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

2009年09月01日

提纲

1. 操作系统的组成、服务、特征
   - System components
   - OS Services
   - System Call
   - system programs
   - 操作系统的特征
2. 操作系统的抽象模型和体系结构
   - 进程模型
   - 线程模型
   - 操作系统的体系结构
   - 虚拟机
3. 小结和作业

Outline

1. 操作系统的组成、服务、特征
   - System components
   - OS Services
   - System Call
   - system programs
   - 操作系统的特征
2. 操作系统的抽象模型和体系结构
   - 进程模型
   - 线程模型
   - 操作系统的体系结构
   - 虚拟机
3. 小结和作业

System components

- Process Management
- Main Memory Management
- Secondary-Storage Management
- I/O System Management
- File Management

- Protection System
- Networking
- Command-Interpreter System

处理器管理

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）**。

一、**进程控制**

- 创建/撤消进程；挂起 / 恢复进程
- 迁移进程的状态
- 一般由**进程控制原语**完成

二、**进程同步**

- 为使多个进程有条不紊地运行，应建立同步机制。
- 包括进程互斥/同步，次序协调；死锁避免、预防、检测和消除

三、进程通信

- 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：
  - **直接：** P_A发msg，P_B收msg
  - **间接：** P_A发msg到中间实体(如$mailbox$)，P_B从中间实体收msg

四、作业与进程调度

- 作业调度：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。
- 进程调度：从就绪队列中选出进程，分配CPU，使之运行。
- 调度算法： FCFS、优先权等

存储管理(Main Memory Management)

- MM is 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

目的：方便用户使用，且提高存贮器利用率

一、内存分配

- 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

二、内存保护

- 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）

三、地址映射

- 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）

四、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

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

文件管理的功能

任务：方便用户，提供安全性

一、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。

二、目录管理

- 使用户按名存取，提高速度。

三、文件的读、写管理和存取控制（即保护）

I/O management

- I/O subsystem
  - 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

设备管理功能

任务：提高I/O利用率和速度，方便用户

一、缓冲管理

缓冲区：用来解决CPU—I/O矛盾，如：CPU快则应多创建缓冲区。

二、设备分配

包括：设备，设备控制器，I/O通信的分配和回收

三、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。

四、设备独立性和虚拟设备

- 独立性，即program与设备无关性，使program易于重定向，增加了可移植性。
- 虚拟设备

Second storage management

- 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)

Command-interpreter system

- The interface between the user and the OS In the kernel or as a special program
- To get the next command statement and execute it
- Example:
  - Control-card interpreter
  - Command-line interpreter (DOS)
  - Shell (UNIX)
  - GUI

用户接口

一、**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

二、**程序接口**

- 系统调用
- 高级语言的库函数，如C库

三、**图形接口**

- 如 win的copy文件，采用“拖”来完成，生动，不需记忆

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

1. 操作系统的组成、服务、特征
   - System components
   - OS Services
   - System Call
   - system programs
   - 操作系统的特征
2. 操作系统的抽象模型和体系结构
   - 进程模型
   - 线程模型
   - 操作系统的体系结构
   - 虚拟机
3. 小结和作业

操作系统提供的服务

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——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

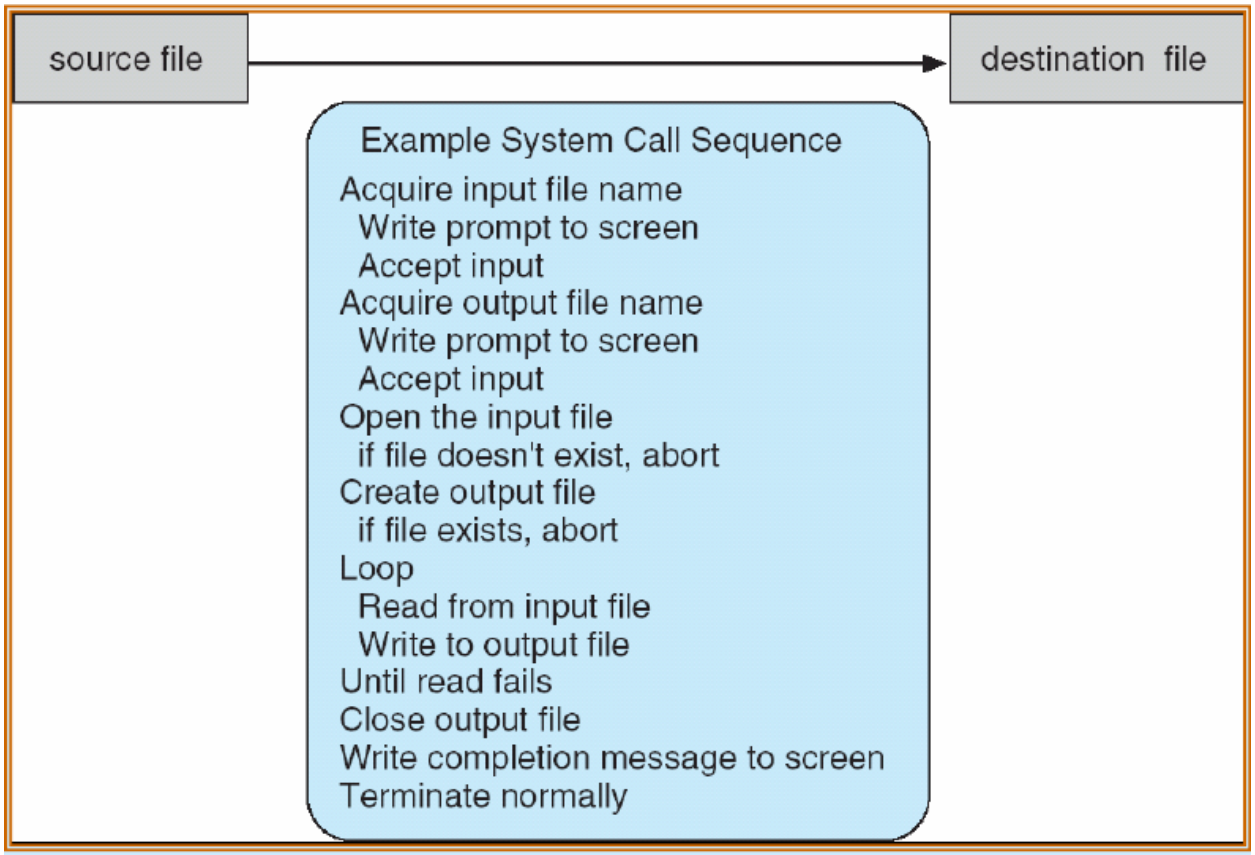
1. 操作系统的组成、服务、特征
   - System components
   - OS Services
   - System Call
   - system programs
   - 操作系统的特征
2. 操作系统的抽象模型和体系结构
   - 进程模型
   - 线程模型
   - 操作系统的体系结构
   - 虚拟机
3. 小结和作业

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?

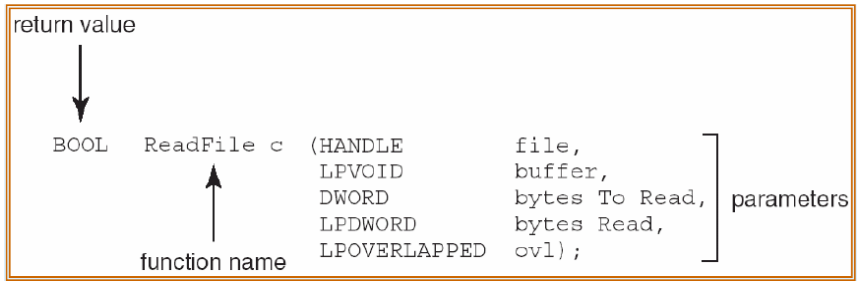
Example of System Calls

- 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

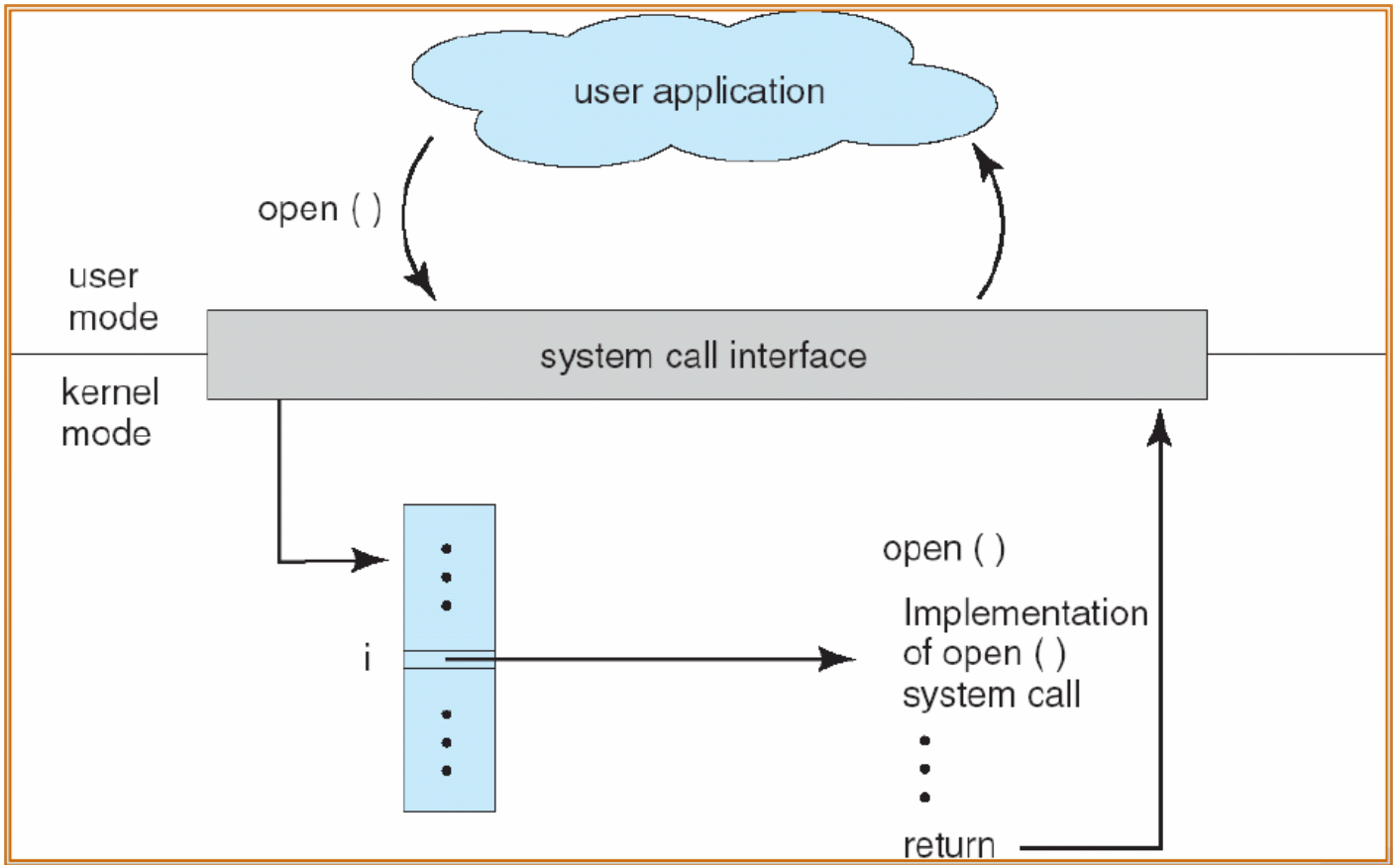


- A description of the parameters passed to ReadFile()
  - **HANDLE** file—the file to be read
  - **LPVOID** buffer—a buffer where the data will be read into and written from
  - **DWORD** bytesToRead—the number of bytes to be read into the buffer
  - **LPDWORD** bytesRead—the number of bytes read during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

System Call Implementation

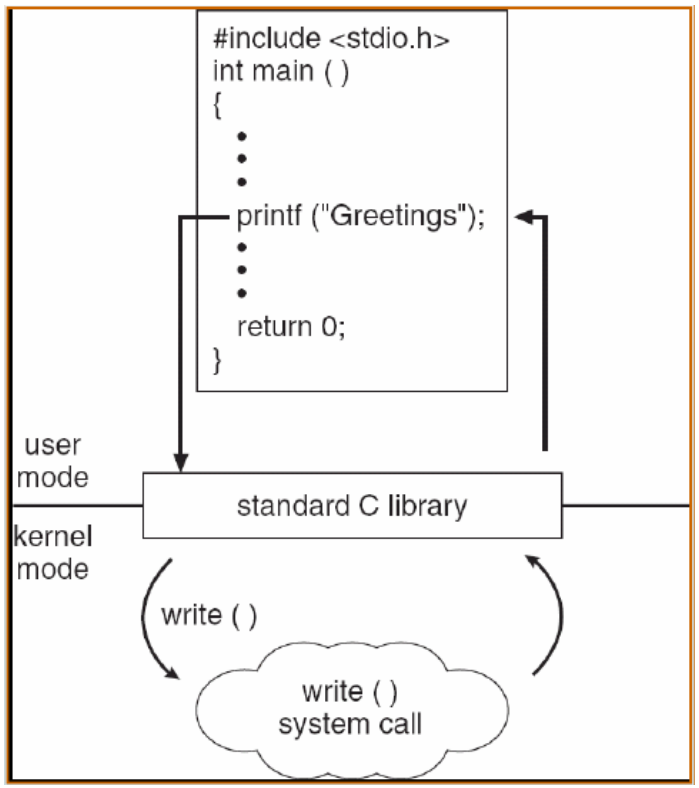
- Typically, **a number associated with each system call**
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know **NOTHING** 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

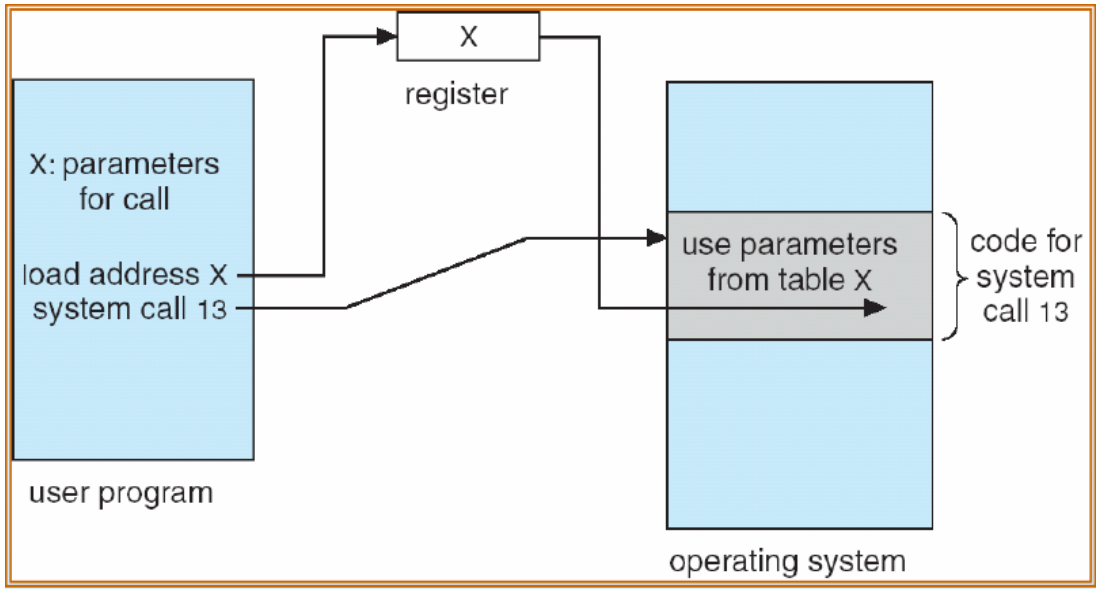


System Call Parameter Passing

- 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



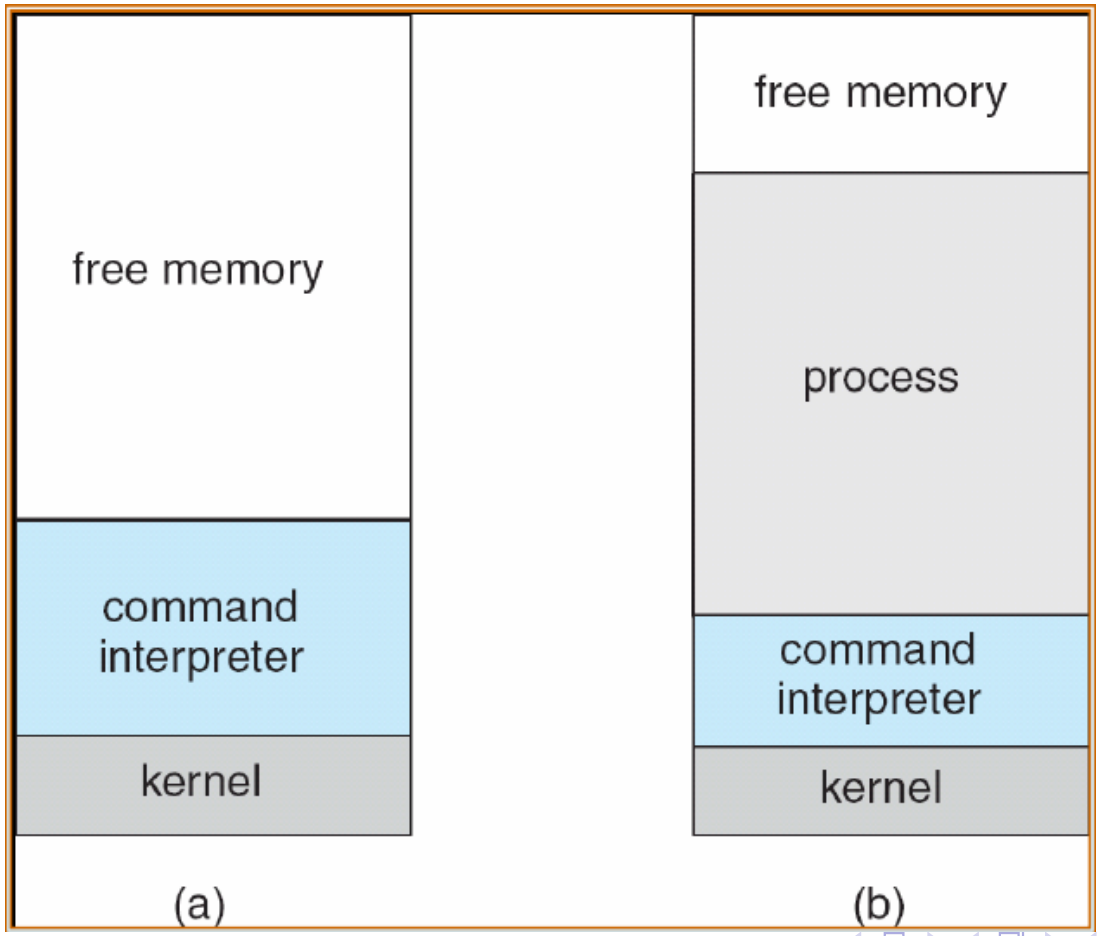
一、进程控制类系统调用

一、进程控制类系统调用

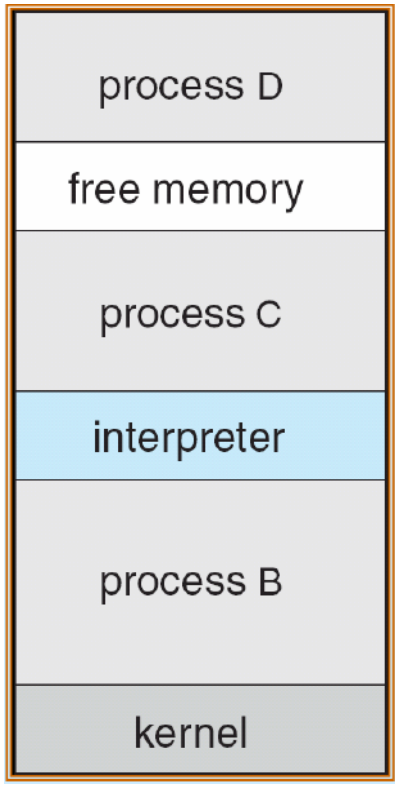
- 命令接口或图形接口
  - 装入并执行一个程序
  - 控制程序的运行：正常或非正常的终止
- 程序接口：
  - 修改进程的属性
  - 终止一个进程的执行
  - 等待一个进程的执行
- 调试类接口，如trace、ptrace等
- 等

example: process control

- MS-DOS （单任务系统）



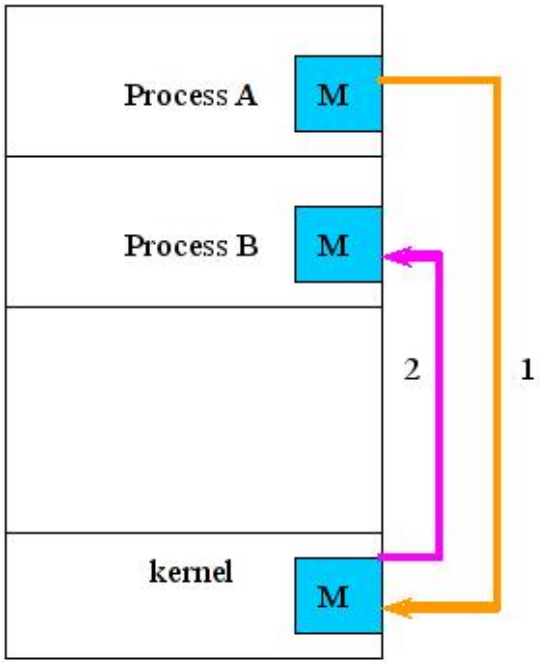
FreeBSD （多任务）


process D
free memory
process C
interpreter
process B
kernel

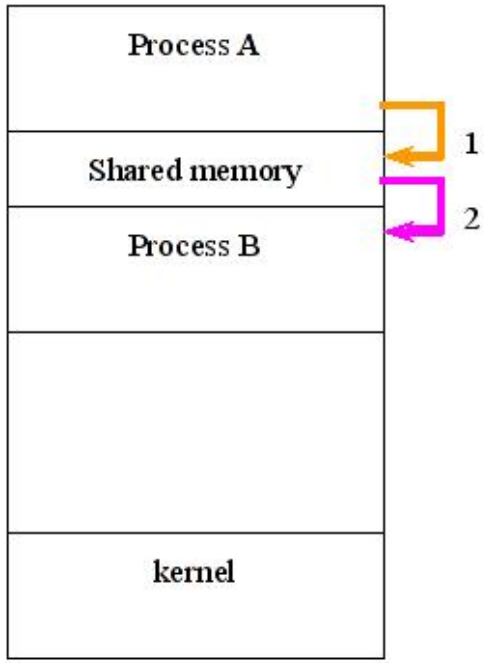
二、文件管理类
三、设备管理类
四、信息维护类

五、通信类系统调用

- 两种常见的通信模型
  - Message-passing VS. Shared-memory



(a) Msg passing



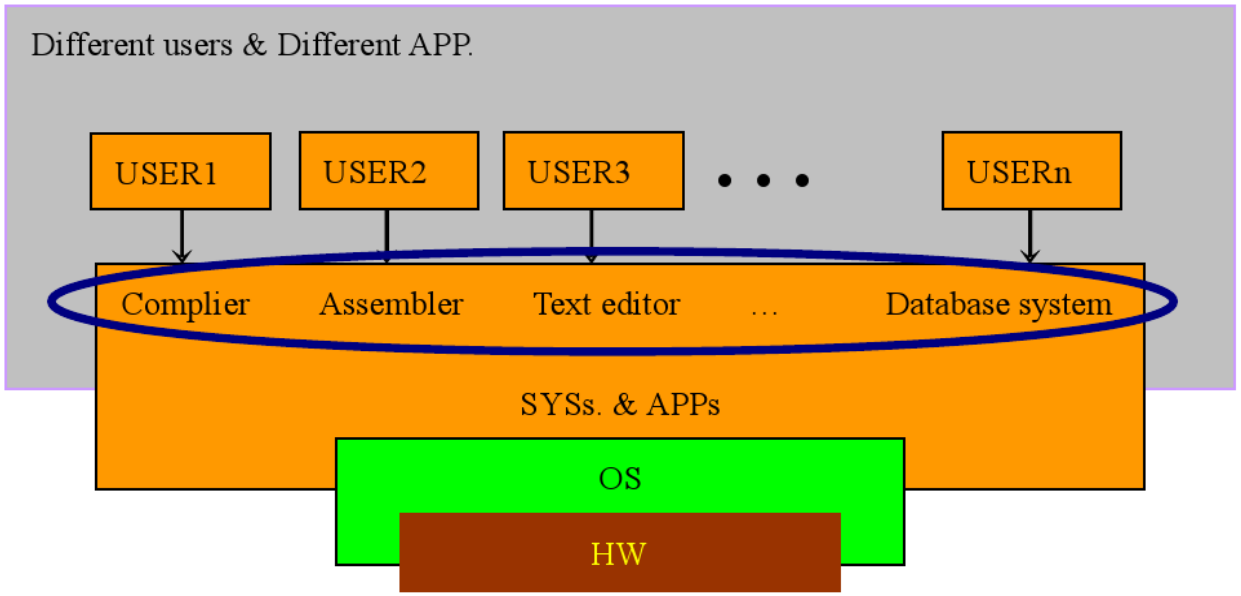
(b) Shared memory

Outline

1. 操作系统的组成、服务、特征
   - System components
   - OS Services
   - System Call
   - system programs
   - 操作系统的特征
2. 操作系统的抽象模型和体系结构
   - 进程模型
   - 线程模型
   - 操作系统的体系结构
   - 虚拟机
3. 小结和作业

system programs

向用户提供一个方便使用的环境。



Outline

1. 操作系统的组成、服务、特征
   - System components
   - OS Services
   - System Call
   - system programs
   - 操作系统的特征
2. 操作系统的抽象模型和体系结构
   - 进程模型
   - 线程模型
   - 操作系统的体系结构
   - 虚拟机
3. 小结和作业

操作系统的特征

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。
- 程序 vs. 进程
  - 程序：**静态实体**；
  - 进程：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。

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。

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。

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

1. 操作系统的组成、服务、特征
   - System components
   - OS Services
   - System Call
   - system programs
   - 操作系统的特征
2. 操作系统的抽象模型和体系结构
   - 进程模型
   - 线程模型
   - 操作系统的体系结构
   - 虚拟机
3. 小结和作业

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

1. 操作系统的组成、服务、特征
   - System components
   - OS Services
   - System Call
   - system programs
   - 操作系统的特征
2. 操作系统的抽象模型和体系结构
   - 进程模型
   - 线程模型
   - 操作系统的体系结构
   - 虚拟机
3. 小结和作业

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

1. 操作系统的组成、服务、特征
   - System components
   - OS Services
   - System Call
   - system programs
   - 操作系统的特征
2. 操作系统的抽象模型和体系结构
   - 进程模型
   - 线程模型
   - 操作系统的体系结构
   - 虚拟机
3. 小结和作业

操作系统的体系结构

- 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？

操作系统体系结构的发展历程

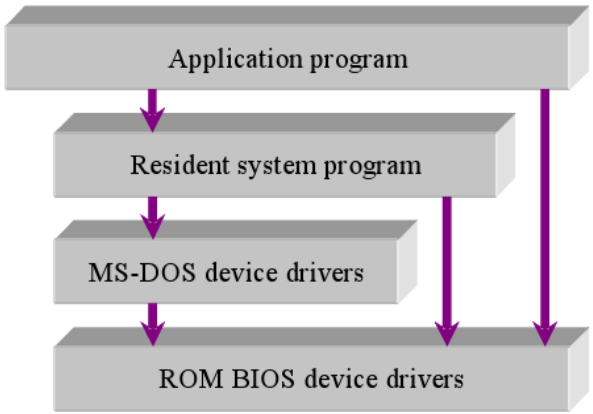
- 无结构，或者说简单结构
- 单一内核结构
- 模块化结构
- 层次式
- 微内核和第二代微内核
- 混合内核
- 外核

简单结构的操作系统

- 操作系统发展初期
  - Not a well-defined structure
    - Started as small, simple, and limited systems
  - 受到硬件性能、软件水平的限制
  - 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，
- 例如：MS-DOS、早期的UNIX系统以及一些早期的或者小型的嵌入式系统。

Example: MSDOS

- 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

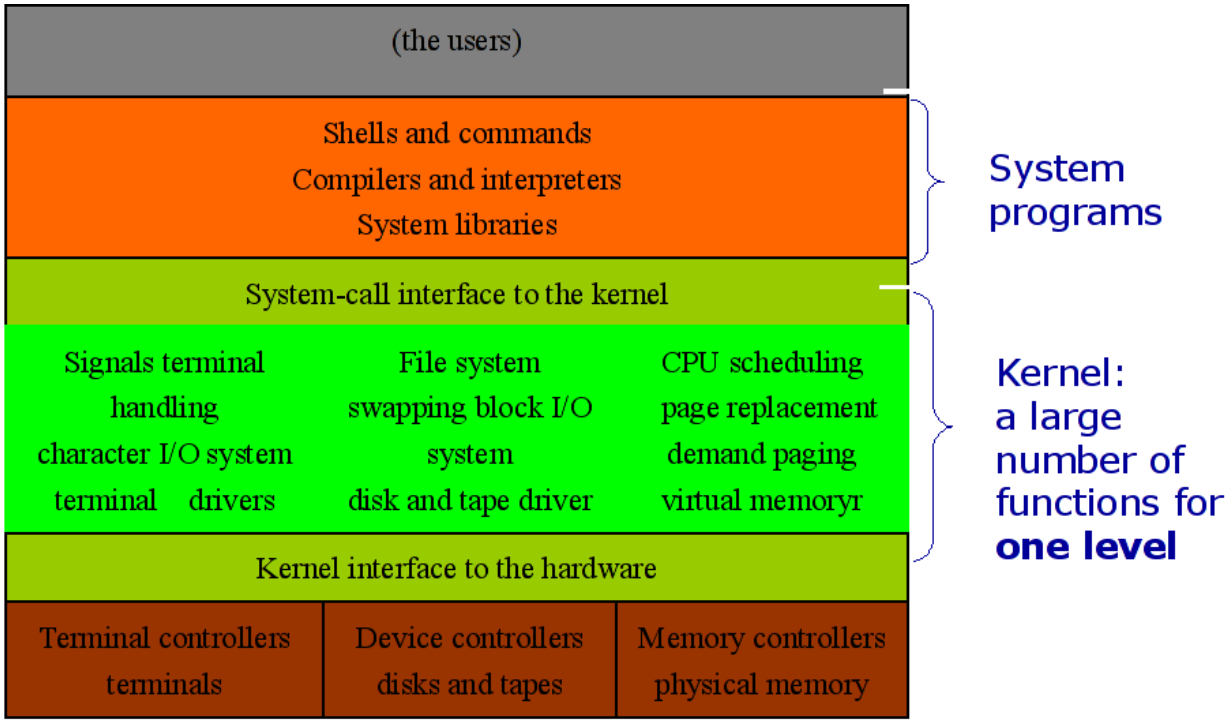


MS-DOS structure

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
  - 系统调用使得操作系统与用户应用程序隔离开来；
  - 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
- 这种结构被称为单一大内核（Monolithic Kernel）结构。
  - 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进程间通信机制进行通信。

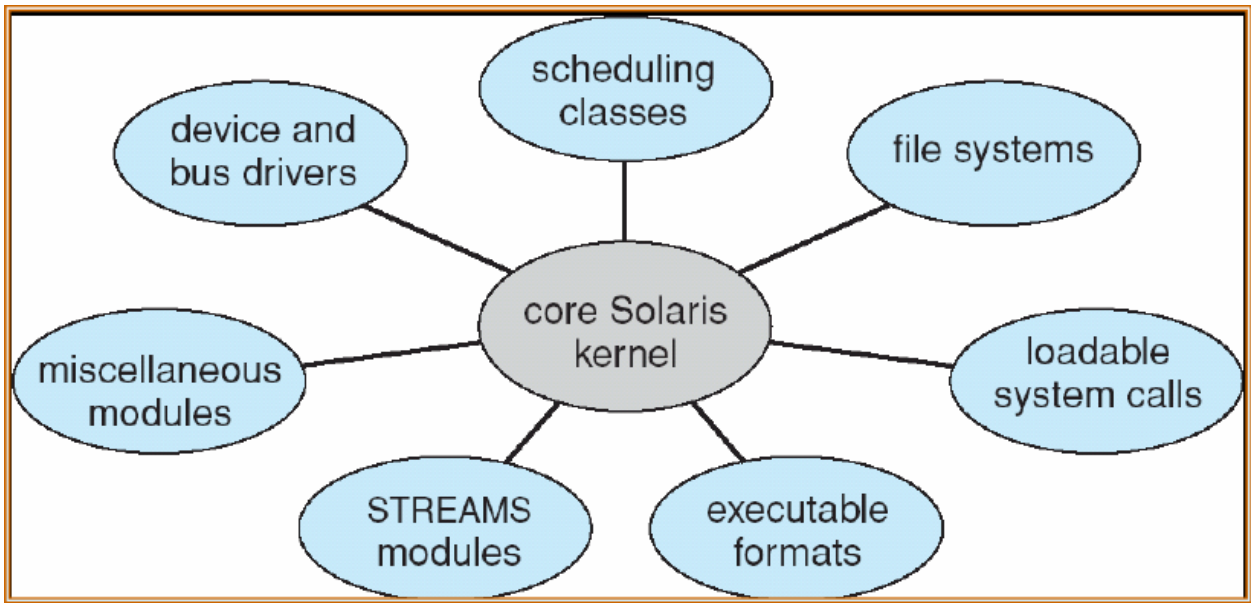
Example


(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

模块化结构

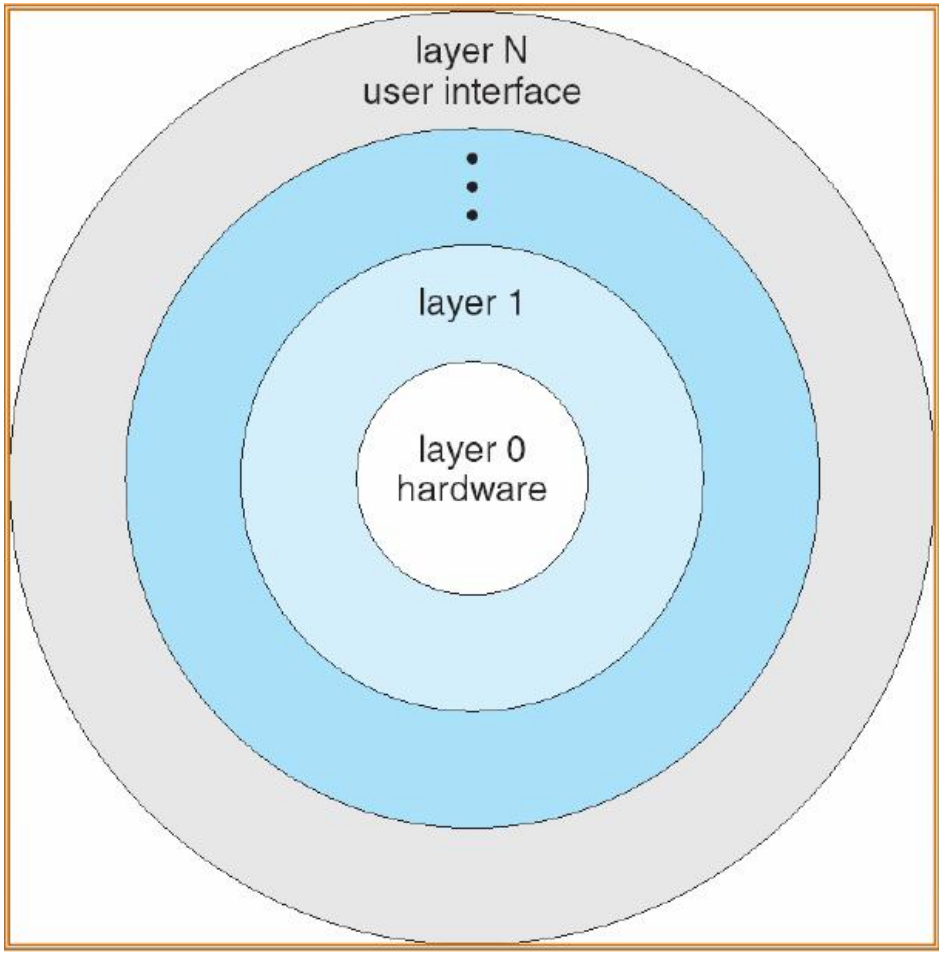
- 随着软件开发技术的发展，单一大内核结构的操作系统逐渐采用了**模块化设计方法**
  - 模块之间定义了很好的以函数调用的形式提供的接口
  - 在一定程度上提高了操作系统的可维护性。
- 常见的例如：UNIX类

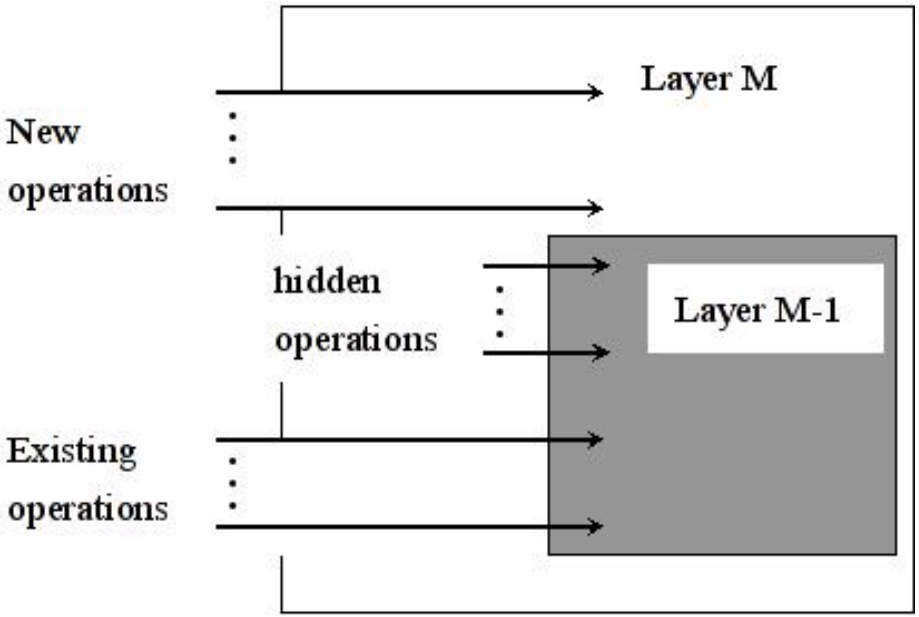
Solaris Modular structure

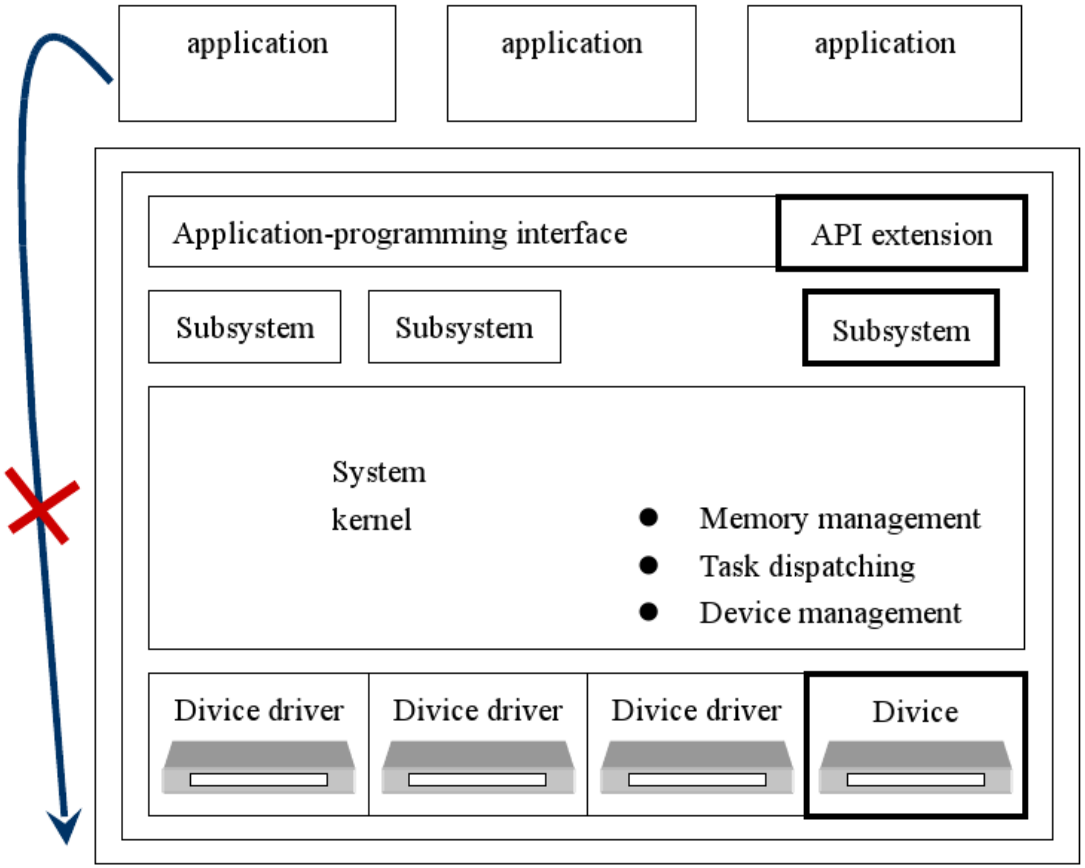


层次结构

- 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
- 在层次结构的操作系统内核中，
  - 系统由若干层次构成，每一层都建立在其下的一层之上
  - 最底层是硬件裸机，最高层则是应用程序。
  - 每一层均公布一定的接口给外层访问，每个层次内部的数据和操作对其他层都是不可见的。
- 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。


layer N
user interface
layer 1
layer 0
hardware


Layer M
New
operations
hidden
operations
Layer M-1
Existing
operations



Fewer layers, more functionality

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

微内核结构

- Richard Rashid在CMU开发Mach时提出了微内核思想
  - 微内核仅仅提供进程管理、线程管理、内存管理、通信和I/O服务等基本功能
  - 其他的功能，诸如文件系统、窗口管理器、WEB服务等都被定义在核外，作为用户态（服务）进程运行
  - 进程之间只能基于消息传递机制通信
- 又称为客户机/服务器结构(C/S)，后来成为分布式操作系统最常使用的一种结构

- 微内核结构的优点：
  - 良好的兼容性、扩充性、灵活性、移植性、可靠性和网络支持
- 微内核存在一个致命的效率问题：
  - 与系统组件之间的基于消息传递的通信效率远低于对单一大内核中相应系统功能原语的调用效率
- 典型的微内核（又称第一代微内核）有Mach、Minix、QNX等。

第二代微内核

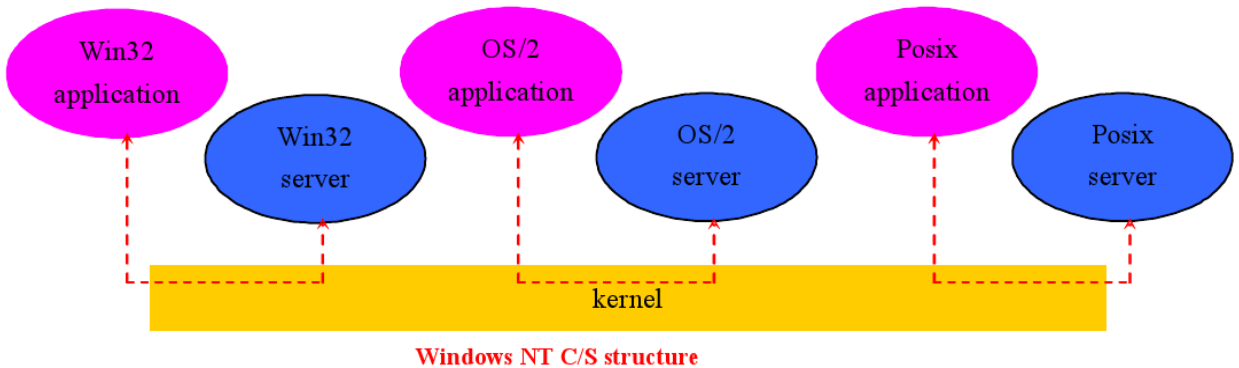
- 解决微内核设计性能问题的一条思路是
  - 对微内核的进程间通信机制加以改进以优化其性能。
  - 由此产生了第二代微内核系统
  - L4。

混合内核

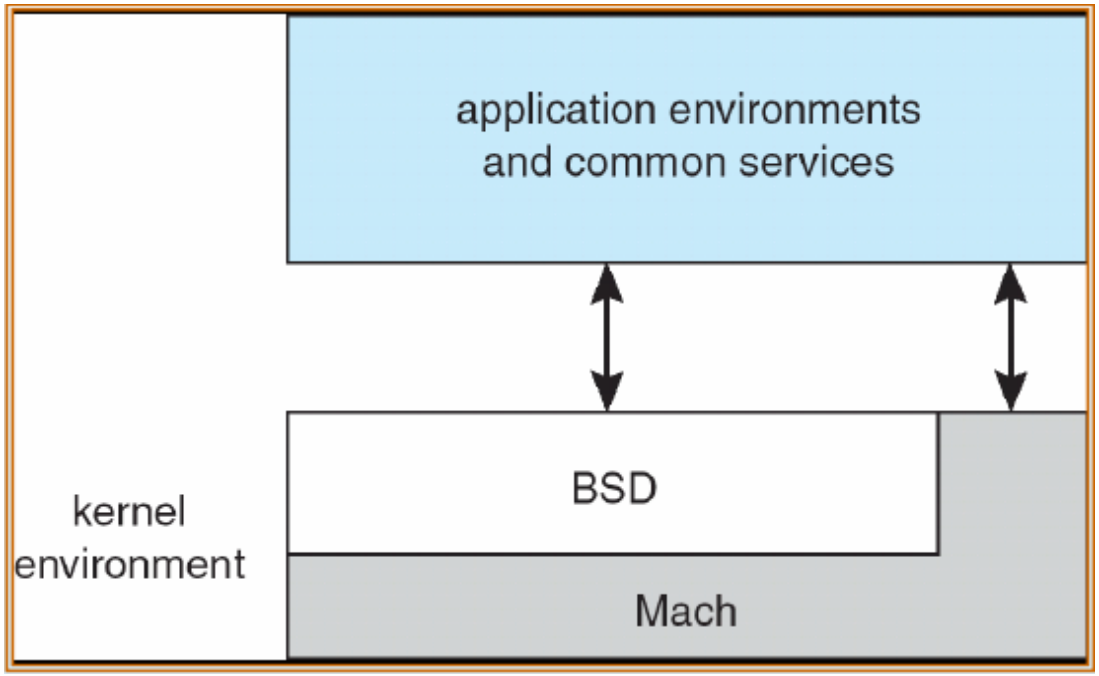
- 混合内核是解决微内核设计性能问题的另一个方法
  - 即扩大微内核并把一些关键的服务程序和驱动程序重新加入到内核中去，
  - 如Windows NT 4.0把图形系统重新加入内核以提高性能。
- 但这种方法削弱了微内核思想在系统的扩充性、灵活性和可靠性等方面所带来的优点

Example

- 微内核，混合内核



Windows NT C/S structure


application environments
and common services
BSD
kernel
environment
Mach

外核（Exokernel ）

- 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：

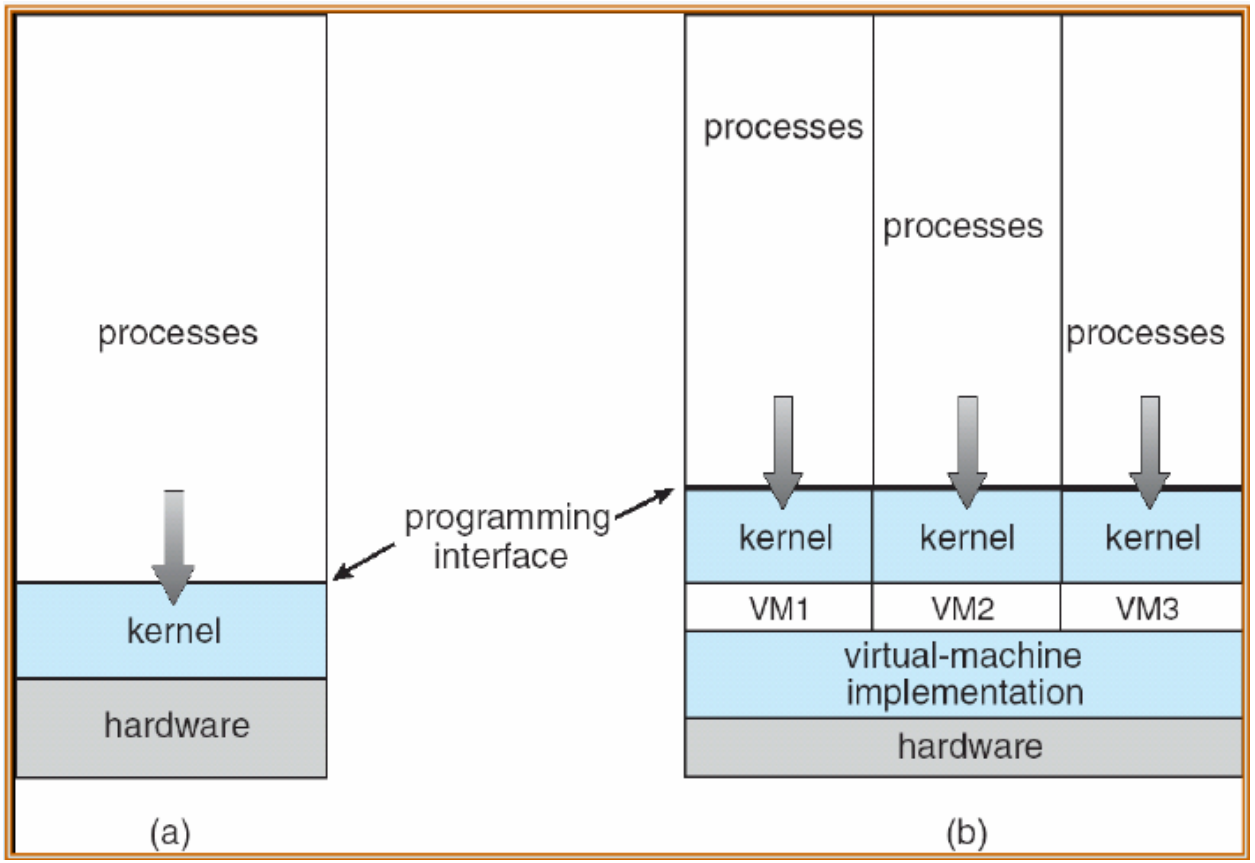
- 除了简单结构以外，操作系统中都存在一个内核
  - 核外的程序通过中断、异常、系统调用的方式访问内核中提供的服务；
  - 内核内部模块之间采用函数调用的形式直接调用（不论接口定义是否良好）；
- 操作系统提供给核外的程序的抽象是进程/线程
  - 包括用户应用和外放到核外运行的操作系统功能模块
  - 进程/线程之间通过进程间通信机制进行通信；

Outline

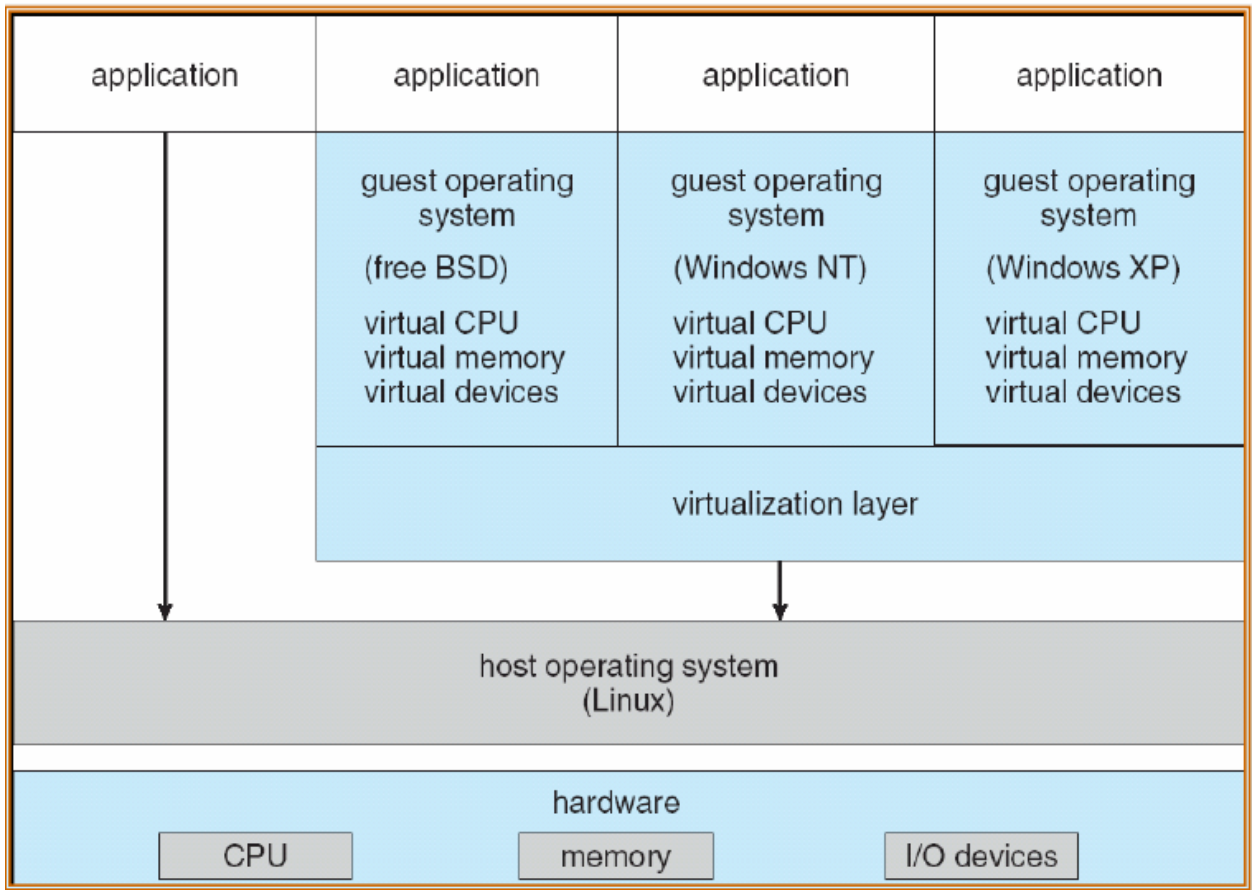
1. 操作系统的组成、服务、特征
   - System components
   - OS Services
   - System Call
   - system programs
   - 操作系统的特征
2. 操作系统的抽象模型和体系结构
   - 进程模型
   - 线程模型
   - 操作系统的体系结构
   - 虚拟机
3. 小结和作业

虚拟机

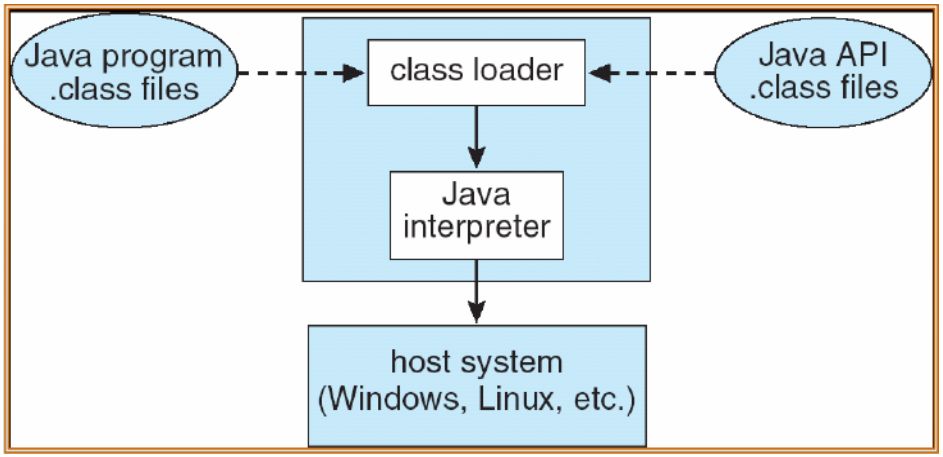
- A **virtual machine** takes the layered approach to its logical conclusion.
- It treats hardware and the OS kernel as though they were all hardware
- A virtual machine provides an interface identical to the underlying bare hardware
- The OS creates the illusion of multiple processes, each executing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


processes
kernel
hardware
(a)
programming interface
processes
processes
processes
kernel
kernel
kernel
VM1
VM2
VM3
virtual-machine implementation
hardware
(b)

VMWare Virtual Machine



Java Virtual Machines



小结

1. 操作系统的组成、服务、特征
   - System components
   - OS Services
   - System Call
   - system programs
   - 操作系统的特征
2. 操作系统的抽象模型和体系结构
   - 进程模型
   - 线程模型
   - 操作系统的体系结构
   - 虚拟机
3. 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 weaknesses of the two approaches?
- 阅读恐龙书第6版翻译版：3.6, 3.7, 3.8，回答习题三的3.15
  华夏班：阅读恐龙书第7版影印版：2.6, 2.7, 2.8, 2.9, 2.10，回答Exercises 2.9, 2.12, 2.13

上机作业

- 在linux中，编写一个程序，该程序能创建一个进程
  - 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
  - 子进程输出自己的进程号
  - 让子进程执行另外一个程序。
  - 提示：getpid，fork，execve
  - 华夏班：父进程创建一个pipe，父子进程通过pipe通信，父进程将自己的进程号传递给子进程，子进程将读到的进程号输出。
    - 提示：pipe，write，read，close
  - 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！