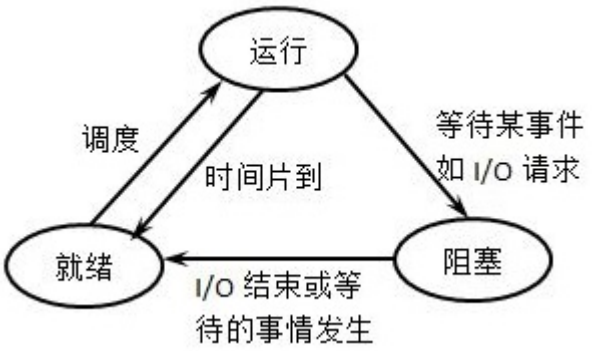
1.进程的常见状态？以及各种状态之间的转换条件？

- 就绪：进程已处于准备好运行的状态，即进程已分配到除CPU外的所有必要资源后，只要再获得CPU，便可立即执行。
- 执行：进程已经获得CPU，程序正在执行状态。
- 阻塞：正在执行的进程由于发生某事件（如I/O请求、申请缓冲区失败等）暂时无法继续执行的状态。



2.进程同步

任务：多个相关进程在执行次序上进行协调，使并发执行的诸进程之间能有效地共享资源和相互合作。

临界资源：是一次仅允许一个进程使用的共享资源。各进程采取互斥的方式，实现共享的资源称作临界资源。属于临界资源的硬件有，打印机，磁带机等；软件有消息队列，变量，数组，缓冲区等。诸进程间采取互斥方式，实现对这种资源的共享。

临界区：每个进程中访问临界资源的那段代码称为临界区（criticalsection），每次只允许一个进程进入临界区，进入后，不允许其他进程进入。不论是硬件临界资源还是软件临界资源，多个进程必须互斥的对它进行访问。

同步机制遵循的原则：

（1）空闲让进：临界区空闲时，可以允许一个请求进入临界区的进程立即进入临界区。

（2）忙则等待：当进程已经进入临界区时，其他试图进入临界区的进程必须等待。

（3）有限等待：对请求访问的进程，应保证能在有限的时间内进入临界区。

（4）让权等待:当进程不能进入临界区时，应立即释放处理器，防止进程忙等待。

3.进程的通信方式有哪些？

- **管道**：**管道是单向的、先进先出的、无结构的、固定大小的字节流，它把一个进程的标准输出和另一个进程的标准输入连接在一起。**

注1：无名管道只能实现父子或者兄弟进程之间的通信，有名管道（FIFO）可以实现互不相关的两个进程之间的通信。

注2：用FIFO让一个服务器和多个客户端进行交流时候，每个客户在向服务器发送信息前建立自己的读管道，或者让服务器在得到数据后再建立管道。使用客户的进程号（pid）作为管道名是一种常用的方法。客户可以先把自己的进程号告诉服务器，然后再到那个以自己进程号命名的管道中读取回复。

- **信号量**：**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，其它进程也访问该资源。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。

- **消息队列**：**是一个在系统内核中用来保存消 息的队列，它在系统内核中是以消息链表的形式出现的。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。**
- **共享内存**：共享内存允许两个或多个进程访问同一个逻辑内存。**这一段内存可以被两个或两个以上的进程映射至自身的地址空间中**，一个进程写入共享内存的信息，可以被其他使用这个共享内存的进程，通过一个简单的内存读取，从而实现了进程间的通信**。如果某个进程向共享内存写入数据，所做的改动将立即影响到可以访问同一段共享内存的任何其他进程。**共享内存是最快的IPC方式，它是针对其它进程间通信方式运行效率低而专门设计的。它往往与其它通信机制（如**信号量**）配合使用，来实现进程间的同步和通信。
- **套接字**：套接字也是一种进程间通信机制，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。

4.上下文切换

对于单核单线程CPU而言，在某一时刻只能执行一条CPU指令。上下文切换(Context Switch)是一种**将CPU资源从一个进程分配给另一个进程的机制。**从用户角度看，计算机能够并行运行多个进程，这恰恰是操作系统通过快速上下文切换造成的结果**。在切换的过程中，操作系统需要先存储当前进程的状态(包括内存空间的指针，当前执行完的指令等等)，再读入下一个进程的状态，然后执行此进程。**

5.进程与线程的区别和联系?

- **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。

进程和线程的关系

（1）一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程。线程是操作系统可识别的最小执行和调度单位。

（2）资源分配给进程，同一进程的所有线程共享该进程的所有资源。 同一进程中的多个线程共享代码段(代码和常量)，数据段(全局变量和静态变量)，扩展段(堆存储)。但是每个线程拥有自己的栈段，栈段又叫运行时段，用来存放所有局部变量和临时变量。

（3）处理机分给线程，即真正在处理机上运行的是线程。

（4）线程在执行过程中，需要协作同步。不同进程的线程间要利用消息通信的办法实现同步。

进程与线程的区别?

（1）进程有自己的独立地址空间，线程没有

（2）进程是资源分配的最小单位，线程是CPU调度的最小单位

（3）进程和线程通信方式不同(线程之间的通信比较方便。同一进程下的线程共享数据（比如全局变量，静态变量），通过这些数据来通信不仅快捷而且方便，当然如何处理好这些访问的同步与互斥正是编写多线程程序的难点。而进程之间的通信只能通过进程通信的方式进行。)

（4）进程上下文切换开销大，线程开销小

（5）一个进程挂掉了不会影响其他进程，而线程挂掉了会影响其他线程

为什么进程上下文切换比线程上下文切换代价高?

进程切换分两步：

1.切换页目录以使用新的地址空间

2.切换内核栈和硬件上下文

对于linux来说，线程和进程的最大区别就在于地址空间，对于线程切换，第1步是不需要做的，第2是进程和线程切换都要做的。

切换的性能消耗：

1、线程上下文切换和进程上下问切换一个最主要的区别是线程的切换虚拟内存空间依然是相同的，但是进程切换是不同的。这两种上下文切换的处理都是通过操作系统内核来完成的。内核的这种切换过程伴随的最显著的性能损耗是将寄存器中的内容切换出。

2、另外一个隐藏的损耗是上下文的切换会扰乱处理器的缓存机制。简单的说，一旦去切换上下文，处理器中所有已经缓存的内存地址一瞬间都作废了。还有一个显著的区别是当你改变虚拟内存空间的时候，处理的页表缓冲（processor's Translation Lookaside Buffer (TLB)）或者相当的神马东西会被全部刷新，这将导致内存的访问在一段时间内相当的低效。但是在线程的切换中，不会出现这个问题。

进程和线程都是一个时间段的描述，是CPU工作时间段的描述，不过是颗粒大小不同。

- 进程(process)与线程(thread)最大的区别是**进程拥有自己的地址空间，某进程内的线程对于其他进程不可见，即进程A不能通过传地址的方式直接读写进程B的存储区域。**进程之间的通信需要通过进程间通信(Inter-process communication，IPC)。与之相对的，**同一进程的各线程间之间可以直接通过传递地址或全局变量的方式传递信息。**
- **进程作为操作系统中拥有资源和独立调度的基本单位，可以拥有多个线程。**通常操作系统中运行的一个程序就对应一个进程。在同一进程中，线程的切换不会引起进程切换。在不同进程中进行线程切换，如从一个进程内的线程切换到另一个进程中的线程时，会引起进程切换。**相比进程切换，线程切换的开销要小很多。线程于进程相互结合能够提高系统的运行效率。**

线程可以分为两类：

- **用户级线程(user level thread)**：对于这类线程，有关线程管理的所有工作都由应用程序完成，内核意识不到线程的存在。在应用程序启动后，操作系统分配给该程序一个进程号，以及其对应的内存空间等资源。应用程序通常先在一个线程中运行，该线程被成为主线程。在其运行的某个时刻，可以通过调用线程库中的函数创建一个在相同进程中运行的新线程。**用户级线程的好处是非常高效，不需要进入内核空间，但并发效率不高。**
- **内核级线程(kernel level thread)**：对于这类线程，有关线程管理的所有工作由内核完成，应用程序没有进行线程管理的代码，只能调用内核线程的接口。内核维护进程及其内部的每个线程，调度也由内核基于线程架构完成。内核级线程的好处是，**内核可以将不同线程更好地分配到不同的CPU，以实现真正的并行计算。**

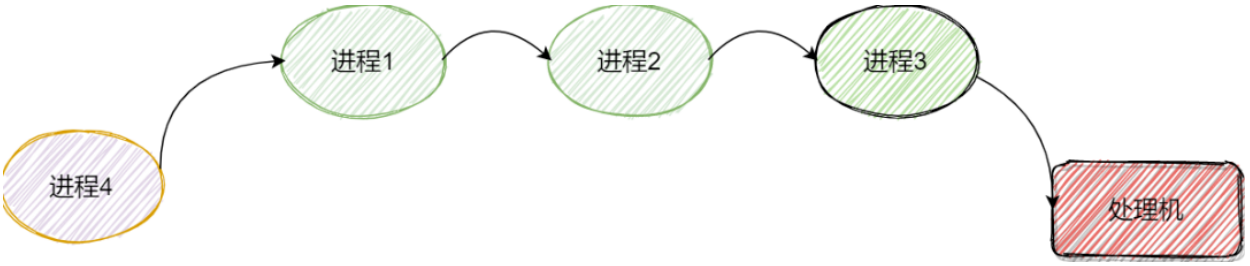
事实上，在现代操作系统中，往往使用组合方式实现多线程，即线程创建完全在用户空间中完成，并且一个应用程序中的多个用户级线程被映射到一些内核级线程上，相当于是一种折中方案。

6.进程调度算法

常用的调度算法有：先来先服务调度算法、时间片轮转调度法、短作业优先调度算法、最短剩余时间优先、高响应比优先调度算法、优先级调度算法等等。

- **先来先服务调度算法**

先来先服务让我们想起了队列的先进先出特性，每一次的调度都从队列中选择最先进入队列的投入运行。



先来先服务

- **时间片轮转调度算法**

先来理解**轮转**，假设当前进程A、B、C、D，按照进程到达的时间**排序**，而且每个进行都有着同样大小的**时间片**。如果这个进程在当前的时间片运行结束，啥事儿没有，直接将进程从队列**移除**完事儿。如果进程在这个时间片跑完都没有结束，进程变为等待状态，放在进程尾部直到所有进程执行完毕。

为什么进程要**切换**，切换无外乎是时间片**够用**或者**不够用**。如果时间片够用，那么进程可以运行到结束，结束后删除启动新的时间片。

- **短作业(进程)优先调度算法**

短作业(进程)优先调度算法SJ(P)F，是指对短作业或短进程优先调度的算法。它们可以分别用于作业调度和进程调度。短作业优先(SJF)的调度算法是从后备队列中选择一个或若干个估计运行时间最短的作业，将它们调入内存运行。而短进程优先(SPF)调度算法则是从就绪队列中选出一个估计运行时间最短的进程，将处理机分配给它，使它立即执行并一直执行到完成，或发生某事件而被阻塞放弃处理机时再重新调度。

7.死锁的条件？以及如何处理死锁问题？

定义:多个进程因循环等待资源而造成无法执行的现象。死锁会造成进程无法执行，同时会造成系统资源的极大浪费(资源无法释放)。

产生死锁的必要条件：

- **互斥条件(Mutual exclusion)**：资源不能被共享，只能由一个进程使用。
- **请求与保持条件(Hold and wait)**：已经得到资源的进程可以再次申请新的资源。
- **非抢占条件(No pre-emption)**：已经分配的资源不能从相应的进程中被强制地剥夺。
- **循环等待条件(Circular wait)**：系统中若干进程组成环路，该环路中每个进程都在等待相邻进程正占用的资源。

如何处理死锁问题：

- **忽略该问题。**例如鸵鸟算法，该算法可以应用在极少发生死锁的的情况下。为什么叫鸵鸟算法呢，因为传说中鸵鸟看到危险就把头埋在地底下，可能鸵鸟觉得看不到危险也就没危险了吧。跟掩耳盗铃有点像。
- **检测死锁并且恢复。**
- 仔细地对资源进行动态分配，使系统始终处于安全状态以**避免死锁。**
- **通过破除死锁四个必要条件之一，来防止死锁产生。**

如何避免死锁？

- **银行家算法**

当进程首次申请资源时，要测试该进程对资源的**最大需求量**，如果系统现存的资源可以满足它的最大需求量则按当前的申请量分配资源，否则就推迟分配。

当进程在执行中继续申请资源时，先测试该进程已占用的资源数与本次申请资源数之和是否超过了该进程对资源的最大需求量。若超过则拒绝分配资源。若没超过则再测试系统现存的资源能否满足该进程尚需的最大资源量，若满足则按当前的申请量分配资源，否则也要推迟分配。

- **安全序列**

是指系统能按某种进程推进顺序（P1, P2, P3, ..., Pn），为每个进程 Pi 分配其所需要的资源，直至满足每个进程对资源的最大需求，使每个进程都可以顺序地完成。这种推进顺序就叫安全序列【银行家算法的核心就是找到一个安全序列】。

- **系统安全状态**

如果系统能找到一个安全序列，就称系统处于安全状态，否则，就称系统处于不安全状态。

8.临界资源

- 在操作系统中，进程是占有资源的最小单位（线程可以访问其所在进程内的所有资源，但线程本身并不占有资源或仅仅占有一点必须资源）。但**对于某些资源来说，其在同一时间只能被一个进程所占用。这些一次只能被一个进程所占用的资源就是所谓的临界资源。**典型的临界资源比如物理上的打印机，或是存在硬盘或内存中被多个进程所共享的一些变量和数据等(如果这类资源不被看成临界资源加以保护，那么很有可能造成丢数据的问题)。
- **对于临界资源的访问，必须是互斥进行。**也就是当临界资源被占用时，另一个申请临界资源的进程会被阻塞，直到其所申请的临界资源被释放。**而进程内访问临界资源的代码被成为临界区。**

9.一个程序从开始运行到结束的完整过程（四个过程）

1、预处理：条件编译，头文件包含，宏替换的处理，生成.i文件。

2、编译：将预处理后的文件转换成汇编语言，生成.s文件

3、汇编：汇编变为目标代码(机器代码)生成.o的文件

4、链接：连接目标代码,生成可执行程序

链接

10.内存池、进程池、线程池。(c++程序员必须掌握)

首先介绍一个概念"池化技术 "。池化技术就是：提前保存大量的资源，以备不时之需以及重复使用。池化技术应用广泛，如内存池，线程池，连接池等等。内存池相关的内容，建议看看Apache、Nginx等开源web服务器的内存池实现。

由于在实际应用当做，分配内存、创建进程、线程都会设计到一些系统调用，系统调用需要导致程序从用户态切换到内核态，是非常耗时的操作。因此，当程序中需要频繁的进行内存申请释放，进程、线程创建销毁等操作时，通常会使用内存池、进程池、线程池技术来提升程序的性能。

线程池：线程池的原理很简单，类似于操作系统中的缓冲区的概念，它的流程如下：先启动若干数量的线程，并让这些线程都处于睡眠状态，当需要一个开辟一个线程去做具体的工作时，就会唤醒线程池中的某一个睡眠线程，让它去做具体工作，当工作完成后，线程又处于睡眠状态，而不是将线程销毁。

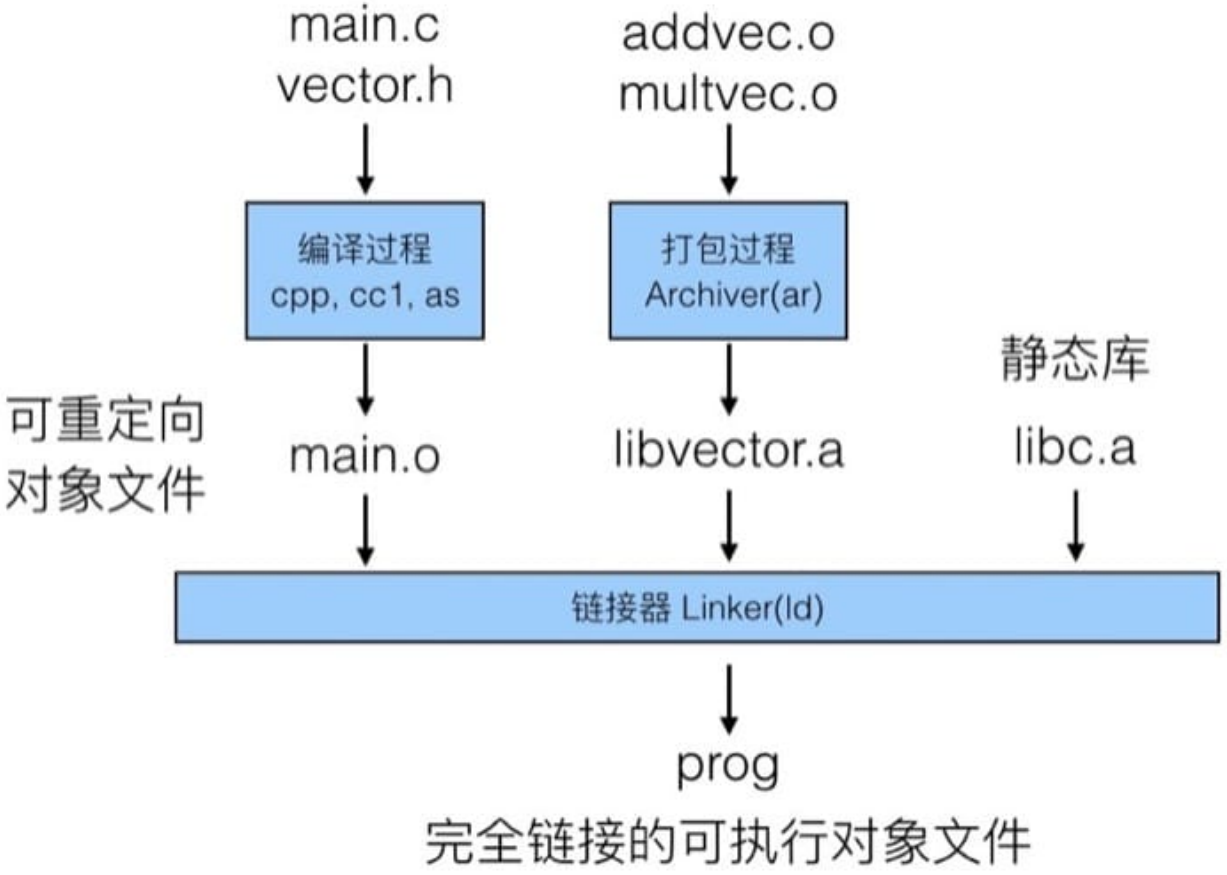
进程池与线程池同理。

内存池：内存池是指程序预先从操作系统申请一块足够大内存，此后，当程序中需要申请内存的时候，不是直接向操作系统申请，而是直接从内存池中获取；同理，当程序释放内存的时候，并不真正将内存返回给操作系统，而是返回内存池。当程序退出(或者特定时间)时，内存池才将之前申请的内存真正释放。

11.动态链接库与静态链接库的区别

静态库

- 静态库是一个外部函数与变量的集合体。静态库的文件内容，通常包含一堆程序员自定的变量与函数，其内容不像动态链接库那么复杂，在编译期间由编译器与链接器将它集成至应用程序内，并制作成目标文件以及可以独立运作的可执行文件。而这个可执行文件与编译可执行文件的程序，都是一种程序的静态创建（static build）。

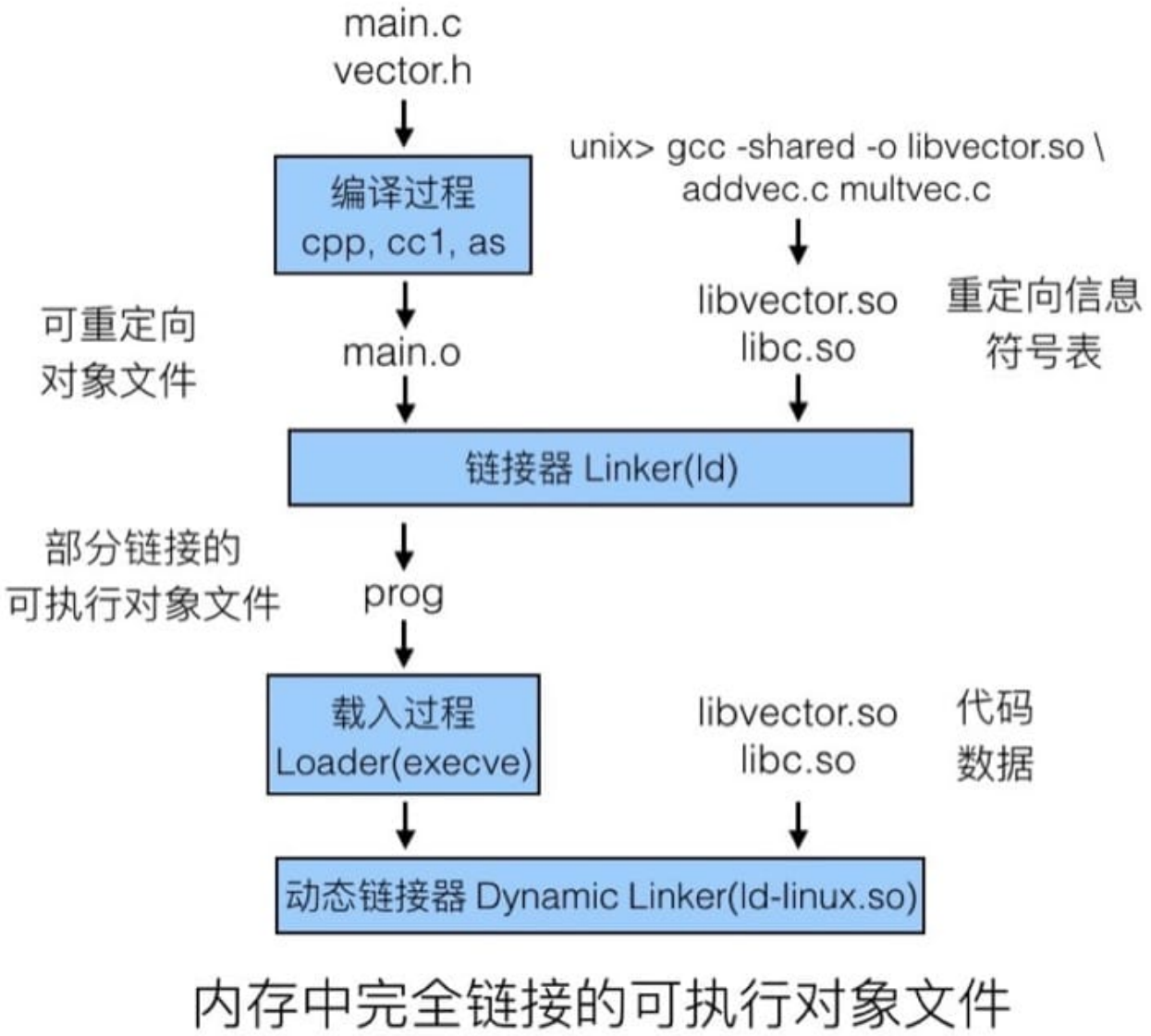


动态库

- 静态库很方便，但是如果我们只是想用库中的某一个函数，却仍然得把所有的内容都链接进去。一个更现代的方法则是使用共享库，避免了在文件中静态库的大量重复。
- 动态链接可以在首次载入的时候执行(load-time linking)，这是 Linux 的标准做法，会由动态链接器ld-linux.so 完成，比方标准 C 库(libc.so) 通常就是动态链接的，这样**所有的程序可以共享同一个库，而不用分别进行封装。**
- 动态链接也可以在程序开始执行的时候完成(run-time linking)，在 Linux 中使用 dlopen() 接口来完成（会使用函数指针），通常用于分布式软件，高性能服务器上。而且共享库也可以在多

个进程间共享。

- **链接使得我们可以用多个对象文件构造我们的程序。可以在程序的不同阶段进行（编译、载入、运行期间均可），理解链接可以帮助我们避免遇到奇怪的错误。**



区别：

1．使用静态库的时候，静态链接库要参与编译，在生成执行文件之前的链接过程中，要将静态链接库的全部指令直接链接入可执行文件中。而动态库提供了一种方法，使进程可以调用不属于其可执行代码的函数。函数的可执行代码位于一个.dll文件中，该dll包含一个或多个已被编译，链接并与使用它们的进程分开储存的函数。

2．静态库中不能再包含其他动态库或静态库，而在动态库中还可以再包含其他动态或者静态库。

3．静态库在编译的时候，就将库函数装在到程序中去了，而动态库函数必须在运行的时候才被装载，所以使用静态库速度快一些。

链接

12.虚拟内存？优缺点？

定义：具有请求调入功能和置换功能，能从逻辑上对内存容量加以扩充得一种存储器系统。其逻辑容量由内存之和和外存之和决定。

与传统存储器比较虚拟存储器有以下三个主要特征：

- 多次性，是指无需在作业运行时一次性地全部装入内存，而是允许被分成多次调入内存运行。

- 对换性，是指无需在作业运行时一直常驻内存，而是允许在作业的运行过程中，进行换进和换出。
- 虚拟性，是指从逻辑上扩充内存的容量，使用户所看到的内存容量，远大于实际的内存容量。

虚拟内存的实现有以下两种方式：

- 请求分页存储管理。
- 请求分段存储管理。

13.页面置换算法

操作系统将内存按照页面进行管理，在需要的时候才把进程相应的部分调入内存。当产生缺页中断时，需要选择一个页面写入。如果要换出的页面在内存中被修改过，变成了"脏"页面，那就需要先写会到磁盘。页面置换算法，就是要选出最合适的一个页面，使得置换的效率最高。页面置换算法有很多，简单介绍几个，重点介绍比较重要的LRU及其实现算法。

一、最优页面置换算法

最理想的状态下，我们给页面做个标记，挑选一个最远才会被再次用到的页面调出。当然，这样的算法不可能实现，因为不确定一个页面在何时会被用到。

二、先进先出页面置换算法（FIFO）及其改进

这种算法的思想和队列是一样的，该算法总是淘汰最先进入内存的页面，即选择在内存中驻留时间最久的页面予淘汰。实现：把一个进程已调入内存的页面按先后次序链接成一个队列，并且设置一个指针总是指向最老的页面。缺点：对于有些经常被访问的页面如含有全局变量、常用函数、例程等的页面，不能保证这些不被淘汰。

三、最近最少使用页面置换算法LRU（Least Recently Used）

根据页面调入内存后的使用情况做出决策。LRU置换算法是选择最近最久未使用的页面进行淘汰。

1.为每个在内存中的页面配置一个移位寄存器。（P165）定时信号将每隔一段时间将寄存器右移一位。最小数值的寄存器对应页面就是最久未使用页面。

2.利用一个特殊的栈保存当前使用的各个页面的页面号。每当进程访问某页面时，便将该页面的页面号从栈中移出，将它压入栈顶。因此，栈顶永远是最新被访问的页面号，栈底是最近最久未被访问的页面号。

链接：分页内存管理（把虚拟内存空间和物理内存空间均划分为大小相同的页面等内容）

链接：分段内存管理

14.中断与系统调用

所谓的中断就是在计算机执行程序的过程中，由于出现了某些特殊事情，使得CPU暂停对程序的执行，转而去执行处理这一事件的程序。等这些特殊事情处理完之后再回去执行之前的程序。中断一般分为三类：

- 由计算机硬件异常或故障引起的中断，称为**内部异常中断**；
- 由程序中执行了引起中断的指令而造成的中断，称为**软中断**（这也是和我们将要说明的系统调用相关的中断）；
- 由外部设备请求引起的中断，称为**外部中断。**简单来说，对中断的理解就是对一些特殊事情的处理。

与中断紧密相连的一个概念就是**中断处理程序**了。当中断发生的时候，系统需要去对中断进行处理，对这些中断的处理是由操作系统内核中的特定函数进行的，这些处理中断的特定的函数就是我们所说的中断处理程序了。

另一个与中断紧密相连的概念就是**中断的优先级。**中断的优先级说明的是当一个中断正在被处理的时候，处理器能接受的中断的级别。中断的优先级也表明了中断需要被处理的紧急程度。**每个中断都有一个对应**

的优先级，当处理器在处理某一中断的时候，只有比这个中断优先级高的中断可以被处理器接受并且被处理。优先级比这个当前正在被处理的中断优先级要低的中断将会被忽略。

典型的中断优先级如下所示：

- **机器错误 > 时钟 > 磁盘 > 网络设备 > 终端 > 软件中断**

在讲系统调用之前，先说下**进程的执行在系统上的两个级别**：用户级和核心级，也称为**用户态和系统态(user mode and kernel mode)。**

用户空间就是用户进程所在的内存区域，相对的，**系统空间就是操作系统占据的内存区域。**用户进程和系统进程的所有数据都在内存中。**处于用户态的程序只能访问用户空间，而处于内核态的程序可以访问用户空间和内核空间。**

用户态切换到内核态的方式如下：

- **系统调用**：程序的执行一般是在用户态下执行的，但当程序需要使用操作系统提供的服务时，比如说打开某一设备、创建文件、读写文件（这些均属于系统调用）等，就需要向操作系统发出调用服务的请求，这就是系统调用。
- **异常**：当CPU在执行运行在用户态下的程序时，发生了某些事先不可知的异常，这时会触发由当前运行进程切换到处理此异常的内核相关程序中，也就转到了内核态，比如缺页异常。
- **外围设备的中断：**当外围设备完成用户请求的操作后，会向CPU发出相应的中断信号，这时CPU会暂停执行下一条即将要执行的指令转而去执行与中断信号对应的处理程序，如果先前执行的指令是用户态下的程序，那么这个转换的过程自然也就发生了由用户态到内核态的切换。比如硬盘读写操作完成，系统会切换到硬盘读写的中断处理程序中执行后续操作等。

用户态和内核态之间的区别是什么呢?

权限不一样。

- **用户态的进程能存取它们自己的指令和数据，但不能存取内核指令和数据（或其他进程的指令和数据）。**
- **内核态下的进程能够存取内核和用户地址某些机器指令是特权指令，在用户态下执行特权指令会引起错误。**在系统中内核并不是作为一个与用户进程平行的估计的进程的集合。

15.C++多线程，互斥，同步

同步和互斥

当有多个线程的时候，经常需要去同步(注：同步不是同时刻)这些线程以访问同一个数据或资源。例如，假设有一个程序，其中一个线程用于把文件读到内存，而另一个线程用于统计文件中的字符数。当然，在把整个文件调入内存之前，统计它的计数是没有意义的。但是，由于每个操作都有自己的线程，操作系统会把两个线程当作是互不相干的任务分别执行，这样就可能在没有把整个文件装入内存时统计字数。为解决此问题，你必须使两个线程同步工作。

所谓同步，是指在不同进程之间的若干程序片断，它们的运行必须严格按照规定的某种先后次序来运行，这种先后次序依赖于要完成的特定的任务。如果用对资源的访问来定义的话，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。少数情况是指可以允许多个访问者同时访问资源。

所谓互斥，是指散布在不同进程之间的若干程序片断，当某个进程运行其中一个程序片段时，其它进程就不能运行它们之中的任一程序片段，只能等到该进程运行完这个程序片段后才可以运行。如果用对资源的访问来定义的话，互斥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。

多线程同步和互斥有几种实现方法

线程间的同步方法大体可分为两类：用户模式和内核模式。顾名思义，内核模式就是指利用系统内核对象的单一性来进行同步，使用时需要切换内核态与用户态，而用户模式就是不需要切换到内核态，只在用户态完成操作。

用户模式下的方法有：原子操作（例如一个单一的全局变量），临界区。

内核模式下的方法有：事件，信号量，互斥量。

1、临界区:通过对多线程的串行化来访问公共资源或一段代码，速度快，适合控制数据访问。

2、互斥量:为协调共同对一个共享资源的单独访问而设计的。

3、信号量:为控制一个具有有限数量用户资源而设计。

4、事 件:用来通知线程有一些事件已发生，从而启动后继任务的开始。

16.逻辑地址 Vs 物理地址 Vs 虚拟内存

- **所谓的逻辑地址，是指计算机用户(例如程序开发者)，看到的地址。**例如，当创建一个长度为100的整型数组时，操作系统返回一个逻辑上的连续空间：指针指向数组第一个元素的内存地址。由于整型元素的大小为4个字节，故第二个元素的地址时起始地址加4，以此类推。事实上，**逻辑地址并不一定是元素存储的真实地址，即数组元素的物理地址(在内存条中所处的位置)，并非是连续的，只是操作系统通过地址映射，将逻辑地址映射成连续的，这样更符合人们的直观思维。**
- 另一个重要概念是虚拟内存。操作系统读写内存的速度可以比读写磁盘的速度快几个量级。但是，内存价格也相对较高，不能大规模扩展。于是，**操作系统可以通过将部分不太常用的数据移出内存，"存放到价格相对较低的磁盘缓存，以实现内存扩展。**操作系统还可以通过算法预测哪部分存储到磁盘缓存的数据需要进行读写，提前把这部分数据读回内存。**虚拟内存空间相对磁盘而言要小很多，因此，即使搜索虚拟内存空间也比直接搜索磁盘要快。唯一慢于磁盘的可能是，内存、虚拟内存中都没有所需要的数据，最终还需要从硬盘中直接读取。**这就是为什么内存和虚拟内存中需要存储会被重复读写的数据，否则就失去了缓存的意义。现代计算机中有一个专门的**转译缓冲区(Translation Lookaside Buffer，TLB)**，用来实现虚拟地址到物理地址的快速转换。

与内存 / 虚拟内存相关的还有如下两个概念：

1) Resident Set

- 当一个进程在运行的时候，操作系统不会一次性加载进程的所有数据到内存，只会加载一部分正在用，以及预期要用的数据。其他数据可能存储在虚拟内存，交换区和硬盘文件系统上。**被加载到内存的部分就是resident set。**

2) Thrashing

- 由于resident set包含预期要用的数据，理想情况下，进程运行过程中用到的数据都会逐步加载进resident set。但事实往往并非如此：**每当需要的内存页面(page)不在resident set中时，操作系统必须从虚拟内存或硬盘中读数据，这个过程被称为内存页面错误(page faults)。当操作系统需要花费大量时间去处理页面错误的情况就是thrashing。**

参考链接：https://blog.csdn.net/newcong0123/article/details/52792070

17.内部碎片与外部碎片

在内存管理中，内部碎片是已经被分配出去的的内存空间大于请求所需的内存空间。

外部碎片是指还没有分配出去，但是由于大小太小而无法分配给申请空间的新进程的内存空间空闲块。

固定分区存在内部碎片，可变式分区分配会存在外部碎片；

页式虚拟存储系统存在内部碎片；段式虚拟存储系统，存在外部碎片

为了有效的利用内存，使内存产生更少的碎片，要对内存分页，内存以页为单位来使用，最后一页往往装不满，于是形成了内部碎片。

为了共享要分段，在段的换入换出时形成外部碎片，比如5K的段换出后，有一个4k的段进来放到原来5k的地方，于是形成1k的外部碎片。

18.同步和互斥的区别

当有多个线程的时候，经常需要去同步这些线程以访问同一个数据或资源。例如，假设有一个程序，其中一个线程用于把文件读到内存，而另一个线程用于统计文件中的字符数。当然，在把整个文件调入内存之前，统计它的计数是没有意义的。但是，由于每个操作都有自己的线程，操作系统会把两个线程当作是互不相干的任务分别执行，这样就可能在没有把整个文件装入内存时统计字数。为解决此问题，你必须使两个线程同步工作。

所谓同步，是指散步在不同进程之间的若干程序片断，它们的运行必须严格按照规定的某种先后次序来运行，这种先后次序依赖于要完成的特定的任务。如果用对资源的访问来定义的话，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。少数情况是指可以允许多个访问者同时访问资源。

所谓互斥，是指散布在不同进程之间的若干程序片断，当某个进程运行其中一个程序片段时，其它进程就不能运行它们之中的任一程序片段，只能等到该进程运行完这个程序片段后才可以运行。如果用对资源的访问来定义的话，互斥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。

19.什么是线程安全

如果多线程的程序运行结果是可预期的，而且与单线程的程序运行结果一样，那么说明是“线程安全”的。

20.同步与异步

同步：

- 同步的定义：是指一个进程在执行某个请求的时候，若该请求需要一段时间才能返回信息，那么，这个进程将会一直等待下去，直到收到返回信息才继续执行下去。
- 特点：

1. 同步是阻塞模式；
2. 同步是按顺序执行，执行完一个再执行下一个，需要等待，协调运行；

异步：

- 是指进程不需要一直等下去，而是继续执行下面的操作，不管其他进程的状态。当有消息返回时系统会通知进程进行处理，这样可以提高执行的效率。
- 特点：

1. 异步是非阻塞模式，无需等待；
2. 异步是彼此独立，在等待某事件的过程中，继续做自己的事，不需要等待这一事件完成后再工作。线程是异步实现的一个方式。

同步与异步的优缺点：

- 同步可以避免出现死锁，读脏数据的发生。一般共享某一资源的时候，如果每个人都有修改权限，同时修改一个文件，有可能使一个读取另一个人已经删除了内容，就会出错，同步就不会出错。但，同步需要等待资源访问结束，浪费时间，效率低。
- 异步可以提高效率，但，安全性较低。

21.系统调用与库函数的区别

- **系统调用(System call)是程序向系统内核请求服务的方式。**可以包括硬件相关的服务(例如，访问硬盘等)，或者创建新进程，调度其他进程等。系统调用是程序和操作系统之间的重要接口。
- **库函数：把一些常用的函数编写完放到一个文件里，编写应用程序时调用，这是由第三方提供的，发生在用户地址空间。**
- 在**移植性方面**，不同操作系统的系统调用一般是不同的，移植性差；而在所有的ANSI C编译器版本中，C库函数是相同的。
- 在**调用开销方面**，系统调用需要在用户空间和内核环境间切换，开销较大；而库函数调用属于"过程调用"，开销较小。

22.守护、僵尸、孤儿进程的概念

- **守护进程**：运行在后台的一种特殊进程，**独立于控制终端并周期性地执行某些任务。**
- **僵尸进程**：一个进程 fork 子进程，子进程退出，而父进程没有wait/waitpid子进程，那么**子进程的进程描述符仍保存在系统中**，这样的进程称为僵尸进程。
- **孤儿进程**：一个**父进程退出，而它的一个或多个子进程还在运行**，这些子进程称为孤儿进程。（孤儿进程将由 init 进程收养并对它们完成状态收集工作）
- **交互式进程**：是由shell启动的进程，它既可以前台运行，也可以后台运行。交互进程在执行过程中，要求与用户进行交互操作。
- **批处理进程**：是一个进程序列。该进程负责按照顺序启动其他进程。

23.Semaphore(信号量) Vs Mutex(互斥锁)

- 当用户创立多个线程 / 进程时，如果不同线程 / 进程同时读写相同的内容，则可能造成读写错误，或者数据不一致。此时，需要通过加锁的方式，控制临界区(critical section)的访问权限。对于semaphore而言，在初始化变量的时候可以控制允许多少个线程 / 进程同时访问一个临界区，其他的线程 / 进程会被堵塞，直到有人解锁。
- Mutex相当于只允许一个线程 / 进程访问的semaphore。此外，根据实际需要，人们还实现了一种读写锁(read-write lock)，它允许同时存在多个阅读者(reader)，但任何时候至多只有一个写者(writer)，且不能于读者共存。

24.IO多路复用

IO多路复用是指内核一旦发现进程指定的一个或者多个IO条件准备读取，它就通知该进程。IO多路复用适用如下场合：

- 当客户处理多个描述字时（一般是交互式输入和网络套接口），必须使用I/O复用。
- 当一个客户同时处理多个套接口时，而这种情况是可能的，但很少出现。
- 如果一个TCP服务器既要处理监听套接口，又要处理已连接套接口，一般也要用到I/O复用。
- 如果一个服务器即要处理TCP，又要处理UDP，一般要使用I/O复用。
- 如果一个服务器要处理多个服务或多个协议，一般要使用I/O复用。
- 与多进程和多线程技术相比，I/O多路复用技术的最大优势是系统开销小，系统不必创建进程/线程，也不必维护这些进程/线程，从而大大减小了系统的开销。

好处：

- IO多路复用是指内核一旦发现进程指定的一个或者多个IO条件准备读取，它就通知该进程。I/O多路复用技术的最大优势是系统开销小，系统不必创建进程/线程，也不必维护这些进程/线程，从而大大减小了系统的开销.

25.线程安全

如果你的代码所在的进程中有多个线程在同时运行，而这些线程可能会同时运行这段代码。如果每次运行结果和单线程运行的结果是一样的，而且其他的变量的值也和预期的是一样的，就是线程安全的。或者说:一个类或者程序所提供的接口对于线程来说是原子操作或者多个线程之间的切换不会导致该接口的执行结果存在二义性,也就是说我们不用考虑同步的问题。

线程安全问题都是由全局变量及静态变量引起的。

若每个线程中对全局变量、静态变量只有读操作，而无写操作，一般来说，这个全局变量是线程安全的；若有多个线程同时执行写操作，一般都需要考虑线程同步，否则的话就可能影响线程安全。

26.线程共享资源和独占资源问题

一个进程中的所有线程共享该进程的地址空间，但它们有各自独立的（/私有的）栈(stack)，Windows线程的缺省堆栈大小为1M。堆(heap)的分配与栈有所不同，一般是一个进程有一个C运行时堆，这个堆为本进程中所有线程共享，windows进程还有所谓进程默认堆，用户也可以创建自己的堆。

用操作系统术语，线程切换的时候实际上切换的是一个可以称之为线程控制块的结构（TCB）,里面保存所有将来用于恢复线程环境必须的信息，包括所有必须保存的寄存器集，线程的状态等。

堆: 是大家共有的空间，分全局堆和局部堆。全局堆就是所有没有分配的空间，局部堆就是用户分配的空间。堆在操作系统对进程初始化的时候分配，运行过程中也可以向系统要额外的堆，但是记得用完了要还给操作系统，要不然就是内存泄漏。

栈: 是个线程独有的，保存其运行状态和局部自动变量的。栈在线程开始的时候初始化，每个线程的栈互相独立，因此，栈是 thread safe的。操作系统在切换线程的时候会自动的切换栈，就是切换 S S / E S P寄存器。栈空间不需要在高级语言里面显式的分配和释放。

线程共享资源	线程独享资源
地址空间	程序计数器
全局变量	寄存器
打开的文件	栈
子进程	状态字
闹铃	
信号及信号服务程序	
记账信息	

27、进程的内存

代码段（只读）：代码段是用来存放可执行文件的操作指令，也就是说是它是可执行程序在内存中的镜像。代码段需要防止在运行时被非法修改，所以只准许读取操作，而不允许写入（修改）操作——它是不可写的。

数据段：数据段用来存放可执行文件中已初始化全局变量，换句话说就是存放程序静态分配[1]的变量和全局变量。

BSS段[2]：BSS段包含了程序中未初始化的全局变量，在内存中 bss段全部置零。

堆（heap）：堆是用于存放进程运行中被动态分配的内存段，它的大小并不固定，可动态扩张或缩减。当进程调用malloc等函数分配内存时，新分配的内存就被动态添加到堆上（堆被扩张）；当利用free等函数释放内存时，被释放的内存从堆中被剔除（堆被缩减）

栈：栈是用户存放程序临时创建的局部变量。除此以外，在函数被调用时，其参数也会被压入发起调用的进程栈中，并且待到调用结束后，函数的返回值也会被存放回栈中。由于栈的

先进先出特点，所以栈特别方便用来保存/恢复调用现场。从这个意义上讲，我们可以把堆栈看成一个寄存、交换临时数据的内存区。

进程内存管理

虚拟空间被划分为许多大小可变的(但必须是4096的倍数)内存区域，这些区域在进程线性地址中像停车位一样有序排列。这些区域的划分原则是“将访问属性一致的地址空间存放在一起”，所谓访问属性在这里无非指的是“可读、可写、可执行等”。

如果你要查看某个进程占用的内存区域，可以使用命令cat /proc/<pid>/maps获得

28、exit() _exit()的区别?

- _exit()执行后会立即返回给内核，而exit()要先执行一些清除操作，然后再将控制权交给内核
- 调用_exit()函数时，会关闭进程所有的文件描述符，清理内存，以及其他一些内核清理函数，但是不会刷新流（stdin，stdout...）。exit()函数是在_exit()函数上的一个封装，他会调用_exit(),并在调用之前先刷新流。
- exit()函数在调用exit系统之前要检查文件打开情况，把文件缓冲区的内容写回文件（保证数据的完整性）。

29、内存管理

https://www.cnblogs.com/CareySon/archive/2012/04/25/2470063.html

30、进程管理

https://www.cnblogs.com/leesf456/p/5413517.html

31、磁盘IO和网络IO

https://www.cnblogs.com/sunsky303/p/8962628.html

32、什么是缓冲区溢出？有什么危害?

官话

> 缓冲区溢出是指当计算机向缓冲区内填充数据时超过了缓冲区本身的容量，溢出的数据覆盖在合法数据上

举个例子

> 一个两升的杯子，你如果想导入三升，怎么做？其他一生只好流出去，不是打湿了电脑就是**那啥**。

1、程序崩溃，导致拒绝服务

2、跳转并且执行一段恶意代码

什么是协程，与线程区别

定义：

一种用户态的轻量级线程，完全由用户调度控制，拥有自己的寄存器上下文和栈，协程调度切换的时候，先将寄存器上下文和栈保存到其他地方，切换回来的时候再恢复之前保存的寄

存器上下文和栈。直接操作栈则基本没有内核切换的开销，可以不加锁的访问全局变量，所以上下文的切换非常快。

但是同一时间只能执行一个协程，大致来说是一系列互相依赖的协程间依次使用CPU，每次只有一个协程工作，而其他协程处于休眠状态，适合对任务进行分时处理；而线程，一次可以执行多个线程，适合执行多任务处理。

与线程的区别：

线程和协同程序的主要不同在于：在多处理器情况下，从概念上来讲多线程程序同时运行多个线程；而协同程序是通过协作来完成，在任一指定时刻只有一个协同程序在运行，并且这个正在运行的协同程序只在必要时才会被挂起。

孤儿进程和僵尸进程，产生的原因，什么样的代码会产生僵尸进程

孤儿进程：当父进程退出时，它的子进程们（一个或者多个）就成了孤儿进程了

僵尸进程：子进程先退出，而父进程又没有去处理回收释放子进程的资源，这个时候子进程就成了僵尸进程

系统调用和函数调用的区别

系统调用会陷入内核态，执行一些权限比较高的操作；函数调用只是调用栈中的函数，处在用户态.

互斥锁和自旋锁

互斥锁：用于保护临界区，确保同一时间只有一个线程访问数据。对共享资源的访问，先对互斥量进行加锁，如果互斥量已经上锁，调用线程会阻塞，直到互斥量被解锁。在完成了对共享资源的访问后，要对互斥量进行解锁

自旋锁：是一种互斥锁的实现方式，相比一般的互斥锁会在等待期间放弃cpu，自旋锁（spinlock）则是不断循环并测试锁的状态，这样就一直占着cpu

自旋锁与互斥锁的区别：线程在申请自旋锁的时候，线程不会被挂起，而是处于忙等的状态

什么是高级缓存，为什么有高级缓存

高速缓存就是为了弥补内存速度和CPU速度不匹配而加入的一级存储层次，靠近CPU，通常由片上的 L1 和 L2 缓存和片外 L3 缓存组成. 缓存的引入使得内存的数据可以缓存在cache中，同时由于cache采用SRAM，读写速度加快，缩小了CPU和内存之间的速度鸿沟.

多核 CPU 怎么保证一致性、顺序性

每个处理器必须按照程序指定的顺序处理内存请求

一个独立的内存模块在服务所有处理器的请求时，必须基于FIFO队列。发起内存请求，即是将请求加入到FIFO队列

什么是 CPU 指令重排

cpu为了提高效率会对指令进行重排序，以适合cpu的顺序运行。但是指令重排会遵守As-if-serial的规则，就是所有的动作(Action)都可以为了优化而被重排序，但是必须保证它们重排序后的结果和程序代码本身的应有结果是一致的。

虚拟内存的原理

虚拟内存 使得应用程序认为它拥有连续的可用的内存（一个连续完整的地址空间），而实际上，它通常是被分隔成多个物理内存碎片，还有部分暂时存储在外部磁盘存储器上，在需要时进行数据交换。