

第二章

操作系统概论

大纲

- 操作系统概观
- 操作系统的形成与发展
- 操作系统提供的服务和用户接口
- 操作系统的结构设计
- 流行操作系统简介

本章教学目标

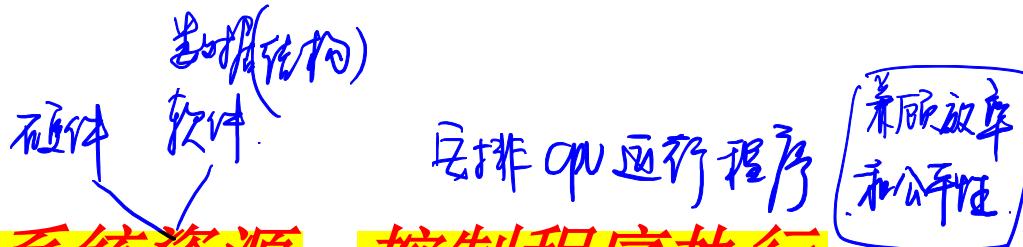
- 理解操作系统的定义、目标和基本功能
- 理解计算机系统的主要特性及其对操作系统设计带来的挑战
- 理解多道程序设计的概念
- 掌握系统调用的执行方式 API
- 理解操作系统的结构设计

操作系统的定义和目标

- 操作系统的定义：

- 没有严格的定义

- 一般认为，是管理系统资源、控制程序执行、
改善人机界面、提供各种服务、合理组织计算机工作流程和为用户计算机提供良好运行环境的一种系统软件。



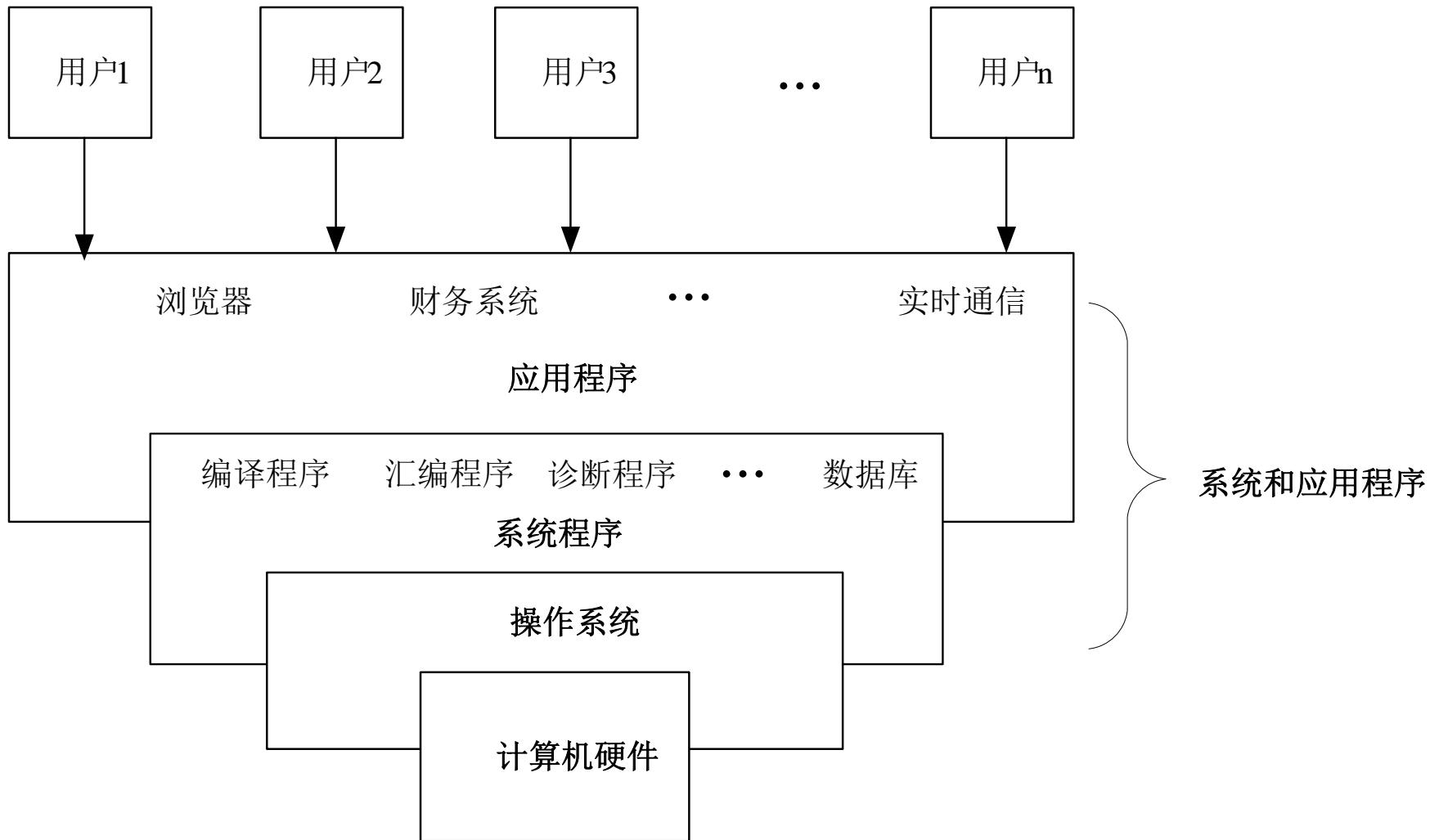
兼顾效率
和公平性

操作系统的定义和目标

- 操作系统的目标是提供一个能使用户**方便**、**高效**地执行程序的环境。
- 具体包括下述目标
 - 方便用户使用
 - 机械装置→命令行→图形用户界面
 - 扩大机器功能 **不用机器指令，用高级语言；**
 - 管理系统资源
 - CPU、存储、文件、...
 - 提高系统效率
 - CPU调度
 - 磁盘调度
 - 构筑开放环境
 - 如POSIX标准

{ 机制：功能实现一肩膀
 策略：算法，一翅膀

计算机系统的层次结构



计算机系统的层次结构

- 硬件层
 - 提供基本的可计算性资源，包括处理器、寄存器、存储器，以及各种I/O设施和设备
 - 是操作系统和上层软件赖以工作的基础
 - 操作系统层
 - **最靠近硬件的软件层**，对计算机硬件作首次扩充和改造
 - 完成**资源的调度和分配，信息的存取和保护，并发活动的协调和控制**
 - 是上层其他软件运行的基础。
 - 系统程序层
 - 各种各样的语言处理程序、数据库管理系统和其他系统程序
 - 实用程序，如连接装配程序、库管理程序、诊断排错程序、分类/合并程序等。
 - 应用程序层
 - 解决用户特定的或不同应用需要的问题
- 上层
下层

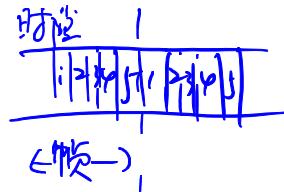
操作系统与上层软件的区别

- 控制与被控制
 - 操作系统有权分配资源，其它程序只能使用资源
访问特权指令
- 操作系统直接作用于硬件之上，隔离其他上层软件，是软件系统的核心，是各种软件的基础运行平台
- 操作系统实现资源管理机制，允许应用程序提供资源管理策略

资源管理技术

- 资源管理技术的目标
 - 解决物理资源有限与竞争使用资源的应用程序众多之间的矛盾，即**资源数量不足**的问题
 - 资源复用 在用户之间合理分配
 - 资源虚拟化 看上去能行吗？
 - 实现资源的易用性 < 应用程序
设备驱动程序
 - 资源抽象

资源管理技术



- 资源复用

- 空分复用 *e.g. 内存*

- 资源可以进一步分割成更多和更小的单位，供进程使用
 - 资源的不同单位同时分配给不同的进程
 - 例如，主存，磁盘

- 时分复用 *e.g. CPU. 时间划分很细*

- 进程可以在一个时间片内以独占方式使用整个物理资源，其它进程则在另外的时间片内使用此资源
 - 例如，磁带机，CPU

资源管理技术

- 资源虚拟化
 - 对资源进行转化、模拟或整合，把一个物理资源转变成逻辑上的多个对应物，创建无需共享的多个独占资源的假象，以达到多用户共享一套计算机物理资源的目的。
- 例子
 - 虚拟内存
 - spooling
 - 虚拟文件系统
- 时分复用、空分复用是基本，资源虚化是表现形式，资源虚化最终都要依赖时分复用和空分复用来实现

排序算法 API. 解耦合

void sort(void *objstart, int objlen, int numberofObject, int (*cmp)(void *obj1,
void *obj2,
int objlen))

任意类型均可赋值给它。无数据
(类型) 数组下标

void quicksort(__) //且实现排序算法

11)

```

int compare( void * obj1, void * obj2, int objlen )
{
    main
    void ( * ) ( _参数列表 ) mysort; //函数指针
    mysort = quicksort;
    mysort( persons, sizeof( Person ), 100, compare );
}

```

资源管理技术

- 资源抽象
 - 资源抽象软件对内封装实现细节，对外提供应用接口
 - 通过分层实现
- 资源抽象的例子：物理设备输出一组字符
 - Layer-0: 硬件接口，控制寄存器、状态寄存器、数据寄存器，I/O操作命令
 - Layer-1: 设备驱动程序，屏蔽物理设备处理I/O操作的细节
 - Layer-2: 系统调用
 - Layer-3: 库函数

- 机器指令
 - load(block, length, device)
 - seek(device, track)
 - out(device, sector)
- 系统调用

```
void write(char *block, int length, int device, int track, int sector)
{
    load(block, length, device);
    seek(device, track);
    out(device, sector);
}
```
- 库函数

```
Int fprintf(fdID, "%s", datum){
    ...
    write( );
}
```

磁盘映射

操作系统中的基础抽象

- 进程抽象
 - 对于进入主存的当前运行程序在处理器上操作的状态集的一个抽象
 - 是并发和并行操作的基础
 - 透明地**时分复用**一个（或多个）处理器
 - **处理器虚化**，每个进程都认为独自拥有一个处理器

操作系统中的基础抽象

- 虚存抽象
 - 物理主存被抽象成虚拟主存，给每个进程造成一种假象，认为它正在独占和使用整个主存
 - 每个进程可以使用**连续的虚拟地址**来引用物理内存单元
 - 进程虚拟内存中的内容可以被透明地交换到磁盘，从而为每个用户提供的**虚拟存储空间可以远大于主存的大小**。

操作系统中的基础抽象

- 文件抽象
 - 文件是磁盘、磁带、光盘等设备的抽象，通过将文件中的字节映射到存储设备的物理块中来实现文件抽象。
无需知道每种设备的存储方式
 - 文件抽象的步骤
 - 磁盘→分区：一个物理磁盘可以被划分成多个逻辑上独立的分区
 - 分区→扇区
 - 扇区→簇：解决不同磁盘的扇区大小可能不同的问题
 - 簇→文件系统分区

进程抽象

虚存抽象

文件抽象

处理器

主存

设备

操作系统的基础抽象

操作系统的作用与功能

- 操作系统作为**用户接口和**服务提供者****
 - 是用户与计算机硬件之间的**接口**①
- 操作系统是计算机系统的**资源管理者和**控制者****
②③

操作系统的作用与功能

- 操作系统是用户与计算机硬件之间的接口
 - OS处于用户与计算机硬件系统之间， 用户通过OS来使用计算机系统。
 - 操作系统为用户屏蔽硬件细节和复杂性。
 - 为用户操作计算机提供多种易用接口
 - 命令方式
 - 系统调用
 - 图形用户界面

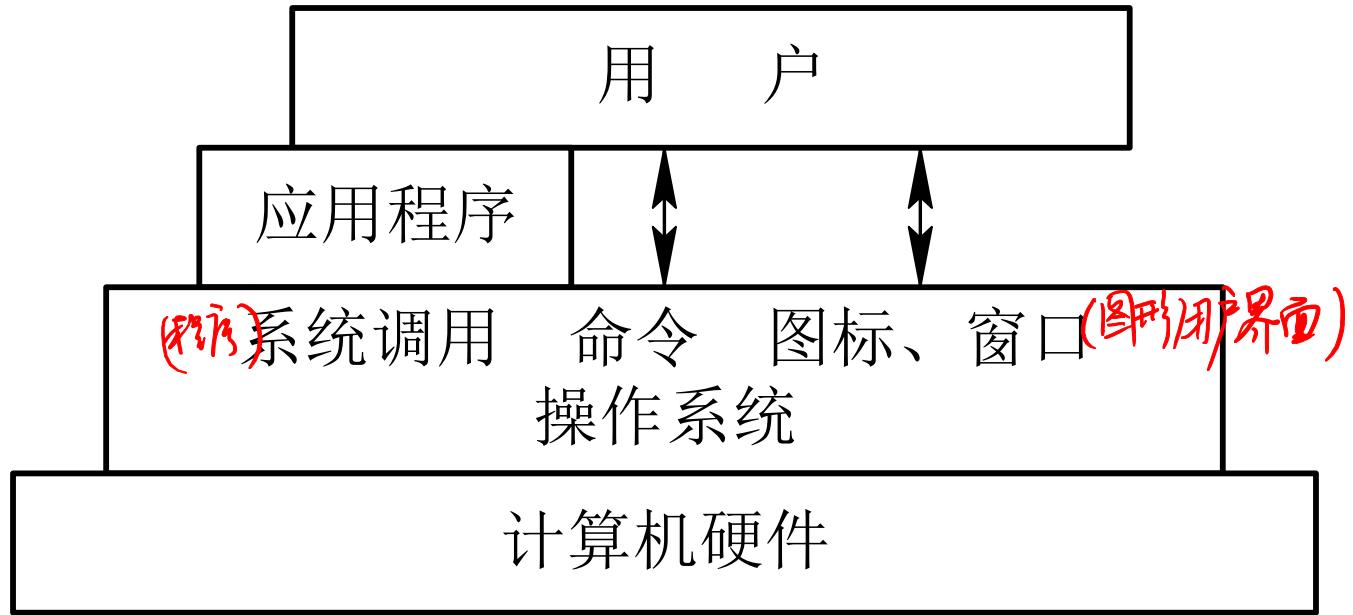


图 OS作为接口的示意图

- (1) 命令方式。这是指由OS提供了一组联机命令接口，以允许用户通过键盘输入有关命令来取得操作系统的服务，并控制用户程序的运行。
- (2) 系统调用方式。OS提供了一组系统调用，用户可在自己的应用程序中通过相应的系统调用，来实现与操作系统的通信，并取得它的服务。
- (3) 图形、窗口方式。这是当前使用最为方便、最为广泛的接口，它允许用户通过屏幕上的窗口和图标来实现与操作系统的通信，并取得它的服务。

操作系统的作用与功能

- 操作系统是计算机系统的**资源管理者**
 - 资源
 - 硬件资源：处理器、存储器、I/O设备等
 - 信息资源：程序+数据
 - 操作系统的任务：
 - 对资源进行抽象，找出各种资源的共性和个性，有序地管理计算机中的硬件、软件资源，跟踪资源使用情况，监视资源的状态，满足用户对资源的需求，协调各程序对资源的使用冲突 *eg. all、动态链接库*
 - 为用户提供简单、有效的资源使用手段，最大限度实现各类资源的共享，提高资源利用率

通用：
即插即用

操作系统的作用与功能

- 上 • 处理机管理/进程管理 期中前
- 下 • 存储管理
- 设备管理
- 文件管理
- ~~• 网络与通信管理~~

操作系统的作用与功能

- 处理机管理
 - 处理中断事件
 - 硬件发现并捕捉中断事件，产生中断信号，但不能处理；
 - 操作系统对中断事件进行处理
 - 处理器调度
 - 单用户单任务系统中，处理器为一个用户的一个任务独占
 - 多道程序或多用户情况下，处理器需要在多个任务和用户之间共享
 - 操作系统引入了**进程和线程**的概念，实现程序的并发执行。处理器的管理和调度最终归结为对进程和线程的管理和调度
 - 将在处理机管理和并发进程两章里详细介绍

操作系统的作用与功能

- 处理器调度的功能
 - 进程控制和管理 跟踪状态.
 - 进程同步和互斥
 - ~~- 进程通信 复制、粘贴.~~
 - ~~- 进程死锁~~
 - 线程控制和管理
 - 处理器调度

操作系统的作用与功能

- 存储管理
 - 目标：管理存储器资源，为多道程序运行提供有力的支撑，便于用户使用存储资源，提高存储空间的利用率。
 - 主要功能包括：
 - 存储分配 *物理空间给进程*
 - 地址转换与存储保护 *(又硬件) < 2. 不访问别人进程空间.*
 - 存储共享 *code+data+堆栈, 代码共享?*
 - 存储扩充
- 存储管理的机制与硬件存储器的组织结构和支撑设密切相关。
- 将在存储管理和虚拟内存两章里详细介绍

操作系统的作用与功能

- 设备管理
 - 目标：管理各类外围设备，完成用户提出的I/O请求，加快I/O信息的传送速度^{加快}，发挥I/O设备的并行性，提高I/O设备的利用率，提供每种设备的设备驱动程序和中断处理程序，为用户隐藏硬件细节，提供方便简单的设备使用方法。
 - 主要功能：
 - 提供外围设备的控制与中断处理
 - 提供缓冲区的管理
 - 提供设备独立性
 - 外围设备的分配和去配
 - 实现共享型外围设备的驱动调度
 - 实现虚拟设备
- 将在设备管理一章里详细介绍

操作系统的作用与功能

- 文件管理
 - 目标
 - 对用户文件和系统文件进行有效管理，实现按名存取；
 - 实现文件的共享、保护和保密，保证文件的安全性；
 - 并提供给用户一整套能方便使用文件的操作和命令。
 - 功能：
 - 提供文件逻辑组织方法
 - 提供文件物理组织方法
 - 提供文件存取和使用方法
 - 实现文件的目录管理
 - 实现文件的共享和存取控制
 - 实现文件的存储空间管理
- 将在文件管理一章里详细介绍

操作系统的作用与功能

- ~~网络与通信管理~~
 - 联网操作系统应具备的功能
 - 网上资源管理功能
 - 数据通信管理功能
 - 网络管理功能

操作系统的主要特性

- 并发性
- 共享性
- 异步性

操作系统的主 要特性

- 并发性
 - 并发性 (**concurrency**) 是指两个或两个以上的事件或活动在同一时间间隔内发生。
 - 操作系统的并发性指它应该具有处理和调度多个程序同时执行的能力。
 - 多个I/O设备同时在输入输出；
 - 设备I/O和CPU计算同时进行；
 - 内存中同时有多个系统和用户程序被启动交替、穿插地执行
 - 并发性使操作系统的[设计](#)和[实现](#)变得复杂化
 - 并发的物理含义
 - 单CPU系统中，实际上是若干道程序之间的空分多路复用
 - 宏观上并发，微观上顺序执行
 - 在多CPU系统中，程序的执行可以是并行的，即两个或两个以上事件或活动在同一时刻发生。并行性是并发性的特例。
- **Note:** 并发和并行的区别？

并发性引发的问题

- 如何从一个程序切换到另一个程序？
- 以什么样的策略选择下一个运行的程序？
- 如何实现各个运行程序的隔离？
- 如何协调多个运行程序对资源的竞争？

操作系统的主要特性

- **共享性**

- 共享指操作系统中的资源（包括硬件资源和信息资源）可被多个并发执行的进程共同使用，而不是被一个进程所独占。
- 资源共享的方式
 - 透明资源共享
 - 允许同一时间段内多个进程对资源进行访问，好像每个进程都独占资源
 - 访问的次序对结果无影响
 - 如**CPU**、主存、磁盘、打印机
 - 显式资源共享
 - 同一时间段内只允许一个进程访问资源，这类资源称为**临界资源**，如磁带机、扫描仪，以及某些数据和表格
 - 操作系统必须**提供显式资源共享机制**（申请/释放资源的系统调用，或锁机制），将资源的互斥访问下放给**用户决策**

操作系统的主要特性

- 共享性和并发性互为依存
 - 资源的共享是因为程序的并发执行而引起的
 - 对资源共享实施有效管理是提高程序并发执行效率的前提

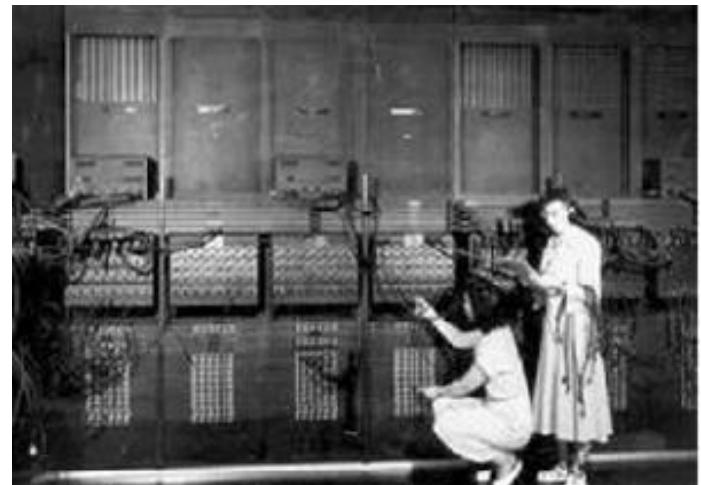
操作系统的主要特性

- 异步性

- 在多道程序环境中，允许多个进程并发执行，由于资源有限而进程众多，多数情况，进程的执行不是一貫到底，而是“走走停停”。 执行环境开放
- 异步性给系统带来了潜在的危险，有可能导致进程产生与时间有关的错误，但只要运行环境相同，操作系统必须保证多次运行进程，都会获得完全相同的结果。
- 异步性的例子
 - 作业到达系统的时间和类型
 - 操作员发出命令或操作的时间和类型
 - 程序运行发生错误或异常的类型和时刻
 - 中断事件发生的时刻

操作系统的形成和发展

- 人工操作阶段
 - 手工方式直接控制、使用计算机硬件
 - 使用机器语言编程
 - 将准备好的程序和数据穿孔在纸带或卡片上
 - 从纸带或卡片输入机将程序和数据输入计算机
 - 通过控制台的按钮、开关和氖灯来操纵和控制程序
- 缺点
 - 用户独占资源
 - 人工干预多
 - 计算时间拉长



操作系统的形成和发展

- 管理程序阶段/简单批处理
 - 自动控制和处理作业流
 - 当一个作业完成时，自动返回到管理程序，选择下一个作业载入运行
 - 管理程序包括
 - 一组控制命令和解释器
 - 设备驱动和I/O控制功能
 - 库程序和程序装配功能
 - 简单的文件管理功能

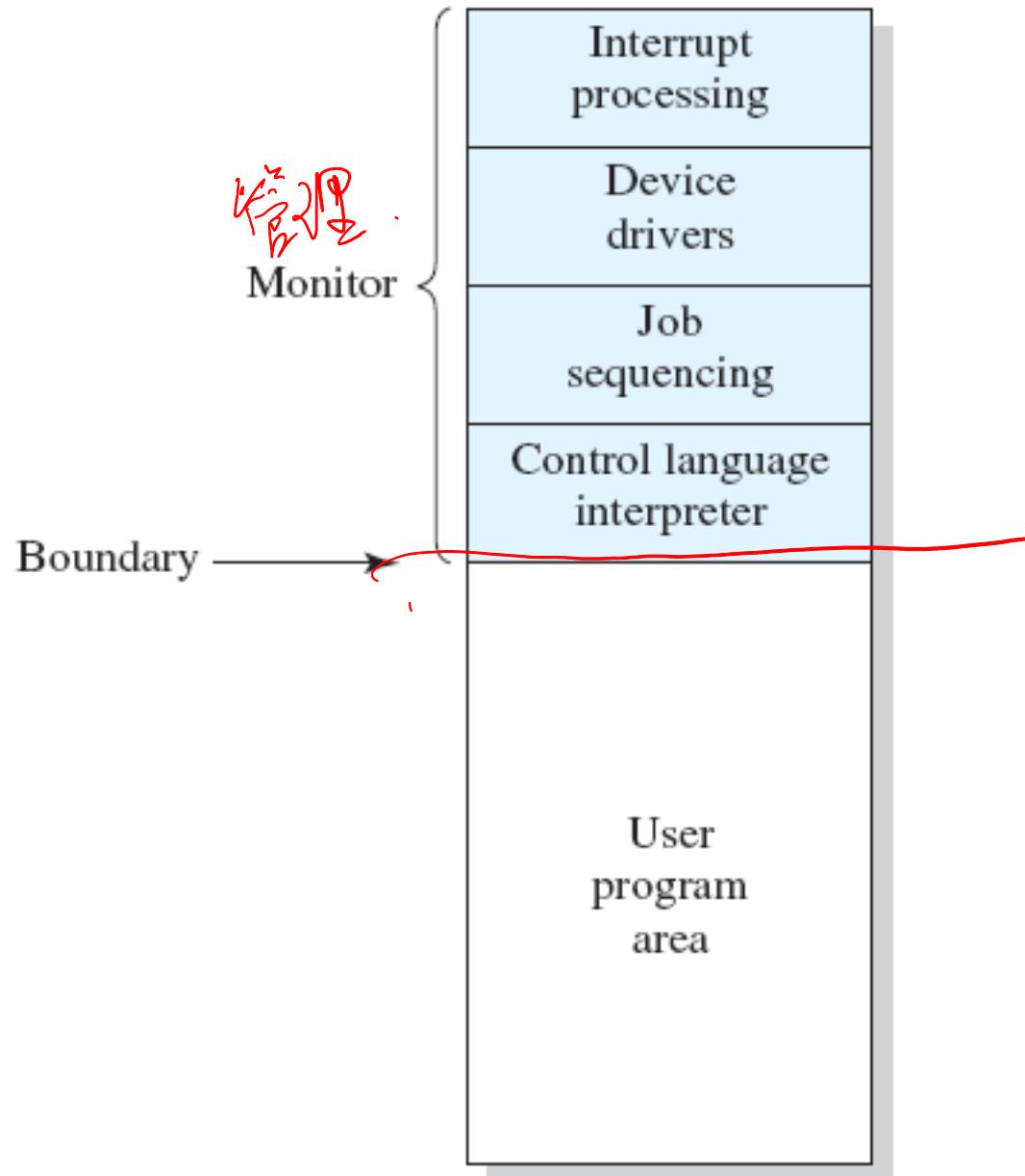
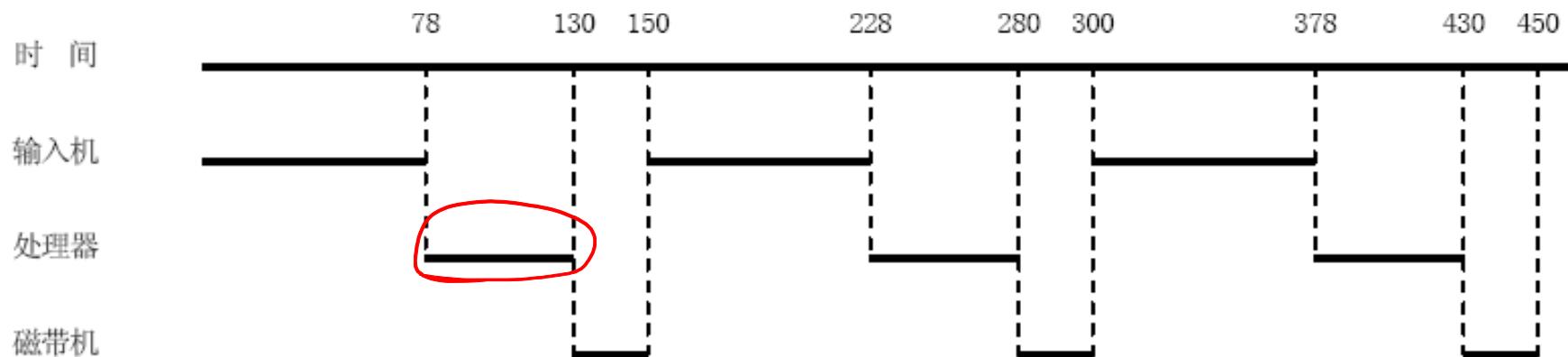


Figure 2.3 Memory Layout for a Resident Monitor

多道程序设计与操作系统的形成

- 早期的单道批处理系统中，内存中仅有单个作业在运行，设备利用率低，系统性能差。
 - CPU, 外设无法同时工作



单道程序设计运行时CPU效率

$$52/150 \approx 35\%$$

多道程序设计

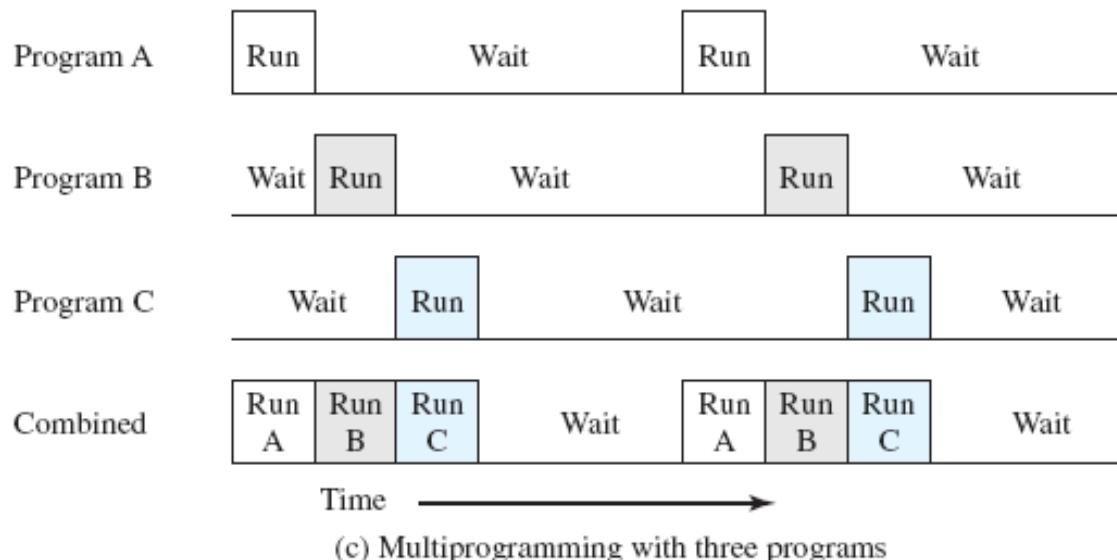
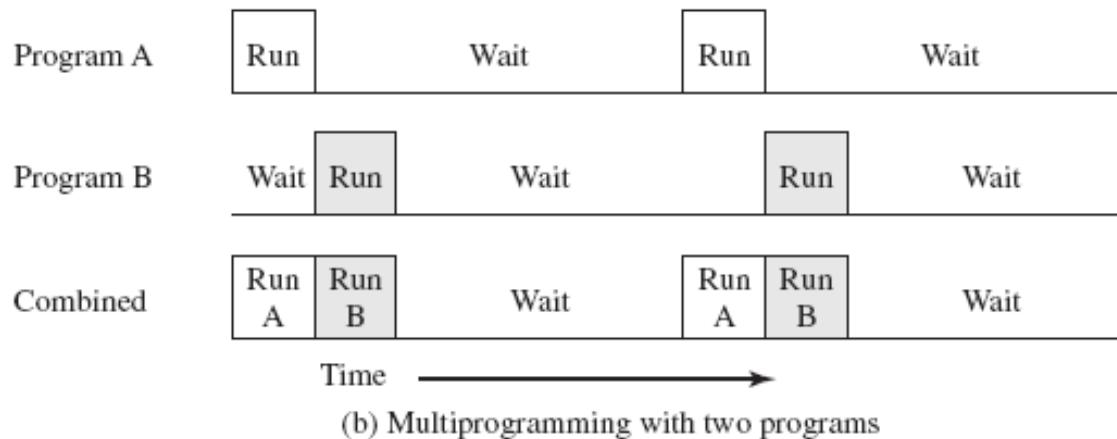
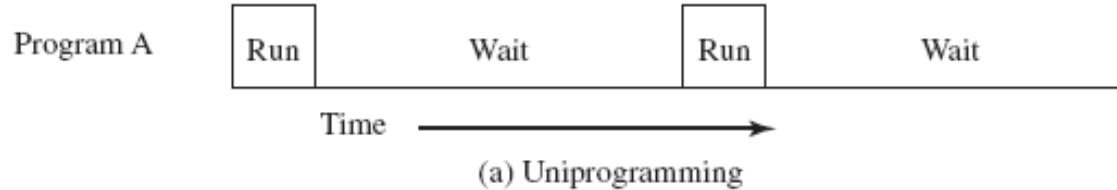
- 中断和通道技术的产生使CPU和I/O设备并行工作成为可能，为多道程序奠定了基础。
- 多道程序设计是指允许多个程序（作业）同时进入一个计算机系统的内存储器并启动进行交替计算的方法。
 - 内存中同时存放了两个以上的程序，它们均处于开始和结束点之间
 - 各程序轮流占用CPU，交替执行



单道程序设计



多道程序设计

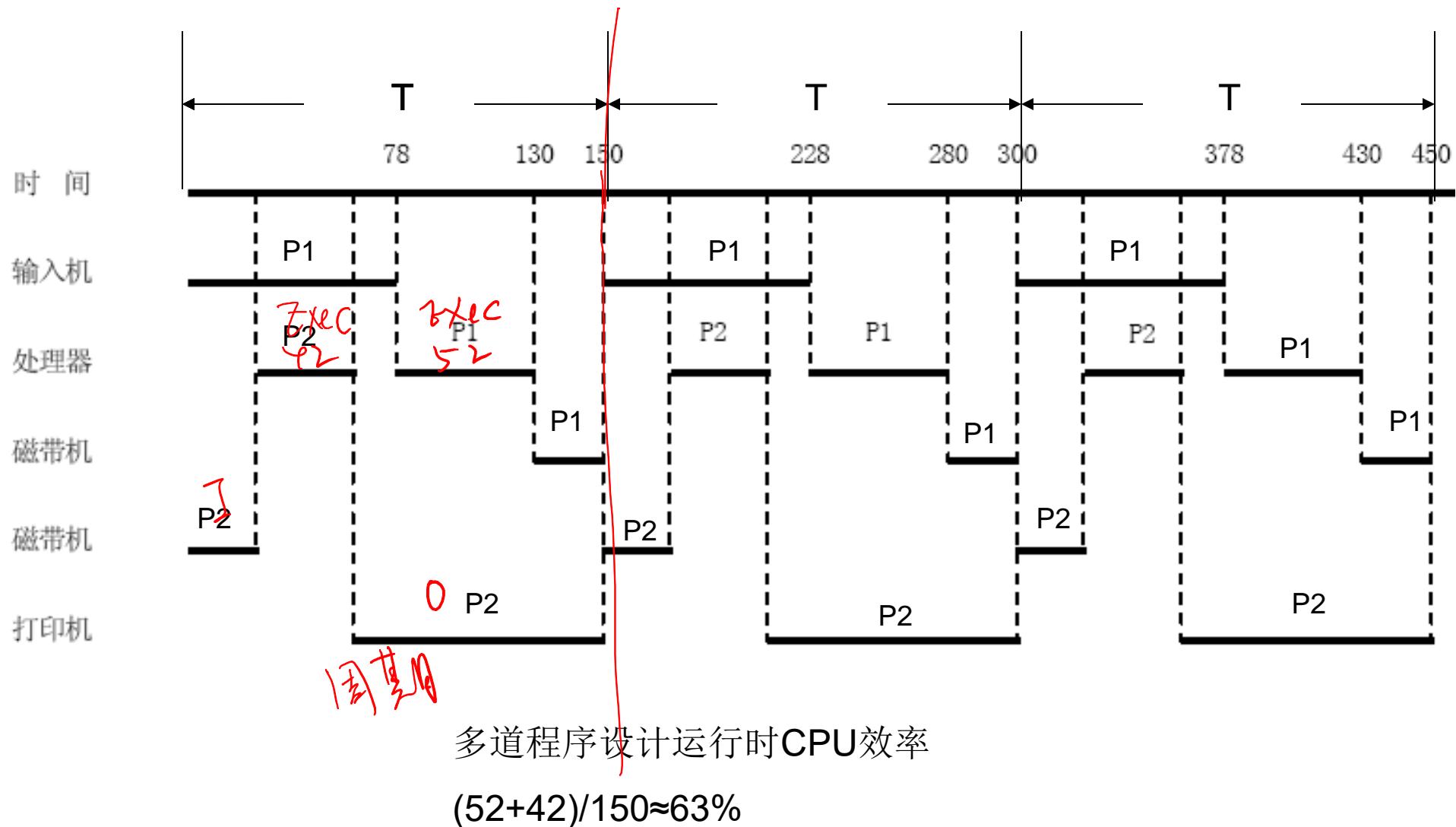


- P1

- a) 输入机输入500个字符(输入机速度6400字符/s), 约78ms
- b) 处理数据, 52ms
- c) 将结果2000个字符存储到磁带机1上(磁带机速度 10^5 字符/s), 20ms

- P2

- a) 从磁带机2上输入2000个字符, 20ms
- b) 处理数据, 42ms
- c) 行式打印机输出两行 (1350行/min), 约88ms



多道程序设计

- 多道程序设计的道数指同时进入内存参与CPU竞争的程序数量
- 假设内存中有n道程序，程序平均等待I/O操作的时间比例为p,且等待操作相互独立，则所有程序都等待I/O操作的概率是 p^n ,从而,

$$\text{CPU利用率} = 1 - p^n \quad \text{同时等待 I/O 操作}$$

- Q: 若进程平均花费80%的时间等待I/O操作，则为了使得CPU利用率不低于80%，应至少有多少道程序在主存中运行？

$$1 - 0.8^n \geq 0.8$$

多道程序设计

- 优点
 - 提高了CPU、内存和I/O设备的利用率
 - 改进了系统的吞吐率
 - 充分发挥了系统的并行性
 - 缺点
 - 延长了作业的周转时间
- 靠软件实现？*
- ↓ 开始运行 → 结束运行 ← 总成竞争*
- ⇒ 总体性能↑*

多道程序设计需要解决的问题

- 存储保护与程序浮动(地址重定向)
 - 多道程序之间避免相互干扰，一道程序的错误不影响其他程序
 - 程序或程序的部分应能从某个内存区域移动到另一区域
- 处理器的管理和分配
 - 一个程序可以处于运行、等待或就绪三个状态；程序可以在三个状态之间切换
 - 合理搭配具有不同特性的多道程序同时运行，如I/O密集型和CPU密集型任务的合理搭配
- 系统资源的管理和调度
 - 资源为多道程序所共享，需要解决好资源的竞争与协作、共享与安全

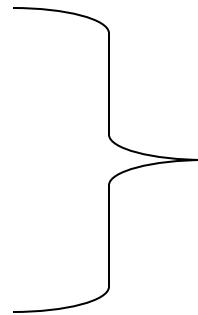
e.g. 磁盘

操作系统发展的动力

- 硬件技术的快速发展
 - 微电子技术是推动计算机技术发展的引擎
 - 8位机->16位机->32位机->64位机
 - 计算机体系结构不断发展
 - 界寄存器→分页或分段设施
 - 中断和通道设施导致多道程序设计的出现
- 提高计算机系统资源利用率的需求
 - 促使各种调度算法、分配策略被研究和采纳
- 新的应用需求不断促使操作系统变革
 - 让用户使用计算机越来越方便的需要
 - 满足用户新的要求，提供用户新服务
 - 实时操作系统

操作系统的类型

- 批处理操作系统
- 分时操作系统
- 实时操作系统
- 并行操作系统
- 网络操作系统
- 分布式操作系统
- 嵌入式操作系统



三种基本类型

批处理

- 批处理系统的特征
 - 用户脱机工作
 - 作业控制语言JCL
 - 程序+数据+作业说明书
 - 用户无法在程序执行过程中对其进行控制 提交任务后不可交互
 - 成批处理作业
 - 操作员集中一批作业并对作业进行预输入
 - 操作系统调度和控制用户作业的执行
 - 单/多道程序运行
 - 单道批处理系统
 - 多道批处理系统
- 优缺点
 - 优点: 系统资源利用率高, 作业吞吐量大
 - 缺点: 作业周转时间长, 不具备交互式能力, 不利于开发和调试

调试机器去运行

单用户
多任务系统

分时操作系统

快速响应

- 多个联机用户通过终端（键盘/显示器）同时直接使用一个计算机系统进行交互式计算
 - 用户在各自终端上进行会话
 - 程序、数据和命令在会话过程中提供，以问答方式控制程序的运行
 - 时间片轮转^{时分复用}将处理器分配给各个终端
- 分时操作系统的特性
 - 同时性
 - 独立性
 - 及时性
 - **交互性**

分时OS和批处理OS的区别

- 目标不同
 - 批处理OS以提高资源利用率和系统吞吐量为目标
 - 分时OS强调公平性和响应时间
- 适应作业的性质不同
 - 批处理OS适应已调试好的大型程序
 - 分时OS适应正在调试的小程序
- 资源使用率不同
 - 批处理OS能最大程度优化系统资源利用率
 - 分时OS能较公平地调配CPU和主存资源

效率↑
公平↑
- 作业控制方式不同
 - 批处理OS通过JCL书写作业控制流
 - 分时OS通过键盘输入以交互式方式控制程序的运行

实时操作系统

- 当外部事件或数据产生时，能对其予以接收并以足够快的速度进行处理，所得结果能够在规定的时间内控制生产过程或对控制对象作出快速响应，并控制所有实时任务协调运行
 - 事件驱动
 - 及时响应
 - 高可靠性
- 实时操作系统的分类
 - 过程控制系统
 - 信息查询系统
 - 事务处理系统
- 过程控制系统的处理步骤
 - 数据采集
 - 加工处理
 - 操作控制
 - 反馈处理

e.g 导弹防御系统. $dd\leq t_{\text{前}}$.

通用操作系统

- 若某个操作系统同时兼具批处理、分时和实时处理的全部或两种功能，则称为通用操作系统

操作系统的服务

- 提供给程序和用户的共性服务包括:
- 创建/执行程序 打开可执行(创建堆栈)
- I/O设备访问
- 信息存取 (文件系统)
- 通信服务 在系统里运行的两个程序、地址空间? {进程间通信 (eg. 复制粘贴)
- 错误检测和处理 避免访问非法地址空间
机器间一网络
- 为保证自身效率和正确性提供的功能
- 资源分配
- 统计
- 保护

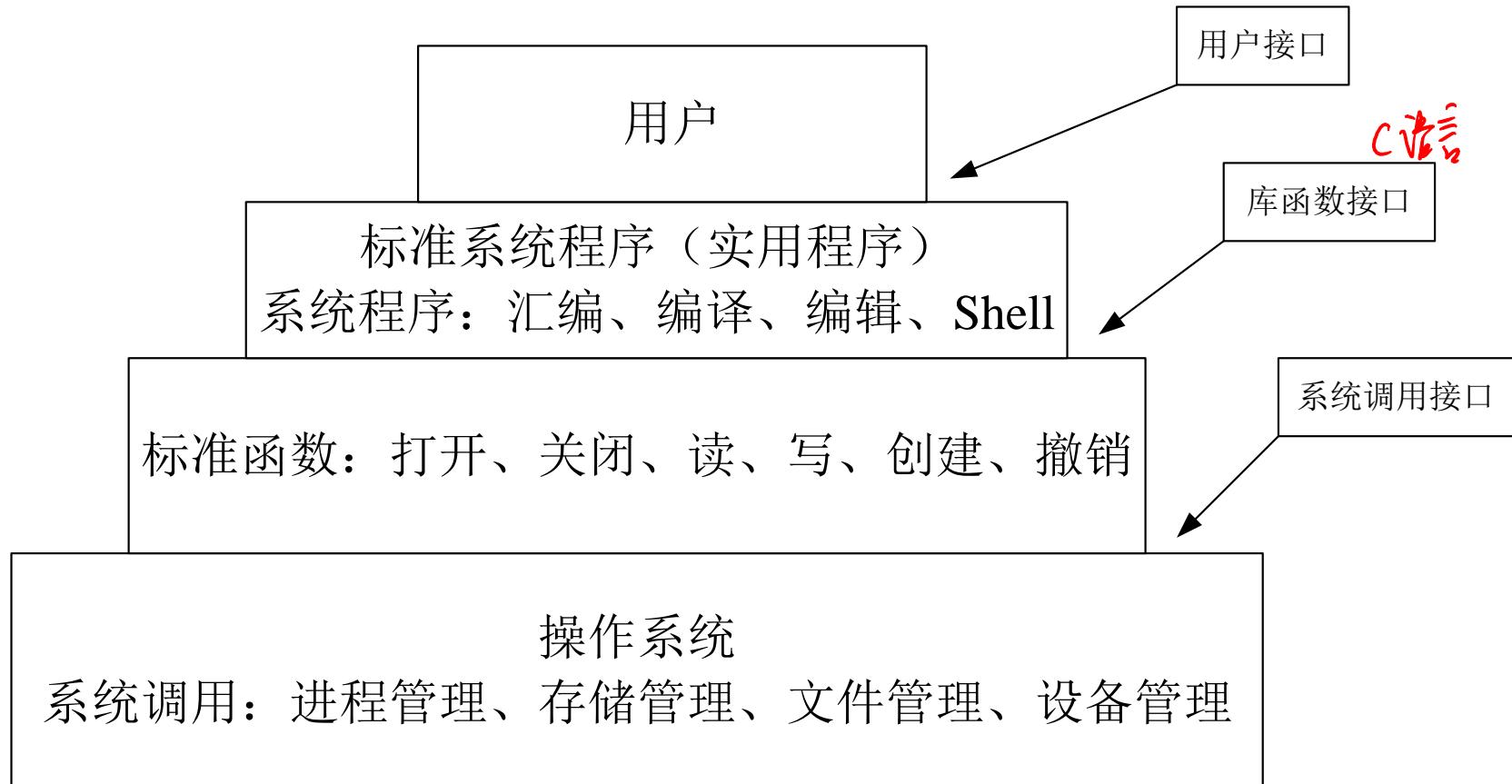
操作系统提供的用户接口

X会和OS类型绑定(不可移植)

- 程序接口
 - 可被应用程序直接使用 编程时使用
 - 由一组系统调用组成 e.g. C语言API(已封装过系统调用)
- 操作接口
 - 通过实用程序提供
 - 用户借助命令管理或shell来请求执行
 - CMD (Windows)
 - (Linux)

系统调用

- 系统调用是为了扩充机器功能、增强系统能力、方便用户使用而建立的。
- 系统调用是用户程序或其他系统程序获得操作系统服务的唯一途径。
- 操作系统内核的主体是系统调用的集合，可以将内核看成特殊的公共子程序
- 应用程序和系统调用分别在用户空间和内核空间运行，在逻辑上相互隔离，起到了保护系统的作用



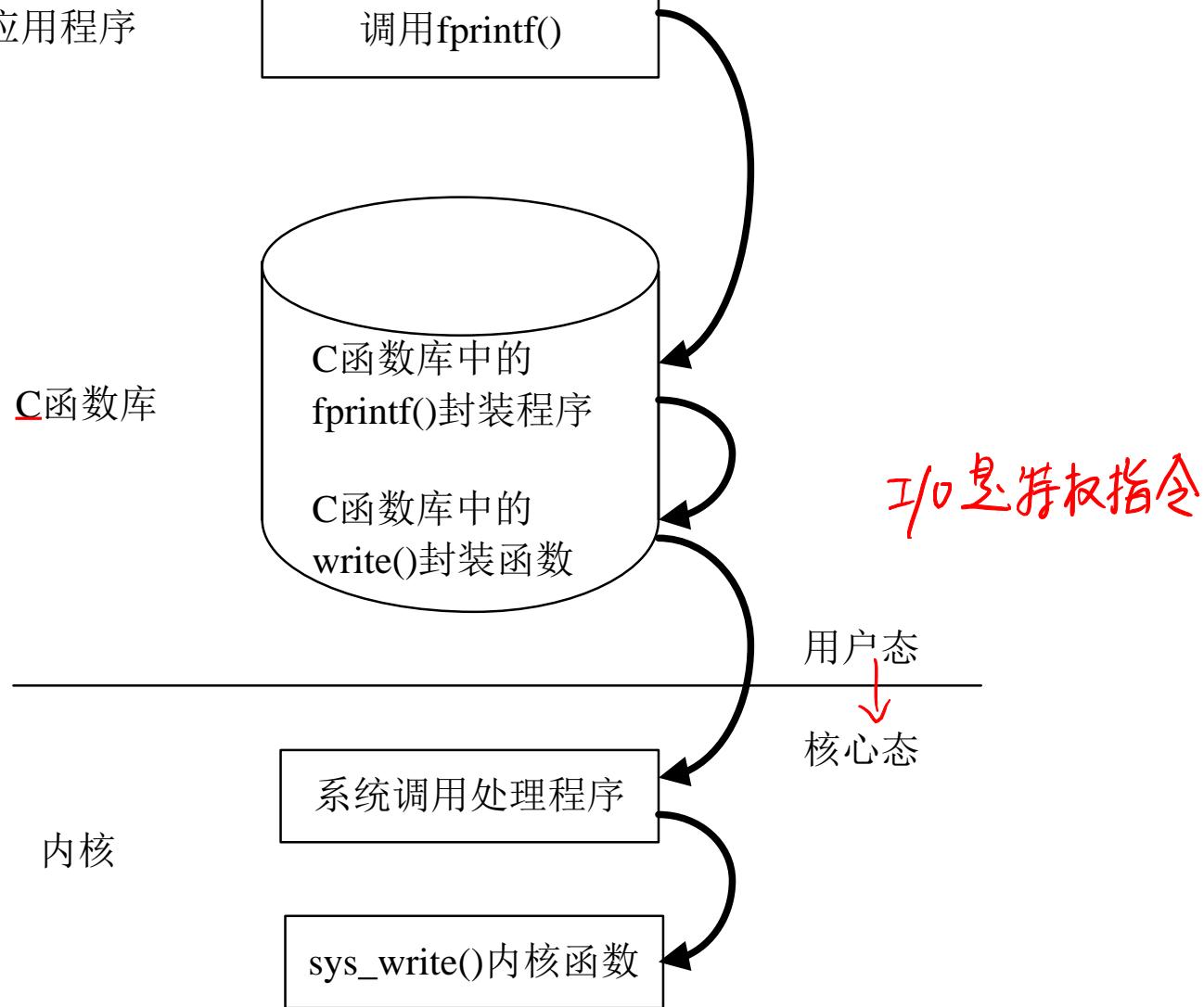
UNIX/Linux操作系统上系统程序、库函数和系统调用的关系

API和系统调用

- 不同操作系统提供的系统调用功能类似，但实现细节不尽相同
- 系统调用属低层接口，接口复杂，跨平台移植困难
- API对系统调用及其它复杂功能进行封装，提供用户所需的服务，标准化，易于使用，易于移植
 - POSIX API标准
 - Win32 API
 - Java API
- API为函数定义，仅说明获得给定服务的方式，不规定过程的实现
- 一个API的实现可能会用到一个或多个系统调用，也可能完全不使用系统调用；多个API可能封装同一个系统调用。
- 调用API在用户态，如果API使用了系统调用，则在执行过程中切换到核心态

应用

OS



UNIX/Linux系统中应用程序执行API fprintf()的过程

系统调用的分类

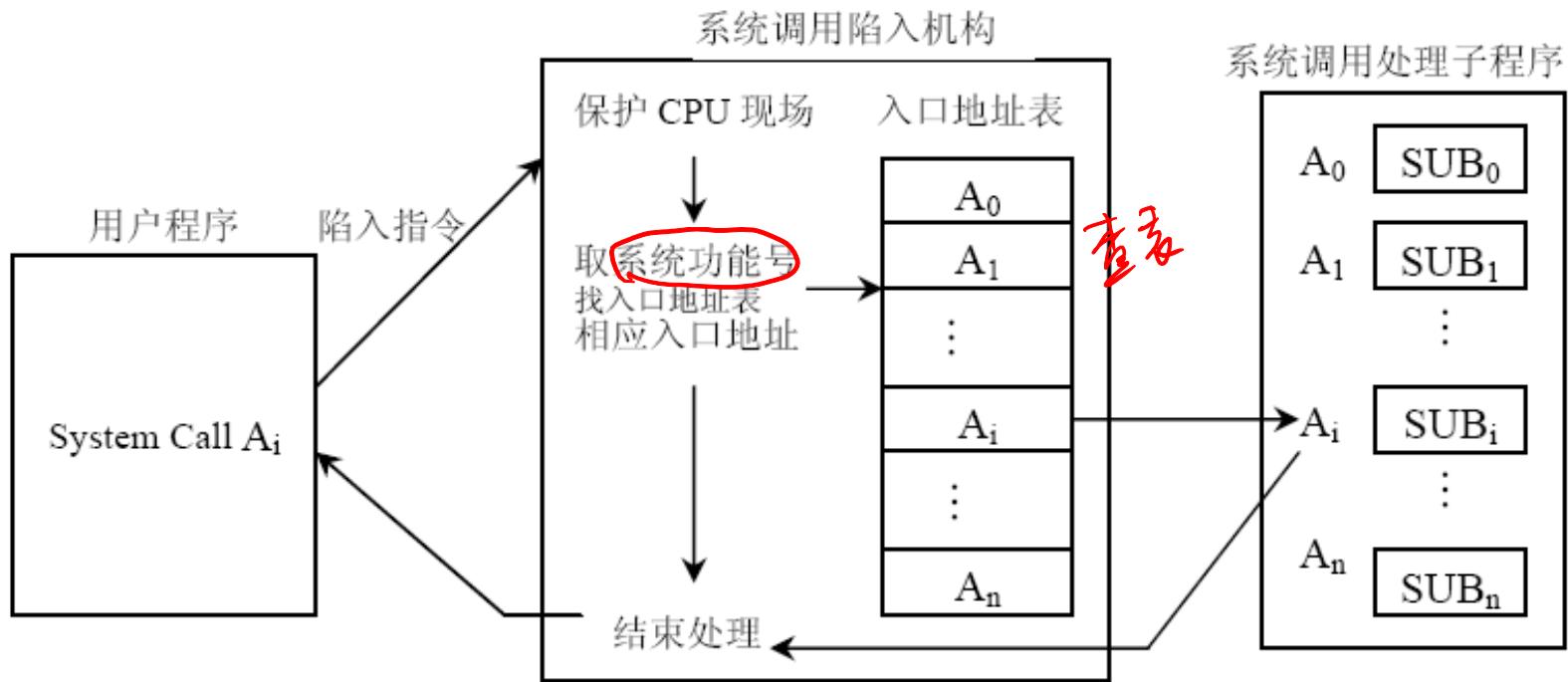
- 进程和作业管理
- 文件操作
- 设备管理
- 内存管理
- 信息维护
- 进程通信

系统调用的实现

- 实现系统调用功能的机制称为**陷入(trap)机制**
- 系统调用将引起处理器中断，相应的机器指令称**访管指令(如INT)**。**软中断**
- 每个系统调用都事先规定了编号，称功能号，在陷入指令中必须指明对应系统调用的功能号

系统调用的实现

- 实现系统调用的步骤
 - 编写系统调用处理程序 ① → 内存地址
 - 设计一张系统调用入口地址表，每个入口地址都指向一个系统调用的处理程序，有些还包含系统调用自带参数的个数
 - 陷入处理机制，需开辟现场保护区，保存发生系统调用时的处理器现场



陷入机构和系统调用处理过程

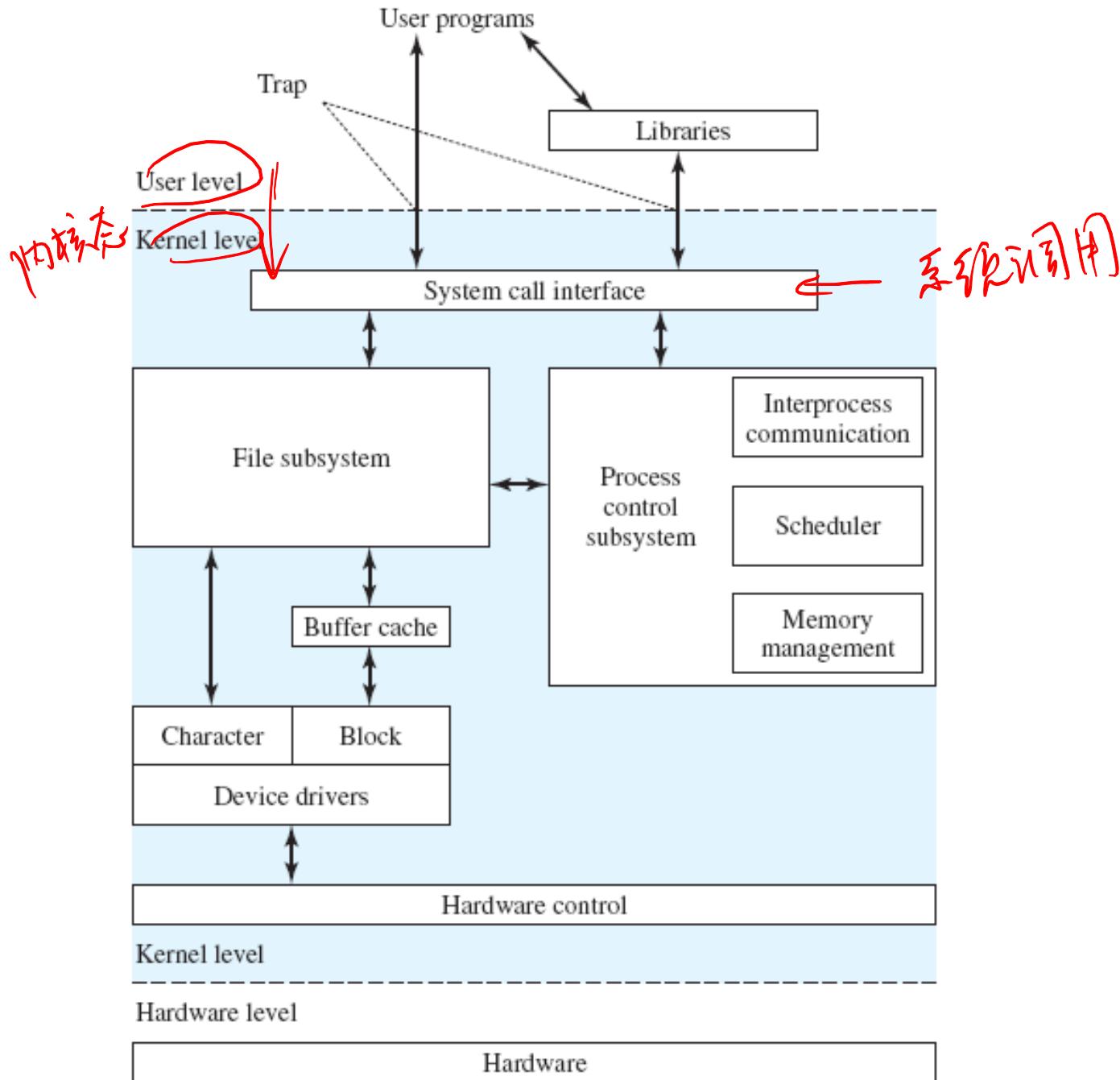


Figure 2.15 Traditional UNIX Kernel

系统调用处理示例

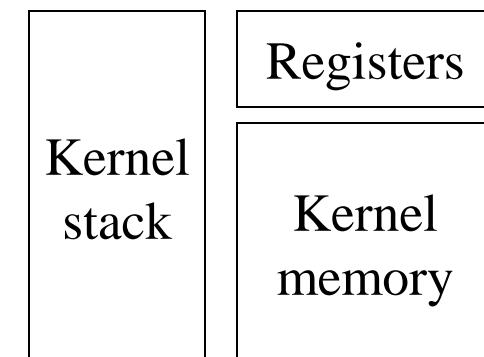
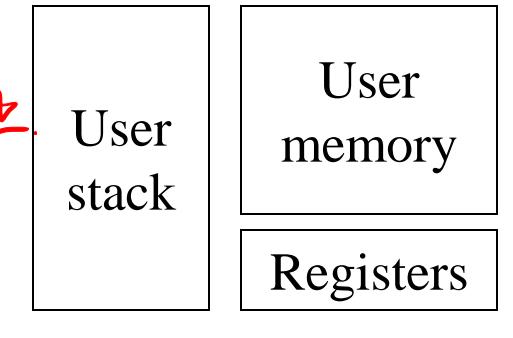
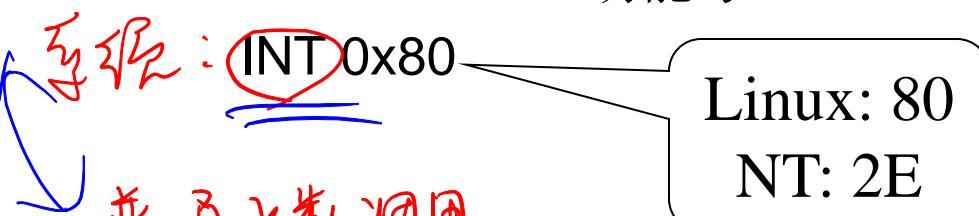
- 文件描述符
- ↓
- 系统调用 `read(fd, buf, size)` :
 - 大致步骤:

Move fd ->R1

Move buf ->R2

Move size ->R3

Move READ (功能号) ->R0



系统调用的参数传递方法

- 陷入指令自带参数
 - 直接参数：规定指令之后若干单元存放参数
 - 间接参数：指令之后紧靠单元存放参数地址，指出参数的存放区
- CPU通用寄存器传递 *Mov AL, 0AH.*
 - 将参数值直接放入通用寄存器，仅适用少量参数传递
 - 在内存中的一个区或表中存放参数，将其首地址送入寄存器
- 堆栈传递

系统调用与普通函数调用的区别

- 调用形式不同
 - 普通函数调用使用一般调用指令，其转向地址是固定不变的，包含在跳转语句中 *JMP*
 - 系统调用中不包含处理程序入口，仅提供功能号，**按功能号调用。** *INT*
- 被调用代码位置不同
 - 普通函数调用是一种静态调用，调用者和被调用者在同一程序内，经过编译连接后作为目标代码的一部分，当函数被修改后，需要重新编译连接。
 - 系统调用是一种动态调用，系统调用的处理代码在调用程序之外（在操作系统中不需要 *内核中* **新编译** *修改* **不变**）
- 提供方式不同
 - 普通函数往往由编译系统提供，不同编译系统提供的函数可以不同
 - 系统调用由操作系统提供，**给定的操作系统上系统调用的功能、种类和数量固定**
- 调用的实现不同
 - 普通函数调用一般使用跳转机器指令(**JUMP**)来实现调用，在用户态运行
 - 系统调用是通过陷入机制实现，需要从**用户态转变为核心态**

中断与陷入关系

被动

主动

- 中断由与现行指令无关的中断信号触发，通常在两条机器指令之间才可以响应中断，一般来说，中断处理程序提供的服务不是为当前进程所需的；
相同
- 陷入则是由处理器正在执行现行指令而引起的，通常，陷入处理程序提供的服务是为当前进程所用的。
- 中断与陷入采用的实现技术基本一样，都需要进行处理器模式切换，保护CPU运行现场，然后转向中断/系统服务例程，待服务结束后恢复CPU现场返回被中断程序

操作接口

- 作业控制方式
 - 联机用户接口
 - 命令行
 - Command arg1 arg2 ... argn
 - 由命令解释器执行
 - 批命令
 - BAT, Shell文件 *← 放批命令让它一起执行*
 - 图形化用户界面
 - 脱机用户接口 *(批处理)*
 - 专为批处理作业用户提供的
 - 作业控制语言JCL

Linux
Shell

命令解释器

- 命令解释器的种类
 - Shell: Bourne shell, C shell, Bourne-Again shell, Korn shell.
 - MS-DOS
- 命令解释器的两种实现方式
 - 解释器自身包含执行命令的代码
 - 能支持的命令数决定了命令解释器的大小，可扩展性差
 - 命令解释器并不理解命令，而只是利用命令定位实现该命令的系统程序，将其载入内存并加以执行
 - 程序员很容易添加新命令 *从PATH里找*
 - 命令解释器可以很小，并无需改变

操作系统的结构设计

- 操作系统的设计呈现下述特征
 - 复杂程度高
 - 生成周期长
 - 正确性难保证
- 操作系统结构设计的内涵
 - 整体结构，包括功能分块、模块交互
 - 局部结构，包括数据结构和控制结构
 - 运行时组织，如采用进程还是线程，在系统空间还是用户空间运行

操作系统的构件

- 内核
- 进程
- 线程
- 管程
- 类程

内核

- 什么是内核?
 - 内核是操作系统对裸机的第一次改造，为进程的并发执行提供了良好的运行环境
 - 内核是一组可信的程序模块
 - 内核运行于核心态
 - 内核具有访问硬件设备和所有主存空间的权限
 - 内核是仅有的能够执行特权指令的程序

内核的分类

- 单内核
 - 整体式结构
 - 层次结构
- 微内核
- 虚拟机

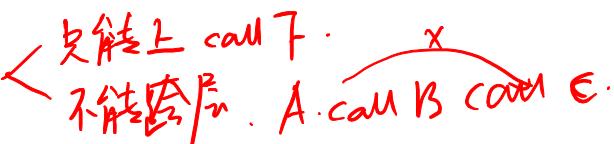
单内核

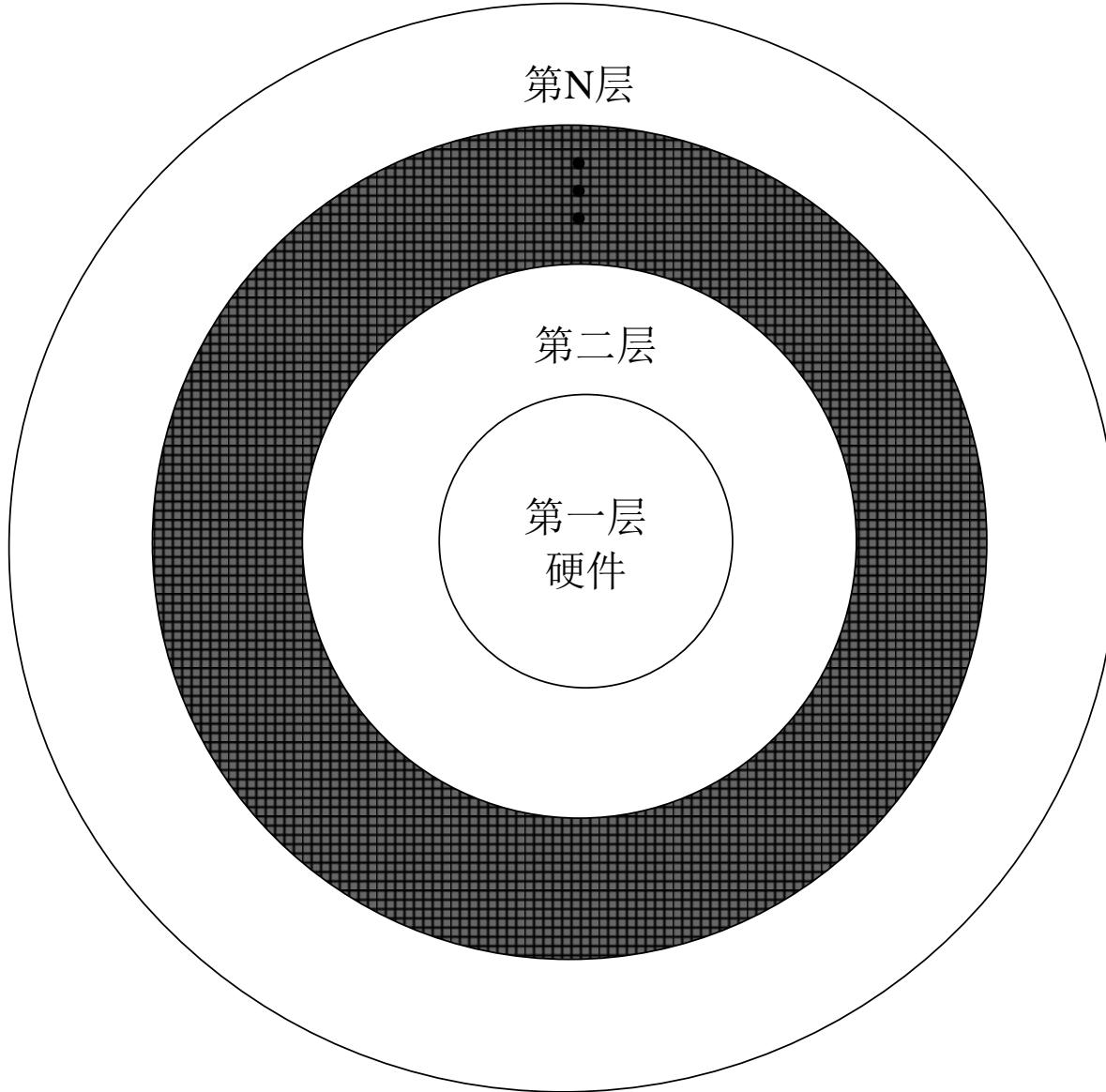
- 运行时是一个大**二进制映像**
- 模块之间的交互通过直接调用其他模块中的函数来实现
- Unix和Linux都是单内核结构，但Linux引入了动态加载模块和卸载模块机制

整体式结构

- 整体式结构的特性 没有解耦合
 - 模块是操作系统的基本单元
 - 按照功能将系统分解为若干具有一定独立功能的模块
 - 模块间可不加控制地自由调用 没有层次(网状)
 - 各模块分别设计、编码、调试，最后将所有模块连接成一个整体
- 优点
 - 模块独立性差，调用关系复杂
 - 系统结构不清晰，正确性难以保证
- 缺点
 - 结构紧密，组合方便
 - 系统效率较高

层次式结构

- 层次式结构的特点
 - 模块按照功能的调用次序排列成若干层次
 - 各层次之间是单向调用关系 
- 优点
 - 接口少而简单
 - 下层模块的正确性为上层模块的正确性提供了基础，使系统的正确性大大提高
 - 增加、修改或替换一个层次不会影响其它层次
- 缺点
 - 合理地定义不同层次的功能较为困难
 - 系统的效率会降低。每一层都可能需要修改参数，传递数据。每一层都可能增加系统调用的开销。



层次化操作系统结构示意

微内核

- 操作系统分为两部分
 - 核心态的内核 *核心放在kernel中*
 - 微内核应该包括哪些服务并没有确切的共识，通常，至少包含进程管理及调度，消息传递和设备驱动，I/O和中断管理，存储管理。
 - 用户态进程 *不需要*
 - 将所有非必要的组件从内核移出，作为系统或用户程序出现，如文件管理服务，统计服务等。
- 不同用户空间的进程之间的通信由消息传递实现。但是，这些进程并不直接通信，而是通过微内核间接通信。*函数调用：压栈*
用户进程空间 copy 内核 copy 用户进程空间
- 优点：
 - 提供一致性接口
 - 可扩充性和易修改性强
 - 可移植性好
 - 对分布式系统提供有力的支撑
- 缺点
 - 效率低，进程之间必须通过内核的通信机制才能进行通信

内核的主要功能

- 资源抽象
- 资源分配
- 资源共享

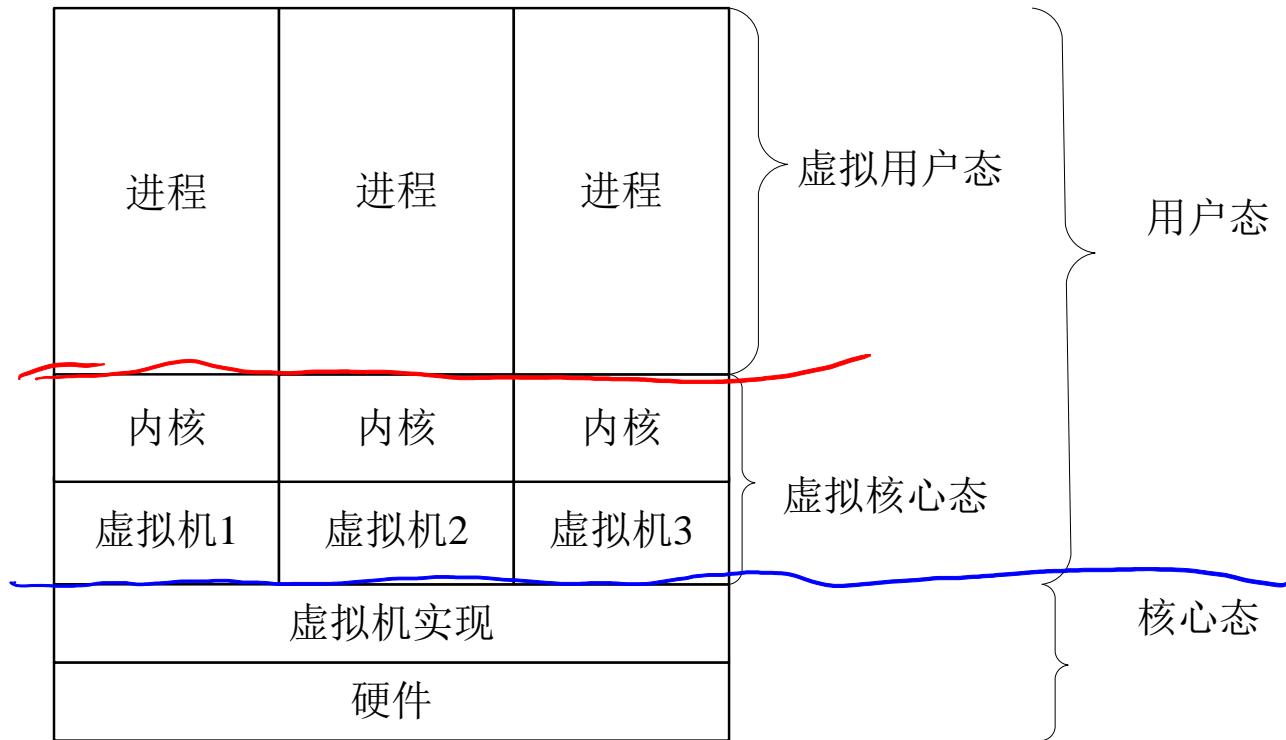
内核执行的属性

- 中断驱动
- 不可抢占
- 在屏蔽中断状态下执行
- 可使用特权指令

虚拟机

物理 . 虚拟

- 将单台计算机的硬件(CPU, 内存, 磁盘驱动, 网卡等) 抽象成多个不同的执行环境, 从而创造每个不同的执行环境都运行在自己的私有计算机上的幻觉。
- 虚拟机仅提供与底层裸硬件等同的接口
- 虚拟机软件可以运行在核心态, 由于它提供操作系统的功能。
- 而虚拟机自身则只能在用户态运行。
- 虚拟机的运行引入了虚拟用户态和虚拟核心态



(a) 非虚拟机结构

(b) 虚拟机结构

操作系统的运行模型

？上不通过进程组织

① 独立运行的内核模型

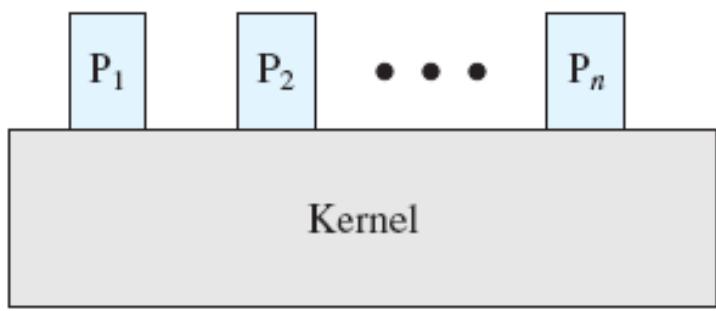
- 进程仅针对应用程序而言
- 操作系统作为独立实体运行在内核模式
- 内核有独立的核心栈，控制函数的调用和返回
- 内核函数难以并发执行

② OS功能在用户进程内执行的模型

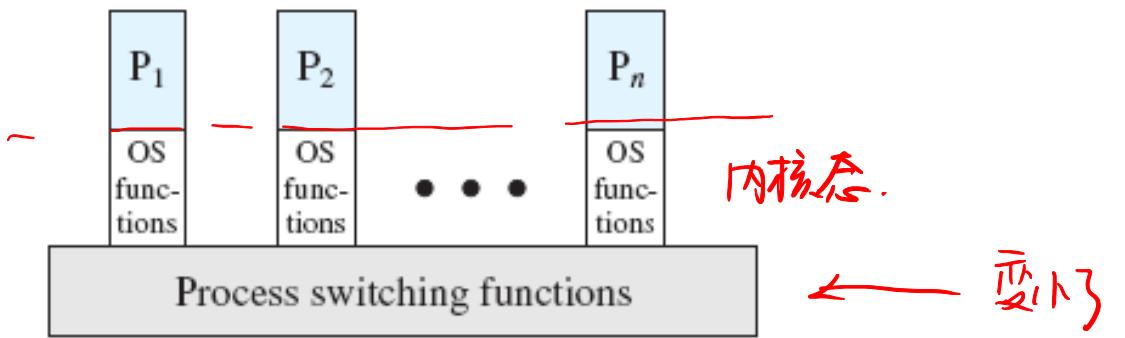
- 创建应用进程时，为其分配一个核心栈，用于运行操作系统的内核函数 依附在用户进程内实现
- 应用程序调用操作系统的功能时进行模式切换

③ OS功能作为独立进程执行的模型

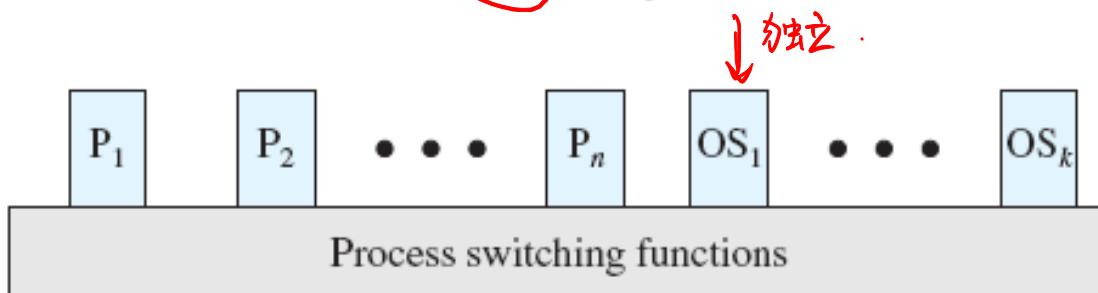
- 操作系统组织成一组系统进程
- 操作系统的小部分功能在核心态运行，大部分功能以独立进程的方式在用户态运行
- 有利于系统性能的提升 并发 ✓



(a) Separate kernel



(b) OS functions execute within user processes



(c) OS functions execute as separate processes

Figure 3.15 Relationship between Operating System and User Processes

流行操作系统简介

- DOS操作系统
- Windows操作系统
- UNIX操作系统
- Linux操作系统 开源 (Android in 核心)
- IBM系列操作系统

DOS

- DOS：单用户单任务的微机操作系统
- DOS的历史：始于1981年的1.0版，至1994年的最后版本DOS 6.22
- DOS的主要功能有：命令处理、文件管理、设备管理、及简单的CPU调度和内存管理
- DOS的主要特点：系统开销小，运行效率高，适用于微型机
- DOS的主要缺点：无法发挥硬件能力，缺乏对数据库、网络通信的支持，没有通用的应用程序接口，用户界面不友善

Windows操作系统

- Xerox公司的图形用户接口(GUI)系统
- Apple公司的Lisa和Macintosh
- 85年推出Windows 1.0,87年推出Windows 2.0
- 90年推出Windows 3.0,92年推出Windows 3.1
- 93年推出Windows NT 3.1,94年3.5,96年4.0
- 95年推出Windows 95 ,98年推出Windows 98
- 2000年推出Windows 2000
- 2000年推出Windows Me
- Windows XP/2003
- Windows Vista
- Windows 7

Windows早期版本的技术特点

- 依赖于DOS操作系统
- 友好、直观、高效的面向对象的图形用户界面
- 丰富的与设备无关的图形操作
- 多任务的操作环境
- 实现了虚存管理，突破640KB限制
- 提供各种系统管理工具
- 允许装入和运行DOS下开发的程序
- 提供数据库接口、网络通信接口
- 丰富的软件开发工具
- 面向对象的程序设计思想

Windows95/98的技术特点

- 独立的32位操作系统，同时能运行16位程序
- 真正的多任务操作系统，在32位的方式下具有抢先多任务能力
- 即插即用和电源管理
- 具有内置的网络功能，直接支持联网和网络通信，并提供对Internet的访问工具
- 新的图形界面，更加高级的多媒体支持
- 支持FAT32文件系统

Windows NT 的技术特点

- 支持对称多处理和多线程
- 支持抢先的可重入多任务处理
- 32位页式授权虚拟存储管理
- 支持多种API，提供源码级兼容性
- 支持多种可装卸文件系统
- 具有各种容错功能，C2安全级
- 可移植性好
- 集成网络计算
- 能与Microsoft SQL Server结合，提供C/S数据
库应用系统的最好组合

Unix

- 美国电报电话公司的贝尔实验室于1969年在DEC公司的小型系列机PDP-7上开发成功
- 73年开发出C语言并改写Unix，从而使得Unix具有高度易读性、可移植性，为迅速推广和普及走出了决定性的一步
- 74年7月，“ Unix分时系统”一文在美国权威杂志CACM上发表，引起了广泛注意
- 75年发布的Unix第6版是最早可获得的Unix
- 78年的Unix第7版，可以看作当今Unix的祖先，该版为Unix走进商界奠定了基础。

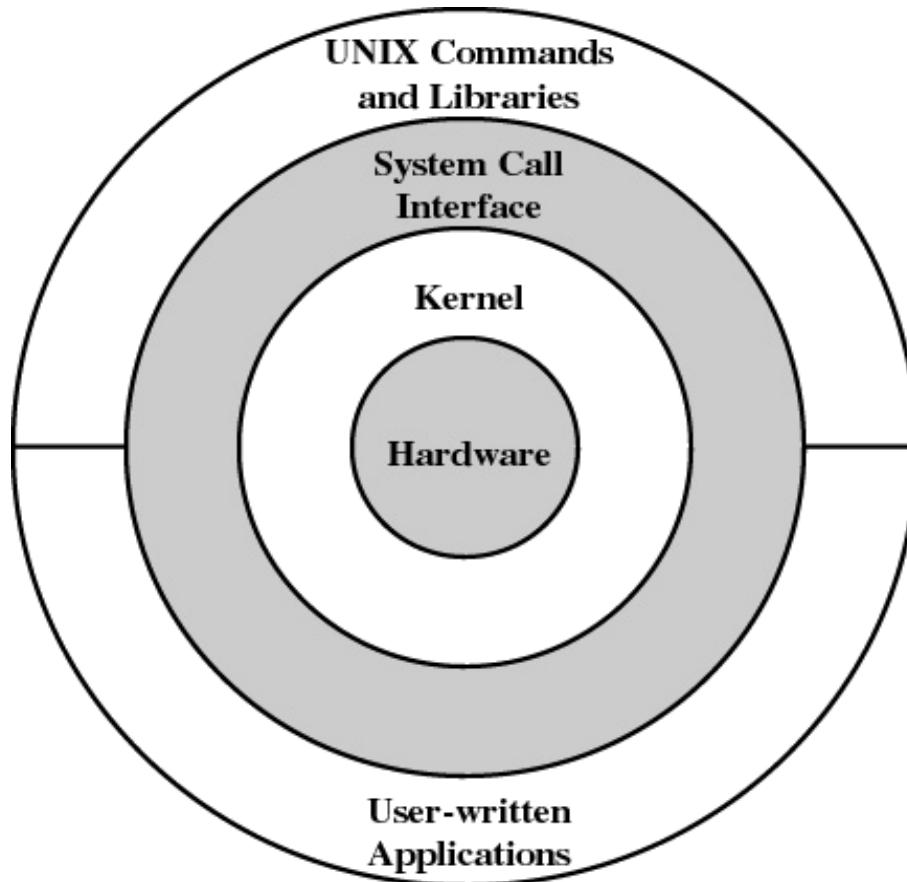
Unix

- Unix商业版本的出现源于1977年的IS/1
- AT&T：81年的System III，83年的SystemV，84年的SVR2，87年的SVR3
- 78年起，SCO和Microsoft的XENIX
- XENIX与AT&T Unix在使用标准上会合于SVR3.2
- Unix BSD：78年的1BSD和2BSD、79年3BSD、80年之后的4.0BSD、4.1BSD、4.2BSD、4.3BSD和4.4BSD
- 4BSD的商业代表Sun OS及其Solaris
- Sun OS和SVR3.2在使用标准上会合于SVR4.0

Unix

- Unix取得成功的最重要原因是系统的开放性，公开源代码，用户可以方便地向Unix系统中逐步添加新功能和工具，从而使得UNIX越来越完善，成为有效的程序开发支撑平台
- Unix是目前唯一可以安装和运行在从微型机、工作站直到大型机和巨型机上的操作系统

UNIX Architecture



Unix

- Unix的结构分核心部分和应用子系统，便于做成开放系统
- 分层可装卸卷的文件系统，提供文件保护功能
- 提供I/O缓冲技术，系统效率高
- 剥夺式动态优先的CPU调度，支持分时操作
- 命令语言丰富齐全
- 具有强大的网络与通信功能
- 支持多用户、多任务
- 请求分页式虚拟存储管理

Modern Unix Systems

- 实际上Unix已不是指一个具体操作系统，许多公司和大学都推出了自己的Unix系统
 - AT&T的SVR，SUN的Solaris，Berkeley的Unix BSD，DEC的Digital Unix（并入Compaq称Tru64 Unix），HP的HP UX，SGI的Irix，CMU的Mach，SCO公司的SCO UnixWare，IBM的AIX
- Unix的国际标准POSIX
 - IEEE拟定了一个Unix标准，称作POSIX
 - POSIX定义了相互兼容的Unix系统必须支持的最少系统调用接口。该标准已被多数Unix支持
 - 其他一些操作系统也在支持POSIX标准。

Solaris

- SUN Microsystem公司开发的Solaris是具有完全对称多处理和多线程支持的32位分布式计算环境的Unix操作系统变种
- Solaris2.x基于SPARC和Intel平台，是一个可移植操作系统，可移植到任何新的主流平台上。
- SUN公司推出64位操作系统Solaris2.7和2.8，在网络特性、可靠性、兼容性、互操作性、易于配置和管理方面均有很好改进

SUN NFS

- SUN NFS(Network File System)是1984年推出的一个网络文件系统产品
- NFS提供了在异种机、异种操作系统的网络环境下共享文件的简单有效的方法
- NFS 基于 C/S 模式实现，使用 UDP 协议和 RPC(Remote Procedure Call)机制
- NFS的主要特点有：
 - 提供透明的文件访问和文件传送，用户使用本地或远程的文件没有区别，真正实现了分布式数据处理
 - 易于扩充新的资源，不需改变现有工作环境
 - 性能高、可靠性好、具有可伸缩性

MINIX

- Minix是荷兰计算机教授Tanenbaum开发的一个与Unix兼容，然而内核全新的操作系统，它非常简洁，短小，故称Minix
- Minix用C编写，可读性好，学生可以通过它来剖析一个操作系统，研究其内部如何运作
- Minix具有多任务处理能力，支持TCP/IP
- Minix版权属于Prentice Hall，可免费下载用于教学：
- <http://www.cs.vu.nl/~ast/>

自由软件与Linux

- 70年代后期起很多软件不再提供源码，使用户无法修改软件中的错误，使用尤为不便。为此在1984年，Stallman启动了GNU计划，并成立了自由软件基金会
- 自由软件（Free Software or Freeware）是指遵循通用公共许可证GPL（General public License）规则，保证您有使用上的自由、获得源程序的自由，可以自己修改的自由，可以复制和推广的自由，也可以有收费的自由的一种软件

自由软件与Linux

- GNU的含义是GNU is not Unix的意思，由Richard stallman指导并启动的一个组织
- Stallman先生通过GNU推出一套和Unix兼容，但同时又是自由软件的Unix系统，GNU完成了大部分外围工作，包括gcc/ gcc++, shell等
- 现在GNU的所有工作继续在向前发展。目前人们熟悉的一些软件如C++编译器、Objective C、FORTRAN77、C库、BSD email、BIND、Perl、Apache、TCP/IP、IP accounting、HTTP server、Lynx Web都是自由软件的经典之作

Linux

- Linux是由芬兰籍科学家Linus Torvalds于1991年编写完成的一个操作系统内核，当时他还是芬兰赫尔辛基大学计算机系的学生，在学习操作系统课程中，自己动手编写了一个操作系统原型，并把这个系统放在Internet上，允许自由下载
- 许多人对这个系统进行改进、扩充、完善，Linux由最初一个人写的原型变化成在Internet上由无数志同道合的程序高手参与的一场运动

Linux

- 继承了Unix的优点，又有了许多更好的改进
- 通用的操作系统，可作为Internet服务器、网关路由器、文件和打印服务器、个人使用
- 内置通信联网功能，可让异种机联网
- 开放的源代码，有利于发展各种操作系统
- 符合POSIX标准，各种Unix应用可方便地移植
- 提供庞大的管理功能和远程管理功能
- 支持大量外部设备
- 支持32种文件系统
- 提供GUI，有多种窗口管理器
- 支持并行处理/实时处理，充分发挥硬件性能
- 可自由获得源代码，开发软件成本低

IBM系列操作系统

- IBM：国际商业机器公司
- IBM操作系统的发展历程
- 目前IBM的主要操作系统
 - RS/6000系列服务器及SP结点群集计算机，运行AIX操作系统
 - S/390企业级服务器，运行OS/390、VM和DOS/VSE操作系统
 - AS/400服务器运行OS400操作系统
 - PC微型机，运行Windows、OS2、DOS等操作系统