操作系统

* 内存对齐
  + 编译器将程序中的每个数据单元都安排在适当的位置，例如按照某种规则将定义的结构体成员放在合适的地址偏移位置上存储
  + 内存是以字节存储，但是处理器存取内存是以块为单位，块可以是2,4,8,6字节大小，内存对齐可以减少CPU读取内存的次数，提升CPU获取内存的效率
  + 一个空的结构体或类的对象是1个字节大小，结构体成员的声明顺序会影响结构体的大小
* 并发和并行
  + 并发
    - 一个CPU，多个任务
    - 一段时间内多个任务同时运行，多个任务互相抢占资源，一个进程的指令和另一个进程的指令是交错运行的
  + 并行
    - 多个CPU，多个任务
    - 同一时刻多个任务并行运行，不是交错运行，多CPU才会并行，不会互相抢占资源
* 进程
  + 资源分配的最小单位，由内核来调度维护，有独立的虚拟地址空间，每启动一个进程，系统都会为其分配地址空间独立的地址空间即是优点也是缺点：一个进程不可能覆盖另一个进程的虚拟内存；进程间共享信息困难，必须使用显式的IPC进程间通信机制，所以进程效率较低
  + 一个经典的定义就是一个执行中程序的实例系统的每个程序都运行在某个进程的上下文中
    - 上下文
      * 上下文是由程序正确运行所需要的状态组成的这个状态包括存放在内存中的程序代码和数据，它的栈，寄存器内容，程序计数器，环境变量，打开文件描述符的集合
* 线程
  + 一个线程就是运行在单一进程上下文中的逻辑流
    - 一个程序至少有一个进程，一个进程至少有一个线程线程依赖于进程而存在
  + 线程是程序执行的最小单位，没有独立的地址空间，使用相同的地址空间共享数据
  + 线程占用的资源要比进程少的多，CPU创建、切换线程比进程的开销小
  + 线程间通信方便，同一进程下，线程共享全局变量，静态变量等数据
* 多线程和多进程区别
  + 数据共享、同步
    - 多进程
      * 数据是分开的，共享复杂，需要用IPC，同步简单
    - 多线程
      * 多线程共享进程数据；共享简单，同步复杂
  + 内存占用
    - 多进程
      * 内存占用多，切换复杂，CPU利用率低
    - 多线程
      * 内存占用少，切换简单，CPU利用率高
  + 创建销毁切换
    - 多进程
      * 进程的创建销毁和切换复杂，速度慢
    - 多线程
      * 简单，速度快
  + 编程调试
    - 多进程
      * 编程简单，调试简单
    - 多线程
      * 编程复杂，调试复杂
  + 可靠性
    - 多进程
      * 一个进程死掉不会对其他进程造成影响（有独立的地址空间）
    - 多线程
      * 一个线程死掉，整个进程都会死掉（共享地址空间），不易维护
  + 分布式
    - 多进程
      * 适用于多核、多机分布；如果一台机器不够，扩展到多台机器比较简单
    - 多线程
      * 适应于多核分布
  + 需要频繁创建切换销毁的选用多线程，要求共享某些变量的用多线程对安全性要求高的用多进程
* 线程同步
  + 当一个线程对内存共享数据进行操作时，其他线程都不可以对这个内存地址进行操作，直到该线程完成操作
  + 互斥锁
    - mutex采用互斥锁保护临界区，防止竞争条件出现。当某个线程无法获取互斥锁时，该线程会被挂起，当其他线程释放互斥锁后，操作系统会唤醒被挂起在这个锁上的线程，让其运行。比如对全局变量的访问，用要加锁，操作完了就解锁
  + 条件锁
    - 就是条件变量，是用来等待而不是用来上锁的，通常条件变量和互斥锁同时使用，条件由互斥量保护，一个线程等待条件变量的条件成立而挂起，另一个线程使条件成立
  + 自旋锁
    - 如果自旋锁已经被别的线程持有，调用线程不会阻塞，就会一直循环检测是否该自旋锁已经被释放，消耗CPU持有锁的时间比较短的时候使用自旋锁
  + 读写锁，乐观锁，悲观锁
    - 适合读多写少；乐观锁以为别的线程不会修改数据不会加锁，更新数据以数据的版本号判断悲观锁以别的线程会去修改数据，所以每次线程获取数据都会加锁
  + 信号量
    - semphare允许同一时刻多个线程访问同一资源，但是需要控制同一时刻访问此资源的最大线程数量
  + 事件（信号）
    - wait/notify，允许一个线程在处理完一个任务后，主动唤醒另一个线程执行任务，还可以实现多线程优先级的比较操作
* 进程间通信
  + 管道pipe
    - 只能用于父子或兄弟进程间的通信
    - 管道是半双工的，数据只能向一个方向流动；需要双方通信时，需要建立两个管道
    - 一个进程向管道中写的内容被管道另一端的进程读出，，写入的内容每次都添加在管道缓冲区的末尾并且每次都是从缓冲区的头部读出数据
  + 命名管道FIFO
    - 除了具有管道的功能外，还允许无情缘关系进程间的通信；命名管道的名字位于文件系统中，内容存在内存中
  + 消息队列message
    - 消息队列是消息的链接表，它克服了上两种通信方式中信号量有限的缺点，具有读写权限的进程可以按照一定的规则向消息队列中添加/读取消息；存放在内核：允许一个或多个进程操作，消息不一定按照先进先出读取，也可以按照消息类型读取
  + 信号signal
    - 用于通知接收进程某个事件已发生，信号可以任何时候发给某个进程，无需知道进程状态，如果进程当前未执行，就会由内核保存起来，直到进程恢复执行并传递给它
  + 共享内存
    - 最用的进程间通信方式，使得多个进程可以访问同一块内存空间，依赖互斥锁和信号量同步
  + 信号量semaphore
    - 主要作为进程之间以及线程之间的同步和互斥手段；初始化操作：PV操作
      * P操作
        + 信号量-1，检测是否小于0，小于则进程进入阻塞状态
      * V操作
        + 信号量+1，若小于等于0，则从队列中唤醒一个等待的进程进入就绪状态
  + 套接字socket
    - 更为一般的进程之间通信机制，网络中不同机器之间进程间通信
* 进程同步机制
  + 原子操作
  + 信号量操作
  + 自旋锁管理
  + 会合、分布式系统
* 进程状态
  + 等待态/阻塞态/睡眠态
    - 进程不具备运行条件，正在等待使用资源或者某事件发生，如外设传输，人工干预
      * 等待-->就绪
        + 资源得到满足或某事件已经发生，如外设传输结束；人工干预完成
  + 就绪态
    - 具备运行条件，等待系统分配处理器以便运行，一个进程创建后就处于就绪态可由等待态等待事件发生变为就绪态，或者运行态的运行时间片到/出现更高优先权进程
      * 就绪-->运行
        + CPU空闲时被调度选中一个就绪进程执行
  + 运行态
    - 占有处理器正在运行
      * 运行-->等待
        + 等待使用资源或某事件发生，如等待外设传输，等待人工干预
      * 运行-->就绪
        + 运行时间片到，或出现有更高优先权进程
* 进程调度策略
  + FCFS(先来先服务)，队列实现，非抢占的，先请求CPU的进程先分配到CPU
  + SJF（最短作业优先调度算法）
    - 平均等待时间最短，
* 死锁
  + 在两个或多个并发进程中，如果每个进程持有某个资源而又等待其他进程释放现在保持着的资源，在未改变这种状态之前都不能向前推进，也就是两个或多个进程无限期的阻塞，相互等待的一种状态
  + 必要条件
    - 互斥
      * 即一次只能一个进程使用，若其他申请使用需要等待该进程释放
    - 占用并等待
      * 一个进程必须占用至少一个资源，并等待另一个资源，而该资源被其他进程占用
    - 非抢占
      * 已经分配给一个进程的资源不能被强制性抢占，资源只能被进程在完成任务后自愿释放
    - 循环等待
      * 若干进程之间形成了一种头尾相接的环形等待资源关系，该环路每个进程都在等待下一个进程所占用的资源
  + 处理方法
    - 鸵鸟策略
      * 解决死锁的代价很高，直接忽略死锁，在发生死锁不会对用户造成多大影响，或发生死锁的概率很低
    - 死锁预防
      * 破坏形成死锁的四个必要条件之一，实用性有限，会降低资源利用率，降低系统性能
    - 死锁避免
      * 动态监测资源分配状态，以确保循环等待条件不成立，从而确保系统处于安全状态即系统能按照某个顺序为每个进程分配资源
    - 死锁解除
      * 终止进程，简单地终止一个或多个进程以打破循环；资源抢占，挂起某些进程并抢占它的资源
* 异常
  + 中断
  + 陷阱
  + 故障
  + 终止
* IO多路复用
  + 单个进程/线程可以同时处理多个IO请求，用户将想要监视的文件描述符添加到select/poll、epoll函数中，由内核监视，函数阻塞，一旦有文件描述符读写就绪，或者超时，函数就会返回，然后该进程可以进行相应的读写操作
  + select
    - 将文件描述符放入集合中，调用select时，将这个集合从用户空间拷贝到内核空间，开销大
  + poll
  + epoll
* 分段和分页
  + 段式存储管理
  + 页式存储管理
  + 不同点
    - 目的不同
      * 分页是由于系统管理的需要而不是用户的需要，它是信息的物理单位；分段的目的是为了能更好地满足用户的需要，它是信息的逻辑单位，它含有一组其意义相对完整的信息；
    - 大小不同
      * 页的大小固定且由系统决定，而端的长度不固定，由其所完成的功能决定
    - 地址空间不同
      * 段向用户提供二维地址空间，页向用户提供一维地址空间
    - 信息共享
      * 段是信息的逻辑单位，便于存储保护和信息的共享；页的保护和共享受到限制
    - 内存碎片
      * 页式存储管理的优点是没有外碎片（因为页的大小固定），但会产生内碎片（一个页可能填充不满）
      * 段的优点是没有外碎片（因为段大小可变，改变段的大小来消除内碎片）但段换入换出时，会产生外碎片
* 页面置换算法
  + FIFO
  + LRU，最近最少使用算法
  + LFU最少使用次数算法
  + OPT最优置换算法
* 同步异步阻塞非阻塞
* 函数库和系统调用
* CPU的内存架构和工作原理
  + 控制单元
  + 逻辑单元
  + 存储单元
* 协程
* 孤儿进程和僵尸进程
* 什么是守护进程，如何创建守护进程
* Bootloader内核从启动到最后加载完成的过程
* linux 文件管理
* 如何设计一个内存池