1. 进程和线程的区别

2. 多线程和多进程的区别，怎么抉择

3. 线程间怎样实现独立访问（线程间同步有哪些方法）？

4. 死锁的条件和解决方法,给资源加锁的常用方法

5. 互斥锁与自旋锁的区别

6. 写一个多线程死锁的伪代码

7. 说明下处理机调度的几种机制

8. 进程间切换的具体流程，画图说明,本质上到底是什么引发的进程切换？

答：时钟中断，

9. 内存中堆和栈的区别，栈底层如何实现？是一上来就分配4MB吗？缺页中断时操作系统怎么做？

10. 描述下进程间通信方式，写下pipe使用的伪代码

11. 内存分页的大小，为什么这样分

12.进程中能开多少个线程？

13.   
链接：<https://www.nowcoder.com/discuss/27739?type=2&order=0&pos=18&page=2>

 一串int型整数存放磁盘上的压缩存储方式，包括写入与读取及内存无法一次性读取时的解决办法

 Bloom过滤器处理大规模问题时的持久化，包括内存大小受限、磁盘换入换出问题

 线程池的了解、优点、调度处理方式和保护任务队列的方式

 对象复用的了解

 零拷贝的了解

 解释哲学家进餐问题，实现一种算法解决哲学家进餐问题

 描述银行家算法

 操作系统内核的划分

 无锁编程

1. 进程和线程的区别

**定义**

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位.

线程是进程的一个实体,是CPU调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.

**区别**：

**1) 简而言之,一个程序至少有一个进程,一个进程至少有一个线程.**

2) 线程的划分尺度小于进程，使得多线程程序的并发性高。

3) 另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。

4) 线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。**但是线程不能够独立执行，**必须依存在应用程序中，由应用程序提供多个线程执行控制。

5) 从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。

2. 多线程和多进程的区别，怎么抉择

**1）需要频繁创建销毁的优先用线程**

这种原则最常见的应用就是Web服务器了，来一个连接建立一个线程，断了就销毁线程，要是用进程，创建和销毁的代价是很难承受的

**2）需要进行大量计算的优先使用线程**

所谓大量计算，当然就是要耗费很多CPU，切换频繁了，这种情况下线程是最合适的。这种原则最常见的是图像处理、算法处理。

**3）强相关的处理用线程，弱相关的处理用进程**

一般的Server需要完成如下任务：消息收发、消息处理。“消息收发”和“消息处理”就是弱相关的任务，而“消息处理”里面可能又分为“消息解码”、“业务处理”，这两个任务相对来说相关性就要强多了。因此“消息收发”和“消息处理”可以分进程设计，“消息解码”、“业务处理”可以分线程设计。当然这种划分方式不是一成不变的，也可以根据实际情况进行调整。

**4）可能要扩展到多机分布的用进程，多核分布的用线程**



3. 线程间怎样实现独立访问

[**线程间实现同步互斥的方法**](http://t240178168.iteye.com/blog/1749315)

.临界区（Critical Section）：适合一个进程内的多线程访问公共区域或代码段时使用。

.互斥量 (Mutex)：适合不同进程内多线程访问公共区域或代码段时使用，与临界区相似。

.事件（Event）：通过线程间触发事件实现同步互斥

.信号量（Semaphore）：与临界区和互斥量不同，可以实现多个线程同时访问公共区域数据，原理与操作系统中PV操作类似，先设置一个访问公共区域的线程最大连接数，每有一个线程访问共享区资源数就减一，直到资源数小于等于零。

4. 死锁的条件和解决方法,给资源加锁的常用方法

**产生死锁的原因：**（1）竞争系统资源 （2）进程的推进顺序不当

**产生死锁的必要条件：**

互斥条件：进程要求对所分配的资源进行排它性控制，即在一段时间内某资源仅为一进程所占用。

请求和保持条件：当进程因请求资源而阻塞时，对已获得的资源保持不放。

不剥夺条件：进程已获得的资源在未使用完之前，不能剥夺，只能在使用完时由自己释放。

环路等待条件：在发生死锁时，必然存在一个进程--资源的环形链。

**解决死锁的基本方法：**

预防死锁：

资源一次性分配：一次性分配所有资源，这样就不会再有请求：(破坏请求条件)

只要有一个资源得不到分配，也不给此进程分配其他的资源：（破坏请保持条件）

可剥夺资源：即当某进程获得了部分资源，但得不到其它资源，则释放已占有的资源（破坏不可剥夺条件）

资源有序分配法：系统给每类资源赋予一个编号，每一个进程按编号递增的顺序请求资源，释放则相反（破坏环路等待条件）

避免死锁:

预防死锁的几种策略，会严重地损害系统性能。因此在避免死锁时，要施加较弱的限制，从而获得较满意的系统性能。由于在避免死锁的策略中，允许进程动态地申请资源。因而，系统在进行资源分配之前预先计算资源分配的安全性。若此次分配不会导致系统进入不安全的状态，则将资源分配给进程；否则，进程等待。其中最具有代表性的避免死锁算法是银行家算法。

检测死锁

首先为每个进程和每个资源指定一个唯一的号码；

然后建立资源分配表和进程等待表。

解除死锁:

当发现有进程死锁后，便应立即把它从死锁状态中解脱出来，常采用的方法有：

**剥夺资源**：从其它进程剥夺足够数量的资源给死锁进程，以解除死锁状态；

**撤消进程**：可以直接撤消死锁进程或撤消代价最小的进程，直至有足够的资源可用，死锁状态消除为止；所谓代价是指优先级、运行代价、进程的重要性和价值等。

5. 互斥锁与自旋锁的区别（C++）

**两种锁的加锁原理**

**互斥锁**：线程会从sleep（加锁）🡪running（解锁），过程中有上下文的切换，cpu的抢占，信号的发送等开销。

**自旋锁**：线程一直是running(加锁🡪解锁)，死循环检测锁的标志位，机制不复杂。

互斥锁属于sleep-waiting类型的锁。例如在一个双核的机器上有两个线程(线程A和线程B)，它们分别运行在Core0和 Core1上。假设线程A想要通过pthread\_mutex\_lock操作去得到一个临界区的锁，而此时这个锁正被线程B所持有，那么线程A就会被阻塞 (blocking)，Core0会在此时进行上下文切换(Context Switch)将线程A置于等待队列中，此时Core0就可以运行其他的任务(例如另一个线程C)而不必进行忙等待。而自旋锁则不然，它属于busy- waiting类型的锁，如果线程A是使用pthread\_spin\_lock操作去请求锁，那么线程A就会一直在 Core0上进行忙等待并不停的进行锁请求，直到得到这个锁为止。

**两种锁的区别**

互斥锁的起始原始开销要高于自旋锁，但是基本是一劳永逸，临界区持锁时间的大小并不会对互斥锁的开销造成影响，而自旋锁是死循环检测，加锁全程消耗cpu，起始开销虽然低于互斥锁，但是随着持锁时间，加锁的开销是线性增长。

**两种锁的应用**

互斥锁用于临界区持锁时间比较长的操作，比如下面这些情况都可以考虑

1 临界区有IO操作

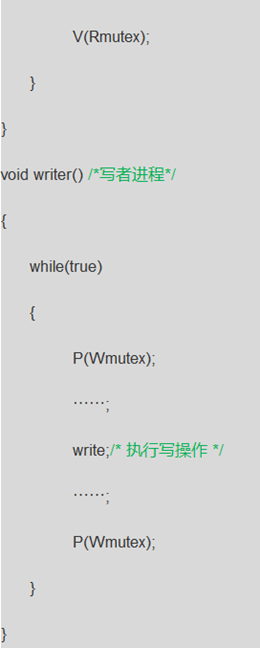
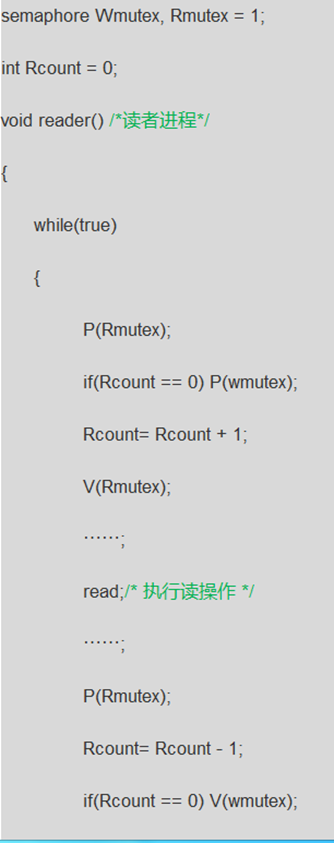
2 临界区代码复杂或者循环量大

3 临界区竞争非常激烈

4 单核处理器

至于自旋锁就主要用在临界区持锁时间非常短且CPU资源不紧张的情况下，自旋锁一般用于多核的服务器。

6. 写一个多线程死锁的伪代码(读者写者问题)



7. 说明下处理机调度的几种机制

调度算法

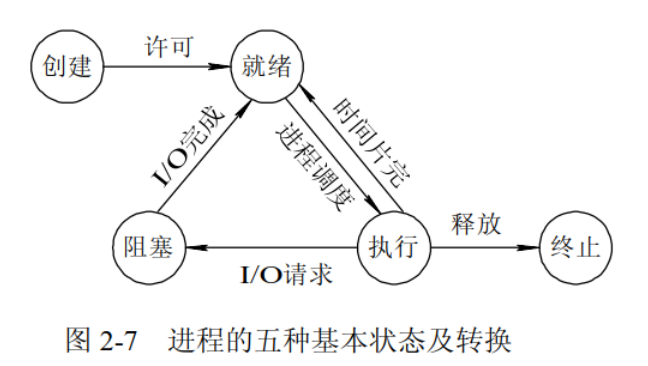
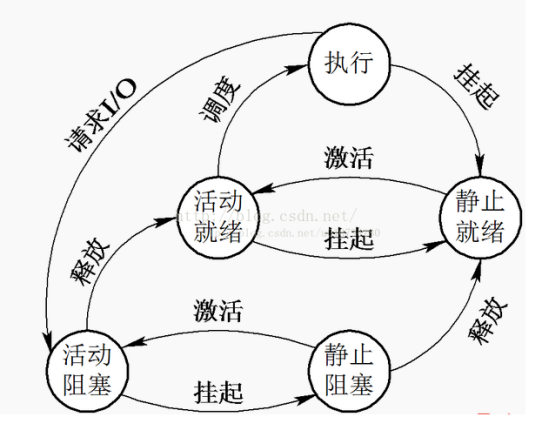


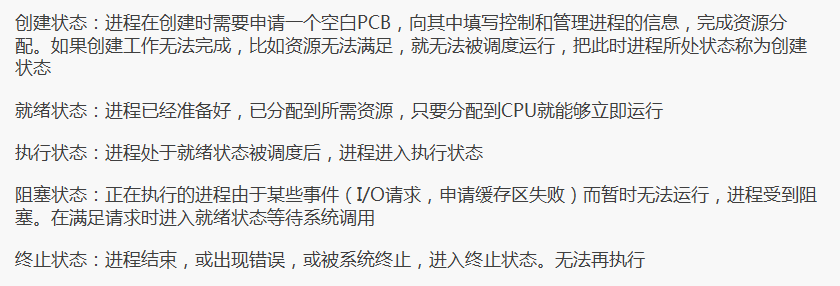
（优先权=（等待时间+需要求服务时间）/要求服务时间）



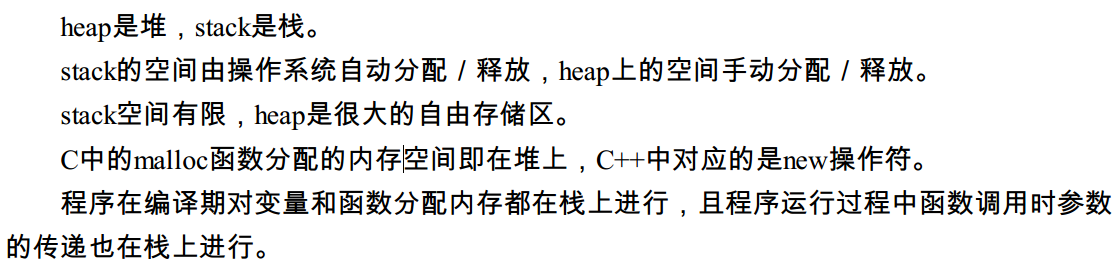


8. 进程间切换的具体流程



9. 内存中堆和栈的区别



栈：在Windows下，栈是向低地址扩展的数据结构，是一块连续的内存的区域。 这句话的意思是栈顶的地址和栈的最大容量是系统预先规定好的，在Windows下，栈的大小是2MB（也有的说是1MB，总之是一个编译时就确定的常数），如果申请的空间超过栈的剩余空间，将提示overflow。 因此，能从栈获得的空间较小。  
堆：堆是向高地址扩展的数据结构，是不连续的内存区域。 这是由于系统是用链表存储空闲内存地址的，自然是不连续的。 而链表的遍历方向是由低地址向高地址，堆的大小受限

**缺页中断发生时的事件顺序**如下：

1) 硬件陷入内核，在内核[堆栈](https://baike.baidu.com/item/%E5%A0%86%E6%A0%88)中保存[程序计数器](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E8%AE%A1%E6%95%B0%E5%99%A8)。大多数机器将当前指令的各种状态信息保存在特殊的CPU[寄存器](https://baike.baidu.com/item/%E5%AF%84%E5%AD%98%E5%99%A8)中。

2) 启动一个汇编代码例程保存[通用寄存器](https://baike.baidu.com/item/%E9%80%9A%E7%94%A8%E5%AF%84%E5%AD%98%E5%99%A8)和其他易失的信息，以免被操作系统破坏。这个例程将操作系统作为一个函数来调用。

3) 当操作系统发现一个缺页中断时，尝试发现需要哪个虚拟页面。通常一个硬件寄存器包含了这一信息，如果没有的话，操作系统必须检索程序计数器，取出这条指令，用软件分析这条指令，看看它在缺页中断时正在做什么。

4) 一旦知道了发生缺页中断的[虚拟地址](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E5%9C%B0%E5%9D%80)，操作系统检查这个地址是否有效，并检查存取与保护是否一致。如果不一致，向进程发出一个信号或杀掉该进程。如果地址有效且没有保护错误发生，系统则检查是否有空闲[页框](https://baike.baidu.com/item/%E9%A1%B5%E6%A1%86)。如果没有空闲页框，执行[页面置换算法](https://baike.baidu.com/item/%E9%A1%B5%E9%9D%A2%E7%BD%AE%E6%8D%A2%E7%AE%97%E6%B3%95)寻找一个页面来淘汰。

5) 如果选择的页框“脏”了，安排该页写回磁盘，并发生一次[上下文切换](https://baike.baidu.com/item/%E4%B8%8A%E4%B8%8B%E6%96%87%E5%88%87%E6%8D%A2)，挂起产生缺页中断的进程，让其他进程运行直至磁盘传输结束。无论如何，该页框被标记为忙，以免因为其他原因而被其他进程占用。

6) 一旦页框“干净”后（无论是立刻还是在写回磁盘后），操作系统查找所需页面在磁盘上的地址，通过磁盘操作将其装入。该页面被装入后，产生缺页中断的进程仍然被挂起，并且如果有其他可运行的用户进程，则选择另一个用户进程运行。

7) 当磁盘中断发生时，表明该页已经被装入，[页表](https://baike.baidu.com/item/%E9%A1%B5%E8%A1%A8)已经更新可以反映它的位置，[页框](https://baike.baidu.com/item/%E9%A1%B5%E6%A1%86)也被标记为正常状态。

8) 恢复发生缺页[中断指令](https://baike.baidu.com/item/%E4%B8%AD%E6%96%AD%E6%8C%87%E4%BB%A4)以前的状态，[程序计数器](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E8%AE%A1%E6%95%B0%E5%99%A8)重新指向这条指令。

9) 调度引发缺页中断的进程，操作系统返回调用它的汇编语言例程。

10) 该例程恢复寄存器和其他状态信息

**10.**[**进程间的通信方式**](http://www.cnblogs.com/mydomain/archive/2010/09/23/1833369.html)

 1**无名管道**( pipe )：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。

 2**高级管道**(popes)：将另一个程序当做一个新的进程在当前程序进程中启动，则它算是当前程序的子进程，这种方式我们成为高级管道方式。

 3**有名管道** (named pipe) ： 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

 4**消息队列**( message queue ) ： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

 5**信号量**( semaphore)：信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

 6**信号** (signal)：信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

 7**共享内存**( shared memory ) ：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号两，配合使用，来实现进程间的同步和通信。

 8**套接字**( socket ) ： 套解口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信。

11.内存分页的大小，为什么这样分

 使用了分页机制之后，4G的地址空间被分成了固定大小的页，每一页或者被映射到物理内存，或者被映射到硬盘上的交换文件中，或者没有映射任何东西。对于一般 程序来说，4G的地址空间，只有小部分映射了物理内存，大部分是没有映射任何东西。物理内存也被分页，来映射地址空间。对于32bit的Win2k，页的大小是4K字节。CPU用来把虚拟地址转换成物理地址的信息存放在叫做页目录和页表的结构里。  
 物理内存分页，一个物理页的大小为4K字节，第0个物理页从物理地址 0x00000000 处开始。由于页的大小为4KB，就是0x1000字节，所以第1页从物理地址 0x00001000 处开始。第2页从物理地址 0x00002000 处开始。可以看到由于页的大小是4KB，所以只需要32bit的地址中高20bit来寻址物理页。

12.进程中能开多少个线程？

默认情况下，一个线程的栈要预留**1M**的内存空间,而一个进程中可用的内存空间只有2G，所以理论上一个进程中最多可以开2048个线程,但是内存当然不可能完全拿来作线程的栈，所以实际数目要比这个值要小。

13.

**哲学家就餐问题**

**解决方法**

1. 同时只允许一位哲学家就餐
2. 对哲学家顺序编号，要求奇数号哲学家先抓左边的叉子，然后再抓他右边的叉子，而偶数号哲学家刚好相反。
3. 仅当一个哲学家左右两边的叉子都可用时才允许他抓起叉子；

银行家算法

