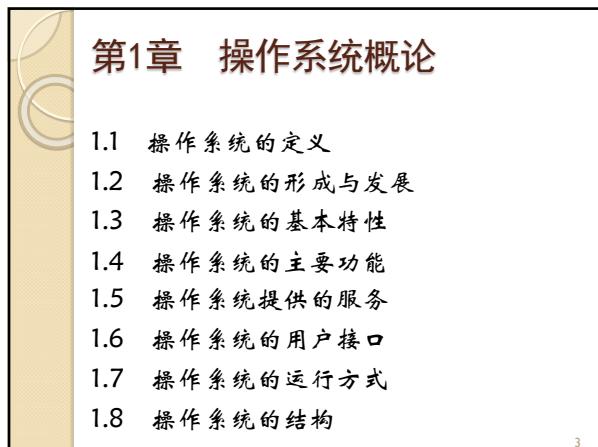
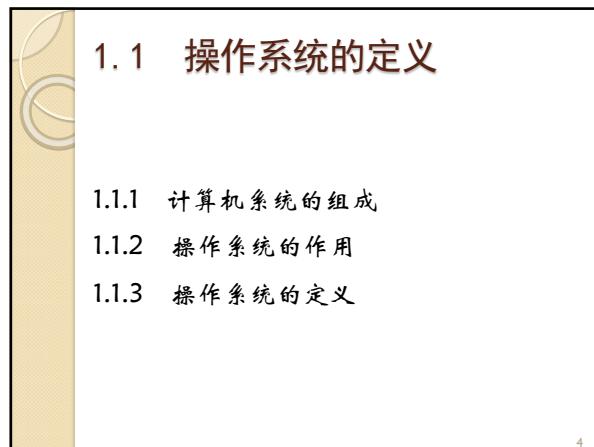


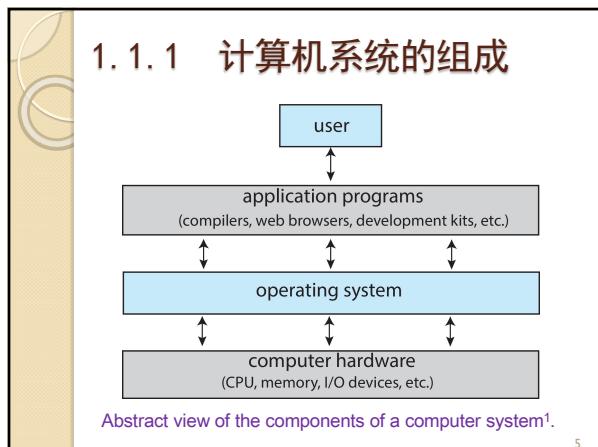
2



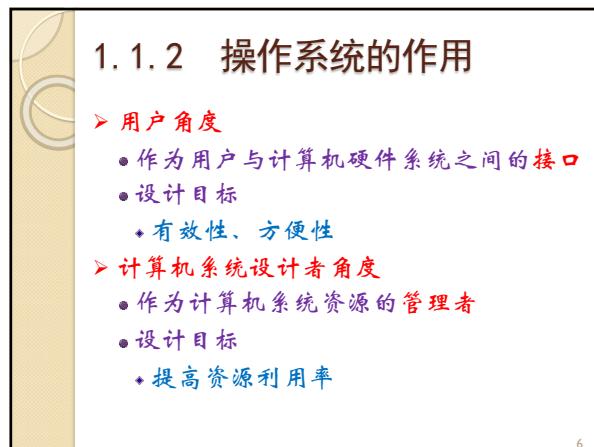
3



4



5



6

1.1.3 操作系统的定义(1)

- 操作系统是计算机系统中的一个**系统软件**，是一些程序**模块**的集合
 - 它们能以**尽量有效、合理**的方式组织和管理计算机的**软、硬件资源**，合理的组织计算机的工作流程，控制程序的执行并向用户提供各种服务功能，使得用户能够**灵活、方便、有效**的使用计算机，使整个计算机系统能高效地运行。是计算机与用户之间的**接口**

7

1.1.3 操作系统的定义(2)

- “The one program running at all times on the computer” is the **kernel**, part of the operating system
- Everything else is either
 - a **system program** (ships with the operating system, but not part of the kernel) , or
 - an **application program**, all programs not associated with the operating system
- Today's OSes for general purpose and mobile computing also include **middleware** – a set of software frameworks that provide addition services to application developers such as databases, multimedia, graphics

8

1.2 操作系统的形成与发展

- 1.2.1 无操作系统的计算机系统
- 1.2.2 单道批处理操作系统
- 1.2.3 多道批处理操作系统
- 1.2.4 分时与多任务操作系统
- 1.2.5 实时操作系统
- 1.2.6 嵌入式操作系统
- 1.2.7 智能移动终端操作系统
- 1.2.8 分布式系统
- 1.2.9 操作系统的发展趋势

9

1.2.1 无OS的计算机系统

- 程序员直接使用计算机硬件系统，效率低下
- **单用户独占全机**
- **CPU等待人工操作**
 - 人工负责输入输出
 - 人工负责计算机的调度
 - 人工负责编排作业的运行顺序

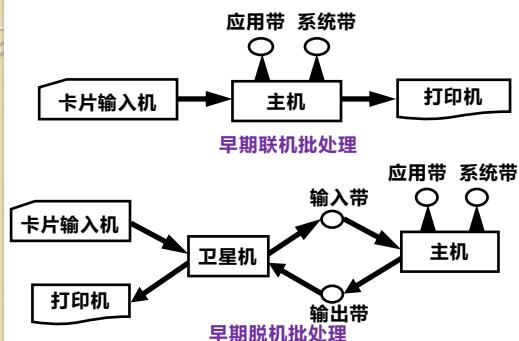
10

1.2.2 单道批处理操作系统(1)

- 以常驻内存的**监控程序**代替人工调度和作业编排，减少人工干预和等待时间，加快作业运行速度
- **硬件结构**
 - **早期联机批处理**
 - 作业的输入、计算和输出都在CPU控制下进行，CPU利用率低
 - **早期脱机批处理**
 - 小型卫星机控制外部设备的输入和输出

11

1.2.2 单道批处理操作系统(2)



12

1.2.2 单道批处理操作系统(3)

优点

- 系统自动化程度高、吞吐量大，资源利用率高
- 减少了CPU的空闲时间
- 提高了I/O速度

缺点

- CPU与外设串行
- 作业周转时间长，用户无法实现对作业的控制

13

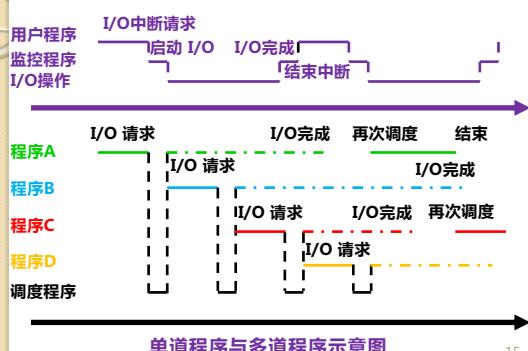
1.2.3 多道批处理操作系统(1)

多道程序设计

- 引入原因：CPU与外设并行执行，提高CPU利用率
- 在内存中同时存放若干道程序，使之在系统中同时处于交叉运行状态
- 特点
 - 内存多道
 - 宏观上并行（不同的作业分别在CPU和外设上执行）
 - 微观上串行（在CPU上交叉运行）

14

1.2.3 多道批处理操作系统(2)



15

1.2.3 多道批处理操作系统(3)

目的

- 提高CPU的利用率
- 充分发挥系统设备的并行性
 - 程序之间
 - CPU与设备之间
 - 设备与设备之间

➤ 单道程序设计示例：主存中有一道程序，读写一个记录各需0.0015s，处理一个记录（执行100条指令）需0.0001s，则CPU利用率为

$$0.0001/(0.0015+0.0001+0.0015) \approx 3.2\%$$

16

1.2.3 多道批处理操作系统(4)

➤ 多道程序设计示例：若一个计算机系统有256K内存(不包含OS)，1个磁盘、1个终端和1台打印机。内存中装有的3个作业对资源的使用情况如下：

作业编号	JOB1	JOB2	JOB3
类型	计算型	I/O型	I/O型
占用内存	50K	100K	80K
使用磁盘	NO	NO	YES
使用终端	NO	YES	NO
使用打印机	NO	NO	YES
运行时间	5分钟	15分钟	10分钟

17

1.2.3 多道批处理操作系统(5)

单道批处理

- ✓ 作业1运行5分钟
- ✓ 作业2等待5分钟运行15分钟
- ✓ 作业3等待20分钟运行10分钟



多道批处理

- ✓ 三个作业同时装入主存，由于几乎不同时使用同类资源，在15分钟内将全部完成



18

1.2.3 多道批处理操作系统(6)

单道/多道程序设计方式下的资源利用率

	单道	多道(三道)
CPU利用率	17% = 5/30	33% = 5/15
存储器利用率	30% = (50/256+100/256+80/256)/3	90% = 230/256
磁盘利用率	33% = 10/30	67% = 10/15
打印机利用率	33% = 10/30	67% = 10/15
全部完成时间	30min.	15min.
吞吐量	6jobs/hour = 3/0.5	12jobs/hour = 3/0.25
平均周转时间	18min. = (5+20+30)/3	10min. = (5+15+10)/3

19

1.2.3 多道批处理操作系统(7)

➤ 衡量批处理系统的性能指标

- **资源利用率**: 给定时间内系统中某一资源(存储器、CPU、外设等)实际使用时间所占比率。
- **吞吐量**: 单位时间内系统所处理的信息量, 通常以每小时或每天所处理的作业个数进行度量。
- **周转时间**: 从作业进入系统到作业退出系统(即完成)所用的时间。
- **平均周转时间**: 系统运行的多个作业的周转时间的平均值。

20

1.2.3 多道批处理操作系统(8)

➤ 多道批处理系统的特点

- 有效地提高了系统资源的利用效率
- 提高了系统的吞吐量
- 用户与作业之间无法交互
- 作业平均周转时间较长
- 适合处理计算量大的成熟的作业

➤ 典型系统

- **IBM OS/360**: 首次将操作系统与计算机相分离, 从专用走向通用的操作系统

21

1.2.4 分时与多任务操作系统(1)

➤ 批处理系统的缺点

- 用户不能直接控制作业运行
- 作业周转时间太长

➤ 产生分时系统的原因

- 用户需求(交互+响应)

➤ 分时概念

- 多个用户分时使用CPU时间, 即将CPU的单位时间划分成若干时间段(每个时间段称为一个**时间片**), 各用户按时间片轮流占用CPU

22

1.2.4 分时与多任务操作系统(2)

➤ 分时系统的特点

- 同时性
- 独立性
- 交互性
- 及时性
 - 响应时间为2-3s
 - 响应时间是衡量分时系统的主要指标
 - 影响响应时间的因素
 - 用户数目
 - 时间片

23

1.2.4 分时与多任务操作系统(3)

批处理系统

- 处理对象为作业
- 目标是提高系统资源利用率
- 适用于比较成熟大型的作业
- 可在后台执行, 不需要用户频繁干预

分时系统

- 处理对象为作业
- 目标是对用户请求快速响应
- 适用于短小作业
- 终端键入命令

24

1.2.4 分时与多任务操作系统(4)

➤ 典型系统

- 第1个分时系统 CTSS(Compatible Time-Sharing System)
- 典型的具有重要意义的分时系统 MULTICS(Multiplexed Information and Computing System)
- UNIX / LINUX

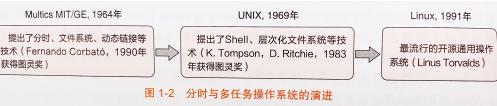


图 1-2 分时与多任务操作系统的演进

1.2.4 分时与多任务操作系统(5)

• DOS/Windows

• macOS

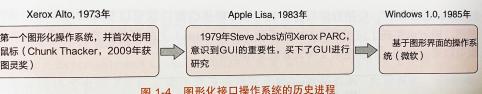


图 1-4 图形化接口操作系统的进程

26

1.2.5 实时系统(1)

➤ 实时概念

- 系统对外部特定输入信号的响应时间足以控制发出实时信号的设备

➤ 实时系统分类

- 应用需求
 - 实时控制系统/实时信息处理系统
 - 实时任务的截止时间
 - 硬实时/软实时

27

1.2.5 实时系统(2)

➤ 实时系统的特点

- 实时性
- 可靠性
- 可确定性

➤ 实时系统适合于一些不强调资源利用效率的专用系统

➤ 实时系统以数据或信息作为处理对象

28

1.2.6 嵌入式操作系统

➤ 嵌入式操作系统

- 以应用为中心、以计算机技术为基础、软硬件可裁剪的专用计算机系统
- 基本都是实时操作系统，只有满足实际需要的有限功能，如任务调度、同步与通信、内存管理、时钟管理等
- 特点
 - 软件要求固化存储
 - 高质量、高可靠性、高实时性
- 典型系统
 - Linux, FreeRTOS

29

1.2.7 智能移动终端操作系统

➤ 智能移动终端操作系统

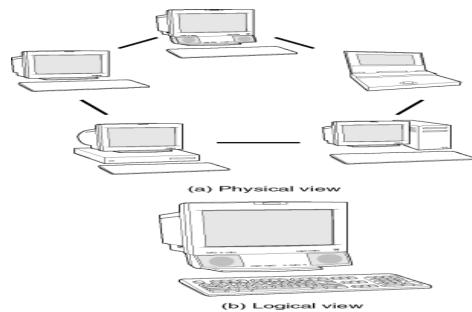
- 在受限的计算能力上提供丰富的用户体验
- 在受限的供电能力上改善系统的能源效率
- 在存储大量用户敏感信息场景下保证用户信息的隐私和安全
- 典型系统
 - Android
 - iOS

30

1.2.7 分布式系统(1)

- Definition of a Distributed System
 - A distributed system is a collection of independent computers appears to its users as a single coherent system.
 - Component: autonomous
 - User: a single system
- A distributed system is the one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages.

1.2.7 分布式系统(2)



1.2.9 操作系统的发展趋势

- 从封闭到开放，再到封闭
- 从专用到通用，再到专用（领域定制）
- 从简单到复杂，到更复杂

33

1.3 操作系统的基本特性

- 并发性
 - 并发性：两个或多个事件在同一时间间隔内发生，它们都已经被启动执行，而且都还没有完成执行
 - 并行性：两个或多个事件在同一时刻发生
- 共享性
 - 互斥共享
- 虚拟性
- 异步性

34

1.4 操作系统的主要功能

- 处理机管理
 - 进程控制/进程同步/进程通信/进程调度
- 存储器管理
 - 内存分配/内存保护/地址映射/内存扩充
- 文件管理
 - 文件存储空间的管理/目录管理/文件的读写管理和保护
- 设备管理
 - 缓冲管理/设备分配

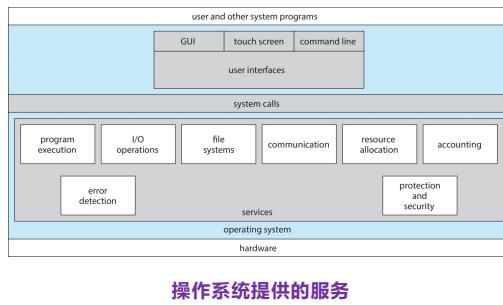
35

1.5 操作系统提供的服务(1)

- 用户接口
 - 命令级接口：Graphical User Interface/Touch Screen Interface/Command-Line Interface
 - 程序级接口（也称系统调用）
- 执行程序
- I/O操作
- 文件系统操作
- 通信服务：共享内存/消息传递
- 错误检测和处理
- 资源分配
- 登录和记账
- 保护和安全

36

1.5 操作系统提供的服务(2)



37

1.6 操作系统的用户接口(1)

操作接口

- 命令行 (命令解释程序)
 - OS内核的一部分(DOS)
 - 特殊程序，任务开始或用户登录时，该程序运行(UNIX的Shell)
 - 作用：执行命令
 - 命令解释程序执行(DOS)
 - 系统程序实现(UNIX的Shell)
- Graphical User Interface
 - 具有窗口界面的解释程序(Windows的Explorer.exe)
- Touch Screen Interface
 - Gesture, Voice Command

38

1.6 操作系统的用户接口(2)

编程接口

- 系统调用
 - 请求操作系统服务或资源
- 常以API形式出现
 - 适用于Windows系统的Win32 API
 - 适用于POSIX系统的POSIX API
 - POSIX: Portable Operating System Interface of UNIX
 - 设计运行于Java虚拟机的Java API
- 应用领域接口
 - Android/iOS应用接口、汽车领域AUTOSAR

39

1.6 操作系统的用户接口(3)

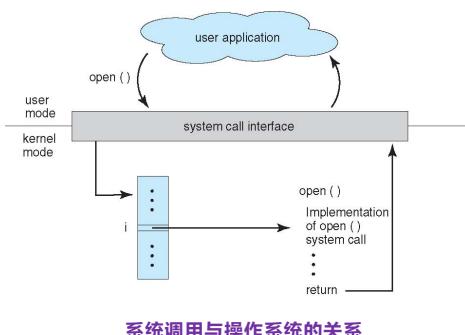
EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

The following illustrates various equivalent system calls for Windows and UNIX operating systems.

	Windows	Unix
Process control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File management	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communications	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shm_open() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

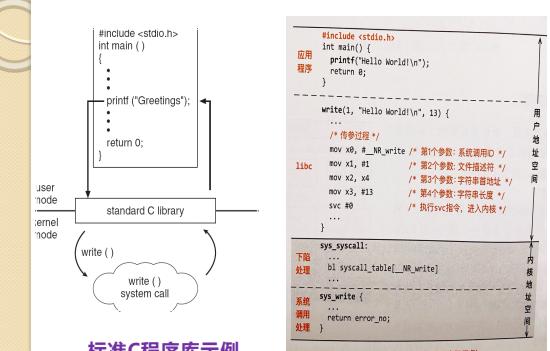
40

1.6 操作系统的用户接口(4)



41

1.6 操作系统的用户接口(5)



42

1.7 操作系统的运行方式(1)

➤ 双重模式操作

- 内核模式 
 - 允许执行全部指令
 - 允许访问所有的寄存器和缓冲区
- 用户模式 
 - 只能执行非特权指令
 - 只能访问指定的寄存器和存储区

43

1.7 操作系统的运行方式(2)

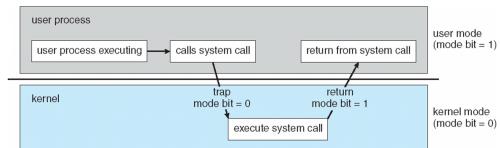
➤ 特权指令

- 能引起损害的机器指令
- E.g. I/O控制, 定时器管理, 中断管理
 - 有关对I/O设备使用的指令: 启动I/O设备指令、测试I/O设备工作状态和控制I/O设备动作的指令等
 - 有关访问程序状态的指令: 操作程序状态字(PSW)的指令等
 - 存取特殊寄存器指令: 存取中断寄存器、时钟寄存器等指令

44

1.7 操作系统的运行方式(3)

➤ 计算机硬件提供模式位



用户模式到内核模式的转换

45

1.7 操作系统的运行方式(4)

➤ 两种模式的转换

- 用户模式-->内核模式
 - 中断、异常、系统调用
- 内核模式-->用户模式
 - 完成中断或异常的处理
 - 新进程或线程启动
 - 进程调度选择执行新的进程
 - 操作系统向进程发送信号

46

1.8 操作系统的结构

- 1.8.1 操作系统的机制与策略
- 1.8.2 操作系统复杂度管理方法
- 1.8.3 操作系统内核架构
- 1.8.4 操作系统框架

47

1.8.1 操作系统的机制与策略

- 目的: 降低操作系统设计的复杂性
- 设计原则: 策略与机制分离
 - 策略(Policy): 做什么
 - 机制(Mechanism): 怎么做
 - 操作系统可仅通过调整策略来适应不同的应用需求
- 示例: 调度
 - 策略: 调度算法, FCFS, RR
 - 机制: 调度队列的设计、调度实体(线程)的表示

48

1.8.2 操作系统复杂度管理方法

➤ 模块化(Modularity)

- 分而治之

➤ 抽象(Abstract)

- 接口与内部实现分离

➤ 分层(Layering)

- 按层次划分，减少模块之间的交互

➤ 层级(Hierarchy)

- 将功能相似的子系统组成一个具有清晰接口的自包含子系统，子系统递归形成大系统

49

1.8.3 操作系统内核架构(1)

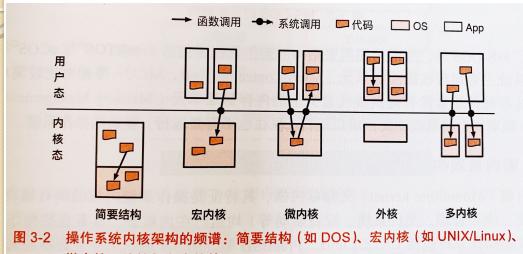


图 3-2 操作系统内核架构的频谱：简要结构(如 DOS)、宏内核(如 UNIX/Linux)、微内核、外核与多内核等

50

1.8.3 操作系统内核架构(2)

➤ 1.8.3.1 简要结构

➤ 1.8.3.2 宏内核架构(Monolithic Kernel)

➤ 1.8.3.3 微内核架构(Microkernel)

➤ 1.8.3.4 外核架构(Exokernel)

➤ 1.8.3.5 多内核架构(Multikernel)

➤ 1.8.3.6 混合内核架构(Hybrid Kernel)

51

1.8.3.1 简要结构(1)

- 功能简单的操作系统，将应用程序和操作系统放置在同一个地址空间中，无须底层硬件提供复杂的内存管理、特权级隔离等功能

➤ 模块化与层次结构

- 将操作系统按其功能划分为若干个具有一定独立性和大小的模块和子模块
- 各模块间通过接口实现交互
 - ◆ 模块间调用关系无序
 - ◆ 模块间耦合紧密

52

1.8.1 简要结构(2)

➤ 优点

- 提高了OS设计的正确性、可理解性和可维护性
- 增强了OS的适应性
- 加速了OS的开发过程

➤ 缺点

- 系统结构不清晰
- 系统可靠性降低
- 未能区分共享资源和独占资源

53

1.8.1 简要结构(3)

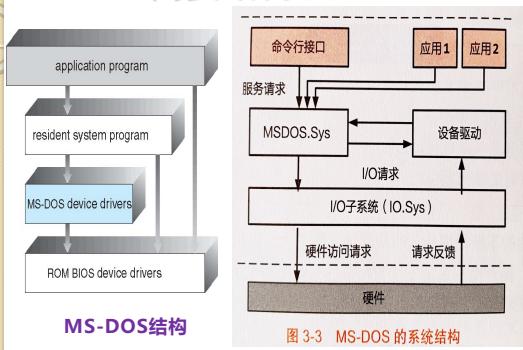


图 3-3 MS-DOS 的系统结构

54

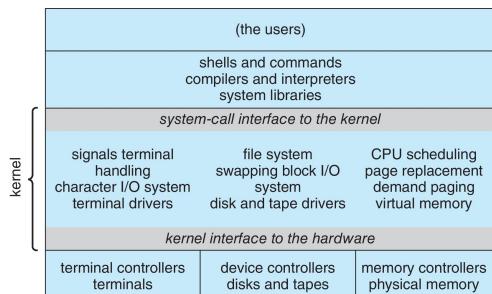
1.8.3.2 宏内核架构(1)

- 操作系统内核的所有模块均运行在内核态（单内核），具备直接操作硬件的能力
- UNIX/LINUX, FreeBSD 等



图 3-4 宏内核的基本结构

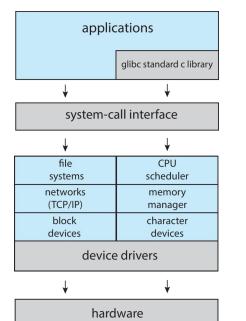
1.8.3.2 宏内核架构(2)



Traditional UNIX system structure.

56

1.8.3.2 宏内核架构(3)



Linux system structure.

57

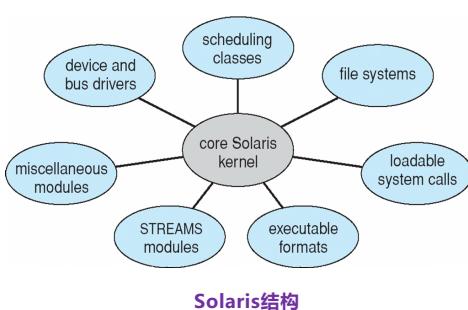
1.8.3.2 宏内核架构(4)

➤ 模块化

- 内核有一组核心部件，以及在启动或运行时对附加服务的动态链接
- 内核提供核心服务，并能动态实现特定的功能
- 可加载内核模块 LKM(Loaded Kernel Module)机制
- 现代操作系统广为采用
 - UNIX/Linux/Windows 等

58

1.8.3.2 宏内核架构(5)



Solaris结构

59

1.8.3.2 宏内核架构(6)

➤ 抽象

- UNIX：一切皆是文件(Everything is a file.)，将数据、设备、内核对象等均抽象为文件，并为上层提供统一接口。

➤ 分层

- 宏内核架构的操作系统一开始就采用了分层结构
- 引入原因
 - 模块间有序调用
- 设计原则
 - 将功能模块排列成若干层

60

1.8.3.2 宏内核架构(7)

- 各层之间的模块只能单向调用，每一层都仅使用其底层（或内层）模块所提供的功能和服务
- 每一层的同层模块之间不存在相互调用关系
- 最底层（第0层）是计算机硬件；最高层（第N层）是用户接口
- 优点
 - 模块化
 - 易于实现
 - 增加了系统的可靠性

61

1.8.3.2 宏内核架构(8)

- 示例：1968年图灵奖获得者Edsger Dijkstra提出“THE”操作系统

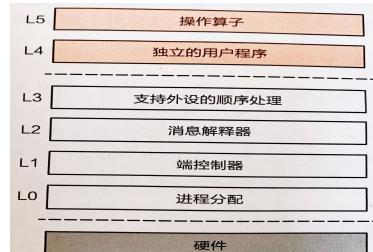


图 3-5 “THE”操作系统的分层结构

62

1.8.3.2 宏内核架构(9)

- 示例：Linux的文件系统

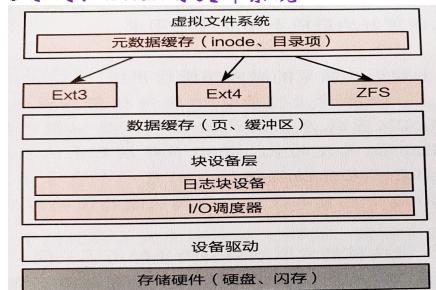
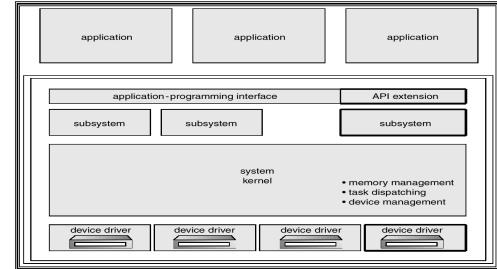


图 3-6 文件系统的分层结构

63

1.8.3.2 宏内核架构(10)

- 困难：难以确切地定义每一层
- 缺点：效率低

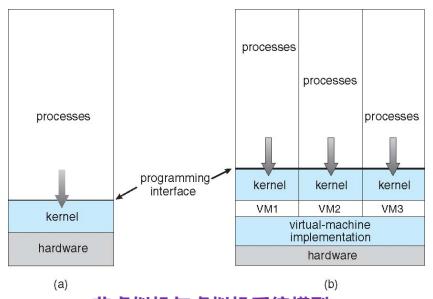


OS/2分层结构

64

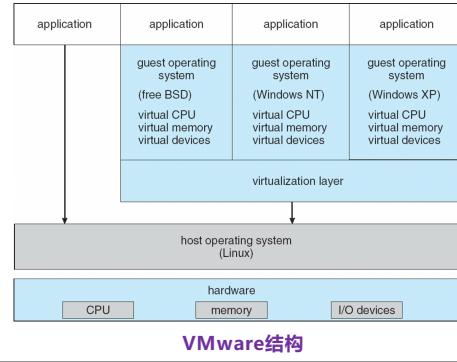
1.8.3.2 宏内核架构(11)

➤ 虚拟机



65

1.8.3.2 宏内核架构(12)



VMware结构

66

1.8.3.2 宏内核架构(13)

➤ 层级

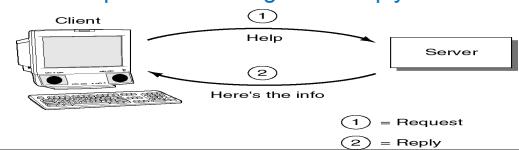
- 广泛应用于内核的资源管理中
 - 调度子系统中对进程优先级的分类
 - 内存分配器对不同内存的分类

67

1.8.3.3 微内核架构(1)

➤ 微内核结构

- 操作系统从内核转移到“用户”空间
- 以客户/服务器模式为基础
 - Server: a process implementing a certain service.
 - Client: uses the service by sending a request and waiting for the reply



1.8.3.3 微内核架构(2)

➤ 微内核的基本功能

- 进程管理
 - 进程作为资源分配的基本单位
 - 线程作为独立运行和调度的基本单位
 - 进程与线程间的同步
- 存储器管理
 - 为进程分配必要的运行空间
 - 提供虚拟存储器管理
 - 处理缺页中断

69

1.8.3.3 微内核架构(3)

- 进程通信管理
 - 用户模块之间通过传递消息进行通信
 - I/O设备管理
 - 设备驱动程序
- 微内核结构的实现方式
- 将最基本、最本质的OS功能保留在核中
 - 用户空间包含所有OS服务进程和用户应用进程
 - 每一OS功能均以单独的服务器进程形式存在

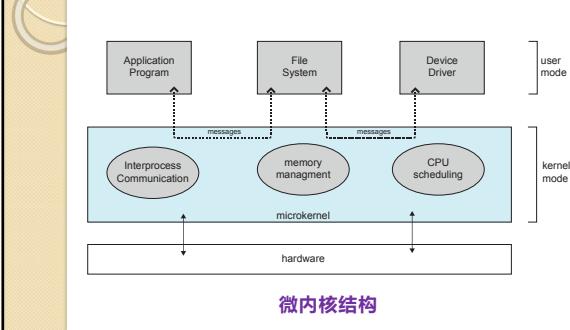
70

1.8.3.3 微内核架构(4)

- 进程以发送消息的方式通过微内核向服务器进程提出服务请求
 - 服务器进程处理完请求服务后，以发送消息的形式通过微内核将结果返回客户进程
- 优点
- 策略与机制进一步分离
 - 易于扩展
 - 易于移植
 - 服务与服务之间完全隔离
 - 更可靠
 - 更安全
- 缺点：性能

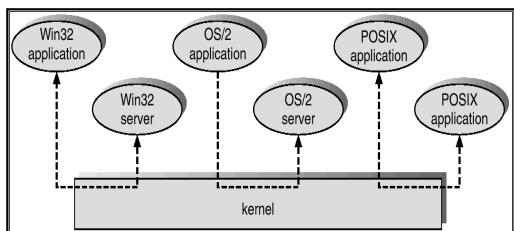
71

1.8.3.3 微内核架构(5)



72

1.8.3.3 微内核架构(6)



Windows NT结构

73

1.8.3.3 微内核架构(7)

➤ 微内核的发展

• 第1代：Mach

• Rick Rashid, Rochester+CMU

• 对进程间通信(IPC)的设计过于通用，使得其性能弱于同时期的宏内核

• 第2代：L4

• Jochen Liedtke, 德国国家信息技术研究中心

• 极大提升了IPC的性能

• 微内核最小化：一个操作系统内核的功能只有在其放在内核态以外会影响整个系统功能时，才能被放置在内核态

74

1.8.3.3 微内核架构(8)

- 第3代：增强安全性
 - Gernot Heiser
 - 将能力(Capability)机制引入微内核操作系统
 - 通过Capability进行IPC的权限判断
 - 代表系统
 - seL4：第一个完成形式化验证的内核
 - Google Fuchsia
 - MINIX：教学用微内核

75

1.8.3.4 外核架构(1)

➤ 引入

- 操作系统内核对硬件资源进行了抽象

• 过度抽象会带来性能损失

• 通用抽象对具体应用往往不是最优选择

➤ 解决方案

• 不提供硬件抽象

• “只要内核提供抽象，就不能实现性能最大化”

• 只有应用才知道最适合的抽象

• 不管理资源，只管理应用

• 保证应用之间的隔离

• 负责应用与资源的绑定

76

1.8.3.4 外核架构(2)

➤ ExoKernel

- 1995年，MIT的Dawson Engler和Frans Kaashoek等提出
- 提出库操作系统(LibOS)概念
 - 将对硬件的抽象封装到LibOS中，以库形式提供，与应用直接链接
 - 按照应用领域特点高度定制化，不同应用动态组装成最适合该领域的LibOS，获得更高性能
 - 操作系统内核很小，多个LibOS之间具有强隔离性，提高安全性与可靠性

77

1.8.3.4 外核架构(3)

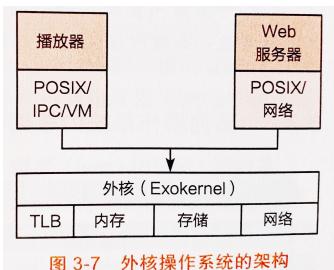


图 3-7 外核操作系统的架构

78

1.8.3.4 外核架构(4)

优点

- OS无抽象，能够在理论上提供最优性能
- 应用对计算有更精确的实时等控制
- LibOS在用户态更易调试

缺点

- 对计算资源的利用效率主要由应用决定
- 定制化过多，维护难度增加
- 缺乏跨场景的通用性，应用生态差

79

1.8.3.4 外核架构(5)

应用

- 一些功能受限、对操作系统接口要求不高但对性能和时延特别敏感的嵌入式场景，可以将数据面与控制面分离
- Unikernel(单内核)
 - 虚拟化环境下的LibOS
 - 每个虚拟机只使用内核态，内核态中只允许一个应用+LibOS，通过虚拟化层实现不同实例间的隔离
 - 适合容器等新的应用场景
 - 每个容器就是一个虚拟机，运行定制的LibOS以提高性能

80

1.8.3.5 多内核架构(1)

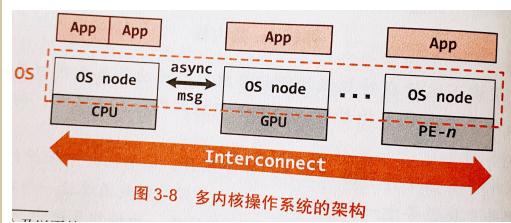
背景

- 硬件结构呈现异构众核特点
- 多内核架构
 - 苏黎世联邦理工学院、微软研究员、雷恩高等师范学校等联合提出的一种实验型的操作系统内核架构
 - 基本想法
 - 将一个众核系统看成一个由多个独立处理器核通过网络互联而成的分布式系统
 - 假设硬件处理器提供全局共享内存语义
 - 为不同处理器核交互提供了一层基于进程间通信的抽象

81

1.8.3.5 多内核架构(2)

- 在每个CPU核上运行一个独立的操作系统节点，节点间的交互由操作系统节点之上的进程间通信来完成



82

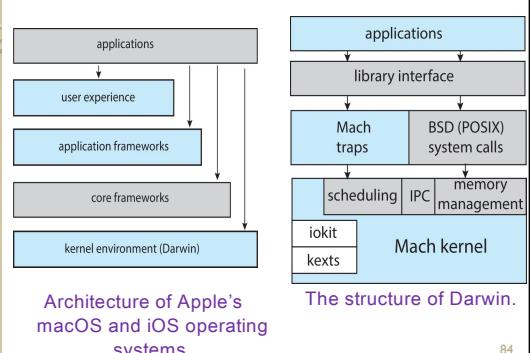
1.8.3.6 混合内核架构(1)

宏内核+微内核

- 将需要性能的模块重新放回内核态
 - Windows NT =
微内核+内核态的系统服务+系统框架
 - macOS/iOS =
Mach微内核+BSD 4.3+系统框架

83

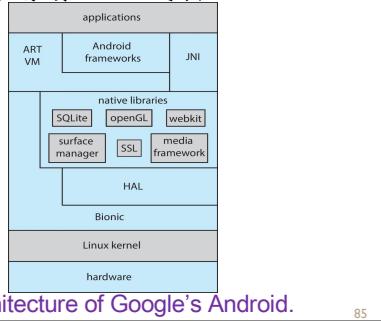
1.8.3.6 混合内核架构(2)



84

1.8.4 操作系统框架

➤ Android: 分层+Linux内核



85

本章要点

- 操作系统的定义
- 操作系统的特性和基本功能
- 操作系统的形成与发展
- 多道程序设计的概念和特点
- 操作系统的分类：批处理系统、分时系统、实时系统
- 操作系统的性能指标：资源利用率、系统吞吐量、作业（平均）周转时间
- 操作系统提供的服务
- 操作系统的两种运行模式
- 操作系统的结构

86