操作系统原理与设计

第 2 章 OS structures

陈香兰

中国科学技术大学计算机学院

March 7, 2014

提纲

操作系统的组成、服务、特征

- System components
- OS Services
- System Calls & Types
- system programs
- 操作系统的特征

操作系统的抽象模型和体系结构

- 进程模型
- 线程模型
- OS structure
- Virtual Machine
- 系统设计与实现

小结和作业

Outline

System components

System components

- Process Management, 进程管理
- Main Memory Management, 内存管理
- Secondary-Storage Management, 辅存/外存管理
- I/O System Management, I/O 管理
- File Management, 文件管理

- Protection System, 保护
- Networking, 网络
- Command-Interpreter System, 命令解释系统

process(or processor) management

多道环境下，处理器的运行及分配都以**进程（process）**为单位，因此处理器管理可归结为**进程管理（process management）**。

1 process control

- create/destroy a process；suspend(挂起)/resume(恢复) a process
- process state(进程状态) transfering
- 一般由**process control primitives(进程控制原语)**完成

2 process synchronization(同步)

- 为使多个进程有条不紊地运行，应建立 synchronization mechanism(同步机制)。
- including process mutual exclution/synchronization(进程互斥/同步)，次序协调；dead-lock avoidance, prevention, detection and resolution(死锁避免、预防、检测和消除)

3 Process communication

- 源于进程合作，如：输入进程、计算进程、打印进程相互间有信息传递
- 类型：
  - **directly**： P_A 发 *msg*， P_B 收 *msg*
  - **indirectly**： P_A 发 *msg* 到中间实体 (如 *mailbox*)， P_B 从中间实体收 *msg*

4 Job scheduling and process scheduling

- Job scheduling：为作业分配必要资源，调入内存建立进程，并使之进入就绪队列。
- Process scheduling：从就绪队列中选出进程，分配 CPU，使之运行。
- Schedule algorithms：FCFS、优先权等

Main Memory Management(存储管理)

- Main Memory, MM:
  - a large array of words or bytes, each with its own **address**
  - A repository of quickly accessible data shared by CPU and I/O devices
  - A **volatile** storage device
  - Maybe the most architecture-specified component of OS
- Activities
  - Keeping track of memory usage
  - Deciding which processes to load
  - Allocating and deallocating memory space

Main Memory Management(存储管理)

目的：方便用户使用，且提高存贮器利用率

1 Memory allocation

- 静态分配：
- 动态分配：作业在内存中可移动

需内存分配的数据结构及内存分配和回收功能

2 Memory protection

- 例：设置上、下界寄存器，每条指令进行越界检查（一般是硬件实现）

3 Memory mapping, 内存映射

- 地址范围地址
- 逻辑空间逻辑地址（相对地址）
- 物理空间物理地址（绝对地址）

4 Memory expansion, 内存扩充

- 利用虚存技术，从逻辑上扩充内存容量
- 系统应有：请求调入/置换功能以支持虚存技术

File management(文件管理)

- A file
  - a collection of related information defined by its creator. Commonly, programs & data
  - **A logical storage unit**
- Activities
  - **File** creation and deletion
  - **Directory(目录)** creation and deletion
  - Support of primitives for manipulating files and directories
  - **Mapping** files onto secondary storage
  - File **backup(备份)** on stable (nonvolatile) storage media

文件管理的功能

任务：方便用户，提供安全性

1 文件存贮空间的管理

- 例：文件系统根据文件长度自动分配连续或离散的扇区，并提供“一句柄”表示该文件。

2 Directory management(目录管理)

- 使用户按名存取，提高速度。

3 文件的读、写管理和存取控制（即保护）

I/O system management

- I/O subsystem(I/O 子系统)
  - To **hide** the peculiarities of specific hardware devices from the user
- Maybe the most complicate component and has largest **line of code (LOC)**
  - Various device
- Consists of
  - Buffering, caching, and spooling
  - A general device-driver interface
  - Drivers for specific hardware device

I/O system management

任务：提高 I/O 利用率和速度，方便用户

1 缓冲管理

Buffer(缓冲区)：用来解决 CPU — I/O 矛盾，如：CPU 快则应多创建缓冲区。

2 device allocation(设备分配)

包括：设备，设备控制器，I/O 通信的分配和回收

3 设备处理

- 指控制设备进行实际的操作，包括读、写等以及向 CPU 发中断。
- 设备处理/驱动程序应能根据用户的 I/O 请求，自动地构成通道程序。

4 Device Independence (设备独立性) and virtual device (虚拟设备)

- Device independence: 即 program 与设备无关性，使 program 易于重定向，增加了可移植性。
- Virtual device

Secondary storage management(辅存管理)

- Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time.
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- **Activities**
  - Free-space management
  - Storage allocation
  - Disk scheduling
- **Some storage need not be fast**
  - Tertiary storage includes optical storage, magnetic tape
  - Still must be managed
  - Varies between WORM (write-once, read-many-times) and RW (read- write)

User Interface

1 命令接口

- 由一组“命令”集组成，分为**联机**和**脱机用户接口**
  - 联机用户接口由一组键盘操作命令及命令解释程序所组成
  - 脱机（批处理用户接口）用 JCL 写作业说明书

2 程序接口

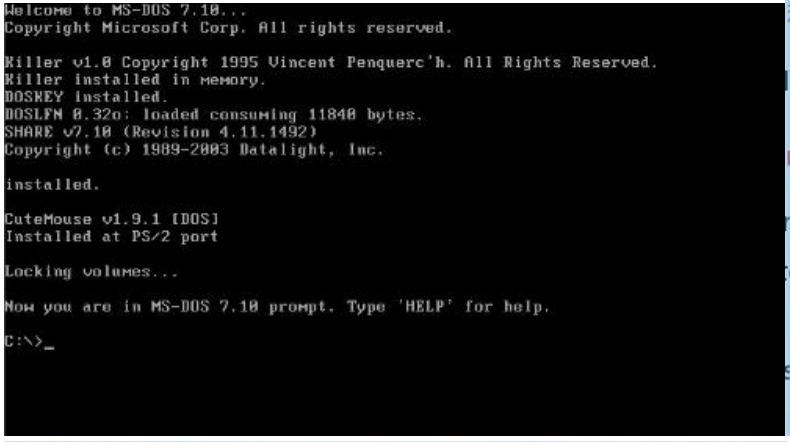
- 系统调用
- 高级语言的库函数，如 C 库

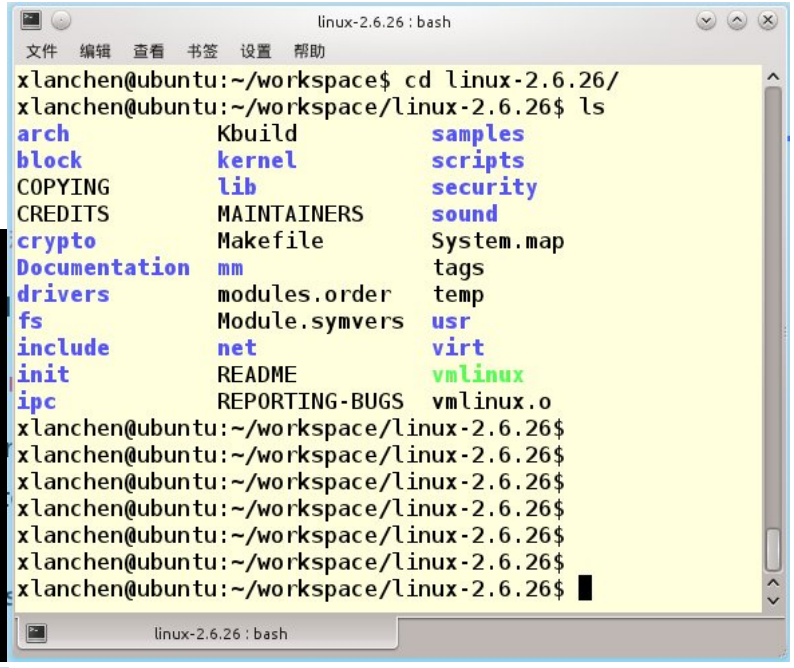
3 图形接口

- 如 win 的 copy 文件，采用“拖”来完成，生动，不需记忆

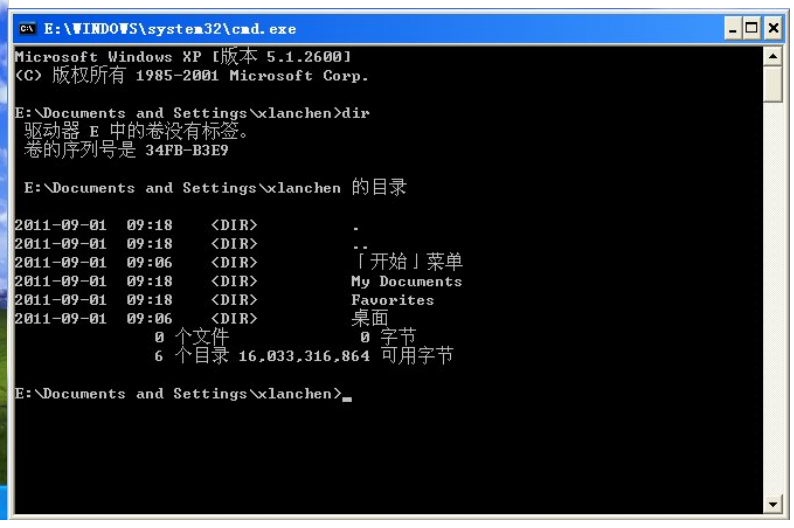
Command-interpreter system(命令解释系统)

- Command-interpreter(命令解释程序):
  - The interface between the user and the OS
  - allows users to directly enter commands
  - In the kernel or as a special program
- Function: To get the next **command statement** and execute it. Examples:
  - **Control-card interpreter(控制卡解释程序)**: batch system
  - **Command-line interpreter(命令行解释程序)**: DOS
  - **Shell**: UNIX
  - **GUI(图形用户界面)**: Macintosh, whindows, ...
- implementation of command
  - **a part of the command interpreter**, **or**
  - **a system program**, such as UNIX.


Welcome to MS-DOS 7.10...
Copyright Microsoft Corp. All rights reserved.
Killer v1.0 Copyright 1995 Vincent Penquerc'h. All Rights Reserved.
Killer installed in memory.
DOSKEY installed.
DOSLFN 0.32o: loaded consuming 11840 bytes.
SHARE v7.10 (Revision 4.11.1492)
Copyright (c) 1989-2003 Datalight, Inc.
installed.
CuteMouse v1.9.1 [DOS]
Installed at PS/2 port
Locking volumes...
Now you are in MS-DOS 7.10 prompt. Type 'HELP' for help.
C:\>_


linux-2.6.26 : bash
文件 编辑 查看 书签 设置 帮助
xlanchen@ubuntu:~/workspace$ cd linux-2.6.26/
xlanchen@ubuntu:~/workspace/linux-2.6.26$ ls
arch Kbuild samples
block kernel scripts
COPYING lib security
CREDITS MAINTAINERS sound
crypto Makefile System.map
Documentation mm tags
drivers modules.order temp
fs Module.symvers usr
include net virt
init README vmlinux
ipc REPORTING-BUGS vmlinux.o
xlanchen@ubuntu:~/workspace/linux-2.6.26$
xlanchen@ubuntu:~/workspace/linux-2.6.26$
xlanchen@ubuntu:~/workspace/linux-2.6.26$
xlanchen@ubuntu:~/workspace/linux-2.6.26$
xlanchen@ubuntu:~/workspace/linux-2.6.26$
xlanchen@ubuntu:~/workspace/linux-2.6.26$
xlanchen@ubuntu:~/workspace/linux-2.6.26$
linux-2.6.26 : bash


回收站
极速酷6
speedupper
temp
speedupper
已经为您的计算机准备好更新
单击此处安装这些更新。
开始
16:52


E:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.
E:\Documents and Settings\xlanchen>dir
驱动器 E 中的卷没有标签。
卷的序列号是 34FB-B3E9
E:\Documents and Settings\xlanchen 的目录
2011-09-01 09:18 <DIR> .
2011-09-01 09:18 <DIR> ..
2011-09-01 09:06 <DIR> 「开始」菜单
2011-09-01 09:18 <DIR> My Documents
2011-09-01 09:18 <DIR> Favorites
2011-09-01 09:06 <DIR> 桌面
0 个文件 0 字节
6 个目录 16,033,316,864 可用字节
E:\Documents and Settings\xlanchen>_

Protection and Security system

- **Protection** – any mechanism for **controlling access** of processes or users to resources defined by the OS
- **Security** – **defense** of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine **who** can do **what**
  - User identities (**user IDs**, security IDs)
    - associated with all files, processes of that user to determine access control
  - Group identifier (**group ID**)
  - **Privilege escalation** allows user to change to effective ID with more rights

Outline

OS Services

操作系统的各种功能，最终要封装成“服务”的形式提供给用户

- 装载并运行程序
- 提供各种 I/O 操作
- 提供文件系统及文件操作
- 提供通信服务
- 提供差错检测服务
- 对资源的分配进行管理
- 对资源的使用进行统计（记账）
- 对系统进行保护

操作系统提供服务的最基本方式——system calls(系统调用)

- 系统调用的类型
  - 进程控制类
  - 文件操作类
  - 设备管理类
  - 通信类，例如消息传送机制
  - 信息维护类，例如日期信息、系统信息等

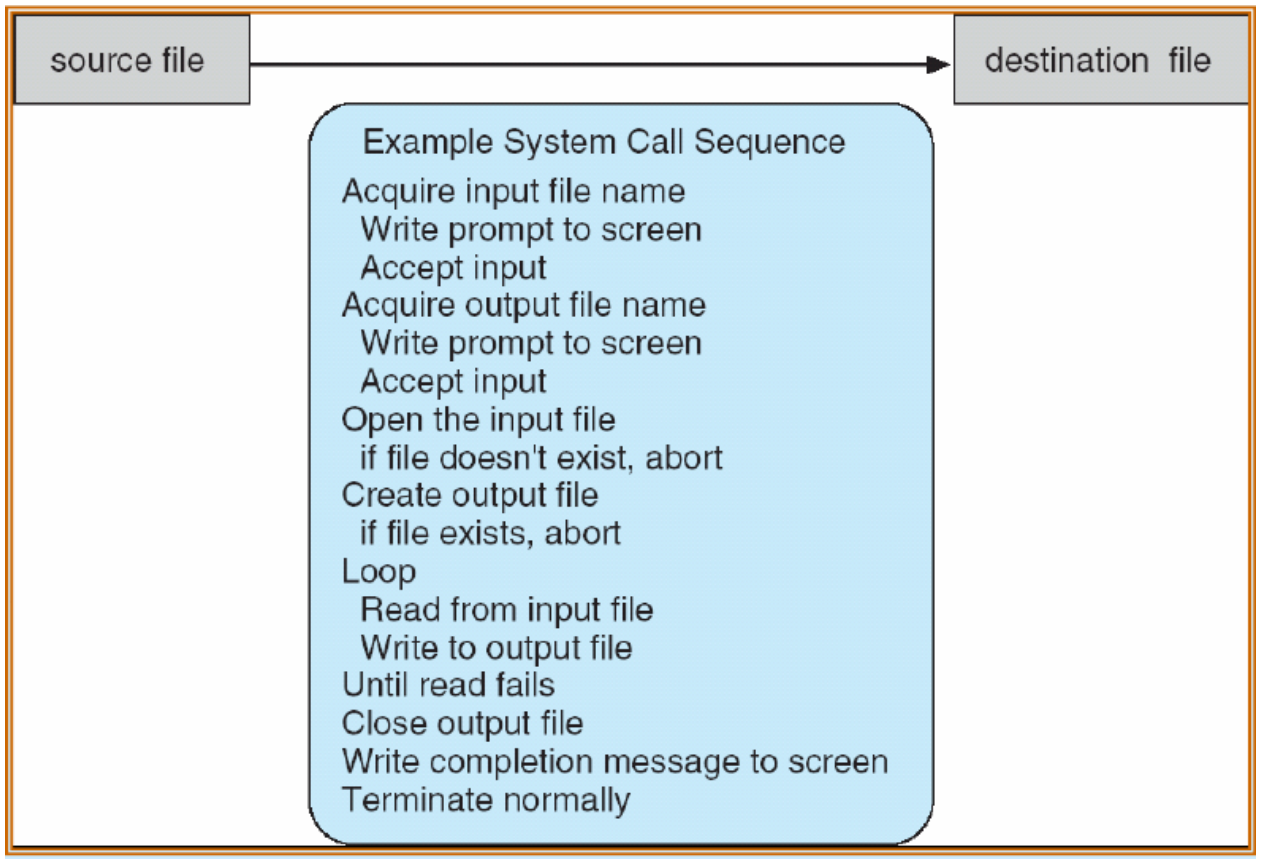
Outline

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 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?

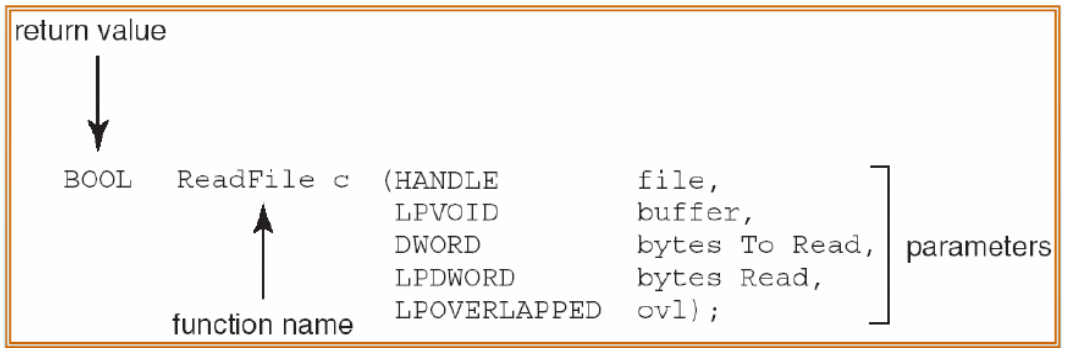
Example of System Calls

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



Example of Standard API

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

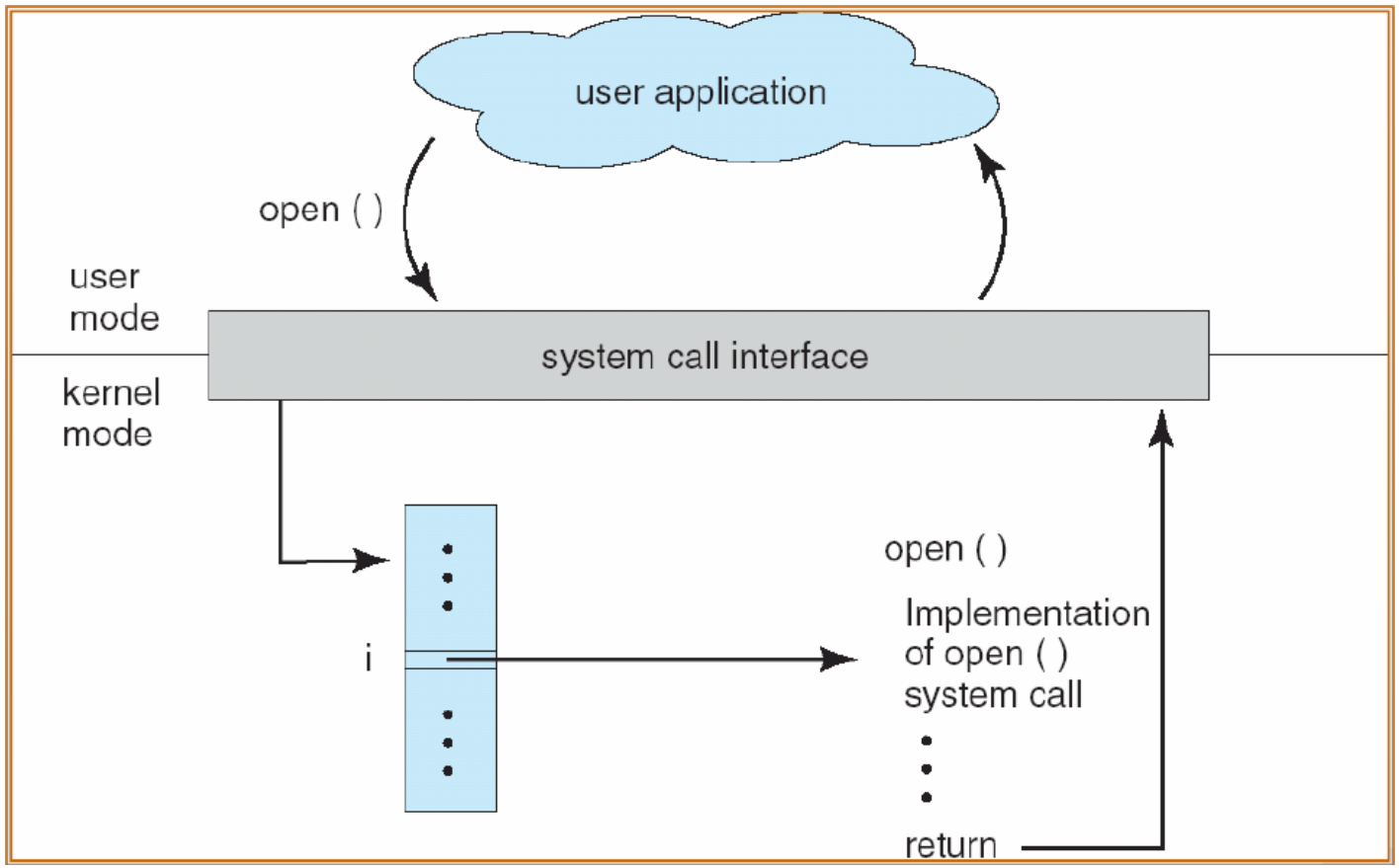


- A description of the parameters passed to ReadFile()
  - **HANDLE** file—the file to be read
  - **LPVOID** buffer—a buffer where the data will be read into and written from
  - **DWORD** bytesToRead—the number of bytes to be read into the buffer
  - **LPDWORD** bytesRead—the number of bytes read during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

System Call Implementation

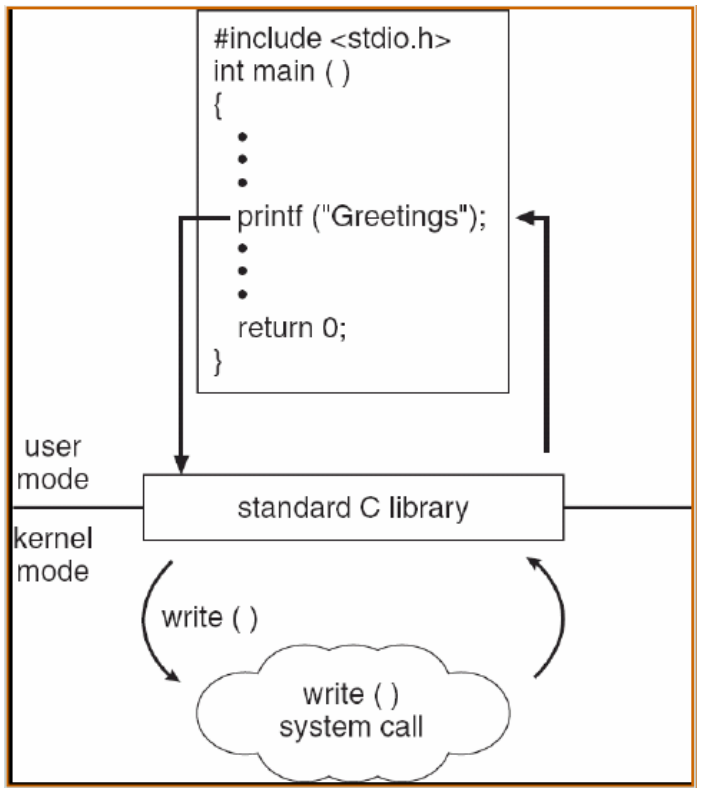
- Typically, **a number associated with each system call**
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know **NOTHING** about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

API – System Call – OS Relationship



Standard C Library Example

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

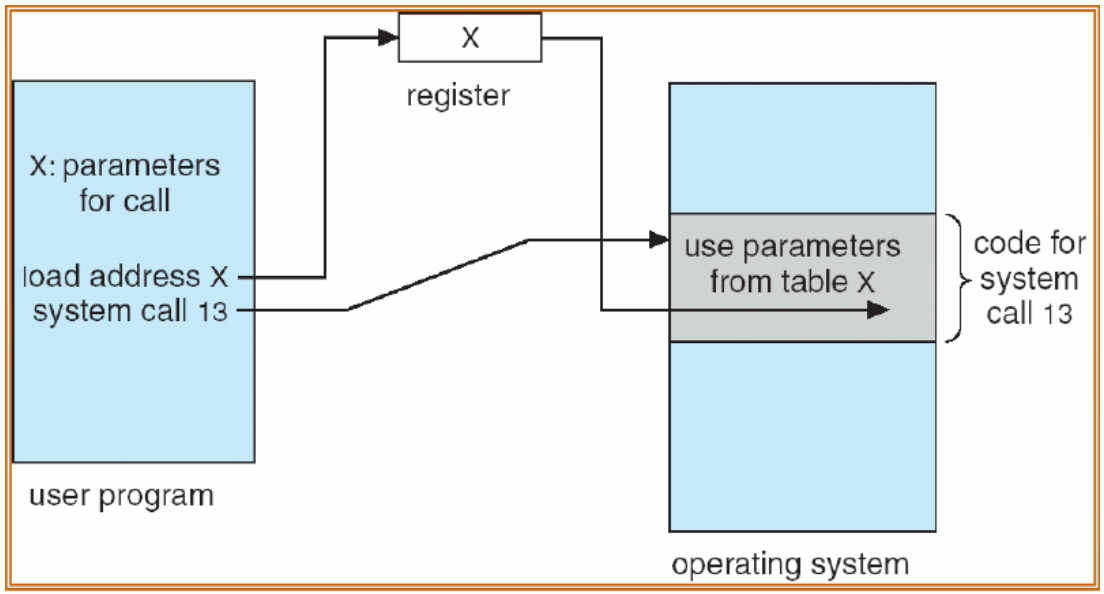


System Call Parameter Passing(参数传递)

- 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
  1. **Simplest**: pass the parameters in **registers**
     - In some cases, **may be more parameters than registers**
  2. Parameters stored in a **block**, or table, in memory, and **address of block passed as a parameter** in a register
     - **Linux** and **Solaris**
  3. 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



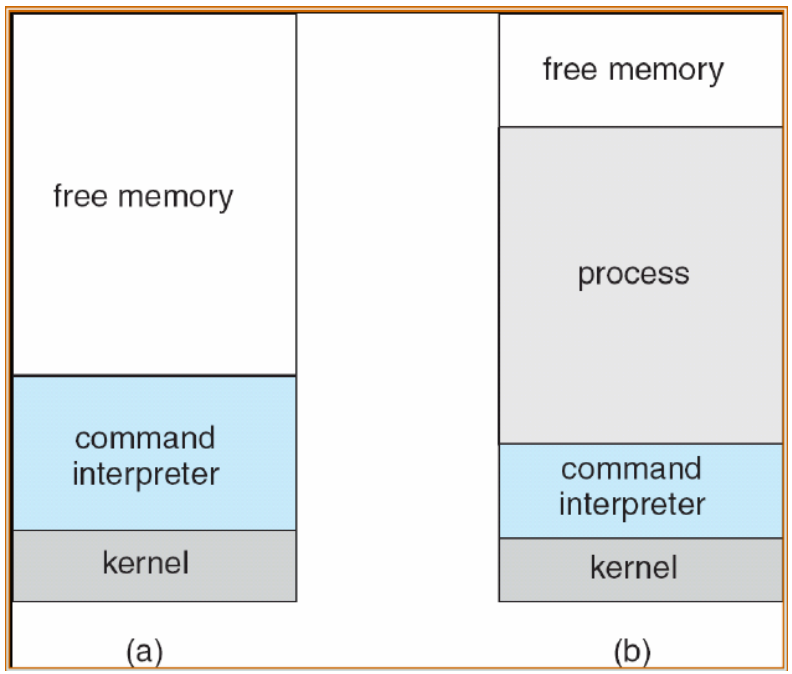
Types of System Calls

1 Process Control

- **Command interface** via the command interpreter, or, **GUI** via a window system
  - **Load** and **execute** a program
  - **Control** a running program: halt its execution either normally (**end**) or abnormally (**abort**)
- **Programming:**
  - **create** process, **terminate** process
  - **get/set process attributes**
  - **wait** for time, wait event, signal event
  - **allocate/free memory**
- **debugging**: trace, ptrace, etc.

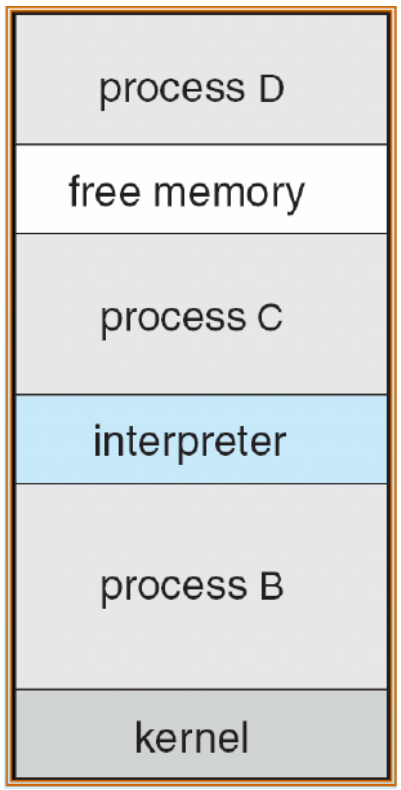
example: process control of MS-DOS

- **MS-DOS: a single-tasking system**
  - start with its command interpreter
  - for a command: does not create a new process
    - load and execute
    - resume



example: process control of FreeBSD

- FreeBSD: a multitasking system
  - a shell is run when a user logs on
  - for a command
    - fork(), exec()
    - wait for the process to finish, or runs the process "in the background"



Types of System Calls

2 file Management

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

3 Device Management

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

4 infomation maintenance

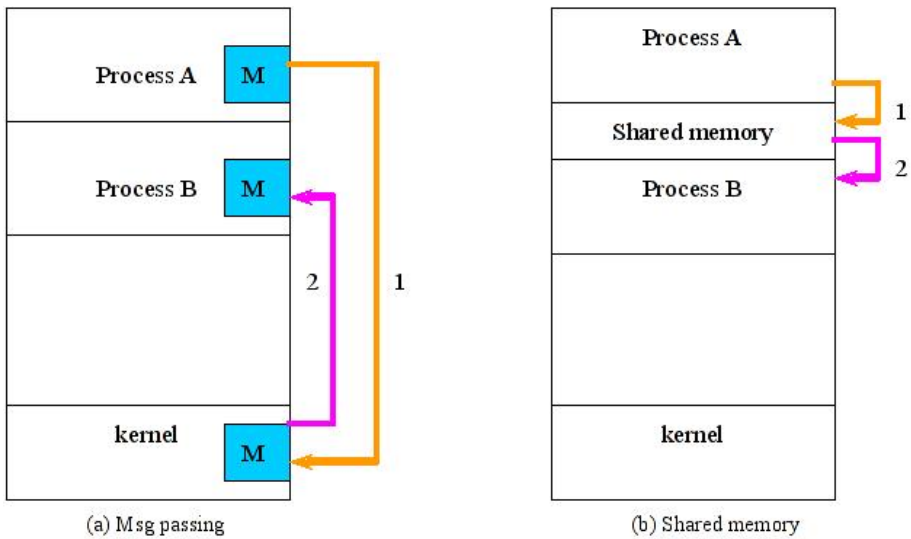
- get/set time/date/system data/process, file, device attributes

Types of System Calls

5 communications

- create/delete communication connection
- send/receive messages
- transfer status information
- attach/detach remote devides

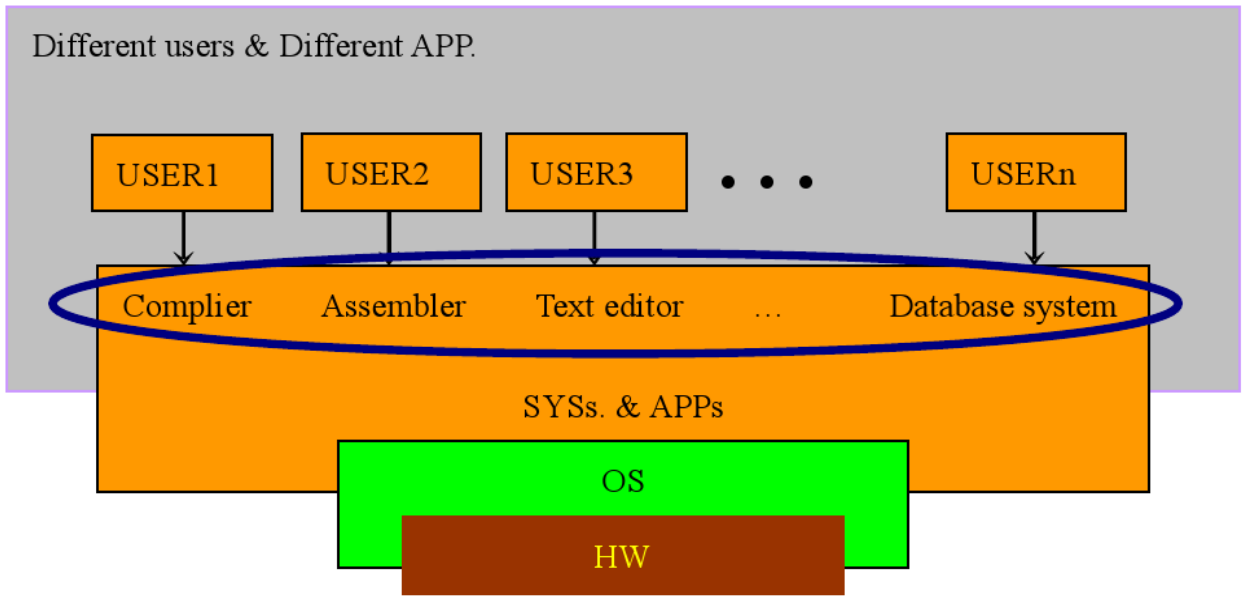
- **Message-passing model**
  - **Message-passing model** and **Shared-memory model**



Outline

system programs

- A collection of system programs
  provide a conveniet environment for program development and execute.
  - file management/modification, status information, programming-language support, program loading and execution, communications



Outline

操作系统的特征

1 并发

- **并行** vs. **并发**
  - 并行是指两或多个事件在同一时刻发生。
  - 并发是两或多个事件在同一时间间隔内发生。
- Program vs. Process(进程)
  - Program：静态实体；
  - Process：动态实体
    - A program in execution.
    - 是系统中能独立运行并作为资源分配的基本单位。
    - 引入线程后，独立运行的单位变为线程。

操作系统的特征

2 共享

- 系统中资源可供内存中多个并发执行的进程共同使用
- 互斥共享 VS. 同时访问
  - 互斥共享：一段时间只允许一个进程访问该资源
  - 同时访问：微观上仍是互斥的临界资源：在一段时间内只允许一个进程访问的资源

并发和共享是操作系统的两个最基本的特征。

操作系统的特征

3 虚拟

- 虚拟：通过某种技术把一个物理实体变为若干个逻辑上的对应物。
- 若 n 是某一物理设备所对应的虚拟的逻辑设备数，则虚拟设备的速度必然是物理设备速度的 1/n。

4 异步

- 运行进度不可预知。

传统操作系统的抽象模型

- 传统的进程模型
- 线程模型

- 服务体/执行流模型

Outline

进程模型

- 进程这一术语，最初是在 20 世纪 60 年代初期，在麻省理工学院（MIT）的 MULTICS 系统和 IBM 公司的 CTSS/360 系统中引入的
- **现代操作系统都以进程（Process）为单位来分配包括处理机、内存、I/O 等在内的各种资源，以实现对计算机系统的并发控制机制。**
- 围绕进程而展开的工作主要包括**进程管理、进程控制、进程同步、进程间通信**以及**进程调度**等

- 进程的**地址空间**
  - 地址空间中的代码、数据和堆栈等内容决定了一个进程所能执行的任务
- **进程描述符及其上下文**
  - 包括程序指针、堆栈指针以及其他硬件寄存器
  - 决定了一个进程的当前执行情况。

- **进程调度**
  - 调度是操作系统实现处理机并发控制的关键。
  - 调度中最能体现进程模型本质的核心功能是进程切换
  - 调度算法仅仅被用来决定在什么时机、切换到哪个进程上。
  - **最常见的调度算法**包括
    - 时间片轮转调度、
    - 基于优先级的可抢占或不可抢占调度
    - 一些实时调度算法
    - 等等

- **进程间通信**机制
  - 用于进程之间交换信息
  - 也是进程之间进行通信的唯一途径。
  - 就模型而言，进程之间实现通信必须有进程调度机制的介入。
  - **常见的进程间通信机制**包括
    - **信号、信号量、管道、消息队列、套接字。**

Outline

线程模型

- 线程模型从进程模型发展而来。
  - 将进程的执行上下文从进程描述符中分离出来，就得到了线程的概念。
- **线程是指令在进程地址空间中的执行轨迹**
  - 在线程模型中，进程可以是单线程的，也可以是多线程的。
  - 传统进程模型中的进程可以看成是单线程的。

- 任何一个线程都属于某个进程。
- 根据是否跨越进程边界，进程/线程在管理、控制、同步、通信和调度上有了两个层次，即
  - 进程内部
  - 和进程之间。
- 通常，现代操作系统在大多数情况下仍然是不区分这两种情况的
- 例外：
  - 进程内部的线程之间可以通过进程地址空间直接共享某些数据，而不必采用传统的进程间通信机制。

Outline

OS structure

- 软件体系结构（Software Architecture）对软件系统的构造具有指导性的作用。
- Bass 在 2003 年关于软件体系结构的定义：
  - “软件体系结构是一种系统结构，该结构包括软件元素、元素的外部可视属性、元素之间的关系”。
- 因此，OS 体系结构包含如下几个方面：
  - OS 的功能模块是如何组成的，即采用什么样的软件元素？
  - OS 的功能模块的哪些信息/属性是相互可见的？
  - OS 的功能模块之间是如何互操作的？

OS structures

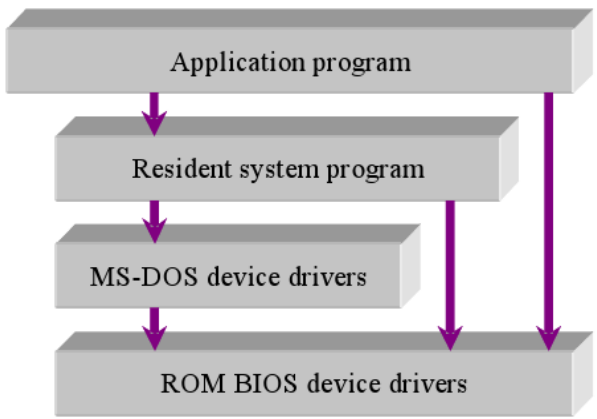
- **OS** as a system software, is **large and complex**
- Software engineering: **to function properly and be modified easily**
  - **partition** into small components (vs. one monolithic system)
  - carefully define the inputs, outputs and function of each module
- How the OS components are interconnected and melded into a kernel
  - **Simple structure(无结构，或者说简单结构)**
  - **Monolithic kernel(单一大内核)**
  - **Modular kernel(模块化结构)**
  - **layered approach(层次结构)**
  - **Microkernel(the first and the second generation)**
  - **Hybrid structure(混合内核)**
  - **Exokernel(外核)**

simple structure(简单结构)

- Initially,
  - Not a well-defined structure
    - Started as small, simple, and limited systems
  - limited by hardware perfromance and software technique at that time
  - no clear system structure(没有清晰的体系结构)
- 操作系统功能模块和用户应用程序混杂在一起，在同一个地址空间上运行，模块之间可以相互任意调用
- Example: MS-DOS, the original UNIX and some early or small embedded system

Example: MS-DOS

- Example: MS-DOS, written to provide the most functionality in the least space
  - Not divided into modules
  - The interfaces & levels of functionality aren't well separated
    - APP can access the basic I/O routines
    - Limited by hardware Intel 8088, no dual mode & hardware protection

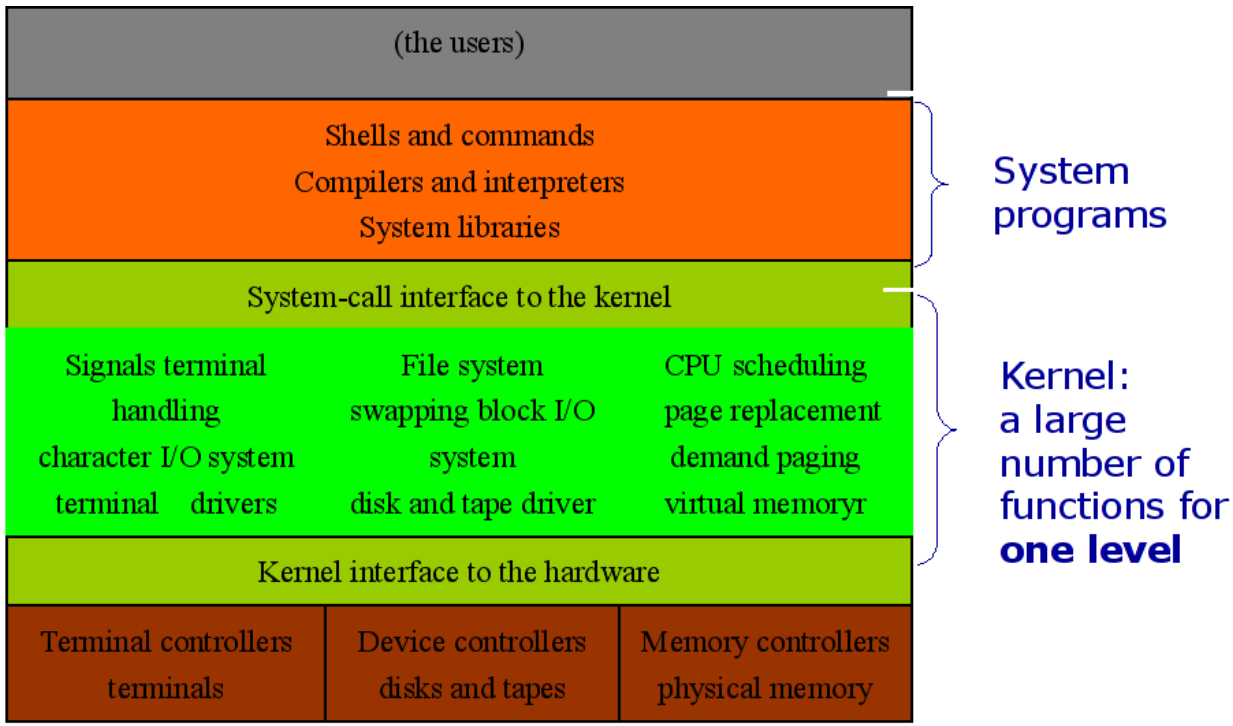


MS-DOS structure

Monolithic structure(单一内核结构)

- 随着硬件平台在性能上、数量上、种类上以及对操作系统的支持上的发展，操作系统的结构有了新的发展
  - **system call(系统调用)**: 使得操作系统与用户应用程序隔离开来;
  - **kernel**: everything below the system call and above the physical HW.
  - 随着操作系统的功能越来越多、模块之间的调用关系也越来越复杂
  - 形成了单一的、体积庞大的操作系统内核
- called **Monolithic structure/ Monolithic kernel(单一大内核)**。
  - 用户应用只能通过中断、异常、系统调用的方式使用操作系统提供的服务
  - 用户应用程序之间通过各种 IPC 进行通信
- **disadvantage**: difficult to implement and maintain.

Example

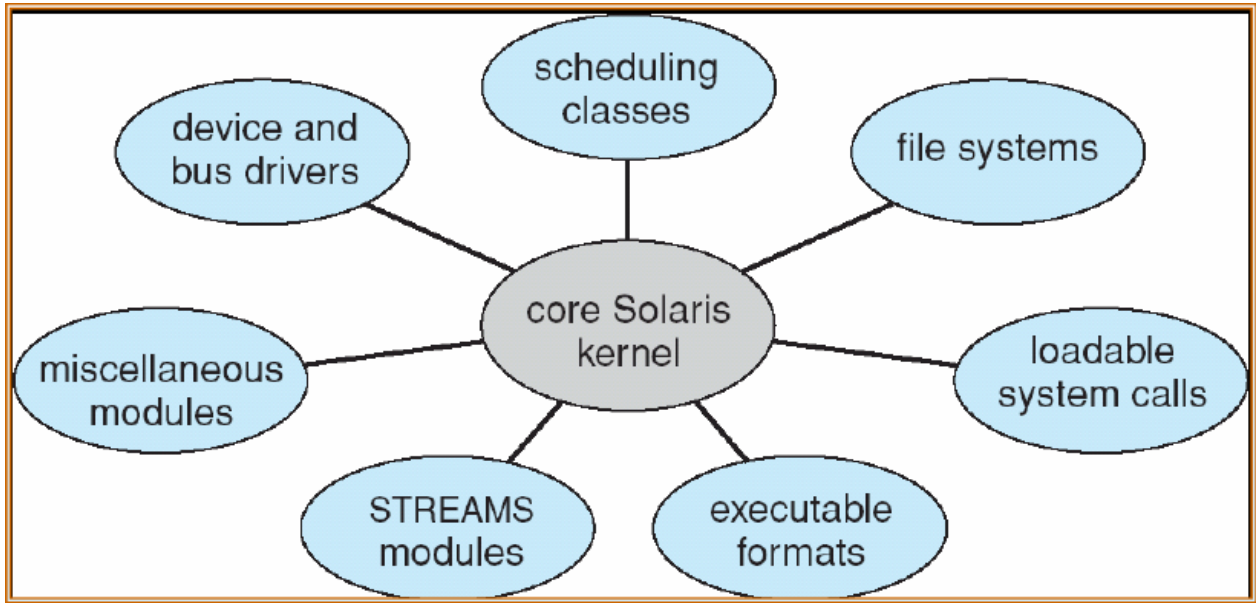

(the users)
Shells and commands
Compilers and interpreters
System libraries
System-call interface to the kernel
Signals terminal handling
character I/O system
terminal drivers
File system
swapping block I/O system
disk and tape driver
CPU scheduling
page replacement
demand paging
virtual memoryr
Kernel interface to the hardware
Terminal controllers
terminals
Device controllers
disks and tapes
Memory controllers
physical memory
System programs
Kernel: a large number of functions for one level

Modular kernel(模块化结构)

- 随着软件开发技术的发展，单一大内核结构的操作系统逐渐采用了**模块化设计方法**
  - 模块之间定义了很好的以函数调用的形式提供的接口
  - 在一定程度上提高了操作系统的可维护性。
- Some techniques
  - **Object-oriented programming**
  - **dynamical link**
  - **dynamical loading**
- Example implementations of UNIX, such as Solaris, Linux, and Mac OS X.

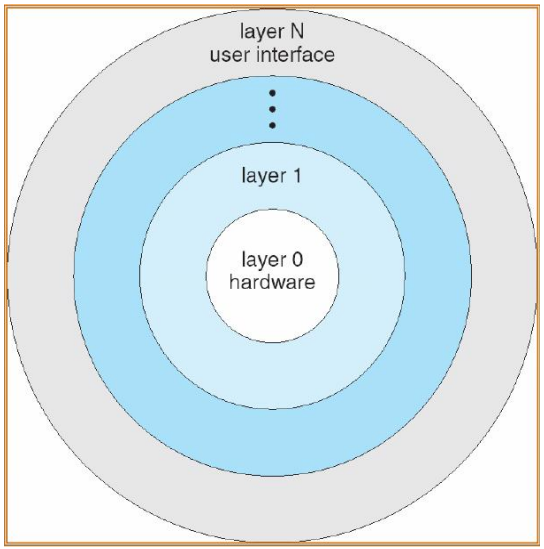
Solaris Modular structure

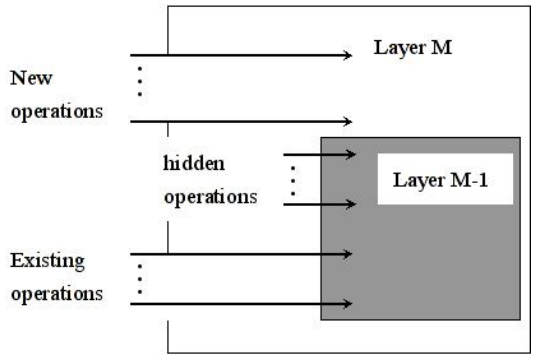
- **the Solaris OS structure = a core kernel + 7 types of loadable kernel modules.**



layered approach(层次结构)

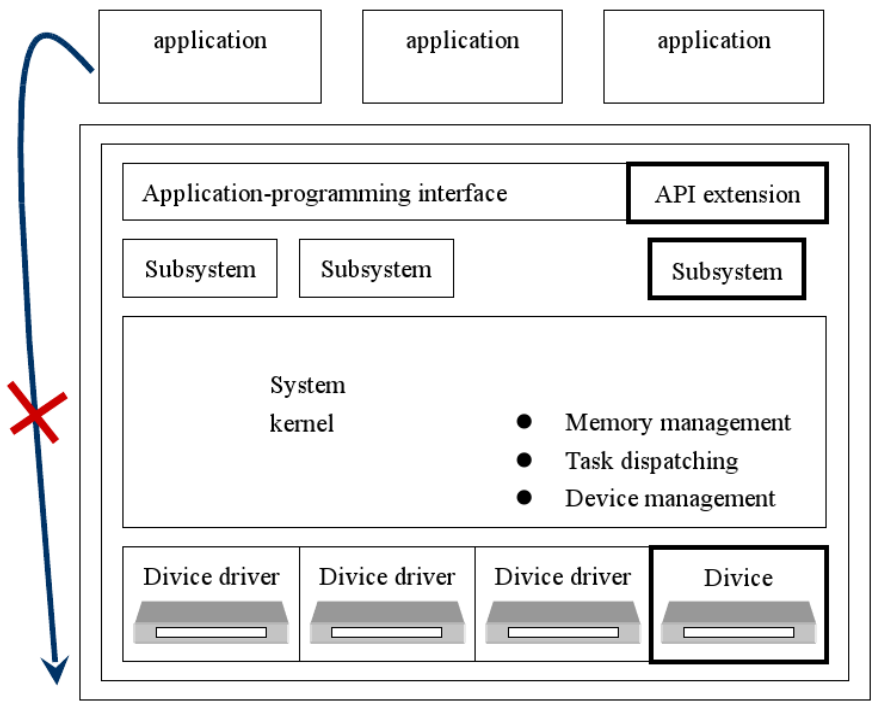
- 为减少 OS 各模块之间紧密依赖和相互调用的关系，特别是消除循环调用现象，实现有序调用
- In the layered approach,
  - the system is broken up into **a number of layers**/levels
    - Bottom layer (layer 0): HW; Highest (layer N): the user interface.
  - each layer can only **use the operations provided by lower-level layers**, and **provides certain operatons** to higher-level layers
  - each layer **hides** the existence of certain data structures, operations and hardware from higher-level layers.





layered approach(层次结构)

- **Advantage** is simplicity of contruction and debugging.
  - debugged from the first layer up to the higher-level layers.
- **Disadvantage:**
  - **Difficulty**: appropriately dedining the various layers.
  - tend to be **less efficient**
    - have caused a small backlash against layering.



Fewer layers, more functionality

OS/2 is a descendant of MS-DOS that adds multitasking & dual-mode operating, ...

layered approach

- 根据每一层内部各模块之间是否存在调用关系，层次结构进一步被划分为
  - 全序的操作系统，如 1968 年 Dijkstra 等开发的 THE 系统
  - 半序的操作系统，如多伦多大学的 SUE 操作系统。

Microkernels(微内核)

- In the **mid-1980s**, **Richard Rashid** developed an OS called **Mach** that modularized the kernel using the **microkernel** approach at **CMU**.
  - all **nonessential components** are removed from the kernel and **implemented as system and user-level programs running in user-space**.
  - a smaller kernel, the function including:
    - provide **minimal process and memory management**
    - mainly, provide **a communication facility** between client and server program: **message passing(消息传递)**
- Benefit：
  - ease of extending the OS.(可扩展性)
  - easy to port from one hardware design to another.(可移植性)
  - more security and reliability.(安全性和可靠性)
- Example:
  - Mach, Tru64 UNIX via Mach kernel, QNX.
  - Windows NT?→Windows XP
- Disadvantage:
  - **performance decrease** due to increased system function overhead.
- the first generation of microkernels (第一代微内核)

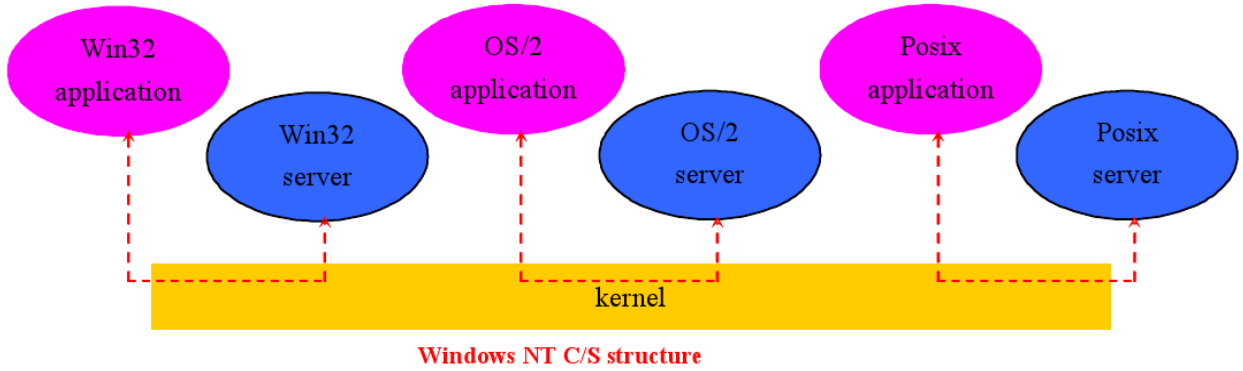
the second generation of microkernels(第二代微内核)

- One approach to solve the performance problem of microkernel
  - IPC↑
  - called the second generation of microkernel
  - typical example: L4

hybrid structure(混合内核)

- another approach to solve the performance problem of micorkernel: hybrid structure
  - 即扩大微内核并把一些关键的服务程序和驱动程序重新加入到内核中去，
  - 如 Windows NT 4.0 把图形系统重新加入内核以提高性能。
- 但这种方法削弱了微内核思想在系统的扩充性、灵活性和可靠性等方面所带来的优点

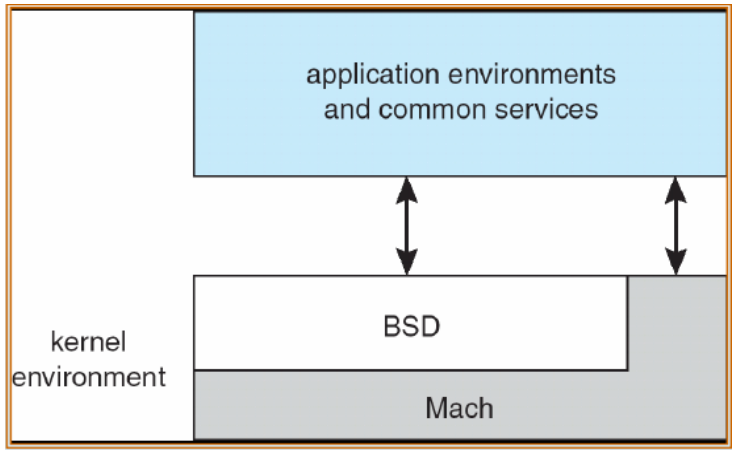
Example



Windows NT C/S structure

- The first release of Windows NT has a layered microkernel organization.
  - Low performance <Windes 95
- Windows NT 4.0: some layers moved from user space to kernel space and integrating them more closely.
- Windows XP: more monolithic than microkernel.

the Mac OS X structure



- **layered** technique with one layer consists of the **Mach** microkernel
- **Mach** provides memory management, support for remote precedure calls (RPCs, 远程过程调用) and IPC facilities include message passing, and thread scheduling.
- **BSD kernel** provides a BSD command line interface, support for networking and file systems, and an inplementation of POSIX APIs, include Pthreads.
- Additionally, an I/O kit for development of device drivers and dynamically loadable modules, which referred as **kernel extensions**

参考：Exokernel(外核)

- Exokernel 提出了外核的概念，它试图将操作系统接口降低到硬件层
  - 在外核结构中，内核只用来负责简单的申请、释放并复用硬件资源，而将内存映射、I/O 和复杂的线程包等所有在传统操作系统内核中提供的抽象都转移到用户空间，以库的形式提供，用户程序通过调用库的形式实现对硬件资源的直接访问。
- 外核结构可以看成是微内核结构的一种极端形式

操作系统体系结构小结

从上述 OS 体系结构的发展过程中可以看出：

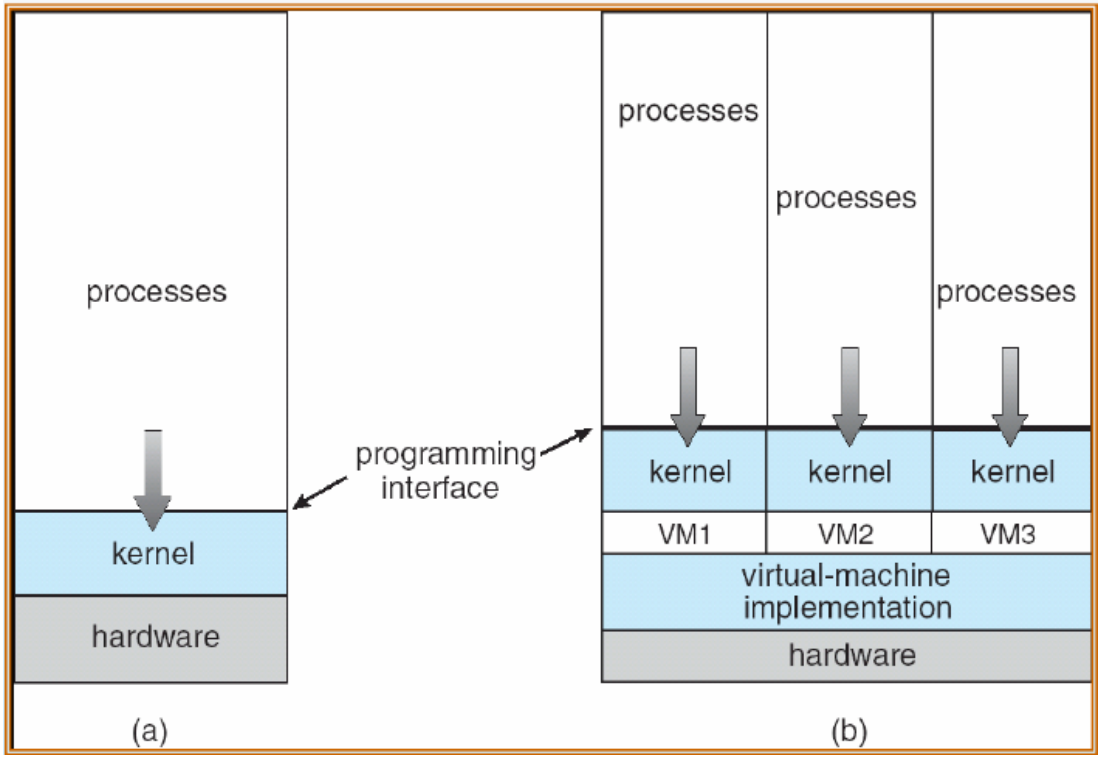
- 除了简单结构以外，操作系统中都存在一个内核
  - 核外的程序通过中断、异常、系统调用的方式访问内核中提供的服务；
  - 内核内部模块之间采用函数调用的形式直接调用（不论接口定义是否良好）；
- 操作系统提供给核外的程序的抽象是进程/线程
  - 包括用户应用和外放到核外运行的操作系统功能模块
  - 进程/线程之间通过进程间通信机制进行通信；

Outline

Virtual Machine(虚拟机)

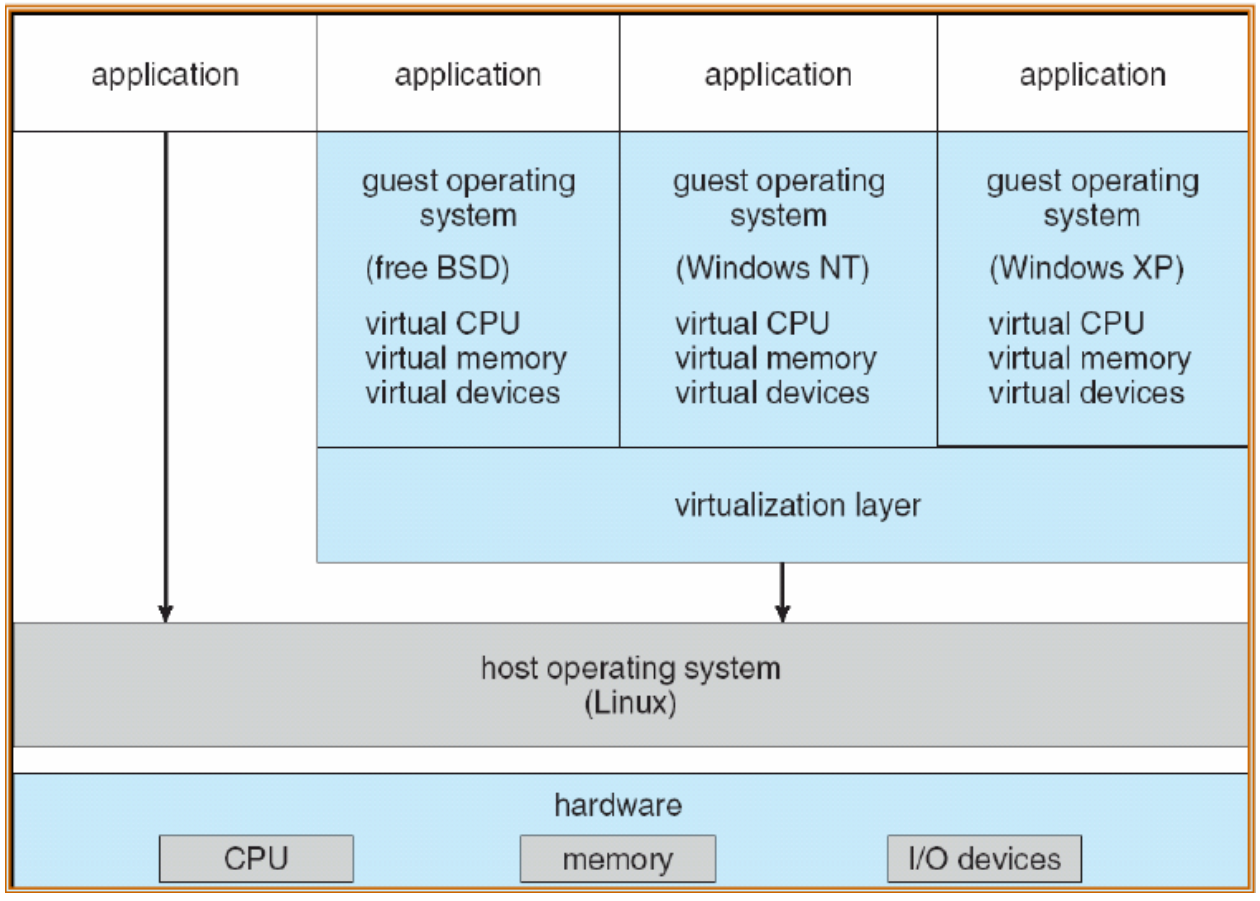
- A **virtual machine** takes the layered approach to its logical conclusion.
  - It treats hardware and the OS kernel as though they were all hardware
  - A virtual machine provides an interface identical to the underlying bare hardware
- The **OS** creates the **illusion** of multiple processes, **each executing on its own processor with its own (virtual) memory**
- 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

system model

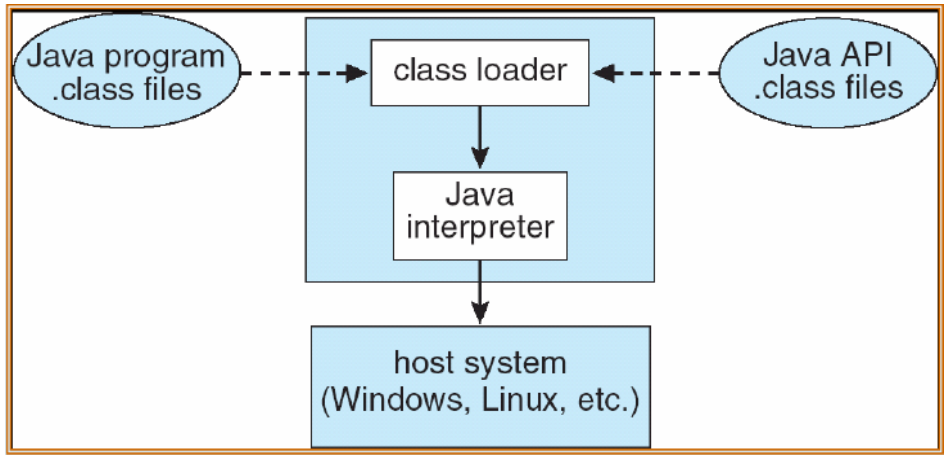


- Advantage:
  - **pretection** via isolation
  - **OS developed** online over virtual machine

Example: VMWare Virtual Machine



Example: Java Virtual Machines



Outline

OS design and implementation

- The first problem in designing a OS is to define goals(目标) and specifications(规格).
  - requirement(需求)
- One important principle: the separation of **policy** from **mechanism**(机制和策略相分离)
  - **mechanisms(机制)** determine **how** to do something(如何来做);
  - **policies(策略)** determine **what** will be done(做什么).
  - Example: timer(定时器)、priority(优先级)；microkernel VS. Apple Macintosh
- implementation
  - Traditionally: assembly language(汇编语言)→Now: higher-level languages such as C/C++
    - advantage: write the OS faster, more cmpact(紧凑), easier o understand and debug, easier to port(移植), ...
    - possible disadvantages: speed↓and storage requirements↑ . No longer a major issue.

小结

操作系统的组成、服务、特征

- System components
- OS Services
- System Calls & Types
- system programs
- 操作系统的特征

操作系统的抽象模型和体系结构

- 进程模型
- 线程模型
- OS structure
- Virtual Machine
- 系统设计与实现

小结和作业

作业

- 操作系统最基本的两个特征是什么？
- 什么是操作系统的体系结构，有哪些？
- 操作系统的组成有哪些？
- Describe three general methods for passing parameters to the operating system.
- What are the two models of interprocess communication? What are the strengths and weakness of the two approaches?
- Why is the separation of mechanism and policy desirable?
  为什么机制与策略分离是一个令人满意的原则？

上机作业

- 在 linux 中，编写一个程序
  - 父进程创建一个 pipe 后创建一个子进程
  - 父进程输出当前的时间和日期
  - 父进程输出自己的进程号和子进程的进程号
  - 父进程将自己的进程号通过 pipe 告诉子进程
  - 子进程输出自己的进程号和父进程的进程号
  - 最后，让子进程执行另外一个程序。
  - 提示：getpid，pipe，write，read，close，fork，execve
  - 使用 strace 跟踪父子进程的运行情况，了解程序是如何在系统中运行起来的。
  - 对可执行程序进行反汇编，了解用户程序是如何请求操作系统的系统调用的。
  - 提供上机报告，说明：
    - 编写程序的过程中参考的材料或者网页
    - 编写程序的过程中出现的错误，解决问题的方法
    - 附上源代码和运行结果的截图

谢谢！