- 操作系统
  - 概述
    - 基本特征
      - 并发
        - 并发是指宏观上在一段时间内能同时运行多个程序
        - 并行则指同一时刻能运行多个指令，并行需要硬件支持，如多流水线、多核处理器或者分布式计算系统。
        - 操作系统通过引入进程和线程，使得程序能够并发运行
      - 共享
        - 共享是指系统中的资源可以被多个并发进程共同使用
        - **共享方式** 互斥共享和同时共享
        - 互斥共享的资源称为临界资源，例如打印机等，在同一时刻只允许一个进程访问，需要用同步机制来实现互斥访问。
      - 虚拟
        - 虚拟技术把一个物理实体转换为多个逻辑实体，主要有两种虚拟技术：时分复用技术和空分复用技术。
        - **时分复用** 多个进程能在同一个处理器上并发执行，让每个进程轮流占用处理器，每次只执行一小个时间片并快速切换。
        - **空分复用** **虚拟内存**将物理内存抽象为地址空间，每个进程都有各自的地址空间。
        - 地址空间的页被映射到物理内存，地址空间的页并不需要全部在物理内存中，当使用到一个没有在物理内存的页时，执行页面置换算法，将该页置换到内存中。
      - 异步 异步指进程不是一次性执行完毕，而是走走停停，以不可知的速度向前推进。
    - 基本功能
      - **进程管理** 进程控制、进程同步、进程通信、死锁处理、处理机调度等。
      - **内存管理** 内存分配、地址映射、内存保护与共享、虚拟内存等。
      - **文件管理** 文件存储空间的管理、目录管理、文件读写管理和保护等。
      - **设备管理**
        - 缓冲管理、设备分配、设备处理、虚拟设备等。
        - 完成用户的 I/O 请求，方便用户使用各种设备，并提高设备的利用率。
    - **系统调用** **如果一个进程在用户态需要使用内核态的功能，就进行系统调用从而陷入内核，由操作系统代为完成。**
    - 宏内核和微内核
      - **宏内核** 宏内核将操作系统功能作为一个紧密结合的整体放到内核。由于各模块共享信息，因此有很高的性能。
      - **微内核**
        - 将一部分操作系统功能移出内核，从而降低内核的复杂性。
        - 在微内核结构下，操作系统被划分成小的、定义良好的模块，只有微内核这一个模块运行在内核态，其余模块运行在用户态
        - 因为需要频繁地在用户态和核心态之间进行切换，所以会有一定的性能损失
    - 中断分类
      - **外中断** 由 CPU 执行指令以外的事件引起，如 I/O 完成中断，表示设备输入/输出处理已经完成，处理器能够发送下一个输入/输出请求。
      - 此外还有时钟中断、控制台中断等。
      - **异常** 由 CPU 执行指令的内部事件引起，如非法操作码、地址越界、算术溢出等。
      - **陷入** 在用户程序中使用系统调用。
  - 进程管理
    - 进程与线程
      - **进程**
        - 进程是**资源分配**的基本单位
        - 进程控制块 (Process Control Block, PCB) 描述进程的基本信息和运行状态，所谓的创建进程和撤销进程，都是指对 PCB 的操作。
      - **线程**
        - 线程是**独立调度**的基本单位
        - **一个进程中可以有多个线程，它们共享进程资源。**
      - **区别**
        - **拥有资源**
          - 进程是资源分配的基本单位
          - 线程不拥有资源，线程可以访问隶属进程的资源。
        - **调度**
          - 线程是独立调度的基本单位
          - 在同一进程中，线程的切换不会引起进程切换，从一个进程中的线程切换到另一个进程中的线程时，会引起进程切换。
        - **系统开销**
          - 由于创建或撤销进程时，系统都要为之分配或回收资源，如内存空间、I/O 设备等，所付出的开销远大于创建或撤销线程时的开销。
          - 进程切换时，涉及当前执行进程 CPU 环境的保存及新调度进程 CPU 环境的设置，而线程切换时只需保存和设置少量寄存器内容，开销很小。
        - **通信**
          - 线程间可以通过直接读写同一进程中的数据进行通信
          - 但是进程通信需要借助 IPC（InterProcess Communication）进程间通信
    - 进程状态切换
      - **创建态 created**
      - **就绪态 ready** 等待被调度【CPU时间】
      - **运行态 running**
        - 只有就绪态和运行态可以相互转换，其它的都是单向转换。
        - 就绪态的进程通过**进程调度算法**从而获得 CPU 时间，转为运行态
        - 运行态的进程，在分配给它的 CPU 时间片用完之后就会转为就绪态，等待下一次调度
      - **阻塞态 waiting** 缺少需要的资源【不包括CPU时间】从运行态转换为阻塞态
      - **终止态 terminated**
    - 进程调度算法
      - 批处理系统
        - **先来先服务 FCFS**
          - **非抢占式**的调度算法，按照请求的顺序进行调度
          - 有利于长作业，但不利于短作业，因为短作业必须一直等待前面的长作业执行完毕才能执行
        - **短作业优先 SJF**
          - **非抢占式**的调度算法，按估计运行时间最短的顺序进行调度。
          - 长作业有可能会饿死，处于一直等待短作业执行完毕的状态。
        - **最短剩余时间优先 SRTN**
          - 最短作业优先的抢占式版本，按剩余运行时间的顺序进行调度。
          - 新的作业到达时，其运行时间与当前进程的剩余时间比较。如果新的进程需要的时间更少，则挂起当前进程，运行新的进程。
      - 交互式系统
        - **时间片轮转**
          - 将所有就绪进程按 FCFS 的原则排成一个队列，每次调度时，把 CPU 时间分配给队首进程，该进程可以执行一个时间片。
          - 当时间片用完时，由计时器发出时钟中断，调度程序便停止该进程的执行，并将它送往就绪队列的末尾，同时继续把 CPU 时间分配给队首的进程。
          - 算法的效率和时间片的大小有很大关系：如果时间片太小，会导致进程切换得太频繁，浪费过多时间。而如果时间片过长，那么实时性就不能得到保证。
        - **优先级调度** 为每个进程分配一个优先级，按优先级进行调度。为了防止低优先级的进程永远等不到调度，可以随着时间的推移增加等待进程的优先级。
        - **多级反馈队列**
          - 时间片轮转调度算法和优先级调度算法的结合。
          - 设置了多个队列，每个队列时间片大小都不同，例如 1,2,4,8,...。进程在第一个队列没执行完，就会被移到下一个队列。
          - 每个队列优先权也不同，最上面的优先权最高。因此只有上一个队列没有进程在排队，才能调度当前队列上的进程。
      - 实时系统
        - 实时系统要求一个请求在一个确定时间内得到响应。
        - 硬实时：必须满足绝对的截止时间
        - 软实时：可以容忍一定的超时
    - 进程同步
      - **临界区**
        - 对临界资源进行访问的那段代码称为临界区。
        - 为了互斥访问临界资源，每个进程在进入临界区之前，需要先进行检查。
      - **同步与互斥**
        - 同步：多个进程因为合作产生的直接制约关系，使得进程有一定的先后执行关系。
        - 互斥：多个进程在同一时刻只有一个进程能进入临界区。
      - **信号量**
        - 信号量（Semaphore）是一个整型变量，可以对其执行 down 和 up 操作，也就是常见的 P 和 V 操作。
        - 如果信号量的取值只能为 0 或者 1，那么就成为了 **互斥量（Mutex）** ，0 表示临界区已经加锁，1 表示临界区解锁。
        - down : 如果信号量大于 0 ，执行 -1 操作；如果信号量等于 0，进程睡眠，等待信号量大于 0；
up ： 对信号量执行 +1 操作，唤醒睡眠的进程让其完成 down 操作。
        - down 和 up 操作需要被设计成原语，不可分割，通常的做法是在执行这些操作的时候屏蔽中断。
        - **使用信号量实现生产者-消费者问题**
      - **管程monitor**
        - 使用信号量机制实现的生产者消费者问题需要客户端代码做很多控制，而**管程把控制的代码独立出来**，不仅不容易出错，也使得客户端代码调用更容易。
        - 在一个时刻只能有一个进程使用管程；进程在无法继续执行的时候不能一直占用管程，否则其它进程永远不能使用管程。
        - 管程引入了条件变量以及相关的操作：wait() 和 signal() 来实现同步操作。对条件变量执行 wait() 操作会导致调用进程阻塞，把管程让出来给另一个进程持有。signal() 操作用于唤醒被阻塞的进程。
    - 经典同步问题
      - 生产者-消费者问题
      - 读者-写者问题
    - 进程通信
      - **进程同步和进程通信区别**
        - 进程同步：控制多个进程按一定顺序执行
        - 进程通信：进程间传输信息
      - **管道**
        - 管道是通过调用 pipe 函数创建的，fd[0] 用于读，fd[1] 用于写。
        - 只支持半双工通信（单向交替传输）；只能在父子进程或者兄弟进程中使用。
      - **FIFO**
        - FIFO也称为命名管道，去除了管道只能在父子进程中使用的限制。
        - FIFO 常用于客户-服务器应用程序中，FIFO 用作汇聚点，在客户进程和服务器进程之间传递数据。
      - **消息队列**
        - 消息队列可以独立于读写进程存在，从而避免了 FIFO 中同步管道的打开和关闭时可能产生的困难
        - 避免了 FIFO 的同步阻塞问题，不需要进程自己提供同步方法；读进程可以根据消息类型有选择地接收消息，而不像 FIFO 那样只能默认地接收。
      - **信号量** 它是一个计数器，用于为多个进程提供对共享数据对象的访问。
      - **共享存储**
        - 允许多个进程共享一个给定的存储区，需要使用信号量用来同步对共享存储的访问。
        - 因为数据不需要在进程之间复制，所以这是最快的一种 IPC-进程间通信。
      - **套接字** 与其它通信机制不同的是，它可用于不同机器间的进程通信。
  - 死锁
    - 产生的必要条件
      - **互斥**：每个资源要么已经分配给了一个进程，要么就是可用的。
      - **占有和等待**：已经得到了某个资源的进程可以再请求新的资源。
      - **不可抢占**：已经分配给一个进程的资源不能强制性地被抢占，它只能被占有它的进程显式地释放。
      - **环路等待**：有两个或者两个以上的进程组成一条环路，该环路中的每个进程都在等待下一个进程所占有的资源。
    - 处理方法
      - **鸵鸟策略** 忽略死锁，因为解决死锁问题的代价很高，且发生死锁时不会对用户造成多大影响
      - **死锁检测与恢复**
        - **死锁检测**
          - 每种类型一个资源的死锁检测
          - 每种类型多个资源的死锁检测
        - **死锁恢复**
          - 利用抢占恢复
          - 利用回滚恢复
          - 通过杀死进程恢复
      - **死锁预防** 破坏互斥条件、占用和等待条件、不可抢占条件、环路等待
      - **死锁避免**
        - 安全状态检测
        - 银行家算法-安全状态检测
  - 内存管理
    - **虚拟内存**
      - 虚拟内存的目的是为了让物理内存扩充成更大的逻辑内存，从而让程序获得更多的可用内存
      - 虚拟内存允许程序不用将地址空间中的每一页都映射到物理内存，也就是说一个程序不需要全部调入内存就可以运行，这使得有限的内存运行大程序成为可能。
      - 假如一台计算机 16 位地址，那么一个程序的地址空间范围是 0~64K。该计算机只有 32KB 的物理内存，虚拟内存技术允许该计算机运行一个 64K 大小的程序。
    - **分页系统地址映射**
      - 内存管理单元（MMU）管理着地址空间和物理内存的转换，其中的页表（Page table）存储着页（程序地址空间）和页框（物理内存空间）的映射表。
      - 一个虚拟地址分成两个部分，一部分存储页面号，一部分存储偏移量。
    - **页面置换算法**
      - **应用场景** 如果要访问的页面不在内存中，发生缺页中断将该页调入内存中。此时如果内存已无空闲空间，则从内存调出一个页面
      - **目标** 使页面置换频率最低（也可以说缺页率最低）
      - **最佳 OPT** 被换出的页面将是最长时间内不再被访问
      - **最近最久未使用 LRU** 最近最久未使用的页面换出
      - **先进先出 FIFO** 选择换出的页面是最先进入的页面
      - **第二次机会算法** 对FIFO的改进
当页面被访问 (读或写) 时设置该页面的 R 位为 1。需要替换的时候，检查最老页面的 R 位。如果 R 位是 0，那么这个页面既老又没有被使用，可以立刻置换掉
如果是 1，就将 R 位清 0，并把该页面放到链表的尾端，修改它的装入时间使它就像刚装入的一样，然后继续从链表的头部开始搜索。
      - **时钟 Clock** 使用环形链表将页面连接起来，再使用一个指针指向最老的页面
    - **分段**
      - **应用场景** 如果使用分页系统的一维地址空间，动态增长的表会导致覆盖问题的出现
      - **做法** 分段的做法是把每个表分成段，一个段构成一个独立的地址空间。每个段的长度可以不同，并且可以动态增长
    - **段页式** 程序的地址空间划分成多个拥有独立地址空间的段，每个段上的地址空间划分成大小相同的页。这样既拥有分段系统的共享和保护，又拥有分页系统的虚拟内存功能。
    - **分页与分段的比较**
      - **对程序员的透明性** 分页透明，但是分段需要程序员显式划分每个段
      - **地址空间的维度** 分页是一维地址空间，分段是二维的
      - **大小是否可以改变** 页的大小不可变，段的大小可以动态改变
      - **出现的原因** 分页用于实现虚拟内存获得更大的地址空间；分段为了使程序和数据可以被划分为逻辑上独立的地址空间并且有助于共享和保护
  - 设备管理
    - **磁盘结构**
      - 盘面（Platter）：一个磁盘有多个盘面
      - 磁道（Track）：盘面上的圆形带状区域，一个盘面可以有多个磁道
      - 扇区（Track Sector）：磁道上的一个弧段，一个磁道可以有多个扇区，它是最小的物理储存单位
      - 磁头（Head）：与盘面非常接近，能够将盘面上的磁场转换为电信号（读），或者将电信号转换为盘面的磁场（写）
      - 制动手臂（Actuator arm）：用于在磁道之间移动磁头
      - 主轴（Spindle）：使整个盘面转动
    - **磁盘调度算法**
      - **先来先服务FCFS** 按照磁盘请求的顺序进行调度
      - **最短寻道时间优先SSTF** 优先调度与当前磁头所在磁道距离最近的磁道
      - **电梯算法/扫描算法 SCAN** 总是按一个方向来进行磁盘调度，直到该方向上没有未完成的磁盘请求，然后改变方向