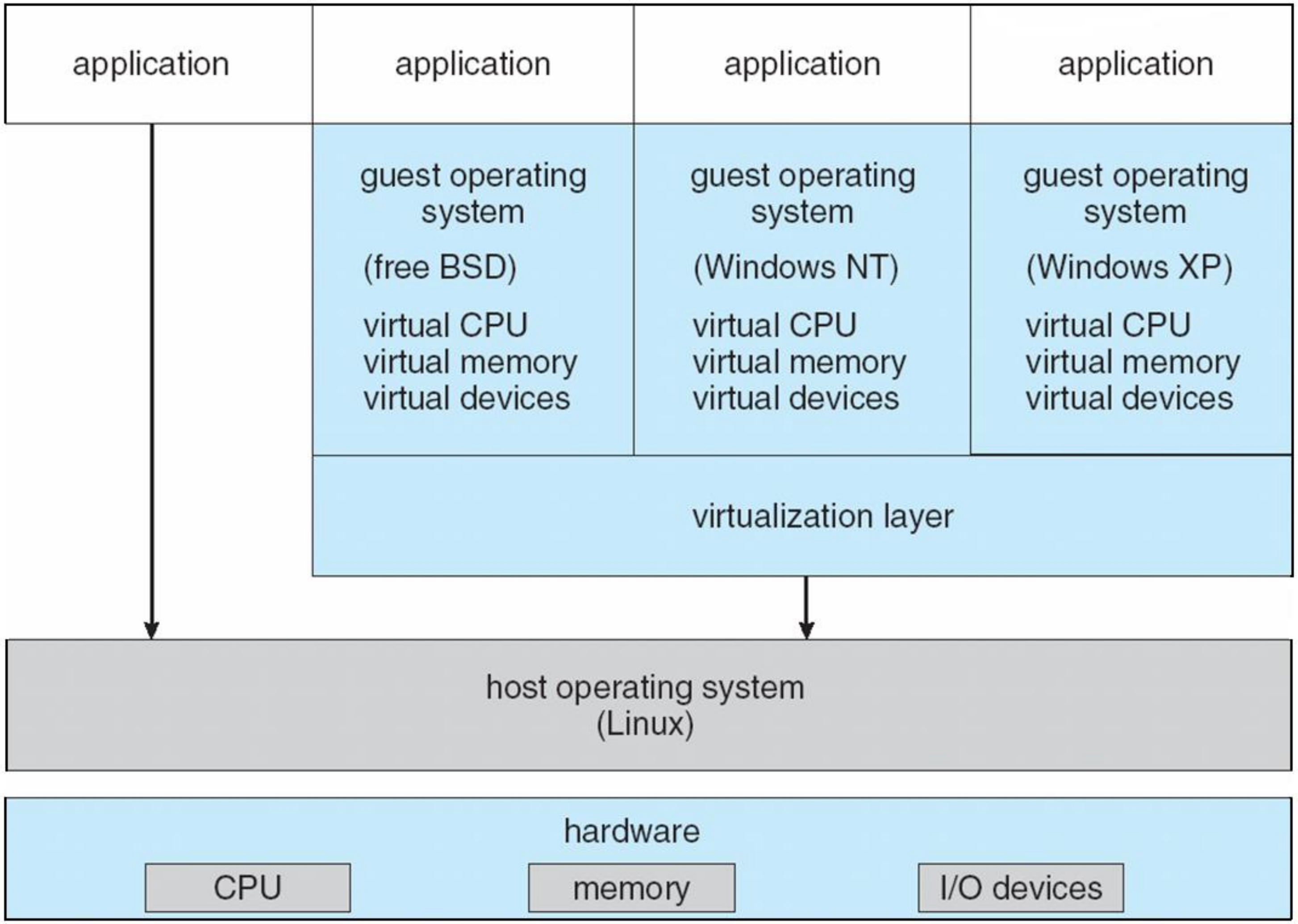


Para virtualization

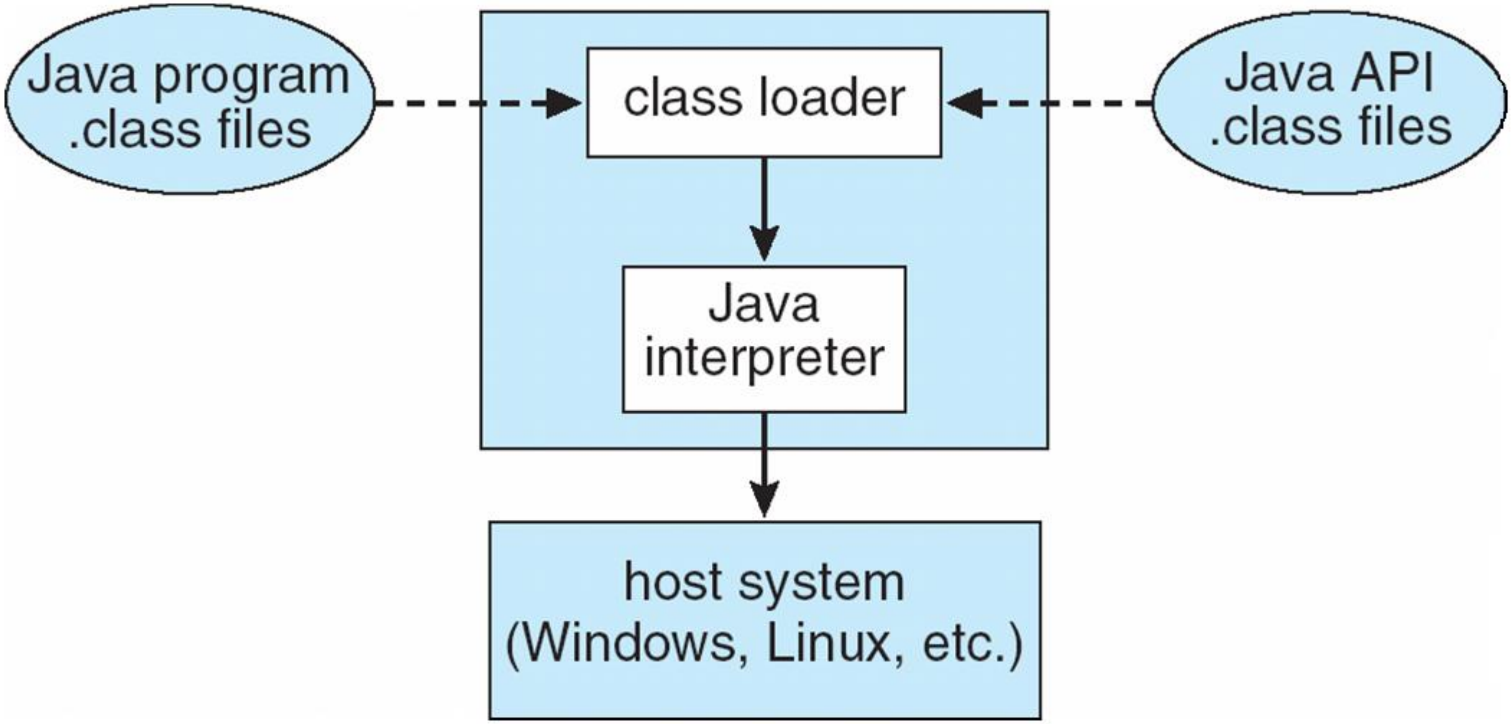
Solaris 10 with Two Containers



VMware Architecture

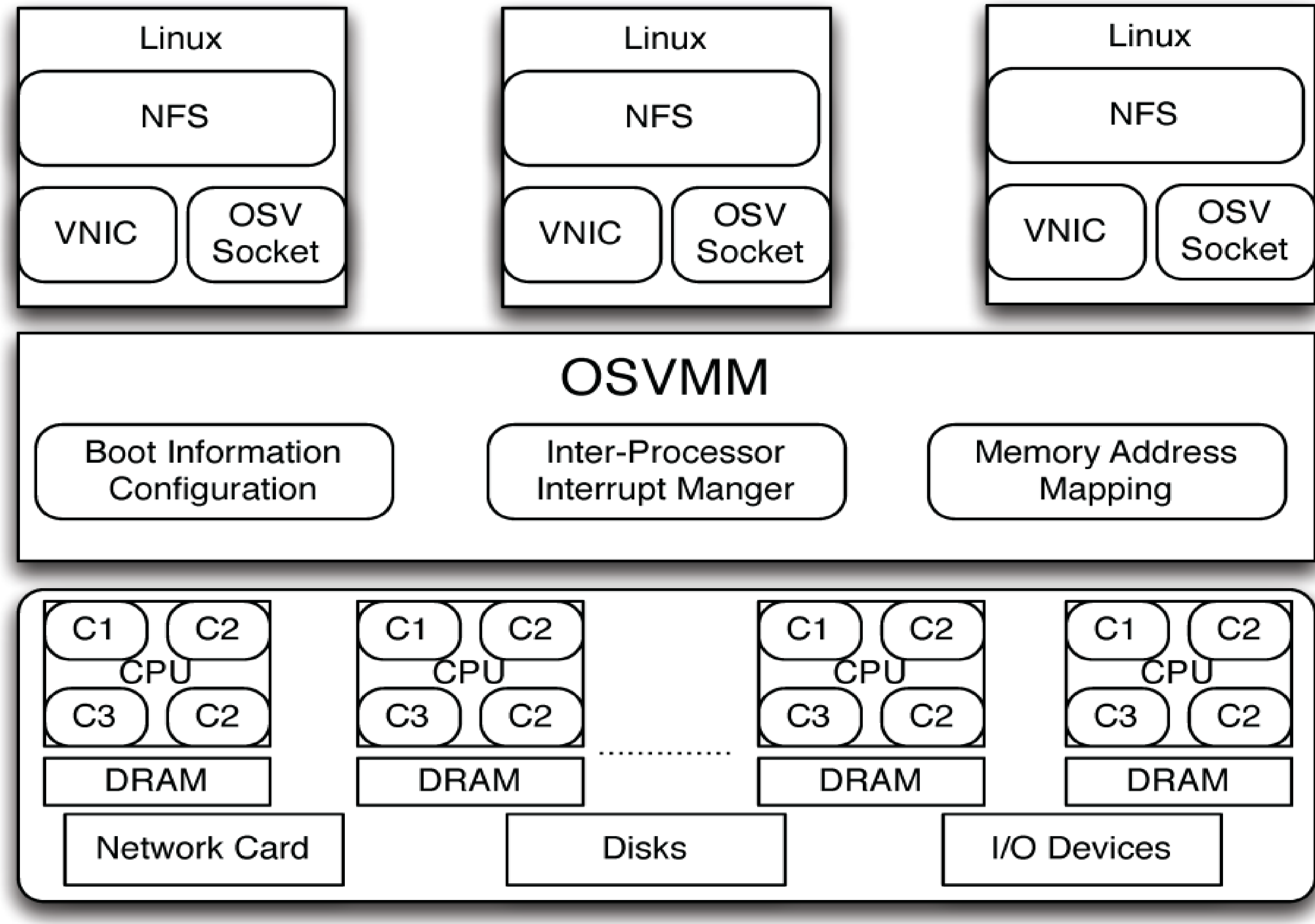


The Java Virtual Machine



A kind of Process Virtual Machine

OSV Architecture(我们团队研发)



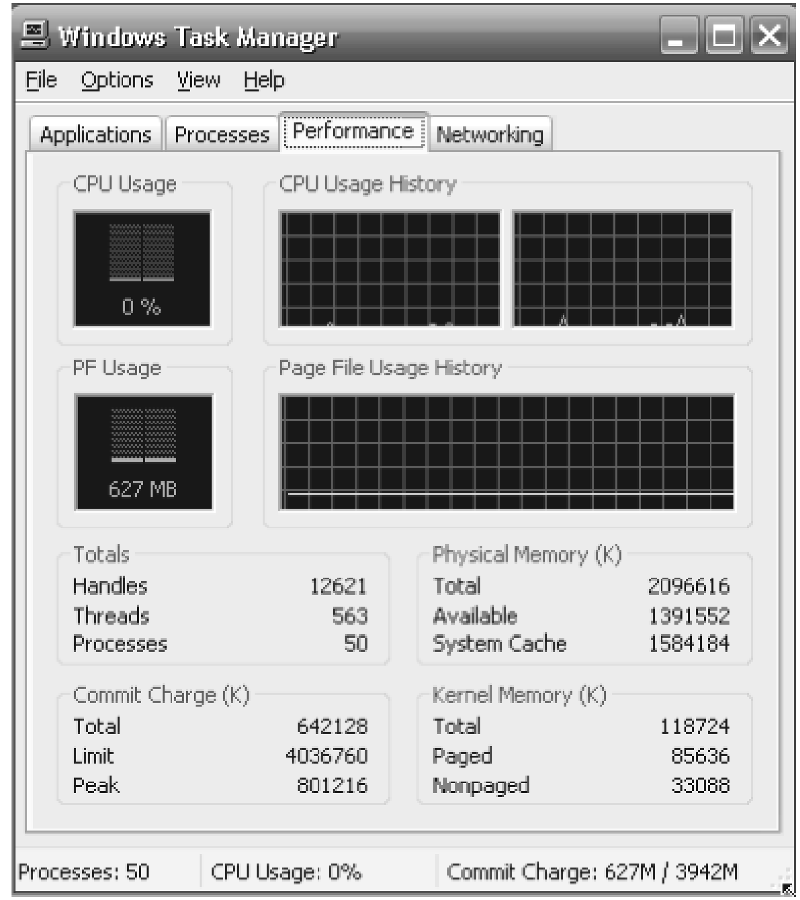
A lightweight VMM. This work is published on ACM VEE 2013

Operating-System Debugging

- Debugging is finding and fixing errors, or bugs
- OSes generate log files containing error information
- Failure of an application can generate core dump(转储) file capturing memory of the process
- Operating system failure can generate crash dump file containing kernel memory
- Beyond crashes, performance tuning can optimize system performance
- DTrace tool in Solaris, FreeBSD, Mac OS X allows live instrumentation on production systems
  - Probes fire when code is executed, capturing state data and sending it to consumers of those probes

Performance Tuning

- Improve performance by removing bottlenecks
- OS must provide means of computing and displaying measures of system behavior
- For example, “top” program or Windows Task Manager



DTrace

- DTrace tool in Solaris, FreeBSD, Mac OS X allows live instrumentation on production systems
- **Probes** fire when code is executed within a **provider**, capturing state data and sending it to **consumers** of those probes
- Example of following XEventsQueued system call move from libc library to kernel and back

```
# ./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
```

Dtrace (Cont.)

DTrace code to record amount of time each process with UserID 101 is in running mode (on CPU) in nanoseconds

```
sched:::on-cpu
uid == 101
{
   self->ts = timestamp;
}

sched:::off-cpu
self->ts
{
   @time[execname] = sum(timestamp - self->ts);
   self->ts = 0;
}
```

```
# dtrace -s sched.d
dtrace: script 'sched.d' matched 6 probes
^C
    gnome-settings-d        142354
    gnome-vfs-daemon        158243
    dsdm                    189804
    wnck-applet             200030
    gnome-panel             277864
    clock-applet            374916
    mapping-daemon          385475
    xscreensaver            514177
    metacity                539281
    Xorg                   2579646
    gnome-terminal         5007269
    mixer_applet2          7388447
    java                  10769137
```

Figure 2.21 Output of the D code.

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
  - What CPU is to be used?
  - How much memory is available?
  - What devices are available?
  - What OS options are desired?
    - Buffer size
    - CPU-scheduling algorithm
    - Maximum number of processes to be supported
    - And so on

System Boot

- *Booting* – starting a computer by loading the kernel
- Finished by *Bootstrap program*
- *Bootstrap program* – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution

System Boot

- *The Bootstrap program* can perform a variety of tasks
  - Run diagnostics to determine the state of the machine
  - Initialize all aspects of the system, from CPU registers to device controllers and the contents of main memory
  - Start the operating system

End of Chapter 2