目录

[1. 进程和线程的区别 2](#_Toc459839791)

[2. 进程与程序的区别 2](#_Toc459839792)

[3. 进程的创建 2](#_Toc459839793)

[4. 进程的终止 3](#_Toc459839794)

[5. 进程的基本状态 3](#_Toc459839795)

[6. PCB：Process Control Block 4](#_Toc459839796)

[7. 引入线程的优点 5](#_Toc459839797)

[8. 使用线程需要注意的地方 5](#_Toc459839798)

[9. 线程机制的实现(用户线程、内核线程、混合) 5](#_Toc459839799)

[10. 同步和互斥的意思 6](#_Toc459839800)

[11. 引起死锁的原因 6](#_Toc459839801)

[12. 产生死锁的四个必要条件 6](#_Toc459839802)

[13. 进程间通信方式 6](#_Toc459839803)

[14. 线程同步的机制 7](#_Toc459839804)

[15. 进程的内核态和用户态 8](#_Toc459839805)

[16. 为什么服务器都是linux的？ 9](#_Toc459839806)

[17. 内存管理方式 9](#_Toc459839807)

[19. 虚拟内存 10](#_Toc459839808)

[20. 内碎片与外碎片 10](#_Toc459839809)

[21. Cache替换算法 11](#_Toc459839810)

[22. 如何理解分布式锁 11](#_Toc459839811)

操作系统是计算机系统中的一个系统软件。

1. 进程和线程的区别

进程是具有独立功能的程序关于某个数据集合上的一次运行活动，是系统进行资源分配和调度的独立单位。从概念上说，每个进程拥有他自己的虚拟CPU。

线程是进程的一个实体，是CPU调度和分配的基本单位，线程自己基本上不拥有系统资源，只拥有在运行中必不可少的资源(如程序计数器、一组寄存器和堆栈、局部变量)，但是它可以与同属一个进程的其他线程共享该进程的全部资源。

线程必须在某个进程中执行。进程是系统资源分配的基本单位，但线程是调度的基本单位，是系统中并发执行的单元。

线程也被称作是轻量级进程，但是进程与线程之间的区别如下：

① 一个线程必定只属于一个进程，而一个进程可以拥有多个线程，并且至少拥有一个线程；

② 属于一个进程的所有线程共享该进程的所有资源，包括打开的文件、创建的socket连接等。不同的进程之间相互独立。

③ 线程被称作轻量级进程，进程有进程控制块，线程也有线程控制块。但是线程控制块比进程控制块要小得多。线程切换代价小，进程间切换代价大；

④ 进程是程序的一次执行，线程可以理解为程序中一段程序片段的执行；

⑤ 每个进程都有独立的内存空间，而线程是共享其所属进程的内存空间；

⑥ 进程让操作系统的并发性成为可能，而线程让进程的内部并发成为可能。

1. 进程与程序的区别

一个进程是某种类型的一个活动，它有程序、输入、输出以及状态。

程序是用适当形式描述的算法。如果一个程序运行了两遍，则算两个进程。

1. 进程更能准确刻画并发，而程序不能
2. 程序是静态的，进程是动态的
3. 进程是有生命周期的，有诞生有消亡，短暂的；而程序是相对长久的
4. 一个进程可以对应多个线程
5. 进程具有创建其他进程的功能
6. 进程的创建

新进程都是由一个已存在的进程执行了一个用于创建进程的系统调用而创建的。

给新进程分配一个唯一标识以及进程块PCB，为进程分配地址空间。然后初始化进程如设置默认值（如：状态为New，…），最后设置相应的队列指针如把新进程加到就绪队列链表中。不同的操作系统提供了不同的创建进程的操作，Unix中主要是fork和exec的配合使用，widows则是createProcess的使用。

主要有以下四个事件导致进程的创建：

① 系统初始化；

② 正在执行的进程所调用的进程创建系统调用；

③ 用户请求创建一个新进程；

④ 一个批处理作业的初始化。

1. 进程的终止

进程的撤消即结束进程，会收回进程所占有的资源，如关闭打开的文件、断开网络连接、回收分配的内存，最重要的是需要撤消该进程的PCB。

进程的终止主要由下列条件引起的：

1 正常退出(自愿的)

2 出错退出(自愿的)

3 严重错误(非自愿的)

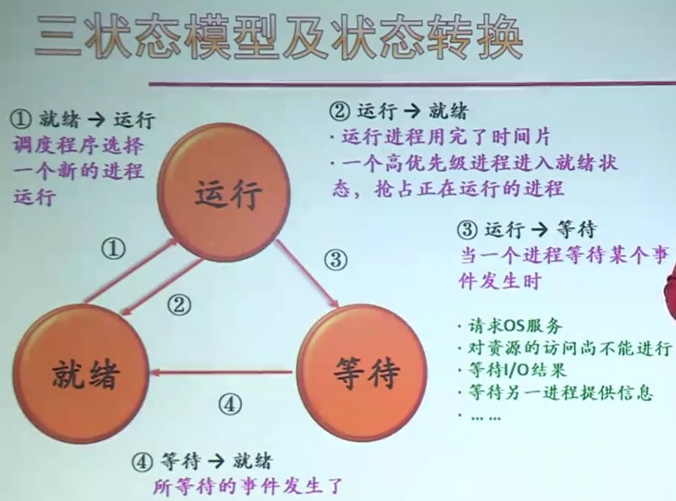
4 被其他进程杀死(非自愿)：在Unix系统中调用exit则撤消了对应的进程，在windows中则是TerminatePrcoess撤消相应进程。

1. 进程的基本状态
2. 进程的三种基本状态有运行态、就绪态、等待态。

运行态（Running）指线程占有CPU，并在CPU上运行；

就绪态（Ready）指线程已经具备运行条件，但是由于没有空闲CPU，而暂时不能运行；

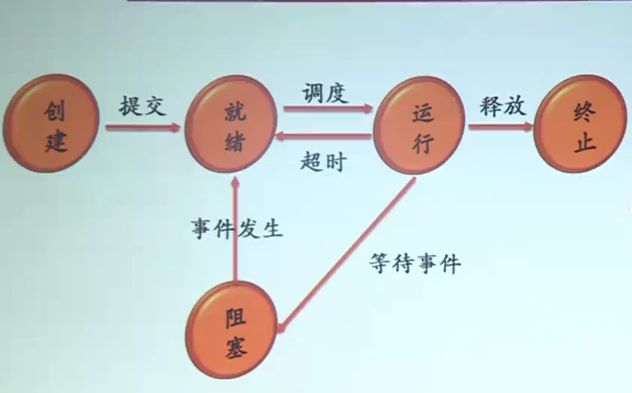
等待态（Waiting/Blocked）指的是线程因等待某一事件而暂时不能运行，如等待读盘结果。等待态又称作阻塞态、封锁态、睡眠态。



(2). 进程的五状态模型

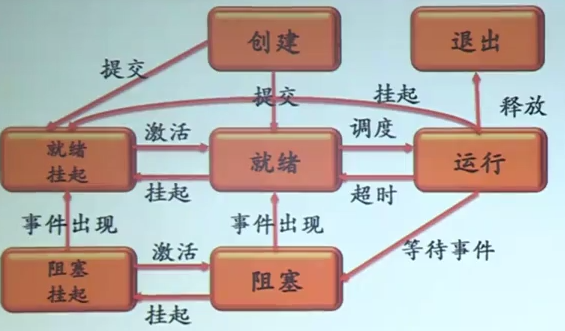
创建：已经完成创建一个进程所必要的工作如PID、PCB，但是尚未统一执行该进程因为资源有限；

终止：进程终止执行后，该进程就会进入该状态。可完成一些数据统计工作，并进行资源回收；



(3). 进程的七状态模型

挂起：当操作系统中如果要进行负载调节时，可能会把一些线程变成这种状态。比如当线程个数比较多的时候就会把若干个线程变成挂起态。一旦线程变成挂起态，操作系统会把其内存空间收回来，然后把线程相关的内容写到磁盘上，当线程解除挂起态时则操作系统再将线程的相关内容读入内存。



1. PCB：Process Control Block

又称进程描述符、进程属性，是操作系统用于管理控制进程的一个专门的数据结构，记录进程的各种属性，描述进程的动态变化过程。

PCB是系统感知进程存在的唯一标志，进程与PCB是一一对应的。

进程表表示所有进程的PCB集合。进程表的大小既可以包含PCB的个数决定了操作系统同时可容纳进程的个数，也叫操作系统的并发度。

PCB包含的信息：

|  |  |
| --- | --- |
| 信息类别 | 包含内容 |
| 进程描述信息 | 进程标识符（Process ID），唯一，通常是一个整数 |
| 进程名，通常基于可执行文件名，不唯一 |
| 用户标识符（User ID） |
| 进程组关系 |
| 进程控制信息 | 当前状态 |
| 优先级（priority） |
| 代码执行入口地址 |
| 程序的磁盘地址 |
| 运行统计信息（执行时间、页面调度） |
| 进程间同步和通信 |
| 进程的队列指针 |
| 进程的消息队列指针 |
| 进程所拥有的资源和使用情况 | 虚拟地址空间的状况 |
| 打开文件列表 |
| CPU现场信息 | 寄存器值（通用寄存器、程序计数器PC、程序状态字PSQ、栈指针） |
| 指向该进程页面的指针 |

1. 引入线程的优点

① 有利于发挥多处理器的功能。通过创建多线程，每个线程都在一个处理器上运行，从而实现应用程序的并行，使每个处理器都得到充分运行

② 开销小：创建线程比创建进程要快，所需要的开销也小；

③ 易于调度；

④ 提高并发性：通过线程可以方便有效地实现并发；多个线程的情况下，有的线程负责计算，有的线程负责I/O。效率高。

1. 使用线程需要注意的地方

① 线程必须在某个进程中执行；

② 在同一个进程中并行运行多个线程，是对同一台计算机上并行运行多个进程的模拟；

③ 当多线程进程在单CPU系统中运行时，线程轮流运行，CPU在线程之间的快速切换，制造了线程并行运行的假象，好像它们在比实际CPU慢一些的CPU上同时运行；

④ 各个线程之间都可以访问其共享的进程地址空间的每一个内存地址，所以一个线程可以读、写或甚至是清除另一个线程的堆栈。

⑤ 线程概念视图实现的是，共享一组资源的多个线程的执行能力，以便这些线程可以为完成某一项任务而共同工作；

⑥ 线程的属性(存放在线程表中)：（1）有标识符ID；（2）有状态及状态转换——>需要提供一些操作；（3）不运行时需要保存的上下文。有上下文环境：程序计数器等寄存器；（4）有自己的栈和栈指针。

⑦ 线程也可以创建、撤消另一个线程。因为线程开始是一个单线程进程方式运行的。

1. 线程机制的实现(用户线程、内核线程、混合)

根据操作系统内核是否能感知到线程的存在可分为用户线程和内核线程。

① 用户线程

不需要内核支持，在用户空间建立线程库：提供一组管理线程的过程。运行时系统：完成线程的管理工作（操作、线程表）。内核管理的是进程，并不知道线程的存在。线程切换不需要内核态特权。

优点：线程切换快；调度算法是应用程序特定的；用户级线程可运行在任何操作系统上（只需要实现线程库）；用户线程能够利用的表空间和堆栈空间比内核线程多的多；

缺点：内核只将处理器分配给进程，同一进程中的两个线程不能同时运行于两个处理器上；大多数系统调用是阻塞的，如果有一个线程使用了系统调用而被阻塞则整个进程都会被挂起；页面失效也会导致该进程所属的进程都会被挂起。

② 内核线程

内核线程的建立和和销毁都是由操作系统负责、通过系统调用完成的，操作系统在调用时，参考进程内的线程状态做出调度决策，如果某个进程内所有的线程都不是就绪态，那么这个进程也不会被调度占用CPU。

内核管理所有的线程，并向应用程序提供API接口；内核维护进程和线程的上下文；线程的切换需要内核支持；以线程为基础进行调度。

内核线程的优缺点和用户线程的优缺点刚好相反。

③ 混合—两者结合方法

线程创建是在用户空间完成的；线程调度等在核心态完成。

1. 同步和互斥的意思

互斥：**是指某一资源同时只允许一个访问者对其进行访问**，具有唯一性和排它性。但互斥无法限制访问者对资源的访问顺序，即访问是无序的。

同步：**是指在互斥的基础上（大多数情况），通过其它机制实现访问者对资源的有序访问。**在大多数情况下，同步已经实现了互斥，特别是所有写入资源的情况必定是互斥的。

显然，同步是一种更为复杂的互斥，而互斥是一种特殊的同步。

1. 引起死锁的原因

死锁指的就是**两个或两个以上的进程在相互等待对方的资源**，如果没有外力的干扰，就会一直在那循环等到，也就是出现了死锁的现象。

（1） 因为系统资源不足。

（2） 进程运行推进的顺序不合适。

（3） 资源分配不当等。

1. 产生死锁的四个必要条件

（1）**互斥条件**：一个资源每次只能被一个进程使用。

（2）**请求与保持条件**：一个进程因请求资源而阻塞时，对已获得的资源保持不放。

（3）**不剥夺条件**: 进程已获得的资源，在末使用完之前，不能强行剥夺。

（4）**循环等待条件**: 若干进程之间形成一种头尾相接的循环等待资源关系。

1. 进程间通信方式

进程间通信(Inter Process Communication, IPC)需要解决的问题有三个：①一个进程如何把信息传递给另一个进程；②确保两个或更多的进程在关键活动中不会出现交叉，例如订票系统中两个用户争夺最后一张票；③进程间正确的执行顺序。进程间相互通信的解决办法同样适用于线程。

进程间通信有：管道、有名管道、消息队列、信号量、共享存储

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；

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。

消息队列( message queue )：消息队列是把有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。

信号量( semophore ) ： 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。

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，如信号量配合使用，来实现进程间的同步和通信。

1. 线程同步的机制

临界区、互斥量、信号量、事件。

① **临界区**：**通过多线程的串行化来访问公共资源，速度快、适合控制数据访问。**

注意：临界区指的是对共享内存进行访问的程序片段，线程在执行代码时，将代码锁定，不允许其他线程执行，只有该线程离开后，其他线程才能使用这些代码。

② **互斥量（Mutex）**：通过锁机制实现线程间的同步。同一时刻只允许一个线程执行一个关键部分的代码。**一个互斥量只能用于一个资源的互斥访问，它不能实现多个资源的多线程互斥问题。**Mutex本质上说就是一把锁，提供对资源的独占访问。Mutex对象的值，只有0和1两个值。这两个值也分别代表了Mutex的两种状态。值为0, 表示锁定状态，当前对象被锁定，用户进程/线程如果试图Lock临界资源，则进入排队等待；值为1，表示空闲状态，当前对象为空闲，用户进程/线程可以Lock临界资源，之后Mutex值减1变为0。

Mutex被创建时可以有初始值，表示Mutex被创建后，是锁定状态还是空闲状态。在同一个线程中，为了防止死锁，系统不允许连续两次对Mutex加锁(系统一般会在第二次调用立刻返回)。也就是说，加锁和解锁这两个对应的操作，需要在同一个线程中完成。

“挂起等待”和“唤醒等待线程”的操作如何实现？每个Mutex有一个等待队列，一个线程要在Mutex上挂起等待，首先在把自己加入等待队列中，然后置线程状态为睡眠，然后调用调度器函数切换到别的线程。一个线程要唤醒等待队列中的其它线程，只需从等待队列中取出一项，把它的状态从睡眠改为就绪，加入就绪队列，那么下次调度器函数执行时就有可能切换到被唤醒的线程。

③ **信号量（Semaphore）**：信号量用于线程的同步，取值可以是非负整数，是一个计数器，用于控制访问有限共享资源的线程数。

Semaphore的模型可以概括为一个计数器、一个等待队列、三个方法。 三个方法原子性分别是init()、down()、up()；

init()：设置计数器的初始值。

down()：将计数器的值减1，如果减了1之后，计数器的值小于0，则当前的线程被阻塞，否则继续执行。

up()：将计数器的值加1，如果加了1之后，计数器的值小于等于0，则唤醒等待队列中的一个线程，并且将它移除出等待队列。(注意是小于等于0，不应该理解为大于等于0，因为大于等于0表明此时没有等待的线程，所以不会有唤醒这个操作。)

简单的理解就是 Semaphore 就是通过这三个方法来改变计数器，通过计数器的值来判断此时的线程是应该加入到等待队列中等待还是成功执行。信号量模型也被称为PV原语，也就是down和up操作最早称为P操作和V操作。

④ 事件：用来通知线程有一些事件已发生，从而启动后继任务开始。

1、  Event  
用事件（Event）来同步线程是最具弹性的了。一个事件有两种状态：激发状态和未激发状态。也称有信号状态和无信号状态。事件又分两种类型：手动重置事件和自动重置事件。手动重置事件被设置为激发状态后，会唤醒所有等待的线程，而且一直保持为激发状态，直到程序重新把它设置为未激发状态。自动重置事件被设置为激发状态后，会唤醒“一个”等待中的线程，然后自动恢复为未激发状态。所以用自动重置事件来同步两个线程比较理想。MFC中对应的类为CEvent.。CEvent的构造函数默认创建一个自动重置的事件，而且处于未激发状态。共有三个函数来改变事件的状态:SetEvent,ResetEvent和PulseEvent。用事件来同步线程是一种比较理想的做法，但在实际的使用过程中要注意的是，对自动重置事件调用SetEvent和PulseEvent有可能会引起死锁，必须小心。

2、  Critical Section  
使用临界区域的第一个忠告就是不要长时间锁住一份资源。这里的长时间是相对的，视不同程序而定。对一些控制软件来说，可能是数毫秒，但是对另外一些程序来说，可以长达数分钟。但进入临界区后必须尽快地离开，释放资源。如果不释放的话，会如何？答案是不会怎样。如果是主线程（GUI线程）要进入一个没有被释放的临界区，呵呵，程序就会挂了！临界区域的一个缺点就是：Critical Section不是一个核心对象，无法获知进入临界区的线程是生是死，如果进入临界区的线程挂了，没有释放临界资源，系统无法获知，而且没有办法释放该临界资源。这个缺点在互斥器(Mutex)中得到了弥补。Critical Section在MFC中的相应实现类是CcriticalSection。CcriticalSection：：Lock()进入临界区，CcriticalSection：：UnLock()离开临界区。

3、  Mutex  
互斥器的功能和临界区域很相似。区别是：Mutex所花费的时间比Critical Section多的多，但是Mutex是核心对象(Event、Semaphore也是)，可以跨进程使用，而且等待一个被锁住的Mutex可以设定TIMEOUT，不会像Critical Section那样无法得知临界区域的情况，而一直死等。MFC中的对应类为CMutex。Win32函数有：创建互斥体CreateMutex() ，打开互斥体OpenMutex()，释放互斥体ReleaseMutex()。Mutex的拥有权并非属于那个产生它的线程，而是最后那个对此Mutex进行等待操作（WaitForSingleObject等等）并且尚未进行ReleaseMutex()操作的线程。线程拥有Mutex就好像进入Critical Section一样，一次只能有一个线程拥有该Mutex。如果一个拥有Mutex的线程在返回之前没有调用ReleaseMutex()，那么这个Mutex就被舍弃了，但是当其他线程等待(WaitForSingleObject等)这个Mutex时，仍能返回，并得到一个WAIT\_ABANDONED\_0返回值。能够知道一个Mutex被舍弃是Mutex特有的。

4、  Semaphore  
信号量是最具历史的同步机制。信号量是解决producer/consumer问题的关键要素。对应的MFC类是Csemaphore。Win32函数CreateSemaphore（）用来产生信号量。ReleaseSemaphore（）用来解除锁定。Semaphore的现值代表的意义是目前可用的资源数，如果Semaphore的现值为1，表示还有一个锁定动作可以成功。如果现值为5，就表示还有五个锁定动作可以成功。当调用Wait…等函数要求锁定，如果Semaphore现值不为0，Wait…马上返回，资源数减1。当调用ReleaseSemaphore（）资源数加1，当时不会超过初始设定的资源总数。

1. 进程的内核态和用户态

进程通常运行在用户态，这时候可以使用CPU和内存来做一些事情，比如数学计算，但进程需要对硬件外设进行操作的时候(如读取磁盘文件、发送网络数据等)就必须切换到内核态，它将拥有更多的权限来控制整个计算机，当在内核态的任务完成后，进程又切换回用户态。

在内核态可以执行任何代码，用户态就不可以。分内核态和用户态最大目的是为了安全，用户态想要去对操作系统底层做一些操作的时候必须切换到内核态。

进程用户态和内核态分离的动机主要在于提高系统底层安全性以及简化开发模型。由于所有进程都必须通过内核提供的系统调用来操作硬件，所以不用担心应用程序对硬件进行非法操作。由于将底层的实现都屏蔽在了系统调用中，也大大简化了用户态开发的难度。

1. 为什么服务器都是linux的？

1.开源

Linux Server 相较Windows Server领先的首要原因是完全免费且可用作开源用途。通过开源方式，您可以轻松查看用于创建Linux内核的代码，也可以对代码进行修改和再创作。通过许多编程接口，您甚至可以开发自己的程序并将其添加到Linux 操作系统中。还可以对Linux Server操作系统进行自定义，以满足使用要求，这是Windows无法实现的。

2.稳定性

Linux系统一直以其稳定性而闻名，它们可以连续运行多年而不发生任何重大问题。事实上，很多Linux用户都从未在自己的环境中遇到过系统崩溃的情况。相对Windows而言，挂起和崩溃完全是一种常态。尽管Windows也可以很好地执行多任务处理，但Linux可以在处理各种任务的同时，仍能提供坚如磐石的性能。当将Linux与Windows进行对比时，对每项系统配置的更改都需要重启Windows Server，而Linux更改大多数配置时都无需重启服务器即可生效，这也确保了Linux服务器最短的停机时间。

3.安全

Linux由最初的多用户操作系统开发的UNIX操作系统发展而来，在安全方面显然比 Windows(win2000以前是单用户伪多用户系统)更强。与Windows作为病毒和恶意软件攻击的首要目标不同，Linux Server只有管理员或特定用户才有权访问Linux内核，而且Linux服务器（相较Windows而言）不会经常受到攻击，并且被发现的任何漏洞都会在第一时间由大批Linux开发人员修复。

4.硬件

当将Linux Server与Windows Server在硬件方面进行比较时，Windows需要经常对硬件进行升级以满足各方面性能需求。而Linux对硬件的需求则很低，也不需要频繁对硬件进行升级更新，并且无论系统架构或处理器如何，都能表现得非常出色。

5.灵活性

Linux是世界上最灵活的操作系统，没有之一，您可以根据需要自定义系统。使用Linux，您可以随心所欲地安装GUI界面或仅使用「终端」管理服务器；使用Linux，您可以选择各种工具和实用程序来管理所有与服务器相关的活动，如：添加用户、管理服务和网络，安装新应用程序以及监控性能等。而Shell是Linux系统中最强大的组件，允许您运行各种程序并允许与内核进行交互。总的来说，Linux为您提供了对服务器的完全控制、掌控权利。

6.总体拥有成本（TCO）和维护

在总体拥有成本方面，由于Linux在使用上属于完全免费。即便您购买了针对企业或组织的Linux发行版，也会比Windows或其他许可软件花费更少。

7.自由

对Linux而言，您不会被商业供应商强加产品和服务，用户可以自由选择适合需求的产品。正是这种自由使得像Amazon和Google这样的大公司选择了基于Linux的服务器来提供服务。

8.访问开源应用程序

Linux为开源应用程序开辟了一个新的世界，有数以千计的开源应用程序正在等待用户探索，您甚至可以使用特殊界面在Linux服务器上运行Windows应用程序。

9.易于变更

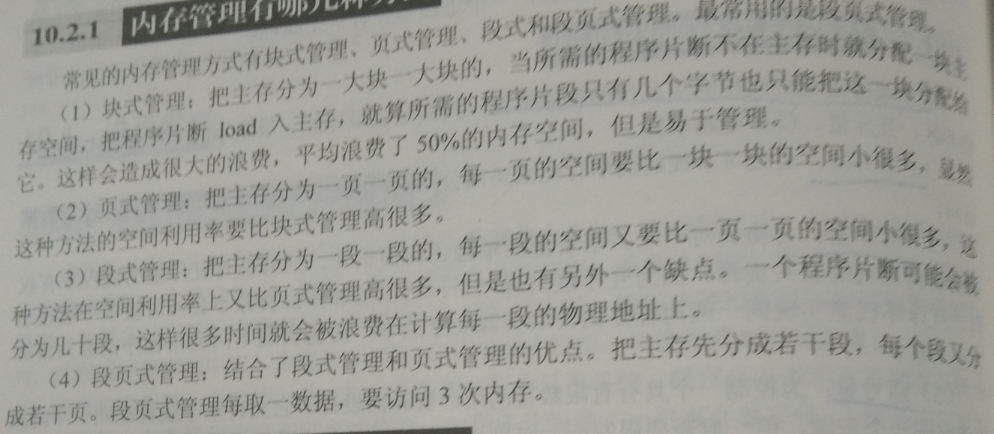
与Windows Servers不同，您可以轻松对Linux Server进行变更，并且无需重启服务器。对于Windows Server而言，要使用微软发布的新功能就得购买下一个版本。

10.社区支持

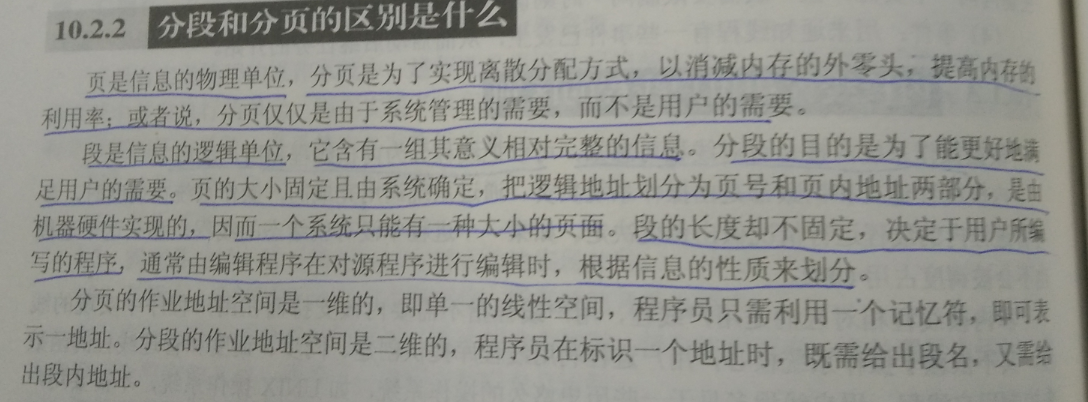
Linux社区在全球都十分活跃和使用广泛，总有数千名志愿者在线活动以解决其他Linux用户的问题，所以几乎在任何Linux论坛上发布的任何问题都会得到即时响应。而选择使用Linux Enterprise版本时，还会附有付费支持选项。

1. 内存管理方式

块式、页式（无外碎片，有内碎片）、段式（无内碎片，有外碎片）、段页式（每取一次数据，要访问3次内存）

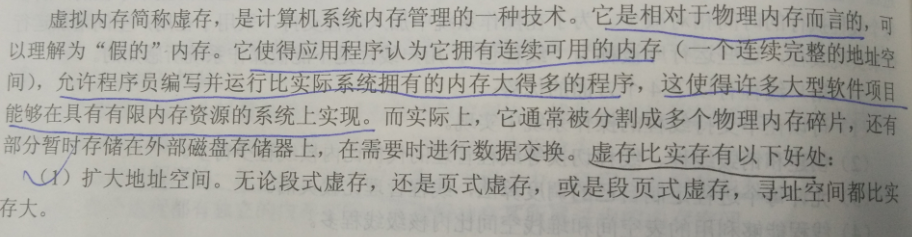


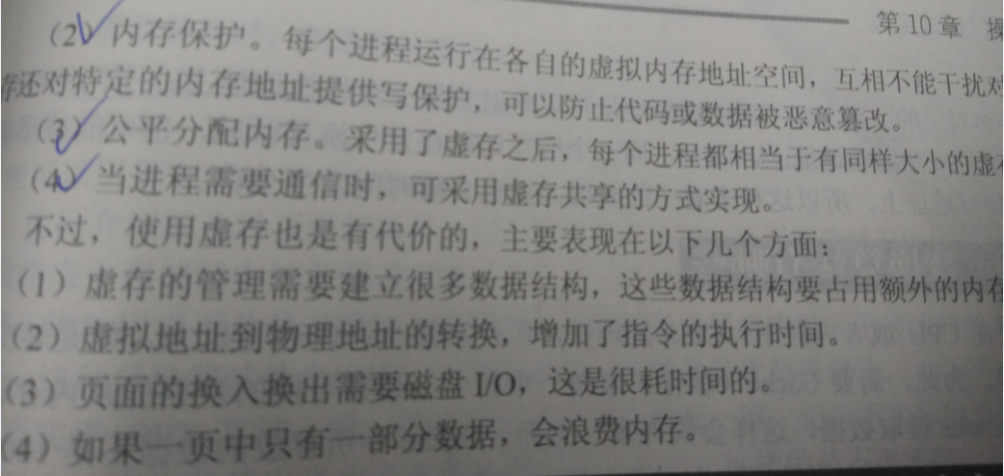
1. 分段和分页的 区别

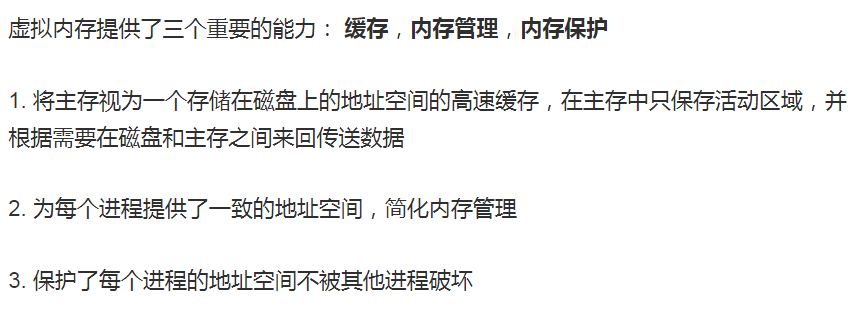


1. 虚拟内存

虚拟内存是相对于物理内存而言的，可以理解为“假的”内存。它使得应用程序认为它拥有连续可用的内存，允许编写并运行比实际系统拥有的内存大得多的程序。



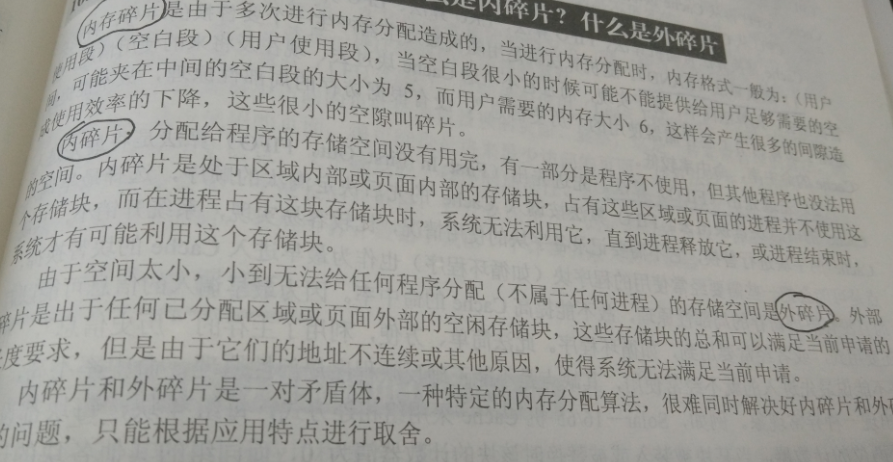




1. 内碎片与外碎片

内碎片是分配给程序的存储空间没有用完，有一部分是程序不使用，但其他程序也没法用的空间；

外碎片是由于空间太小，小到无法给任何程序分配（不属于任何进程）的存储空间。



1. Cache替换算法

随机算法：用随机数发生器产生一个要替换的块号，将该块替换出去。

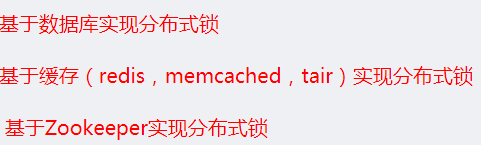
先进先出算法：将最先进入Cache的信息块替换出去。

近期最少使用算法（LRU）：将近期最少使用的Cache中的信息块替换出去。

最优替换算法（OPT）：必须先执行一次程序，统计Cache的替换情况，有了这样的先验信息，在第二次执行该程序时可以用最有效的方式来替换，以达到最优的目的。

1. 如何理解分布式锁

分布式锁是控制分布式系统之间同步访问共享资源的一种方式。在分布式系统中，常常需要协调他们的动作。如果不同的系统或是同一个系统的不同主机之间共享了一个或一组资源，那么访问这些资源的时候，往往需要互斥来防止彼此干扰来保证一致性，在这种情况下，往往会用到分布式锁。



## 1.基于数据库实现分布式锁

使用数据库来实现分布式锁的方式，两种方式都是依赖数据库的一张表，一种是通过表中的记录的存在情况确定当前是否有锁存在，另外一种是通过数据库的排他锁来实现分布式锁。

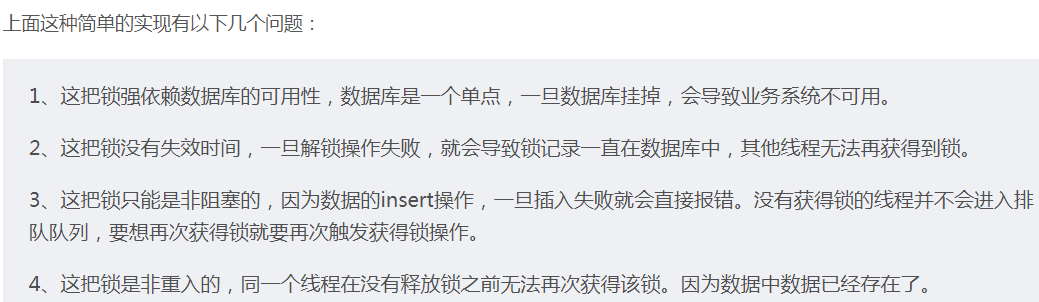
### 1.1锁住某个方法

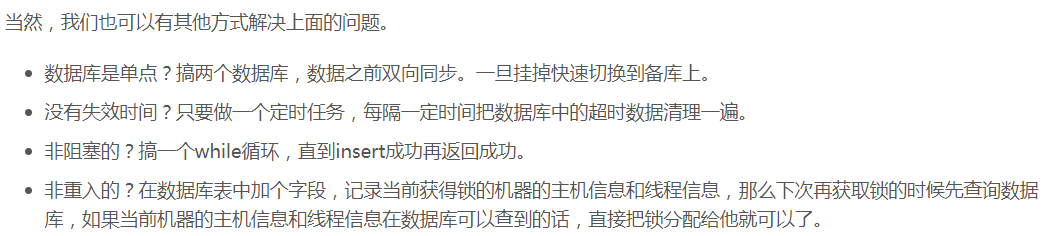


因为我们对method\_name做了唯一性约束，这里如果有多个请求同时提交到数据库的话，数据库会保证只有一个操作可以成功，那么我们就可以认为操作成功的那个线程获得了该方法的锁，可以执行方法体内容。

当方法执行完毕之后，想要释放锁的话，需要执行以下Sql:



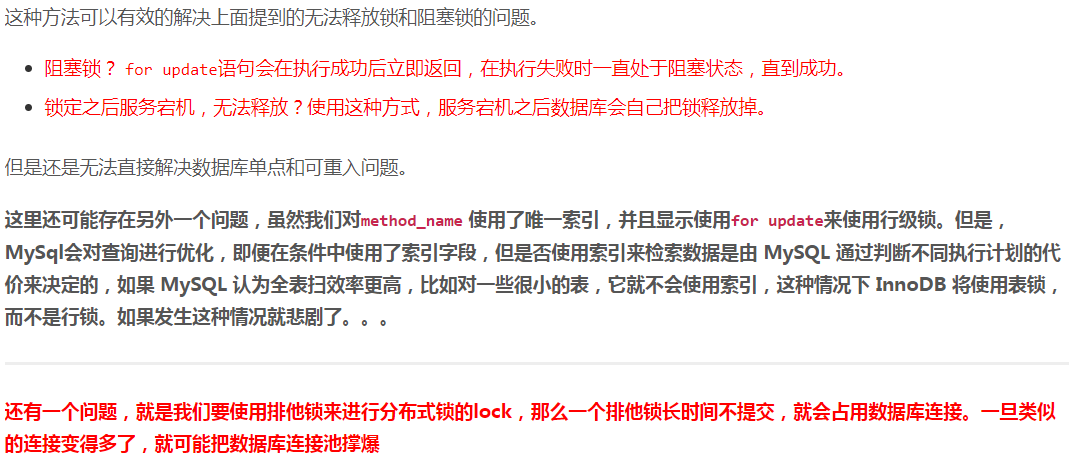




### 1.2借助数据中自带的锁来实现分布式的锁

基于MySql的InnoDB引擎，在查询语句后面增加for update，数据库会在查询过程中给数据库表增加排他锁（这里再多提一句，InnoDB引擎在加锁的时候，只有通过索引进行检索的时候才会使用行级锁，否则会使用表级锁。这里我们希望使用行级锁，就要给method\_name添加索引，值得注意的是，这个索引一定要创建成唯一索引，否则会出现多个重载方法之间无法同时被访问的问题。重载方法的话建议把参数类型也加上。）。当某条记录被加上排他锁之后，其他线程无法再在该行记录上增加排他锁。

我们可以认为获得排它锁的线程即可获得分布式锁，当获取到锁之后，可以执行方法的业务逻辑，执行完方法之后，再通过connection.commit()操作来释放锁。



### 1.3总结

数据库实现分布式锁的优点：

1.直接借助数据库，容易理解。

数据库实现分布式锁的缺点：

1.会有各种各样的问题，在解决问题的过程中会使整个方案变得越来越复杂。

2.操作数据库需要一定的开销，性能问题需要考虑。

3.使用数据库的行级锁并不一定靠谱，尤其是当我们的锁表并不大的时候。

## 2.基于缓存

相比较于基于数据库实现分布式锁的方案来说，基于缓存来实现在性能方面会表现的更好一点。而且很多缓存是可以集群部署的，可以解决单点问题。目前有很多成熟的缓存产品，包括Redis，memcached。

使用缓存实现分布式锁的优点：性能好，实现起来较为方便。

使用缓存实现分布式锁的缺点：通过超时时间来控制锁的失效时间并不是十分的靠谱。

## 3.zookeeper

大致思想即为：每个客户端对某个方法加锁时，在zookeeper上的与该方法对应的指定节点的目录下，生成一个唯一的瞬时有序节点。判断是否获取锁的方式很简单，只需要判断有序节点中序号最小的一个。当释放锁的时候，只需将这个瞬时节点删除即可。同时，其可以避免服务宕机导致的锁无法释放，而产生的死锁问题。

来看下Zookeeper能不能解决前面提到的问题。

1.锁无法释放？使用Zookeeper可以有效的解决锁无法释放的问题，因为在创建锁的时候，客户端会在ZK中创建一个临时节点，一旦客户端获取到锁之后突然挂掉（Session连接断开），那么这个临时节点就会自动删除掉。其他客户端就可以再次获得锁。

2.非阻塞锁？使用Zookeeper可以实现阻塞的锁，客户端可以通过在ZK中创建顺序节点，并且在节点上绑定监听器，一旦节点有变化，Zookeeper会通知客户端，客户端可以检查自己创建的节点是不是当前所有节点中序号最小的，如果是，那么自己就获取到锁，便可以执行业务逻辑了。

3.不可重入？使用Zookeeper也可以有效的解决不可重入的问题，客户端在创建节点的时候，把当前客户端的主机信息和线程信息直接写入到节点中，下次想要获取锁的时候和当前最小的节点中的数据比对一下就可以了。如果和自己的信息一样，那么自己直接获取到锁，如果不一样就再创建一个临时的顺序节点，参与排队。

4.单点问题？使用Zookeeper可以有效的解决单点问题，ZK是集群部署的，只要集群中有半数以上的机器存活，就可以对外提供服务。

23.存储系统：寄存器、快速缓存、内存、外存

在内存和处理器之间有个cache主要解决内存和处理器的速度差别

内存和外存之间有个辅助存储，页式（按照物理的大小进行分解） 段式（程序的逻辑） 所以出现了段页式

页式（进程在内存中不一定要连续分配，但一定要一次性装入）

24.文件

文件的物理结构：连续 链式的 索引的

文件怎么取

目录的组织形式:B+ 树

磁盘空间：空闲空间的管理

软连接和硬链接：软连接如windows快捷方式，新建的链接可以只想不存在的文件，硬链接共享inode号（所以说硬链接不能跨越磁盘分区）