操作系统

1. **进程和线程**

进程其实是应用程序在内存里完成工作的一个过程，要完成这个过程就需要分配CPU资源，进程就是操作系统进行资源分配的基本单位；而线程可以理解为一个执行流，一个进程拥有了CPU资源之后需要做很多任务，它把这些任务分为一个或多个执行流，这些执行流就是该进程内的线程，需要操作系统的调度来执行，线程是系统调度的基本单位 。 每个进程都有独立的地址空间，一个进程崩溃不影响其它进程，但是进程之间的切换开销也很大；一个进程中的多个线程共享该进程的地址空间，所以线程间切换的开销就比较小，但是一个线程的非法操作会使整个进程崩溃。总的来说多进程指的是操作系统中同时运行的多个程序而多线程指的是在同一个进程中同时运行的多个任务。

Go程序内的协程goroutine是一种用户态的轻量级的线程，比线程所占内存更小，一个goroutine大概只占用4kb的内存，一个Go程序创建几百万个goroutine是没有问题的，而且goroutine之间的切换都是由runtime管理的，处于用户态，开销更小。goroutine在用户态就可以创建，而线程需要操作系统来创建。

1. **进程间通信**

常见的进程间的通信方式主要有以下6种：1管道通信、2消息队列、3共享内存、4信号、5信号量、6socket。

1. 管道有匿名管道和命名管道，匿名管道在Linux中是一条竖线，只能用来在父子进程间通信，用完了就会被销毁。我们也可以用mkfifo创建命名管道，命名管道可以在不相关的进程间通信。往管道里面发送数据的进程会阻塞，直到有别的进程将数据取走，所以管道通信效率比较低，不适合进程间频繁地交换数据。【管道本质上就是内存中的缓冲区，也是一个文件，用来存放不同的进程间进行通信的数据，他有一些特点：互斥访问，写到满阻塞，读到空阻塞，没写满不可以读，没读空不可以写。】
2. 消息队列是进程间的数据交换以格式化消息为单位，存入内核中的消息链表，通过操作系统提供的原语来进行数据交换。与管道不同的是，存入消息的进程不会阻塞，存入后他就可以去坐别的事了，等到取消息的进程需要了再过来取就行。
3. 前面说的管道和消息队列都会发生用户态与内核态之间的转换，这个过程会耗费性能。而共享内存的方式可以解决这一点。

共享内存是共享一块大家可以互斥访问的内存空间，原理就是将两个进程的一部分虚拟内存映射到相同的物理块。

他又分为基于数据结构的共享和基于存储区的共享，前者限制多比如只能放数组等是一种低级通信，而后者没有这些限制、速度更快是一种高级通信。

1. 信号量

信号量其实是一个整型的计数器，主要用于实现进程间的互斥与同步。有两种可以控制信号量的原子操作，也就是PV操作。在使用共享资源之前执行P操作，表示计数器减1，归还共享资源后执行V操作，表示计数器加1。信号量初始化为1表示互斥信号量，同一时刻始终只能有一个进程可以访问。信号量初始化为0表示同步信号量。例如生产和消费问题，生产进程结束后执行V操作加1，消费进程在执行P操作后才能执行，反着来的话消费进程就会被阻塞。

1. 信号

信号是进程间唯一的一种异步通信,一般用于异常情况下的通信。进程需要为信号设置相应的监听处理，当收到特定信号时，执行相应的操作。比如在Linux中杀死和重启一个进程就是一种信号。在Go语言中主要用os包中的signal产生信号，用signal包里的Notify函数做相应的处理。

1. Socket

前面都是在同一台主机上的进程间通信。实际开发中我们更多的是需要跨网络与不同主机上的进程通信，这就需要Socket通信了。我们可以通过IP加端口号的方式唯一确定需要和我们进行通信的进程。

1. 说到通信的话我们Go语言不主张用共享内存的方式来通信，而是主张用通信的方式共享内存。主要体现在通道上。
2. **死锁**

死锁主要是由于资源分配和并发进程推进顺序不当导致的，它的必要条件有四个：互斥、部分分配、不可抢占、循环等待。我们可以破坏它的必要条件来预防死锁，比如使资源可以被抢占等,或者通过银行家算法来判断是否处于安全状态。当已经发生死锁后我们可以采用剥夺资源、进程回滚或撤销进程的方式解除死锁。

1. **CPU调度(执行进程的时候）**
   1. 先来先服务
   2. 短作业优先
   3. 按优先级
   4. 时间片轮转算法
   5. 多级队列

按性质分为多队列，不同的队列采用不同的调度算法

* 1. 多级反馈队列

分为多个执行队列，执行完时间片后进入下一级队列

* 1. 高响应比优先 作业周转时间(等待时间+执行时间)/作业执行时间

1. **什么是虚拟内存**

虚拟内存是计算机系统[管理](https://so.csdn.net/so/search?q=%E5%86%85%E5%AD%98%E7%AE%A1%E7%90%86&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/zhanyd/article/details/_blank)内存的一种技术，他让应用程序以为自己分配到了符合自己大小的一片连续的内存，实际上他通常是被映射到多个内存碎片，还有部分在磁盘上，只有需要用到时才会被交换进入内存。也就是说虚拟内存可以只把程序运行的部分换入内存，并且充分利用每一块内存碎片，极大提高了内存的利用率。

1. **内存分配**

有连续内存分配（现代os已经不采用）和离散内存分配两种方式；连续分配方式有单一连续区分配（单一进程）、固定分区分配（有内碎片）和可变分区分配（最先、最佳、最坏适应法；有外碎片）；离散分配有分页，分段和段页式管理；分段主要是基于程序结构考虑的，把有不同功能的各部分分成不同的段，这些段以可变分区分配的方式放入内存。而分页主要是基于系统考虑的，将内存分为大小相同的页框，然后也再将程序分页，然后由页表将程序的分页映射到分散的页框中。现在的系统一般都采用页式管理加虚拟内存的方式来管理内存，只装入经常使用到的页，其它部分在后面运行的过程中按需装入。段页式就是两种方式的结合，先将程序按功能结构分段，再将这些段分页。

1. **页面置换**
   1. 理想淘汰算法
   2. 随机淘汰算法
   3. 先进先出算法

balady现象（分配的物理页面越多，缺页中断次数反而增加）

* 1. 最近最久未使用页面置换算法
  2. 时钟页面置换算法（设置访问位，先进先出+跳过访问的页）

1. **抖动**

抖动指的就是刚被调出内存的页又要马上被调回内存， 这些调页的时间甚至大于进程执行时间，降低了CPU的利用率。

1. 防止抖动的根本手段：给进程分配足够多的物理页框
2. 那么我们怎样确定分配够了，而不用分配过多呢？

只需要使驻留集稍微大于工作集就可以使进程顺利运行，合理地解决抖动问题

1. 驻留集：在当前时刻，进程实际驻留在内存中的页集合。
2. 工作集：工作集就是指在某段时间间隔Δ 里，进程实际所要访问页面的集合。
3. **文件的结构**

文件的结构分为连续结构、链接结构和索引结构；连续结构现在已经不采用了，链接结构不适用与数据的随机访问，目前一般都使用多级索引结构

1. **数据传输控制方式**
2. 程序直接控制方式

CPU不断检查IO设备的准备情况，浪费CPU时间，而且IO准备就绪后CPU参与数据传送工作，而不能执行原程序。

1. 中断方式

CPU向IO发出命令后可以执行其他工作，当收到IO中断后转到相应的中断处理程序。

1. DMA方式

在外围设备和内存之间开辟数据交换通路，由DMA控制器控制。需要硬件支持。

1. 通道方式

通道其实是一种特殊的执行I/O指令的处理机，控制设备与内存直接进行数据交换。

1. **缓冲技术**

主要是减少对CPU的中断频率和提高CPU和IO的并行性

有专用的硬件缓冲器或者是在内存中划出一个专用缓冲区，存放输入输出的数据。

1. **五中网络IO模型 //没看懂**
   1. [阻塞IO（blocking IO）](https://blog.csdn.net/qq_42956653/article/details/125613170?ops_request_misc=%7B%22request%5Fid%22%3A%22167220856516800222843957%22%2C%22scm%22%3A%2220140713.130102334.pc%5Fall.%22%7D&request_id=167220856516800222843957&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v1~rank_v31_ecpm-3-125613170-null-null.142^v68^control,201^v4^add_ask,213^v2^t3_control2&utm_term=%E7%BD%91%E5%92%AFio%E6%A8%A1%E5%9E%8B&spm=1018.2226.3001.4187" \l "_IOblocking_IO_21" \t "https://blog.csdn.net/qq_42956653/article/details/_self)
   2. [非阻塞IO（non-blocking IO）](https://blog.csdn.net/qq_42956653/article/details/125613170?ops_request_misc=%7B%22request%5Fid%22%3A%22167220856516800222843957%22%2C%22scm%22%3A%2220140713.130102334.pc%5Fall.%22%7D&request_id=167220856516800222843957&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v1~rank_v31_ecpm-3-125613170-null-null.142^v68^control,201^v4^add_ask,213^v2^t3_control2&utm_term=%E7%BD%91%E5%92%AFio%E6%A8%A1%E5%9E%8B&spm=1018.2226.3001.4187" \l "_IOnonblocking_IO_76" \t "https://blog.csdn.net/qq_42956653/article/details/_self)
   3. [多路复用IO（IO multiplexing）](https://blog.csdn.net/qq_42956653/article/details/125613170?ops_request_misc=%7B%22request%5Fid%22%3A%22167220856516800222843957%22%2C%22scm%22%3A%2220140713.130102334.pc%5Fall.%22%7D&request_id=167220856516800222843957&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v1~rank_v31_ecpm-3-125613170-null-null.142^v68^control,201^v4^add_ask,213^v2^t3_control2&utm_term=%E7%BD%91%E5%92%AFio%E6%A8%A1%E5%9E%8B&spm=1018.2226.3001.4187" \l "_IOIO_multiplexing_118" \t "https://blog.csdn.net/qq_42956653/article/details/_self)
   4. [异步IO（Asynchronous I/O）](https://blog.csdn.net/qq_42956653/article/details/125613170?ops_request_misc=%7B%22request%5Fid%22%3A%22167220856516800222843957%22%2C%22scm%22%3A%2220140713.130102334.pc%5Fall.%22%7D&request_id=167220856516800222843957&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v1~rank_v31_ecpm-3-125613170-null-null.142^v68^control,201^v4^add_ask,213^v2^t3_control2&utm_term=%E7%BD%91%E5%92%AFio%E6%A8%A1%E5%9E%8B&spm=1018.2226.3001.4187" \l "_IOAsynchronous_IO_189" \t "https://blog.csdn.net/qq_42956653/article/details/_self)
   5. [信号驱动IO（signal driven I/O， SIGIO）](https://blog.csdn.net/qq_42956653/article/details/125613170?ops_request_misc=%7B%22request%5Fid%22%3A%22167220856516800222843957%22%2C%22scm%22%3A%2220140713.130102334.pc%5Fall.%22%7D&request_id=167220856516800222843957&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v1~rank_v31_ecpm-3-125613170-null-null.142^v68^control,201^v4^add_ask,213^v2^t3_control2&utm_term=%E7%BD%91%E5%92%AFio%E6%A8%A1%E5%9E%8B&spm=1018.2226.3001.4187" \l "_IOsignal_driven_IO_SIGIO_222" \t "https://blog.csdn.net/qq_42956653/article/details/_self)
2. **并发和并行的区别**

宏观上要是同时执行，在微观上的话，并行也是同时执行的，而并发是交替执行的

1. **同步和异步的区别**

同步是指协调多个程序顺序访问一个资源，是阻塞式的。异步是指通过缓冲区的方式通信，不用一直等待，是非阻塞式的。

我们可以拿Go语言中的Channel举例。当channel没有缓冲时，goroutine之间就是同步通信，当channel有缓冲时，goroutine之间就是异步通信。

1. **常见缓存淘汰算法**
2. 最不经常使用算法（LFU）

使用一个计数器来记录条目被访问的频率，淘汰计数最小的条目。

1. 最近最少使用算法（LRU）

运用队列的思想，把使用过后的条目重新放进一头，另一头就是最近没被使用的。

1. 自适应缓存替换算法(ARC)
2. 先进先出算法（FIFO）
3. 最近最常使用算法（MRU）