```yaml
title: 大学课程 | 计算机操作系统
tags:
  - 操作系统
  - 大学课程
categories: 学习笔记
abbrlink: 8398
date: 2019-09-16 21:09:06
reward: true
copyright: true
cover: https://npm.elemecdn.com/justlovesmile-img/233900-1579621140c81d.jpg
top_img: https://npm.elemecdn.com/justlovesmile-img/233900-1579621140c81d.jpg
```

作者博客：[Justlovesmile's BLOG]

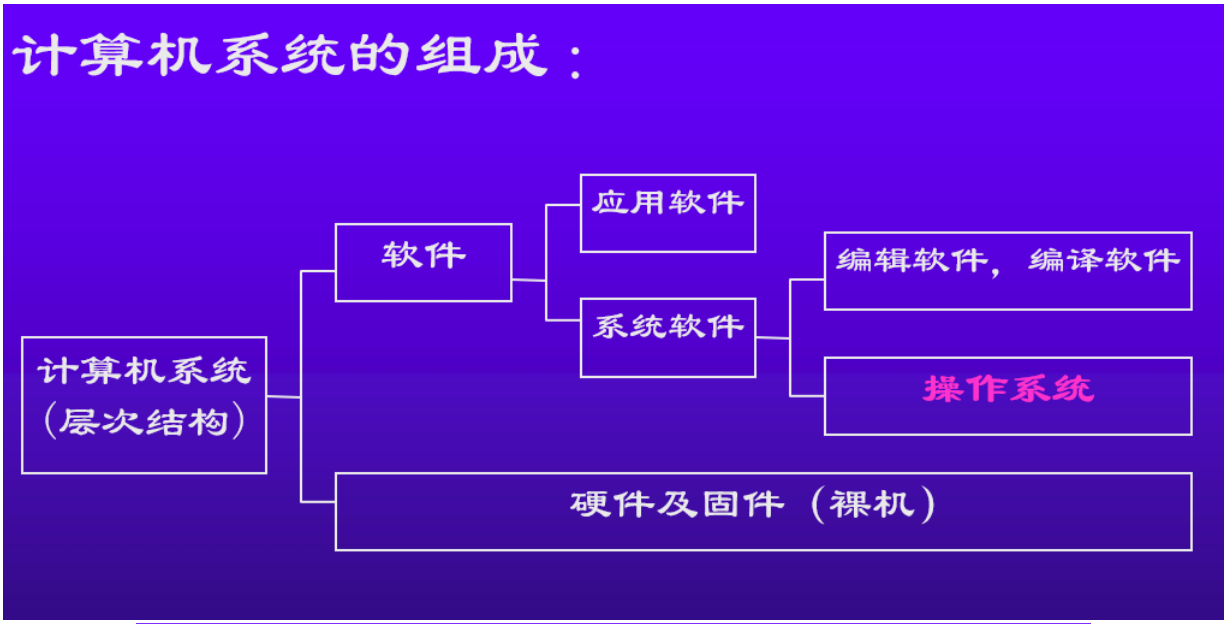
大二计算机操作系统课程笔记

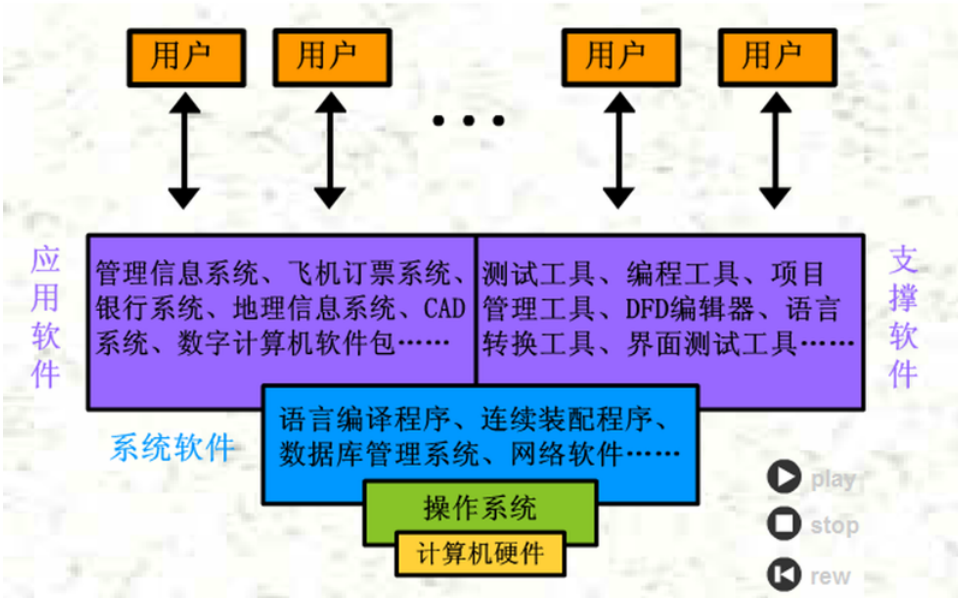
计算机操作系统

第一章 操作系统引论

- 操作系统是配置在计算机硬件上的第一层软件，是对硬件系统的首次扩充。

1.1 操作系统的目标和作用

- 

- 

- 操作系统的地位：紧贴于系统硬件之上，所有其他软件之下（是其他软件的共同环境）。

1.1.1 操作系统的目标

---

1. **方便性**：对**用户**方便，提供良好的，一致的用户接口
2. **有效性**：对**系统管理人员**方便
   （1）提高了系统资源的利用率
   （2）提高系统吞吐量
3. **可扩充性**：与OS结构相关，方便添加新的功能和模块
   早期无结构→模块化结构→层次结构→微内核结构和客户服务器模式
4. **开放性**：开放系统互连OSI国际标准，实现应用的可移植性和互操作性

1.1.2 操作系统的作用

> 操作系统的非形式化定义（关键点）：**系统软件，程序模块的集合，资源管理和用户接口功能**

1. OS作为用户与计算机硬件系统之间的接口

- 通过三种方式使用计算机：**命令方式**，**系统调用方式**，**图标—窗口方式**

2. OS作为计算机系统资源的管理者

- 对这些资源：处理机，存储器，I/O设备以及文件（数据和程序）**有效的管理**。

3. OS实现了对计算机资源的抽象（OS是扩展机，是虚拟机器）

- 在裸机上添加：设备管理、文件管理、存储管理（针对内存和外存）、处理机管理（针对CPU）
- 合理组织工作流程：作业管理、进程管理。

1.1.3 推动操作系统发展的主要动力

1. 不断提高计算机资源利用率和系统性能
2. 方便用户
3. 器件的不断更新换代
4. 计算机体系结构的不断发展
5. 不断提出新的应用需求

1.2 操作系统的发展过程

1.2.1 未配置操作系统的计算机系统

一， 人工操作方式：

1. 计算机的工作特点

- **用户独占全机**
  （用户是程序元，计算机专业人员）
  （编程语言是机器语言）
  （输入/输出：纸带或卡片）
- **CPU等待人工操作**：严重降低计算机资源的利用率，存在人机矛盾

2. 主要矛盾（**人机矛盾**）：

- 计算机处理能力的提高，手工操作的低效率（造成浪费）
- 用户独占全机的所有资源

3. 提高效率的途径：
   专门的操作员，批处理

二，脱机输入/输出方式：

1. 工作方式：

- 程序和数据的输入输出都是在外围机的控制下完成的，或者说，是在脱离主机的情况下进行的

2. 优点：

- 减少了CPU空闲时间
- 提高了I/O速度

1.2.2 单道批处理系统

1. 单道批处理系统的处理过程：

- 通过作业控制语言将磁带（或卡片）上的若干个作业编成作业执行序列，自动实现作业间的自动转换。每个批作业由一个专门的**监督程序**自动依次处理。
- 程序和数据虽然是成组成批提交，但是任一时间只有一个作业运行，因此称为单道批处理。
- 批处理中的作业的组成：包括用户程序、数据和作业说明书（作业控制语言）

2. 单道批处理系统的特征：

- 自动性
- 顺序性
- 单道性

3. 优点：

- 同一批内各作业的自动依次更替，改善了主机CPU和I/O设备的使用效率，提高了吞吐量。

4. 缺点：

- 主要缺点：系统的**资源得不到充分的利用。**CPU和I/O设备使用忙闲不均
- 磁带或磁盘需要人工装卸，作业需要人工分类
- 监督程序易遭到用户程序的破坏（由人工干预才可恢复）。
- I/O与CPU之间的速差造成的CPU空闲

1.2.3 多道批处理系统

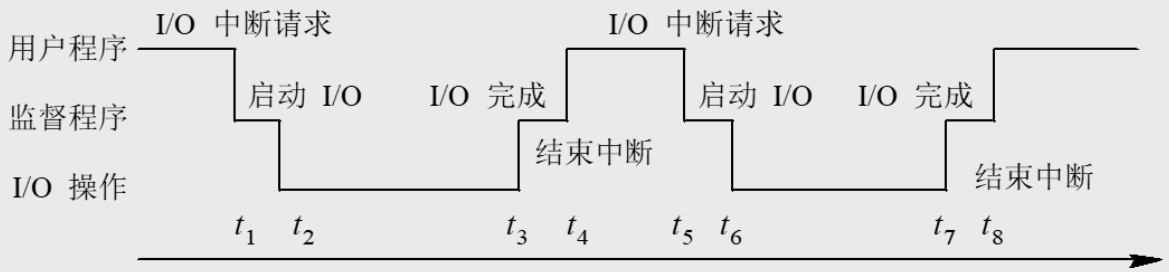
1. 多道批处理系统的概念：在系统中，用户所提交的作业先存放在外存上，并排列成一个队列，称为“后备队列”，然后由作业调度程序按一定的算法，从后备队列中选出若干个作业调入内存，使它们共享CPU和系统中的各种资源。在程序A因I/O操作而空闲CPU时，运行程序B，使**多道程序交替运行。**

- 内存中同时存放几个作业：
  （1）**宏观上并行运算**
  （2）**微观上串行运算**
- 特点：
  （1）**多道性**：
  多道程序驻留内存：提高了资源的利用率；
  程序并发执行：提高了系统的吞吐量
  （2）**无序性**：
  作业进入内存先后顺序和完成的先后顺序无对应性
  （3）**调度性**：
  作业调度
  进程调度

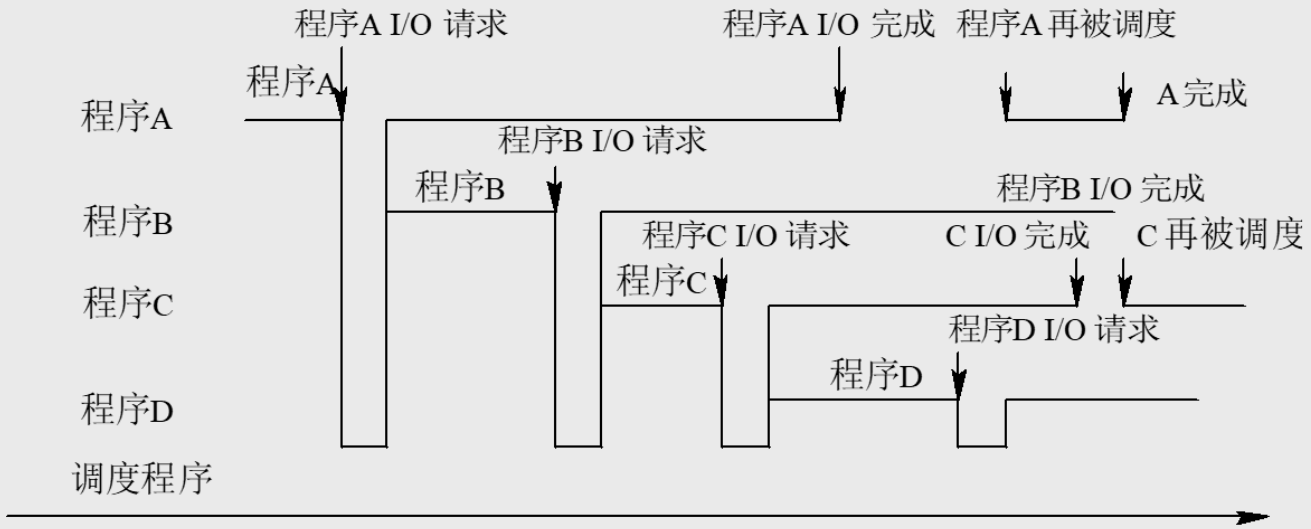
2. 优点：
   （1）资源利用率高：**并发执行**
   （2）系统吞吐量大：①保持忙碌②系统开销小
3. 缺点：
   （1）平均周转时间长
   （2）无交互能力
4. 多道批处理系统需要解决的问题：
   （1）处理机争用问题
   （2）内存分配和保护问题
   （3）I/O设备分配问题
   （4）文件的组织和管理问题
   （5）作业管理问题
   （6）用户与系统的接口问题

- 操作系统是一组控制和管理计算机硬件和软件资源，合理地对各类作业进调度，以及方便用户使用的程序集合
- 单道与多道的区别：
-

单道与多道程序运行情况



(*a*) 单道程序运行情况



(*b*) 四道程序运行情况

单道与多道批处理的比较

	单道	多道
内存使用	每次一个作业	每次多个作业（充分利用内存）
作业次序	顺序，先进先出	无确定次序

	单道批处理	多道批处理
内存中驻留程序数目	一道	多道
占用CPU的情况	独占	交替占用
是否需要作业和进程调度	不需要	需要
程序完成次序与其进入内存次序间的关系	严格对应	不严格对应

1.2.4 分时系统

1. 分时系统的引入，用户需求具体表现在：
   （1）人机交互
   （2）共享主机
   （3）便于用户上机

2. 分时系统实现中的关键问题：
   （1）及时接收
   （2）及时处理：
   - 作业直接进入内存
   - 采用轮转运行方式：引入**时间片**
   - 分为抢先式和非抢先式的OS（CPU让出OS的方式是自动还是强迫）

3. 分时系统的特征：
   （1）多路性：共享主机
   （2）独立性：感觉像是用户独占主机
   （3）交互性
   （4）及时性

1.2.5 实时系统

- 主要特征：将**时间**作为关键参数
- 特点：（1）**可靠性强**（2）**响应时间短**

1. 实时系统的类型：
   工业（武器）控制系统，信息查询系统，多媒体系统，嵌入式系统
2. 实时任务的类型：
   （1）周期性，非周期性：截至时间：①开始截止时间②完成截止时间
   （2）硬实时，软实时：
   - 硬实时：满足对截止时间的要求
   - 软实时：不严格要求截止时间
   -

1.2.6 *微机操作系统的发展

- 软件工程的出现：
  操作系统在结构、规模、数量、接口等方面都得到了简化：结构从复杂到简单，规模从大到小，核心外移。
  实际系统的数量由于通用、开放和标准化而大量减少。
- 小型化与网络：
  微机操作系统和个人操作系统

- 网络操作系统：
  （1）通过通信设施将物理上分散的具有自治功能的多个计算机系统互连起来的实现信息交换、资源共享、可互操作和协作处理的系统。
  （2）在各种计算机操作系统上，按网络体系结构协议标准开发的软件
  （3）包括网络管理、通信、安全、资源共享和各种网络应用
  （4）目标：是相互通信及资源共享
- 分布式操作系统：
  1）特征：
  （1）是一个统一的操作系统
  （2）资源进一步共享
  （3）透明性：资源共享与分布对用户是透明的
  （4）自治性：处于分布式系统的多个主机处于平等地位，无主从关系
  （5）处理能力增强、速度更快、可靠性增强
  2）网络和分布式的区别：
  （1）分布式具有各个计算机间相互通讯，无主从关系；网络有主从关系
  （2）分布式系统资源为所有用户共享；而网络有限制地共享
  （3）分布式系统中若干个计算机可相互协作共同完成一项任务
- 嵌入式系统：
  （1）在各种设备、装置或系统中，完成特定功能的软硬件系统。它们是一个大设备、装置或系统中的一部分，这个大设备、装置或系统可以不是“计算机”。由于它们被嵌入在各种设备、装置或系统中，因此称为嵌入式系统
  （2）EOS（Embedded Operating System）在嵌入式系统中的OS，运行在嵌入式智能芯片环境中
  （3）典型嵌入式操作系统的特性：
  - 完成某一项或有限项功能；不是通用型的
  - 在性能和实时性方面有严格的限制
  - 能源、成本和可靠性通常是影响设计的重要因素
  - 占有资源少、易于连接
  - 系统功能可针对需求进行裁剪、调整和生成，以便满足最终产品的设计要求

---

1. 单用户单任务操作系统：

- 只允许一个用户上机，且只允许用户程序作为一个任务运行
- CP/M， MS-DOS

2. 单用户多任务操作系统：

- 只允许一个用户上机，但允许用户把程序分为若干个任务，并使他们并发执行
- Windows

3. 多用户多任务操作系统：

- 允许多个用户通过各自的终端，使用同一台机器，共享系统中的各种资源，而每个用户程序又可进一步分为几个任务，使它们能并发执行
- UNIX OS

> 操作系统的发展规律:
> 操作系统的发展是由底层硬件、软件技术与上层应用需求的发展推动的。
> 操作系统的每一步发展都是权衡的结果：可能性与必要性的权衡，性能与代价的权衡，方便和效率的权衡。

项目	网络OS	分布式OS	多处理机OS
单处理机	不像	像	像
相同OS（同质）	不需要	需要	需要
有多少个OS	n个	n个	1个
通信	共享文件	通讯协议	共享存储器
网络协议	不需要	不需要	需要
运行队列	不需要	不需要	需要
文件共享是否有限制	没有	有，需要特殊文件说明	有限制，需特殊文件

1.3 操作系统的基本特性

- 四个基本特性：**并发，共享，虚拟，异步**

1.3.1 并发

1. 并行与并发：

- 并行性：两个或多个时间在**同一时刻**发生
- 并发性：两个或多个时间在**同一时间间隔内**发生

2. 引入进程

- 进程：**在系统中能独立运行并作为资源分配的基本单位**
- 引入线程是现代操作系统的重要标志

1.3.2 共享

- OS环境下的资源共享或称资源复用，是指系统中的资源可供内存中多个并发执行的进程共同使用

1. **互斥共享方式**

- 资源可以提供给多个进程，但应规定在一段时间内，只允许访问一个进程访问该资源
- 资源分配后到释放之前，不能被其他进程所用
- 临界资源：一段时间内只允许一个进程访问的资源

2. **同时访问方式**

- 资源允许在一段时间内由多个进程“同时”对它们进行访问
- 可重入代码，磁盘文件
- **并发和共享是多用户OS的两个最基本的特征**
- 资源分配难以达到最优化

1.3.3 虚拟

- 一个物理实体映射为若干个对应的逻辑实体
- **虚拟是操作系统管理系统资源的重要手段，可提高资源利用率。**
- CPU - - 每个用户（进程）的“虚处理机”
- 存储器 - - 每个进程都占有的地址空间（指令 + 数据 + 堆栈）
- 显示设备 - - 多窗口或虚拟终端(virtual terminal)

1. **时分复用技术**：
   （1）虚拟处理机技术
   （2）虚拟设备技术
2. **空分复用技术**：

> 虚拟的实现：如果是采用分时复用的方法，即对某一物理设备进行分时使用，设N是某物理设备所对应的虚拟的逻辑设备数，则每台虚拟设备的平均速度必然等于或低于物理设备速度的1/N。类似的如果是采用空分复用技术，此时一台虚拟设备平均占用的空间必然也等于或小于物理设备所拥有的空间的1/N

1.3.4 异步

- 也称不确定性，指进程的执行顺序和执行时间的不确定性；
- **进程的运行速度不可预知**：分时系统中，多个进程并发执行，“时走时停”，不可预知每个进程的运行推进快慢
- 判据：无论快慢，应该结果相同， 通过进程互斥和同步手段来保证
- **难以重现**系统在某个时刻的状态（包括重现运行中的错误）
- 性能保证：实时系统与分时系统相似，但通过资源预留以保证性能

1.4 操作系统的主要功能

1.4.1 处理机管理功能

- 处理机管理的主要功能：创建和撤销进程，对诸进程的运行进行协调，实现进程之间的信息交换以及按照一定的算法把处理机分配给进程

1. **进程控制**：为作业创建进程，撤销（终止）已结束的进程以及控制进程在运行过程中的状态转换
2. **进程同步**：

- **重要功能**
- 主要功能：为多个进程的运行进行协调
- 常用的协调方式：
  （1）进程互斥方式
  （2）进程同步方式：常用信号量机制

3. 进程通信
4. 调度：
   （1）作业调度：从后备队列中按照一定的算法选择除若干个作业，为它们分配资源，将其调入内存后，为其建立进程，将它们插入就绪队列
   （2）进程调度：从进程的就绪队列中按照一定的算法选出一个进程，将处理机分配給它，并为它设置运行现场，使其投入执行

1.4.2 存储器管理功能

- 为多道程序的运行提供良好的环境，提高存储器的利用率，方便用户使用，并能从逻辑上扩充内存

1. **内存分配**
   （1）静态分配方式：每个作业的内存空间是在作业装入时确定的，在作业装入后的整个运行期间不允许作业再次申请新的内存空间，也不允许在内存中移动
   （2）动态分配方式：允许作业在运行期间继续申请新的附加内存空间，以适应程序和数据的动态增长，也允许作业在内存中移动。
2. **内存保护**：保证进程间互不干扰、相互保密；如：访问合法性检查、甚至要防止从“垃圾”中窃取其他进程的信息
3. **地址映射**：

- **主要功能**
- 进程逻辑地址到内存物理地址的映射；

4. **内存扩充**：

- 从**逻辑上**扩充内存容量
- 设置内存扩充机制，以实现请求调入功能和置换功能

1.4.3 设备管理功能

- 主要任务：
  （1）完成用户进程提出的I/O请求
  （2）提高CPU和I/O设备的利用率

1. **缓冲管理**：单缓冲机制，双缓冲机制，公用缓冲池机制
2. **设备分配**：
3. **设备处理**：

- 设备处理程序又叫：设备驱动程序，实现CPU和设备控制器之间的通信
- 根据中断请求的类型，调用相对应的中断处理程序

1.4.4 文件管理功能

- 主要任务：是对用户文件和系统文件进行管理以方便用户使用，并保证文件的安全性

1. **文件存储空间的管理**：为每一个文件分配必要的外存空间，提高外存的利用率，进而提高文件系统的存取速度
2. **目录管理**：方便存取，实现共享，提高文件检索的速度
3. **文件的读/写管理和保护**：
   （1）文件的读写/管理
   （2）文件保护：防止未经核准的用户存取文件，防止冒名顶替存取文件，防止以不正确的方式使用文件
4. **软件管理**：软件的版本、相互依赖关系、安装和拆除等

1.4.5 操作系统与用户之间的接口

1. **用户接口**：用户通过该接口直接或间接地控制自己的作业
   （1）联机用户接口
   （2）脱机用户接口
   （3）图形用户接口
2. **程序接口**：为用户程序在执行中访问系统资源而设定的，是用户程序取得操作系统服务的唯一途径

1.4.6 现代操作系统的新功能

1. 系统安全：
   （1）认证技术
   （2）密码技术
   （3）访问技术
   （4）反病毒技术
2. 网络的功能和服务
   （1）网络通信
   （2）资源管理
   （3）应用互操作
   （4）支持多媒体
   （1）接纳控制功能
   （2）实施调度
   （3）多媒体文件的存储

1.5 OS结构设计

1.5.1 传统操作系统结构

1. 无结构操作系统

2. 模块化结构OS
   （1）基本概念：按功能精心地划分为若干个具有一定独立性和大小的模块，并仔细规定好各模块的间的接口，使各模块之间能通过接口实现交互。然后再细分为子模块，这种设计方法称为模块-接口法
   （2）模块独立性：

   - 内聚性：子模块内部各部分间联系的紧密程度，内聚性越高，模块独立性高
   - 耦合度，指模块间相互联系和相互影响的程度，耦合度越低，模块独立性越好

   （3） 模块接口法的优缺点：
   优点：
   ①提高OS设计的正确性，可理解性，可维护性
   ②增强OS的可适应性
   ③加速OS的开发过程
   缺点：
   ①接口很难满足实际需求
   ②无序模块法，无法寻找一个可靠的决定顺序

3. 分层式结构OS
   （1）基本概念：有序分层法，每一步设计都建立在可靠的基础上，**高层仅依赖于紧邻它的底层**
   （2）优点：
   ①易保证系统的正确性
   ②易扩充和易维护性
   （3）缺点：系统效率低

1.5.2 客户/服务器模式

1. 客户/服务器模式的由来：
   （1）客户机（2）服务器（3）网络系统
2. 交互：
   （1）客户发送请求消息
   （2）服务器接受消息
   （3）服务器回送消息
   （4）客户机接受消息
3. 优点：
   （1）数据的分布式处理和存储
   （2）便于集中管理
   （3）灵活性和可扩充性
   （4）易于改编应用软件

4. 缺点：

- 不可靠性和瓶颈问题

1.5.3 面向对象的程序设计

1. 基本概念：（1）对象（2）对象类（3）继承
2. 优点：（1）通过“重用”提高质量和效率（2）易修改性和易扩展性
   （3）易于保证正确性和可靠性

1.5.4 微内核OS结构

- 适用于分布式系统环境

1. 基本概念：
   （1）足够小的内核：最基本的部分：①与硬件处理紧密相关部分②较基本功能③C/S之间的通信
   （2）基于C/S模式
   （3）应用“机制和策略分离”原理：机制在低层，策略在系统高层
   （4）采用面向对象编程
2. 微内核的功能：
   （1）进程（线程）管理：进程（线程）之间的通信是微内核OS最基本的功能
   （2）低级存储器管理：实现用户空间的逻辑地址变换为内存空间的物理地址的页表机制和地址变换机制
   （3）中断和陷入处理
3. 微内核操作系统的优点：
   （1）提高可扩展性
   （2）增强系统可靠性
   （3）可移植性增强
   （4）支持分布式系统
   （5）融入面向对象技术
4. 微内核操作系统存在的问题

- 运行效率降低

第二章 进程的描述与控制

2.1 前趋图和程序执行

2.1.1 前趋图

- **有向无循环**图，可记为DAG，用于描述进程之间的先后顺序。
- 图中结点：进程或程序段
- 有向边：结点之间的偏序或前趋关系
- 权值：每个结点的重量

2.1.2 程序的顺序执行

1. 程序的顺序执行
2. 程序的顺序执行时的特征：
   （1）顺序性：每一个操作必须在下一个操作开始之前结束
   （2）封闭性：程序运行时独占全机资源，程序一旦开始执行，其执行结果不受外界因素影响
   （3）可再现性：只要程序执行时的环境和初始条件相同，当程序重复执行时。都可以得到相同的结果

2.1.3 程序并发执行

- 只有在**不存在前驱关系**的程序之间才有可能并发执行，否则无法并发执行
- 程序并发执行时的特征：
  （1）间断性
  （2）失去封闭性
  （3）不可在现性

2.2 进程的描述

2.2.1 进程的定义和特征

1. 进程的定义：
   （1）进程是程序的一次执行
   （2）进程是一个程序及其数据在处理机上顺序执行时所发生的活动
   （3）进程是具有**独立功能**的**程序**在一个**数据集合**上**运行的过程**，它是系统进行资源分配和调度的一个独立单位。【数据集合说明：需要数据，运行的过程说明：有生命周期】

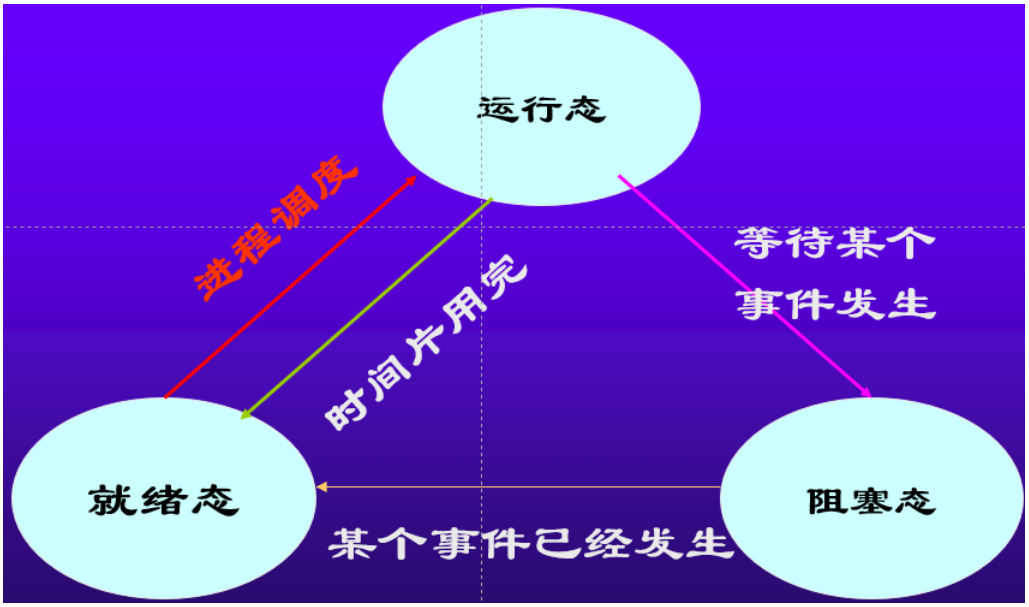
- 进程控制块（PCB）：描述进程的基本情况和活动过程，进而控制和管理进程
- 进程实体（进程映像）：包括：程序段，数据段，PCB
- 一个程序可能对应多个进程（程序的多次执行）
- 一个进程也可能对应多个程序（进程的多次调用）

2. 进程的特征：
   （1）动态性：进程动，程序静
   （2）并发性
   （3）独立性
   （4）异步性：进程暂时的，程序永久的
   （5）结构性：程序段，数据段，PCB

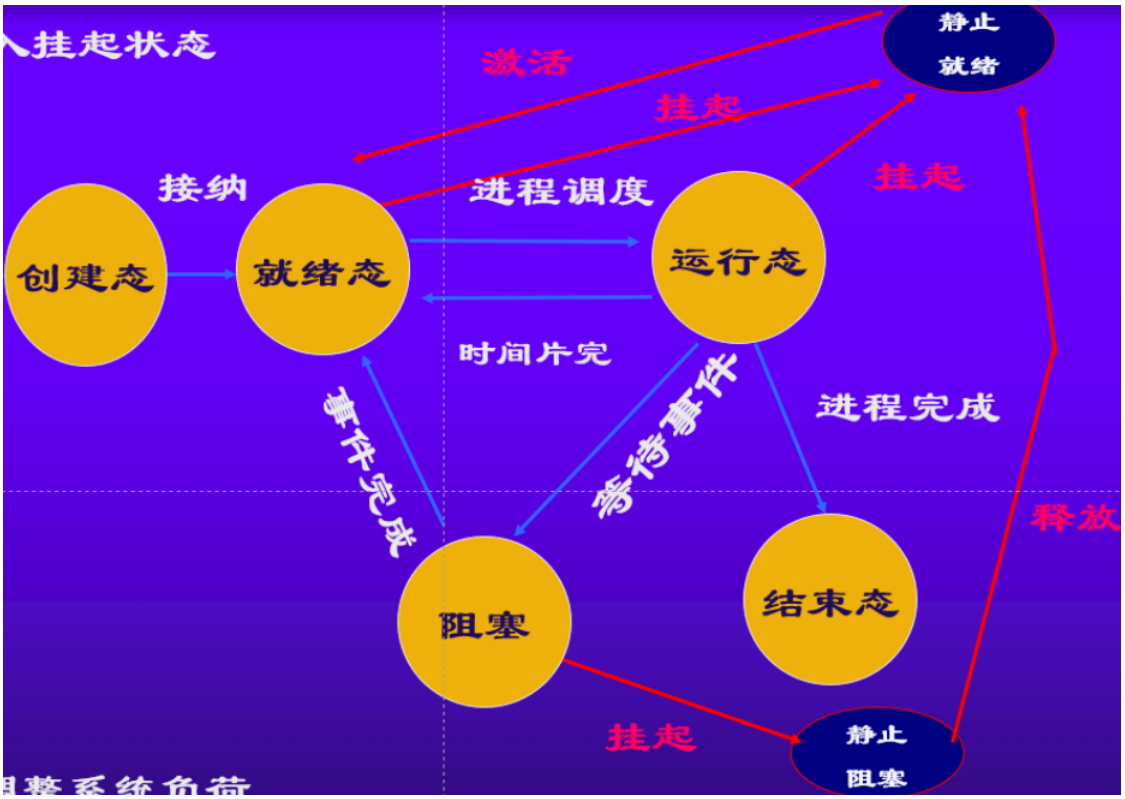
2.2.2 进程的基本状态和转换

1. 进程的三种基本状态：
   （1）**就绪状态**：n个进程最多n-1个，多个就绪状态的进程排成就绪队列
   （2）**执行状态**：单处理机只有一个，多处理机多个
   （3）**阻塞状态**：最多n个，正在执行的进程由于发生某事件（如I/O请求，申请缓冲区失败等）暂时无法继续执行时的状态

2. 三种基本状态的转换：



3. 创建状态和终止状态



（1）创建状态：进程由创建而产生，创建一个进程的步骤：首先，进程申请一个空白PCB，并向PCB中填写用于控制和管理进程的信息；然后为进程分配运行时所必须的资源；最后，把该进程转入就绪状态并插入就绪队列之中。如果进程所需的资源得不到满足，比如系统尚无足够的内存使进程无法装入其中，此时创建工作尚未完成，进程不能被调度运行，此时进程所处的状态被称为**创建状态**

（2）终止状态

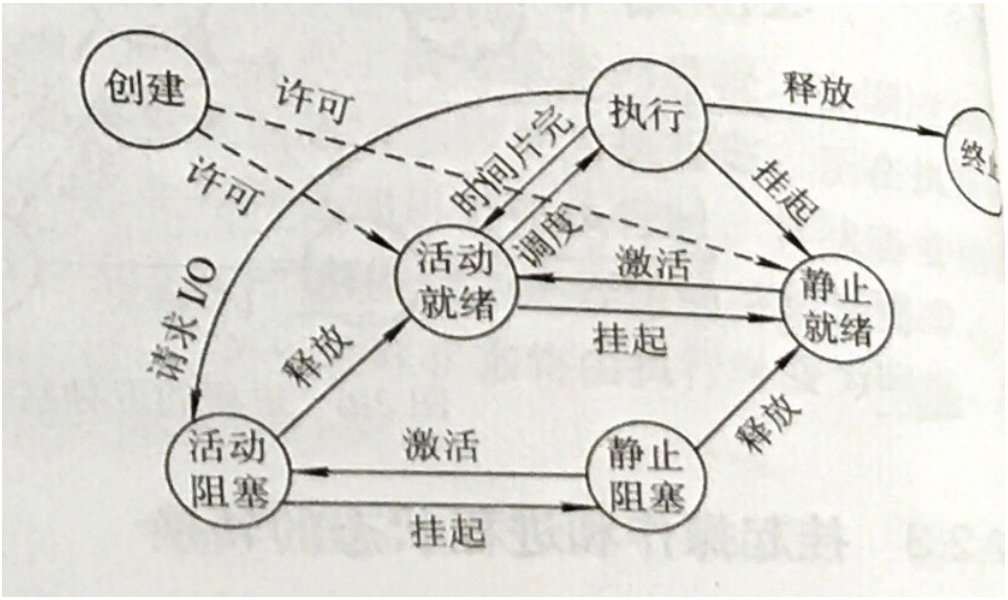
首先，等待操作系统进行善后处理，最后将其PCB清零，并将PCB空间返回系统。当一个进程到达了自然结束点，或是出现了无法克服的错误清零，或是被操作系统所终结，或是被其他有终止权的进程所终结，它将进入**终止状态**。进入终止态的进程以后不能再执行，但在操作系统中依然保留一个记录，其中保存状态码和一些计时统计数据，供其他进程收集。

2.2.3 挂起操作和进程状态的转换

- 当操作作用于某个进程时，该进程将被**挂起**，意味着此时此刻进程处于静止状态

1. 挂起操作的引入：
   （1）终端用户的需要
   （2）父进程的请求
   （3）负荷调节的需要
   （4）操作系统的需要

2. 引入挂起原语操作后三个进程状态的转换
   （1）活动就绪→静止就绪
   （2）活动阻塞→静止阻塞
   （3）静止就绪→活动就绪
   （4）静止阻塞→活动阻塞



3. 引入挂起操作后五个进程状态的转换
   （1）NULL→创建：
   （2）创建→活动就绪
   （3）创建→静止就绪
   （4）执行→终止

2.2.4 进程管理中的数据结构

1. 操作系统中用于管理控制的数据结构

- 内存表
- 设备表
- 文件表
- 进程表PCB，进程控制块。有总数限制，

2. 进程控制块PCB的作用

- 在OS的核心为每一个进程专门定义了一个PCB，记录了操作系统所需的，用于描述进程的当前情况以及管理进程运行的全部信息，是操作系统中最重要的记录型数据结构。
- PCB的作用是**使一个在多道程序环境下不能独立运行的程序（含数据）成为一个能独立运行的基本单位，一个能与其他进程并发执行的进程**
  （1）**作为独立运行的基本单位的标志**：系统使通过PCB感知进程的存在的，PCB已成为进程存在的唯一标志
  （2）**能实现间断性运行方式**：将CPU现场信息保存在中断进程的PCB中，供该进程再次被调度执行时恢复CPU现场时使用
  （3）**提供进程管理所需要的信息**
  （4）**提供进程调度所需要的信息**
  （5）**实现与其他进程的同步与通信**

3. 进程控制块中的信息：
   （1）进程标识符：唯一地标识一个进程
   - ①外部标识符：由数字，字母构成
   - ②内部标识符：为每一个进程赋予唯一的数字标识符，是一个进程的序号
     （2）处理机状态：
   - 处理机状态信息：①通用寄存器②指令计数器③程序状态字PSW④用户栈指针
   - CPU现场保护结构：寄存器值，主要由CPU的各种寄存器组成
     （3）进程调度信息：①进程状态②进程优先级③进程调度所需其他信息，与进程调度算法有关④事件，即阻塞原因
     （4）进程控制信息：①程序和数据的地址②进程同步和通信机制③资源清单④链接指针
4. 进程控制块的组织方式：
   （1）线性方式
   （2）链接方式
   （3）索引方式

2.3 进程控制

- 进程控制一般是由OS内核中的原语来实现的

2.3.1 操作系统内核

- OS内核：一般将OS划分为若干层次,将紧靠硬件的软件层次常驻内存
- 处理机的执行状态分为系统态和用户态：
  - 系统态：又称管态，内核态，具有较高特权，能执行一切指令，访问所有的寄存器和存储区
  - 用户态：又称目态，具有较低权限

1. 支撑功能：
   （1）**中断处理**：内核中最基本的功能，是整个操作系统赖以活动的基础
   （2）**时钟管理**
   （3）**原语操作**：原语：是由若干条指令组成的，用于完成一定功能的一个过程，是**原子操作**，即要么都做，要么都不做
2. 资源管理功能：
   （1）进程管理
   （2）存储器管理
   （3）设备管理

2.3.2 进程的创建

1. 进程的层次结构

- 把创建进程的进程称为父进程，而把被创建的进程称为子进程，子进程还可以创建孙进程

2. 进程图：描述进程间关系的一颗有向树
3. 引起创建进程的事件：
   （1）用户登陆
   （2）作业调度
   （3）提供服务
   （4）应用请求
4. 进程的创建：
   步骤：
   （1）申请空白PCB，为新进程获取唯一的数字标识符
   （2）为其分配运行时所需的资源，包括物理和逻辑
   （3）初始化进程控制块PCB：①初始化标识信息②初始化处理机状态信息③初始化处理机控制信息
   （4）如果进程就绪队列能够接纳新进程，便将新进程插入就绪队列

2.3.3 进程的终止

1. 引起进程终止的事件：
   （1）正常结束：通常会在程序的最后安排一条Halt指令，用于向OS表示运行已结束
   （2）异常结束：
   - 越界错
   - 保护错
   - 非法指令
   - 特权指令错
   - 运行超时
   - 等待超时
   - 算术超时
   - I/O故障
     （3）外界干预：
   - 操作员或操作系统干预
   - 父进程请求
   - 因父进程终止
2. 进程的终止过程：
   （1）根据被终止进程的标识符，从PCB集合中检索该进程的PCB，从中读出该进程的状态
   （2）若被终止进程正处于执行状态，应立即终止该进程的执行，并置调度标志为真，用于指示该进程被终止后应重新进行调度
   （3）若该进程还有子孙进程，还应将其所有子孙进程也都给予终止，以防他们成为不可控进程
   （4）将被终止进程所拥有的全部资源或者归还其父进程，或者归还系统
   （5）将被终止进程（PCB）从所在队列（或链表）中移除，等待其他程序来搜集信息

2.3.4 进程的阻塞与唤醒

1. 引起进程阻塞和唤醒的事件：
   （1）向系统请求共享资源失败
   （2）等待某种操作的完成
   （3）新数据尚未到达
   （4）等待新任务的到达
2. 进程阻塞过程

- 进程主动调用阻塞原语block将自己阻塞

3. 进程唤醒过程

- 当被阻塞进程所期待的事件已经发生时，有关程序调用唤醒原语wakeup
- wakeup：首先从阻塞队列中移除，将其PCB中的现行状态改为就绪，然后将该PCB插入到就绪队列

2.3.5 进程的挂起与激活

1. 进程的挂起：

- 原语suspend：检查被挂起进程的状态，若处于活动就绪状态→静止就绪，对于活动阻塞状态→静止阻塞，为方便用户或父进程考察进程的运行情况，把该进程PCB复制到某指定的内存区域，若被挂起的进程正在执行，则转向调度程序重新调度

2. 进程的激活过程

- OS利用激活原语active：将进程从外存调入内存，检查该进程的现行状态，若静止就绪→活动就绪，若静止阻塞→活动阻塞

2.4 进程同步

- 使程序的执行具有可再现性

2.4.1 进程同步的基本概念

1. 两种形式的制约关系
   （1）间接相互制约关系：多个进程互斥访问资源
   （2）直接相互制约关系：多个进程相互合作
2. 临界资源

- **硬件或软件，多个进程在对其访问时，必须互斥进行**

3. 临界区

- **临界区**：每个进程中**访问临界资源的那段代码**
- 进入区：临界区前面增加一段用于进行检查的代码
- 退出区：临界区后面用于将临界区正在被访问的标志恢复为未被访问的标志
- 剩余区：除了上诉区外的其他部分代码

4. 同步机制应遵循的规则
   （1）**空闲让进**：临界资源空闲，应当允许一个请求进入临界区的进程立即进入自己的临界区，若有多个请求，必须挑一个
   （2）**忙则等待**：若有临界资源正在被使用，其他进程需要等待。每次最多一个进程
   （3）**有限等待**：每一个进程逗留有限时间
   （4）**让权等待**：让出CPU竞争（不参与就绪，参与阻塞状态）

2.4.2 硬件同步机制

1. 关中断：

- 实现互斥的最简单的方法之一
- 在进入锁测试之前关闭中断，直到完成锁测试并上锁之后才能打开中断

2. 利用Test-and-Set指令实现互斥
   该指令读出标志后设标志置为TRUE

```
boolean TS(boolean *lock) {
   boolean old;
   old = *lock;
   *lock = TRUE;
   return old;
}
```

- lock表示资源的两种状态：TRUE表示正被占用，FALSE表示空闲
- 利用TS实现进程互斥：每个临界资源设置一个公共布尔变量lock，初值为FALSE
- 在进入区利用TS进行检查：有进程在临界区时，重复检查；直到其它进程退出时，检查通过；

```
do{
...
    while TS(&lock);
    critical section;
    lock = FALSE;
    remainder section;
}while(TRUE);
```

3. 利用Swap指令实现进程互斥

- Swap指令（或Exchange指令）
  交换两个字（字节）的内容

```
   void SWAP(int *a, int *b)
    {
        int temp;
        temp = *a;
        *a = *b;
        *b = temp;
    }
```

利用Swap实现进程互斥：每个临界资源设置一个公共布尔变量lock，初值为FALSE。每个进程设置一个私有局部布尔变量key

```
do{
    key = TRUE;
    do{
        swap(&lock,&key);
    }while  (key !=FALSE) ;
    临界区操作;
    lock = FALSE;
    ...
}while (TRUE);
```

2.4.3 信号量机制

1. 整型信号量：

- wait和signal操作可描述为：

```
wait(int s)
{
while(s<=0);
s--;
}

signal(int s)
{
s++;
}
```

- 为临界资源设置一个互斥信号量mutex(MUTual Exclusion)，其初值为1；

- 在每个进程中将临界区代码置于wait(mutex)和signal(mutex)原语之间
必须成对使用wait和signal原语：遗漏wait原语则不能保证互斥访问，遗漏signal原语则不能在使用临界资源之后将其释放（给其他等待的进程）；

- wait、signal原语不能次序错误、重复或遗漏:

2. 记录型信号量机制

- 每个信号量s除一个整数值外s.count，还有一个进程等待队列s.queue，其中是阻塞在该信号量的各个进程的标识
- 信号量只能通过初始化和两个标准的原语来访问 - - 作为OS核心代码执行，不受进程调度的打断。

3. AND型信号量
4. 信号量集

2.4.4 信号量的应用

1. 利用信号量实现进程互斥：

- 利用整型信号量机制实现进程互斥时应注意，wait(mutex)和signal(mutex)必须成对出现
- 缺少wait(mutex)会导致系统混乱，不能保证对临界资源的互斥访问
- 缺少signal(mutex)将会使临界资源永远不被释放，从而使因等待该资源而阻塞的进程不再被唤醒

2. 利用信号量实现前趋关系：

- 设有两个并发进程P1和P2。P1中有语句S1，P2中有语句S2，希望在执行完S1后执行S2
- 为实现这种前趋关系，可让进程P1和P2共享一个公用信号量S，并赋初值为0，将signal(S)操作放在语句S1后面；而在S2语句前插入wait(S)操作
- 进程P1， S1;signal(S);
- 进程P2， wati(S); S2;
- 由于S被初始化为0，若P2先执行，必定阻塞，只有在进程P1执行完使S增为1后，P2才能执行S2操作

2.4.5 管程机制

1. 管程的定义：代表共享资源的**数据结构**以及由对该共享数据结构实施**操作**的一组过程所组成的资源管理程序共同构成了一个操作系统的资源管理模块
2. 一个管程定义了一个数据结构和能为并发进程所执行（在该数据结构上）的一组操作，这组操作能同步进程和改变管程中的数据
3. 管程的组成部分：
   （1）管程的名字
   （2）局部于管程的共享数据结构说明
   （3）对该数据结构进行操作的一组过程
   （4）对局部于管程的共享数据设置初始值的语句
4. 所有进程要访问临界资源时，都只能通过管程间接访问，而管程每次只准许一个进程进入管程，执行管程内的过程，从而实现了进程互斥
5. 管程的特性：
   （1）模块化
   （2）抽象数据结构
   （3）信息掩蔽
6. 进程与管程的区别：
   （1）进程定义私有数据结构PCB，管程定义的是公共数据结构如消息队列
   （2）进程是由顺序程序执行有关操作，管程主要是进行同步操作和初始化操作
   （3）设置进程的目的在于实现系统的并发性，管程的设置是解决共享资源的互斥使用问题
   （4）进程通过调用管程中的过程对共享数据结构实行操作，因而管程被动，进程主动
   （5）进程之间能并发执行，管程不能
   （6）进程具有动态性，管程是操作系统中的一个资源管理模块，供进程使用
7. 条件变量

2.5 经典进程的同步问题

2.5.1 生产者-消费者问题

```
int in= 0,out= 0;
item buffer[n];
semaphore mutex = 1,empty = n,full = 0;
void producer(){
    do{
        producer an item nextp;
```

```
        ...
        wait(empty);
        wait(mutex);
        buffer[in] = nextp;
        in = (in + 1) % n;
        signal(mutex);
        signal(full);
    }while(TRUE);
 }
 void consumer(){
    do{
    wait(full);
    wait(mutex);
    nextc = buffer[out];
    out = (out+1)%n;
    signal(mutex)
    signal(empty)
    consumer the item in nextc;
    ...
    }while(TRUE);
 }
 void main(){
    cobegin
        producer();
        consumer();
    coend
}
```

2.5.2 哲学家进餐问题

略

2.5.3 读者-写者问题

略

2.6 进程通信

- 进程通信：进程间的信息交换
- 低级进程通信：进程的互斥和同步在进程间交换信息
  （1）效率低（2）通信对用户不透明
- 高级通信：进程间传送大量数据
  （1）使用方便（2）高效地传送大量数据

2.6.1 进程通信的类型

- 高级通信机制分为：**共享存储器系统，管道通信系统，消息传递系统，客户机-服务器系统**

1. 共享存储器系统：
   （1）基于共享数据结构的通信方式：仅适用于相对少量的数据，通信效率低，属于低级通信
   （2）基于共享存储区的通信方式：在内存中划出一块共享存储区域，诸进程可通过对该区域的读或写交换信息，实现通信，数据的形式和位置甚至访问控制都是由进程负责，而不是OS，属于高级通信
2. 管道通信系统

- 管道：指用于连接一个读进程和一个写进程以实现它们之间通信的一个共享文件，又名pipe文件。
- 提供互斥，同步，确定对方是否存在三个方面的协调能力

3. 消息传递系统

- 以格式化的消息为单位，将通信的数据封装在消息中，并利用操作系统提供的一组通信命令（原语），在进程间传递消息，完成进程间的数据交换
- （1）直接通信方式，如消息缓冲机制
- （2）间接通信方式，如电子邮箱系统

4. 客户机-服务器系统

- 实现的三种方法：套接字，远程过程调用，远程方法
- （1）套接字：是一个通信标识类型的数据结构，包含了通信目的地址，端口号，协议等，是进程通信和网络通信的基本构件
- 分类：①基于文件型②基于网络型
- （2）远程过程调用和远程方法调用：远程过程调用PRC，是一个通信协议，用于通过网络连接的系统。如果涉及的软件采用面向对象编程，那么远程过程调用亦可称做远程方法调用

2.6.2 消息传递通信的实现方式

1. 直接消息传递系统：
   （1）对称寻址原语：不足：一旦改变进程的名字，则可能需要检查所有其他进程的定义
   （2）非对称寻址方式
   （3）消息的格式
   （4）进程的同步方式
   （5）通信链路：为使在发送进程和接受进程之间能进行通信，必须在两者之间建立一条通信链路

- 分为：单向通信链路，双向通信链路

2. 信箱通信：

- 信箱通信属于间接通信方式，即进程之间的通信，需要通过某种中间实体来完成。该实体建立在随机存储器的公用缓冲区上，用来暂存发送进程发送给目标进程的消息；接受进程可以从该实体中取出发送进程发送给自己的消息，通常把这种中间实体称为邮箱（或信箱），每个邮箱都有唯一的标识符。
  （1）信箱的结构：信箱头，信箱体
  （2）信箱通信原语
  （3）信箱的类型：①私用信箱②公用信箱③共享信箱

2.7 线程的基本概念

- 线程：比进程更小的基本单位
- 线程目的：提高程序并发执行的程度

2.7.1 线程的引入

- 引入进程的目的：为了多个程序能并发执行，以提高资源利用率和系统吞吐量
- 引入线程的目的：减少程序在并发执行时所付出的时空开销，使OS具有更好的并发性

1. 进程的两个基本属性：
   （1）进程是一个可拥有资源的独立单位
   （2）进程同时又是一个可独立调度和分派的基本单位
2. 程序并发执行所需付出的时空开销
   （1）创建进程，分配所需资源，建立PCB
   （2）撤销进程，执行回收操作，撤销PCB
   （3）进程切换，需保留当前CPU环境，建立新的CPU环境
3. 线程 ——作为调度和而分派的基本单位

2.7.2 线程与进程的比较

- 传统进程：重型进程，线程：轻型进程/进程元

1. **调度的基本单位**：

- 线程间通信比进程更方便
- 同一进程的线程，计数器，堆栈资源不同
- 线程的切换，建立，撤销均快于进程
- 线程的切换不会引起进程的切换，但从一个进程中的线程切换到另一个进程的线程时，必然会引起进程的切换。

2. **并发性**：一个进程中的多个线程之间亦可以并发执行，甚至还允许在一个进程中所有线程都能并发执行
3. **拥有资源**：仅有一点必不可少的，能保证独立运行的资源，如：TCB等
4. **独立性**：同一进程中的不同线程之间的独立性要比不同进程之间的独立性低
5. **系统开销**：线程低于进程
6. 支持多处理机系统

2.7.3 线程的状态和线程控制块

1. 线程的三种状态：（1）执行（2）就绪（3）阻塞
2. **线程控制块TCB**：将所有用于控制和管理线程的信息记录其中
3. 多线程OS中的进程属性：
   （1）进程是一个可拥有资源的基本单位
   （2）多个线程可以并发执行
   （3）进程已不是可执行的实体，**在多线程OS中，是把线程作为独立运行的基本单位**

2.8 线程的实现

2.8.1 线程的实现方式

1. 内核支持线程KST
   优点：
   （1）在多处理器系统中，内核能够同时调度同一进程中的多个线程并行执行
   （2）如果进程中的一个线程被阻塞，内核可以调度该进程中的其他线程占用处理器运行，也可以运行其他进程中的线程
   （3）内核支持线程具有很小的数据结构和堆栈，线程切换快，开销小
   （4）内核本身采用多线程技术，可以提高系统的执行速度和效率
   缺点：
   系统开销大
2. 用户级线程ULT
   优点：
   （1）线程切换不需要转换到内核空间
   （2）调度算法可以是进程专用的
   （3）用户级线程的实现与OS平台无关，因为对于线程管理的代码是属于用户程序的一部分
   缺点：
   （1）系统调用的阻塞问题
   （2）进程中只有一个线程能执行
3. 组合方式

2.8.2 线程的实现

1. 内核支持线程的实现：
2. 用户级线程的实现：（中间系统）
   （1）运行时系统：实质上是用于管理和控制线程的函数的集合
   （2）内核控制线程：**轻型进程LWP**，可看作是ULT和KLT之间的映射，它把一个或多个ULT映射到一个KLT上。LWP还是核心独立调度的单位，它可以在多个处理器上并行执行

- 注意：核心向用户程序提供的界面是LWP，不提供核心线程和用户级线程，用户程序接触不到核心线程，只能看到LWP和用户态线程。
- 把这些LWP做成一个线程池，只有连接到LWP上的线程才能与内核通信
- LWP实现理内核和用户级线程之间的隔离

2.8.3 线程的创建和终止

- 线程具有生命周期
- 创建完成后返回一个线程标识符
- 线程被终止后并不立即释放它所占有的资源，只有当进程中的其他线程执行分离函数后，被终止线程才与资源分离

第三章 处理机调度与死锁

3.1 处理机调度的层次和调度算法的目标

- 调度的实质是一种资源分配，处理机调度是对处理机资源进行分配

3.1.1 处理机调度的层次

1. 高级调度：又称长程调度或**作业调度**，调度对象是作业，高级调度主要用于多道批处理系统中，而在分时和实时系统中不设置
2. 低级调度：又称**进程调度**或短程调度，调度对象是进程，最基本的调度
3. 中级调度：又称**内存调度**，目的：提高内存利用率和系统吞吐量。将那些暂时不能运行的进程，调至外存等待，此时进程的状态称为挂起

- 其中进程调度的运行频率最高
- 作业调度周期最长
- 中级调度的运行频率介于两者之间

3.1.2 处理机调度算法的目标

1. 处理机调度算法的共同目标
   （1）资源利用率：
   cpu的利用率=cpu有效工作时间/（cpu有效工作时间+cpu空闲等待时间）
   （2）公平性
   （3）平衡性
   （4）策略强制转换
2. 批处理系统的目标：
   （1）平均周转时间短
   （2）系统吞吐量高
   （3）处理机利用率高
3. 分时系统的目标
   （1）响应时间快
   （2）均衡性
4. 实时系统的目标
   （1）截至时间的保证
   （2）可预测性

3.2 作业与作业调度

3.2.1 批处理系统中的作业

1. 作业和作业步：
   （1）作业：
   - 作业是用户提交给系统的以向相对独立的工作
   - 在批处理系统中是以作业为基本单位从外存调入内存的

（2）作业步：

- 在作业运行期间，每个作业都必须经过若干个相对独立的，又相互关联的顺序加工步骤才能得到结果。我们把其中的每一加工步骤叫做一个作业步
- 一个典型的作业可以分成：编译作业步，链接装配作业步，运行作业步
  2. 作业运行的三个阶段和三种状态：

（1）收容阶段：操作员把用户提交的作业通过某种输入方式或SPOOLing系统输入到硬盘上，在为该作业建立JCB，并把它们放到作业后备队列中。此时作业的状态也叫“后备状态”
（2）运行阶段：“运行状态”
（3）完成阶段：“完成状态”

3.2.2 作业调度的主要任务

- 主要任务：根据JCB中的信息，检查系统中的资源能否满足作业对资源的需求，以及按照一定算法，将作业调入内存，创建进程，分配资源，将新建立的进程排在就绪队列上等待调度，故作业调度也叫接纳调度。
- 接纳多少个作业
- 接纳那些作业

3.2.3 先来先服务（FCFS）和短作业有限（SJF）调度算法

1. **先来先服务调度算法（FCFS）**：

- 按照作业的到达的先后次序来调度，即优先考虑等待时间最长的作业
- 缺点：对于短进程而言，带权周转时间可能非常的大
- 适用于作业调度和进程调度

2. **短作业有限的调度算法（SJF）**：

- 按照作业的长短来计算优先级，作业越短，其优先级越高

- 缺点：
  （1）必须预知作业的运行时间
  （2）对长作业非常不利
  （3）在采用SJF时，人机交互无法实现
  （4）该调度算法完全未考虑作业的紧迫程度，故不能保证紧迫作业能得到及时处理

3.2.4 优先级调度算法和高响应比调度算法

1. **优先级调度算法（PSA）**

- 根据作业的优先级进行调度

2. 高响应比优先调度算法：

优先权=（等待时间+要求服务时间）/要求服务时间=响应时间/要求服务时间

- 动态优先级
- 更公平但增加了系统开销

3.3 进程调度

3.3.1 进程调度的任务，机制和方式

1. 进程调度的任务：
   （1）保存处理机的现场信息
   （2）按某种算法选取进程
   （3）把处理器分配给进程
2. 进程调度机制
   三个部分
   （1）排队器
   （2）分派器
   （3）上下文切换器
3. 进程调度方式
   （1）非抢占方式：

> 在采用非抢占调度方式时，可能引起进程调度的因素可归结为这样几个：①正在执行的进程执行完毕，或因发生某事件而不能再继续执行；②执行中的进程因提出I/O请求而暂停执行；③在进程通信或同步过程中执行了某种原语操作，如P操作(wait操作)、Block原语、Wakeup原语等。这种调度方式的优点是实现简单、系统开销小，适用于大多数的批处理系统环境。但它难以满足紧急任务的要求——立即执行。

（2）抢占方式
主要原则：
（1）优先权原则。
（2）短作业(进程)优先原则。
（3）时间片原则。

3.3.2 轮转调度算法（RR）

- 基于时间片的轮转，非常公平
- 每个进程每次大约可获得1/n的处理机时间

1. 轮转法的基本原理：系统根据FCFS策略，将所有的就绪进程排成一个就绪队列，并每隔一段时间产生一次中断，激活系统中的进程调度，将CPU分配给队首进程，令其执行
2. 进程切换时机：
   （1）若一个时间片尚未用完，正在运行的进程便已经完成，就立即激活调度程序，将它从就绪队列中删除，再调度就绪队列队首的进程运行，并启动新的时间片
   （2）在一个时间片用完时，计时器中断处理程序被激活，如果进程尚未运行完，调度程序把它送入就绪队列末尾
3. 时间片大小的确定

- 小：频繁地执行进程调度和进程上下文切换，增加系统开销
- 大：每个进程都能在一个时间片内完成，算法退化为FCFS

3.3.3 优先级调度算法

1. 优先级调度算法分类：
   （1）非抢占式优先级调度算法：优先级低的正在运行时，优先级高的不能抢占CPU
   （2）抢占式优先级调度算法
2. 优先级的类型：
   （1）静态优先级：
   - 在创建进程时确定
   - 依据：进程类型，进程对资源的需求，用户需求
   - 简单易行，系统开销小，但不够精确
     （2）动态优先级：

- 其值随进程的推进或时间的增加而改变，以获得更好的性能

3.3.4 多队列调度算法

- 该算法将系统中的进程就绪队列从一个拆分成若干个，将不同类型或性质的进程固定分配在不同的就绪队列，不同的就绪队列采用不同的调度算法，一个就绪队列中的进程可以设置不同的优先级，不同的就绪队列本身也可以设置不同优先级

3.3.5 多级反馈队列调度算法

1. 调度机制：
   （1）设置多个就绪队列。第一个队列优先高，时间片最短，其后依此降低优先级，增大时间片

（2）每个队列内采用FCFS算法
（3）调度算法的性能：
- 能较好的满足用户的需要

3.3.6 基于公平原则的调度算法

1. 保证调度算法：n个相同类型的进程，每个都获得相同的处理机时间1/n
2. 公平分享调度算法：分配给每个进程相同的处理机时间，公平性是针对用户而言，要考虑每一个用户的进程数目

3.4 实时调度

3.4.1 实现实时调度的基本条件

1. 提供必要的信息：
   （1）就绪时间
   （2）开始截止时间和完成截止时间
   （3）处理时间
   （4）资源要求
   （5）优先级
2. 系统处理能力强
   （1）单处理机，必须满足下面的限制条件：

c是处理机时间，p是周期时间
（2）多处理机，限制条件改为：

3.采用抢占式调度机制

- 在含有HRT任务的实时系统中，广泛采用抢占式机制，满足对截止时间的要求

4. 具有快速切换机制
   （1）对中断的快速响应
   （2）快读的任务分派能力

3.4.2 实时调度算法的分类

1. 非抢占式调度算法
   （1）非抢占式轮转调度算法
   （2）非抢占式优先调度算法
2. 抢占式调度算法
   （1）基于时钟中断的抢占式优先级调度算法
   （2）立即抢占的优先级调度算法

3.4.3 最早截止时间优先EDF算法

- 任务的截止时间越早，其优先级越高，具有最早截止时间的任务排在队列的队首

1. 非抢占式调度方式用于非周期实时任务
2. 抢占式调度方式用于周期实时任务

3.4.4 最低松弛度优先LLF算法

- 任务紧急程度越高，赋予该任务的优先级就越高，以使之优先执行
- 松弛度=必须完成时间-其本身的运行时间-当前时间

3.4.5 优先级倒置

- 高优先级进程被低优先级进程延迟或阻塞

3.5 死锁

3.5.1 资源问题

1. 可重用性资源和消耗性资源

- （1）可重用性资源：用户可以重复使用多次的资源
  - 性质：（1）每一个可重用资源中的单位只能分配给一个进程使用
    （2）进程在使用时的顺序：①请求资源②使用资源③释放资源
    （3）系统中每一类可重用资源的单元数目都是固定的，进程在运行期间既不能创建也不能删除
  - 对资源的请求和释放都是利用系统调用来实现的
- （2）可消耗性资源：又称临时资源
  - 性质：（1）每一个可消耗性资源的单元数目在进程运行期间是可以不断变化的

2. 可抢占性资源和不可抢占性资源：
   （1）可抢占性资源：这类资源不会引起死锁
   （2）不可抢占性资源：例如磁带机，打印机

3.5.2 计算机系统中的死锁

- 死锁：多个进程对资源的争夺，不仅对不可抢占性资源争夺会引起死锁，对可消耗性资源的争夺也会引起死锁

1. 竞争不可抢占性资源引起死锁
2. 竞争可消耗资源引起死锁
3. 进程推进顺序不当引起死锁

3.5.3 死锁的定义，必要条件和处理方法

1. 死锁的定义：如果一组进程中的每一个进程都在等待仅由该组进程中的其他进程才能引发的事件，那么该组进程是死锁的
2. **产生死锁的必要条件**：

- （1）互斥条件
- （2）请求和保持条件
- （3）不可抢占条件
- （4）循环等待条件

3. 处理死锁的方法：

- （1）预防死锁：设置某些限制条件，破坏产生死锁的四个必要条件
- （2）避免死锁：在资源的动态分配过程中采取某种方法，使之保持安全状态
- （3）检测死锁
- （4）解除死锁：撤销进程，回收资源

3.6 预防死锁

- 互斥条件是非共享设备所必须的，不仅不能改变，还应加以保证，因此主要破坏产生死锁的后三个条件

3.6.1 破坏“请求和保持条件”

1. 第一种协议：所有进程在开始运行钱，必须一次性申请所需全部资源
   - 简单，易行且安全，但①资源被严重浪费，②进程经常发送饥饿现象
2. 第二种协议：
   - 允许进程只获取运行初期所需的资源，便可开始运行，在逐步释放已用毕的全部资源，再请求新的资源
   - 更快，提高设备的利用率，减少饥饿机率

3.6.2 破坏“不可抢占”条件

- 一个已经保持了某些不可抢占资源的进程，提出新的资源请求而不能得到满足时，必须释放已经保持的多有资源，待以后再申请
- 实现复杂，延长了进程的周转事件，增加系统开销，降低系统吞吐量

3.6.3 破坏“循环等待” 条件

- 对系统所有资源类型进行线性排序，并赋予不同的序号，必须按照序号递增的顺序请求资源
- 如果需要多个同类资源单元，则必须一起请求
- 假如进程已请求到一些序号较高的资源后，后来它又想请求序号低的资源，必须先释放所有具有相同和更高序号的资源后才能申请

避免死锁

- 再资源的动态分配过程中，防止系统进入不安全状态，以避免发送死锁

3.7.1 系统安全状态

1. 安全状态：是指系统能按某种顺序，来为每个进程分配其所需资源，直至最大需求，使每个进程都可顺利完成。
2. 安全序列：
   一个进程序列{P1，...，Pn}是安全的，如果对于每一个进程Pi(1≤i≤n），它以后需要的资源量不超过系统当前剩余资源量与所有进程Pj (j < i)当前占有资源量之和，系统处于安全状态。
   (安全状态一定是没有死锁发生的)

3.7.2 利用银行家算法避免死锁

> 银行家算法（Banker's Algorithm）

- 概念
  银行家算法是一个避免死锁产生的算法。以银行借贷分配策略为基础，判断并保证系统处于安全状态。
  客户在第一次申请贷款时，声明所需最大资金量，在满足所有贷款要求并完成项目时，及时归还
  在客户贷款数量不超过银行拥有的最大值时，银行家尽量满足客户需要

3.8 死锁的检测和解除

允许系统进入死锁状态
维护系统的资源分配图
定期调用死锁检测算法来搜索图中是否存在死锁
出现死锁时，用死锁恢复机制进行恢复

- 解除死锁：
  （1）资源剥夺法：挂起某些死锁进程并抢夺它的资源，以便让其他进程继续推进
  （2）撤销进程法：强制撤销部分、甚至全部死锁进程并剥夺这些进程的资源
  （3）进程回退法：让进程回退到足以回避死锁的地步
- 算法：
  (1)银行家算法为死锁避免算法；
  (2)死锁检查算法和资源分配图化简法为死锁检测；
  (3)资源有序分配算法为死锁预防策略；
- 所谓CPU繁忙型的作业，是指该类作业需要大量的CPU时间进行计算，而很少请求I/O操作。I/O繁忙型的作业是指CPU处理时，需频繁的请求I/O操作。
  (1)周转时间 = 作业完成时间 - 作业提交时间；
  (2)平均周转时间 = （作业1的周转时间 + ... + 作业n的周转时间）/ n；
  (3)带权周转时间 = 作业周转时间 / 作业实际运行时间；
  (4)平均带权周转时间 = （作业1的带权周转时间 + ... + 作业n的带权周转时间）/ n；
  (5)响应比Rp = (等待时间 + 要求服务时间) /要求服务时间；

第四章 存储器管理

内存管理：

- 引入目的：更好的支持多道程序并发执行，提升系统性能
  -‘程序的编译’：由编译程序将用户源代码编译成若干个目标模块；
  -‘程序的链接’：由链接程序将编译后形成的一组目标模块，以及所需库函数链接在一起，形成一个完整的装入模块；
  （1）静态链接：在程序运行之前链接
  （2）装入时动态链接：在装入内存时，采用边装入边链接的链接方式
  （3）运行时动态链接：在程序执行中需要该目标模块时，才对它进行链接
  -‘程序的装入’：由装入程序将装入模块装入内存运行；
  （1）绝对装入：适合单道程序环境
  （2）静态重定位：适合装入之后不再移动的情况
  （3）动态重定位：适合装入之后还会移动的情况
- ‘地址空间’：
  （1）逻辑地址空间：是指一个源程序在编译或者链接装配后指令和数据所用的所有相对地址的空间；
  （2）物理地址空间：内存中物理单元的集合；
- ‘地址重定位’：通过地址转换将逻辑地址转换为物理地址。
- ‘内存保护’：
  （1）上、下限寄存器：分别与上、下限寄存器比较
  （2）基址、限长寄存器：与限长寄存器比较，与基址寄存器相加
- 管理方式：
  （1）连续分配：产生内部碎片；用户进程（或作业）在主存中都是连续存放的
  - 单一连续分配：分配到内存固定区域，只适合单任务系统；
  - 固定分区分配：分配到内存中不同的固定区域，分区可以相等也可以不等；
    - 产生内部碎片
  - 动态分区分配：
    - 产生外部碎片
    - 基本概念：按照程序的需要进行动态的划分
    - 分配算法：
      （1）首次适应：空闲区按地址从小到大为序，分配第一个符合条件的分区；
      （2）最佳适应：空闲区按空间大小从小到大排序，分配第一个符合条件的分区；
      （3）最坏适应：空闲区按空间从大到小排序，分配第一个符合条件的分区；
      （4）邻近适应：空闲区按地址地址递增的次序排列，分配内存时从上次查找结束的位置开始继续查找；

（2）非连续分配：允许一个程序分散地装入到不相邻的内存分区中，需要额外的空间去存储分散区域的索引

- 基本分页：内存分为固定的块，按物理结构划分，会有内部碎片；
  - 主存、进程都划分为大小固定的块，进程在执行时，以块为单位申请主存中的块空间；
  - 进程中的块为页，内存中的块为页框。系统为每个进程建立一张页表，页表记录页面在内存中对应的物理块号，实现从页号到物理块号的地址映射；
  - 页式管理中地址空间是一维的；
- 基本分段：内存块的大小不固定，按逻辑结构划分，会有外部碎片；
  - 段式管理方式按照用户进程中的自然段划分逻辑空间。段内要求连续，段间不要求连续。段号和段内偏移量必须由用户显示提供。
  - 方便编程、共享、保护、动态链接和增长。
- 段页式：基本分段和基本分页的结合，会有内部碎片；
  - 作业的逻辑地址分为：段号、页号和页内偏移量；采用分段方法来分配和管理用户地址空间，采用分页方法来管理物理存储空间；开销大。
- 请求分页存储管理：采用虚拟技术，开始运行时不必将作业全部一次性装入内存；
- 多级页表：将页表的10页空间也进行地址映射，建立上一级页表，用于存储页表的映射关系；

多道程序下的内存扩充：

- 覆盖：预先设定覆盖段，覆盖掉暂时不用的内容，通常在同一个程序之中进行；
- 交换：把处于等待的程序暂时移到外存，通常在不同程序之间进行；

- 虚拟内存：只能基于非连续分配技术。
  - 引入原因：在逻辑上扩充内存
  - 时间局部性：程序中存在着大量的循环操作；
  - 空间局部性：程序在一段时间内所访问的地址，可能集中在一定的范围内；
  - 组成部分：
    （1）页表机制：通过查表获取相关信息；
    （2）中断机制：要访问页不在内存时产生缺页中断
    （3）地址变换机构：把逻辑地址变换成物理地址
    （4）内存和外存：需要一定容量的内存和外存支持
  - 置换算法：
    （1）最佳置换算法（OPT）：选择以后不用的页面
    （2）先进先出（FIFO）：选择最先装入的页面
    （3）最近最久未使用（LRU）：选择最近最近未使用的页面
    （4）时钟置换算法（最近未用算法）：选择最近未用的页面
    （5）改进型CLOCK：考虑页面修改问题
  - 地址翻译：TLB->页表（TLB不命中）->Cache->主存->外存

页面分配策略：

- 固定分配局置换：每个进程分配一定数目的物理块，在整个运行期间不变，缺页时只在该进程在内存中的页面中进行置换；
- 可变分配全局置换：为每个进程分配一定数目的物理块，操作系统自身也保持一个空闲物理块队列；
- 可变分配局部置换：若进程在运行中频繁地缺页，系统再为该进程分配若干物理块；
- 抖动（颠簸）：刚换出的页面马上又要换入内存；刚换入的页面马上就要换出内存；
- 工作集（驻留级）：
  - 指在某段时间间隔内，进程要访问的页面集合。
- 虚拟内存空间大小：
  - <=内存容量和外存容量之和<
  - <=计算机的地址位数能容纳的最大容量
- 虚拟存储的页表项：
  - 页号
  - 物理块号
  - 状态位P：用于指示该页是否已调入内存，供程序访问参考；
  - 访问字段A：用于记录本页在一段时间内被访问的次数，或记录本页最近已有多长时间未被访问，供置换算法换出页面时参考；
  - 修改位M：标识该页在调入内存后是否被修改过；
  - 外存地址
- Belady现象：进程的缺页次数随着分配给进程的页框个数的增加而增加，只有FIFO队列式页面置换算法才有。
- 快表（联想寄存器TLB）：用来存放当前访问的若干页表项，以加速地址变换的过程，若所需访问页号在快表中则可减少一次内存访问。

第五章 虚拟存储器

略

第六章 输入输出系统

- I/O管理概述：状态跟踪、设备存取、设备分配、设备控制
- 设备分类：按传输速率分：
  低速：如磁盘、鼠标中速：如行式打印机、激光打印机高速：如磁带机、磁盘机、光盘机按信息交换单位分：
  块设备：如磁盘字符设备：如键盘、打印机
- 控制方式：程序直接控制：程序直接对设备特环测试中断驱动：引入中断机制，当设备准备完成时发生中断DMA：在I/O设备与主存之间开辟直接数据通路，彻底“解放”CPU。
  基本数据单位是块；传送的数据从设备直接送入内存（或相反）仅在传送一个或多个数据块的开始和结束时，才需要CPU干预，整块数据的传送是在DMA控制器的控制下完成的；包含的四类控制器：
  命令/状态寄存器（CR）内存地址寄存器（MAR）数据寄存器（DR）数据计数器（DC）
- 通道控制：引入专门的I/O处理机进行管理
- I/O子系统层次：用户层I/O软件：实现与用户交互的接口设备独立性软件：实现用户程序与设备驱动器的统一接口、设备命令、设备保护以及设备分配与释放设备驱动程序：与硬件直接相关，负责具体实现系统对设备发出的操作指令中断处理程序：用于处理中断相关事项硬件设备：包括一个机械部件（设备本身）和一个电子邮件（控制器）
- I/O核心子系统：
  I/O调度：确定一个好的顺序来执行这些I/O请求
  磁盘高速缓存：指利用内存中的存储空间来暂存从磁盘上读出的一系列盘块中的信息；逻辑上属于磁盘，物理上属于内存；1：在内存中开辟一个单独的存储空间作为磁盘高速缓存，大小固定2：把未利用的内存空间作为一个缓冲池，供请求分页系统和磁盘时I/O共享
  缓冲区：位于内存区域
  特点：当缓冲区的数据非空的时候，不能往缓冲区冲入数据，只能从缓冲区把数据传出；为空时，可以冲入数据，但必须充满遗憾才能再传出。
  引入缓冲区的目的：缓和CPU与I/O设备间速度不匹配的矛盾减少对CPU的中断频率，放宽对CUP中断响应时间的限制解决基本数据单元大小不匹配的问题提高CPU和I/O设备之间的并行性
  单缓冲
  双缓冲
  循环缓冲
  缓冲池

比较	高度缓存	缓冲区
存放数据	存放的是低速设备上的某些数据的复制数据	存放的是低速设备传递给高速设备的数据（或相反）
目的	高速缓存存放的是高速设备经常要访问的数据	高速设备和低速设备的通信都要经过缓冲区，高速设备永远不会直接去访问低速设备
相同点	都是介于高速设备和低速设备之间	都是介于高速设备和低速设备之间

- 设备的分配与回收：
  - 分类：
    （1）独点式使用设备：设备被使用时不再允许其他进程使用设备
    （2）分时共享式使用设备：设备没有独占使用的要求时，可以通过分时共享使用
    （3）SPOOLing技术：将独占设备改造成共享设备,实现了虚拟设备的功能；以空间换时间，必须先有独占设备
  - 设备分配的数据结构：
    （1）设备控制表（DCT）：每个设备配置一张DCT，以记录本设备的情况；
    （2）控制器控制表（COCT）：每个控制器有一张COCT；
    （3）通道控制表（CHCT）：每个通道配置一张CHCT；
    （4）系统设备表（SDT）整个系统只有一张SDT，记录已连接到系统中的所有物理设备的情况
    （5）SDT中有一个DCT指针，DCT中有一个COCT指针，COCT中有一个CHCT指针，CHCT中有一个COCT指针。
  - 分配原则：即要求充分发挥设备的使用效率，又要避免造成进程死锁，还要将用户程序和具体设备隔离开
  - 分配方式：
    （1）静态分配：在用户作业开始执行前，由系统一次性分配该作业所要求的全部设备
    （2）动态分配：在进程执行过程中根据执行需要进行分配
  - 设备分配的安全性：
    （1）安全分配方式：每当进程发出I/O请求后便进入阻塞状态，直到其I/O操作完成时才被唤醒。
    （2）不安全分配方式：进程发出多个I/O请求并继续运行，仅当进程所请求的设备已被另一进程占用时，才进入阻塞状态。
  - 设备独立性是指应用程序独立于具体使用的物理设备
  - SPOOLing技术：主要包括输入井、输出井、输入缓冲区和输出缓冲区以及输入进程和输出进程。
  - 输入井和输出井是在磁盘上开辟的两大存储空间；
    （1）输入井是模拟脱机输入时的磁盘设备，用于暂存I/O设备输入的数据
    （2）输出井是模拟脱机输出时的磁盘，用于暂存用户程序的输出数据

第七章 文件管理

- 文件控制块（FCB），类似进程管理的PCB，存放控制文件需要的各种信息的数据结构。
  - 基本信息：包括文件物理位置
  - 存取控制信息
  - 使用信息
  - 索引结点一个
- 文件对应一个FCB，而一个文件目录项就是一个FCB。
- 打开文件操作是讲该文件的FCB存入内存的活跃文件目录表，而不是将文件内容负责到主存，找到指定文件目录是打开文件之前的操作。

文件系统基础：

- 逻辑结构：
  - 无结构文件（流式文件）：将数据按顺序组织成记录并积累保存，（流式文件）则被看成是一个字符流，以字节（Byte）为单位；
  - 有结构文件：
    （1）顺序文件：
    - 串结构：记录之间的顺序与关键字无关
    - 顺序结构：记录之间的顺序与关键字有关
      （2）索引文件：为变长文件建立索引表，提高查找速度
      （3）索引顺序文件：顺序文件和索引文件的结合，将顺序文件中的所有记录分为若干组，为顺序文件建立一张索引表，在索引中为每组的第一个记录建立一个索引项，其中含有该记录的关键字值和指向该记录的指针
      （4）直接文件（查找文件）Hash File：通过哈希函数直接决定记录地址
- 目录结构：
  - 单级：全部文件都放在一个目录下
  - 两级：在目录下分出用户目录
  - 多级：将两级结构加以推广，采用树形结构
  - 无环图：在树形结构上加入一些有向边，便于共享

文件共享：

- 基于索引结点（硬链接）：共享文件指向同一个索引节点；链接计数count；
- 基于符号链（软链接）：有文件拥有者才拥有指向其索引结点的指针，共享该文件的其他用户则只有该文件的路径；

文件保护：

- 口令保护：通过口令访问文件
- 加密保护：对文件进行加密处理
- 访问控制：根据访问者的身份进行限制

文件系统实现：

- 目录实现：线性列表：
  - 无序：查找文件较慢，新建文件较快
  - 有序：查找文件较快，新建文件较慢
  - 哈希表：查找、新建文件都较快，要处理冲突
- 文件实现：
  - 连续分配：在磁盘上连续存放文件
  - 链接分配：隐式：采用类似链表的结构，显式：把隐式文件中的指针单独抽离出来
  - 索引分配：每个文件所有的盘块号都集中存放，建立索引表

存储空间管理：

- 空闲表：把所有空闲块组织成表
- 空闲链表法：把所有空闲块组织成链表
- 位示图：利用二进制的每位记录空闲块
- 成组链接：空闲表和空闲链表的结合，适合大的文件系统

第八章 磁盘存储器的管理

磁盘管理

- 磁盘地址结构：柱面号、盘面号、扇区号
- 读写时间：
  （1）寻道时间：将磁头移动到指定磁道所需要的时间
  （2）延迟时间：磁头定位到某一磁道的扇区所需要的时间
  （3）传输时间：从磁盘读出或向磁盘写入数据所经历的时间
  （4）启动时间（一般忽略）：控制器的启动时间
- 调度算法：
  （1）先来先服务（FCFS）：根据进程请求访问磁盘的先后顺序进行调度
  （2）最短寻找时间优先（SSTF）：选择当前磁头所在的磁道距离最近的磁道
  （3）扫描（SCAN）算法（电梯算法）：在磁头当前移动方向上选择与当前磁头所在磁道距离最近的请求
  （4）循环扫描（C-SCAN）：在扫描算法的基础上规定磁头单向移动来提供服务
- 磁盘管理：
  - 初始化：对磁盘进行低级格式化和逻辑化
  - 引导块：存放自举程序
  - 坏块：对于损坏扇区的处理