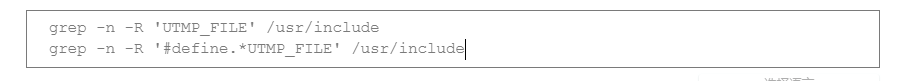
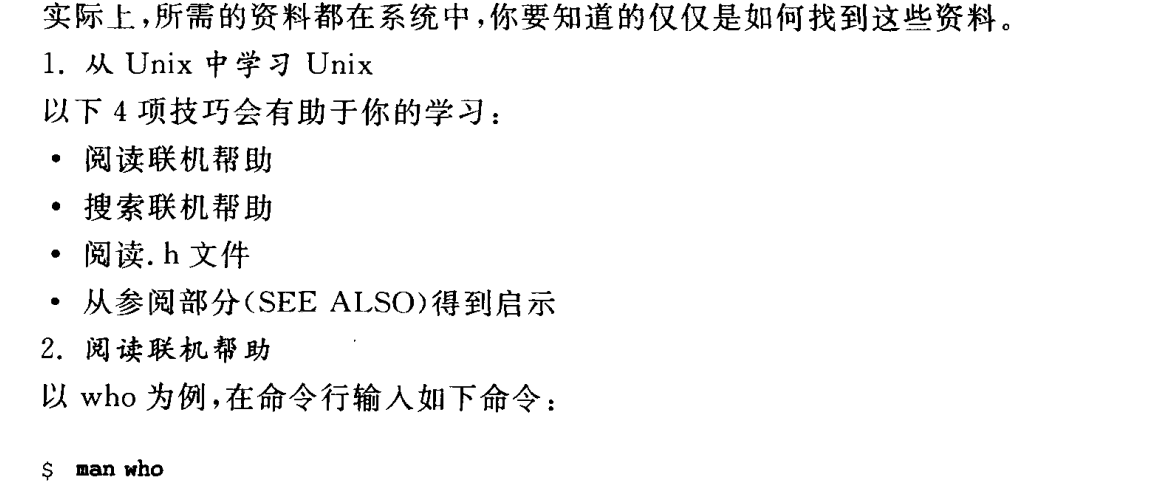
1、文件/dev/tty是键盘和显示器的文件描述符，向这个文件写相当于显示在屏幕上，读相当于从键盘获取用户的输入。

2、搜索某个变量在哪个头文件的时候，可以使用以下命令：



1. 如何找相关信息：

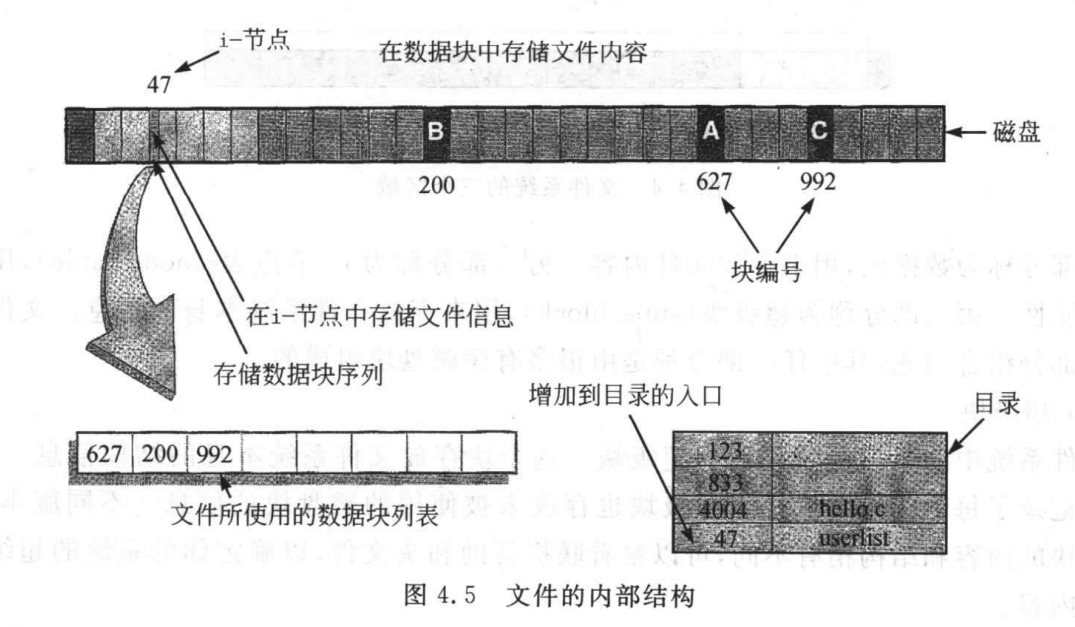


1. man详解

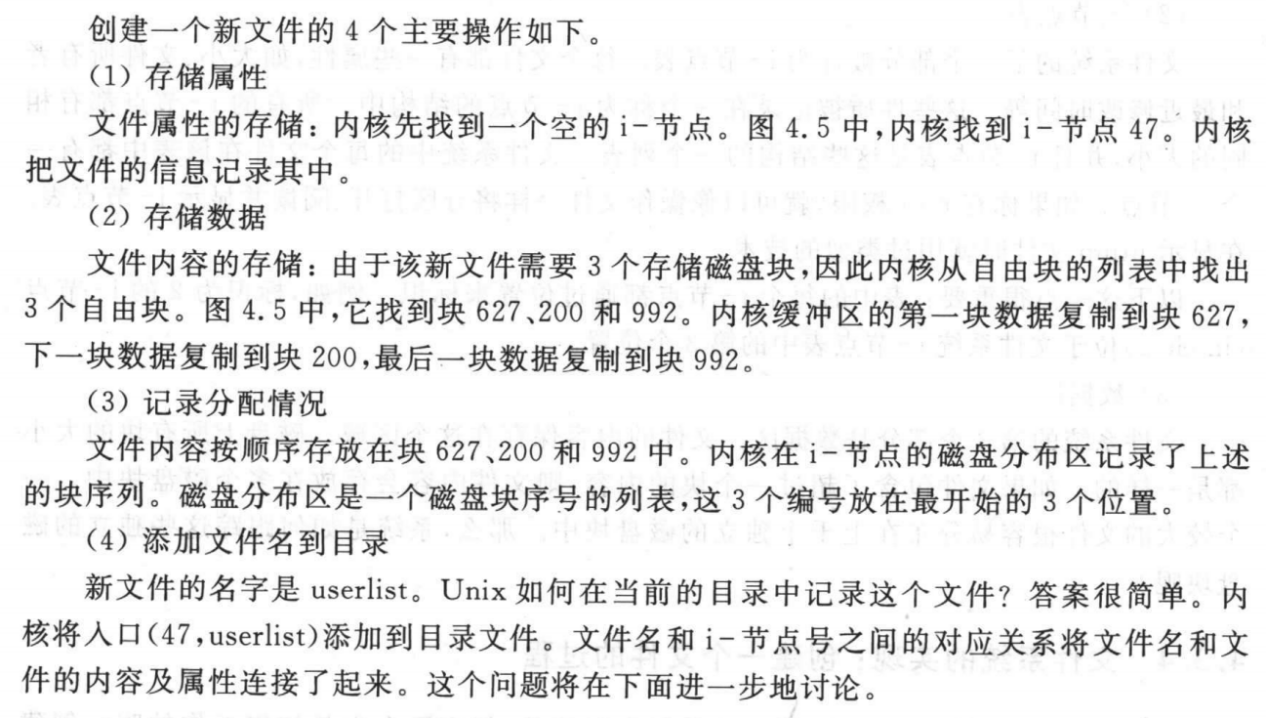


5、可以使用man -k keyword | grep 筛选条件 这样的形式去查找手册里面的联机帮助。

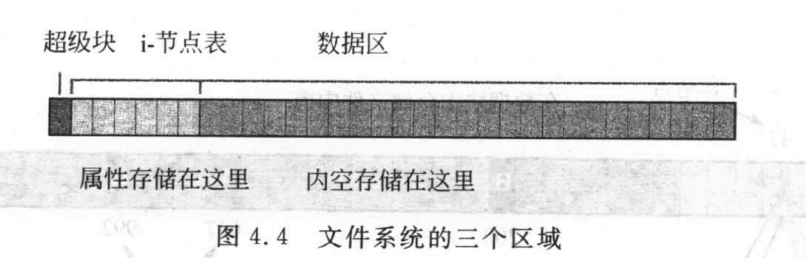
6、磁盘是如何存储数据的？



过程如下：



1. 按照书本字面上的理解，磁盘块上的结构是一样的，都是如下：



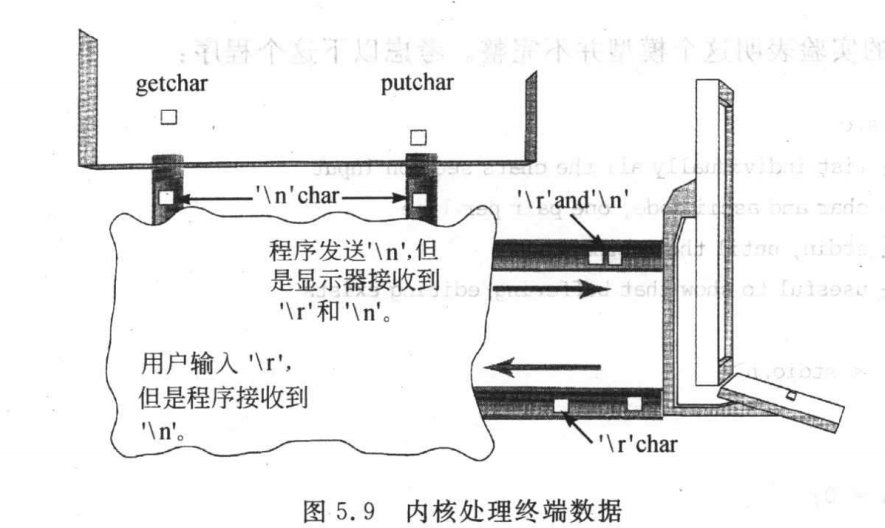
磁盘块可以和扇区大小一样，也可以不一样。

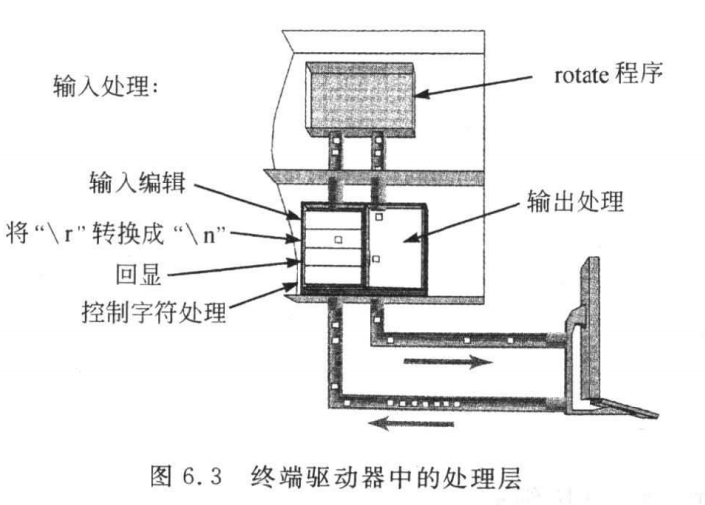
8、在Linux中，文件是一个i-结点和一些数据块的结合，链接是对i-结点的引用，因此，所谓的文件名，不是真正的文件名，文件没有文件名，其实是链接的名字。

9、如此不难理解，readdir返回的结构中包含的内容，大致是i-node链接和文件名。stat是通过i-node找到文件信息并填充stat结构返回。

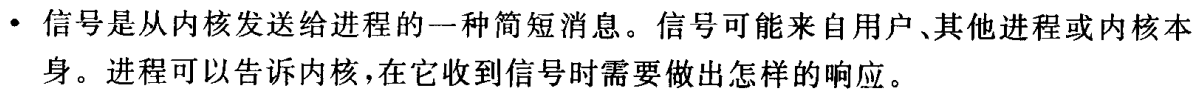
10、设备文件的i结点存储的是指向内核子程序的指针，内核中，传输设备数据的子程序被称为设备驱动程序。

11、内核处理终端数据：

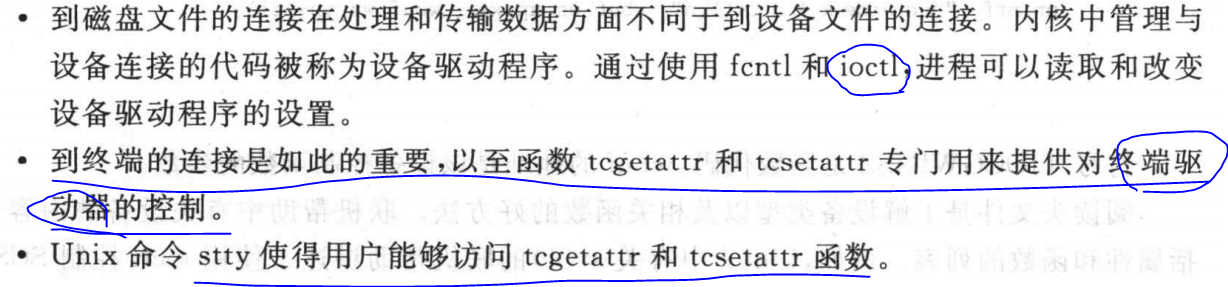




1. 信号：



1. 搞清楚一些细节方面的东西：



上图可以知道，通过ioctl可以对其他任何设备进行编程，而tcgetattr和tcsetattr只是针对终端的。

1. Linux的信号机制：

<https://www.cnblogs.com/hoys/archive/2012/08/19/2646377.html>

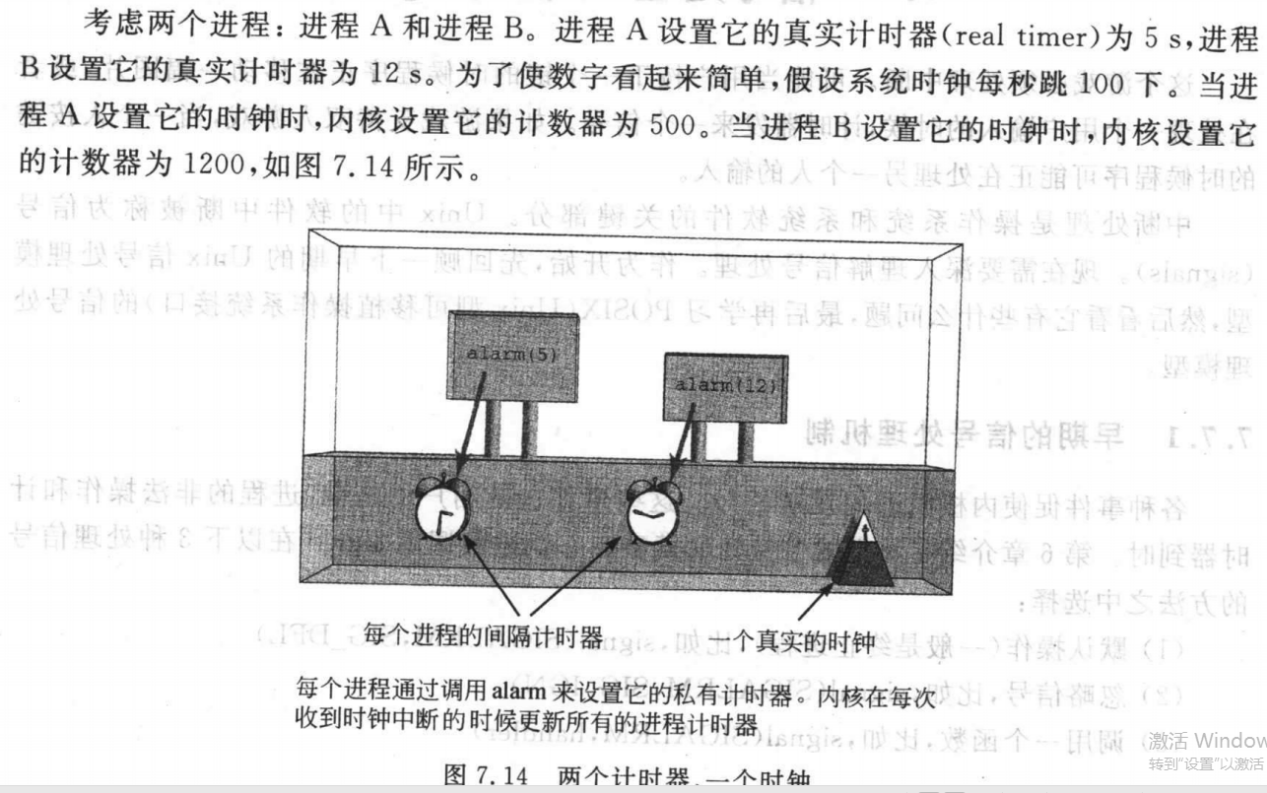
15、编写程序的时候，遇到没有curses.h这个文件的时候如下做法：

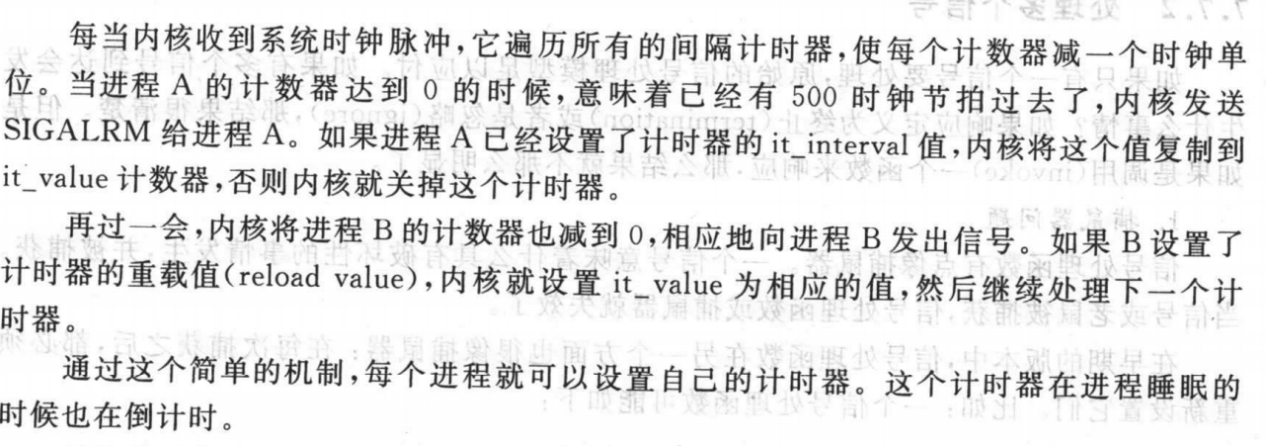
yum install ncurses-libs yum install ncurses-devel

16、sleep函数是调用alarm实现的。

17、计算机有几个时钟？

一个系统只需要一个时钟来设置节拍，一个硬件时钟的脉冲是计算机里唯一需要的时钟，每个进程设置自己的计数时间，操作系统在每过一个时间片后为所有的计数器数值做递减。





1. 测试所用的系统一些特性

1. 调用后还起作用（也就是不是捕鼠器模型）

2. x信号被y信号打断跳到y信号的处理函数执行

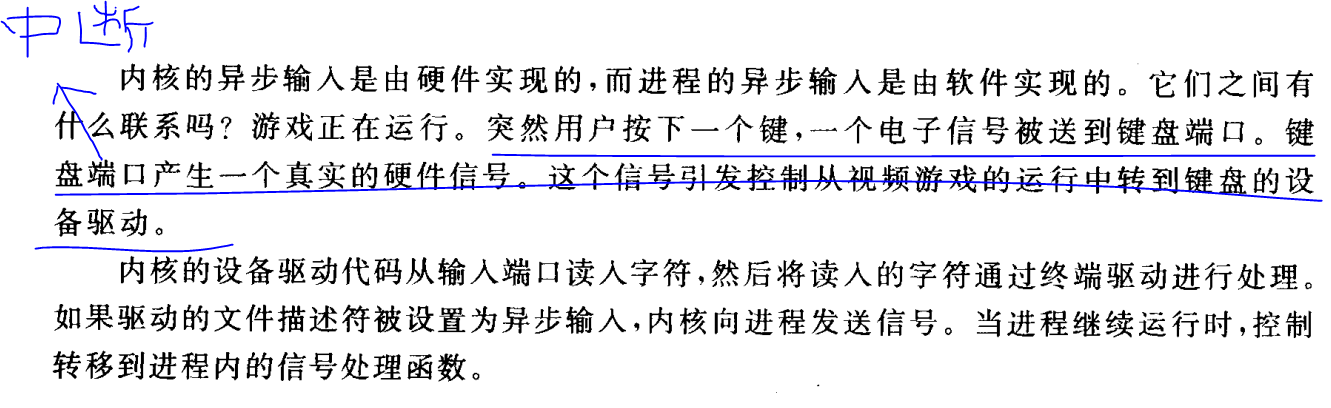
3. 第二个相同的信号过来第二个信号被阻塞直到第一个信号被处理完。（有可能丢失信号）

4. 程序在等待的时候接受信号，这时候会抛弃前面的字符，重新接受。

5. 接受信号进行处理的时候，如果这个时候有输入，则有时候会丢失，也有可能阻塞后接受输入字符。

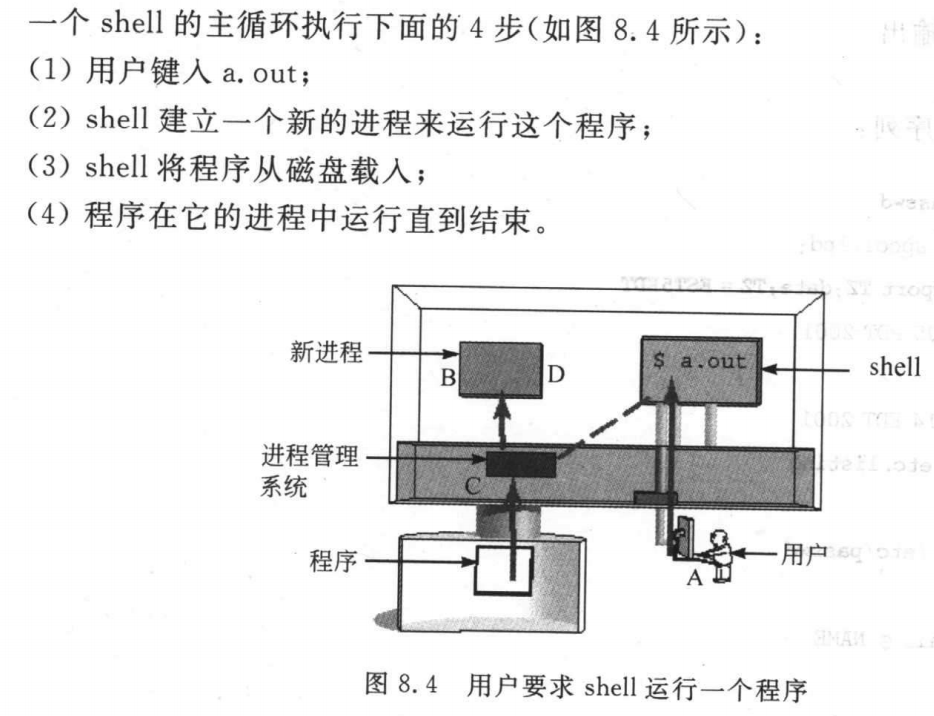
19、使用sio\_read的时候，编译的时候使用-lrt,同时，如果想多次使用aio\_read，要写多次，不可以只写一次。

20、操作系统内核是要异步输入（回顾一下中断的三种方式，其中的第二种），内核的异步输入是由软件实现的



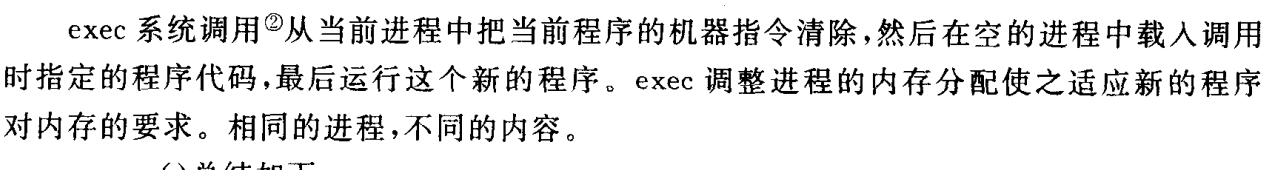
21、一共有五种IO模型，分别是阻塞IO模型，非阻塞IO模型，IO复用模型，信号驱动IO模型，异步IO模型。其中，前面四种是同步IO，只有后面那种才是真正的异步IO。区别于同步与异步的根本区别是数据从内核到用户态进程是否有阻塞，只有异步IO模型才是完全没有阻塞的，即只有他才是真正的异步。

22、按照书本上的说法，shell是如何运行程序的？

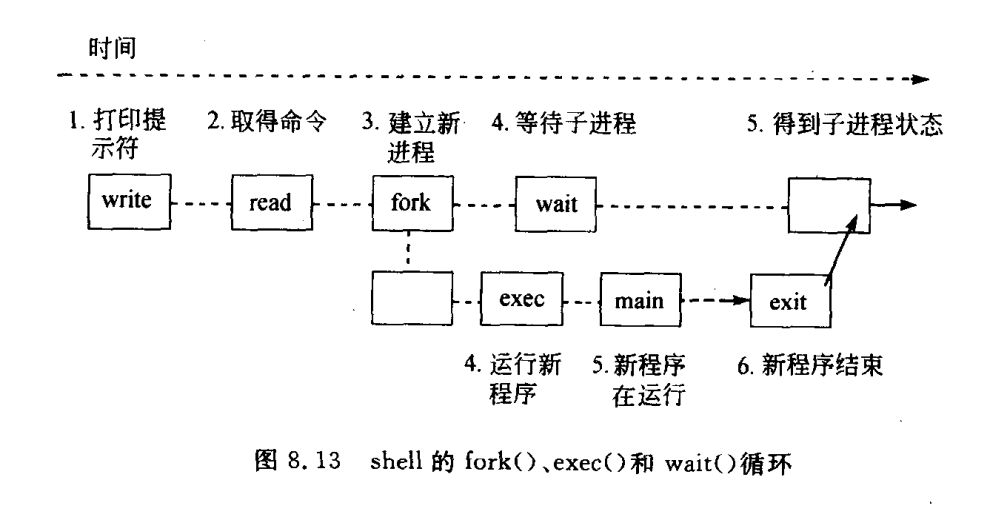


也就是shell建立一个新进程，在那个进程里将程序载入直到运行结束。而execvp则是在当前进程里载入程序代替当前的数据和代码。

1. exec会把当前进程的机器指令清除。



1. unix里面的shell模型如下图：



25、键盘的信号发送给所有连接的进程。所以最初的时候shell模型如果在等待子进程执行的时候键盘按下ctrl-c的时候就会导致shell也会退出。

26、zombie：僵尸进程：根据文档描述如下：

If the parent has not indicated that it is not interested in the exit status, but is not waiting, the exiting process turns into a "zombie" process (which is nothing but a container for the single byte representing the exit status) so that the parent can learn the exit status when it later calls one of the wait(2) functions.

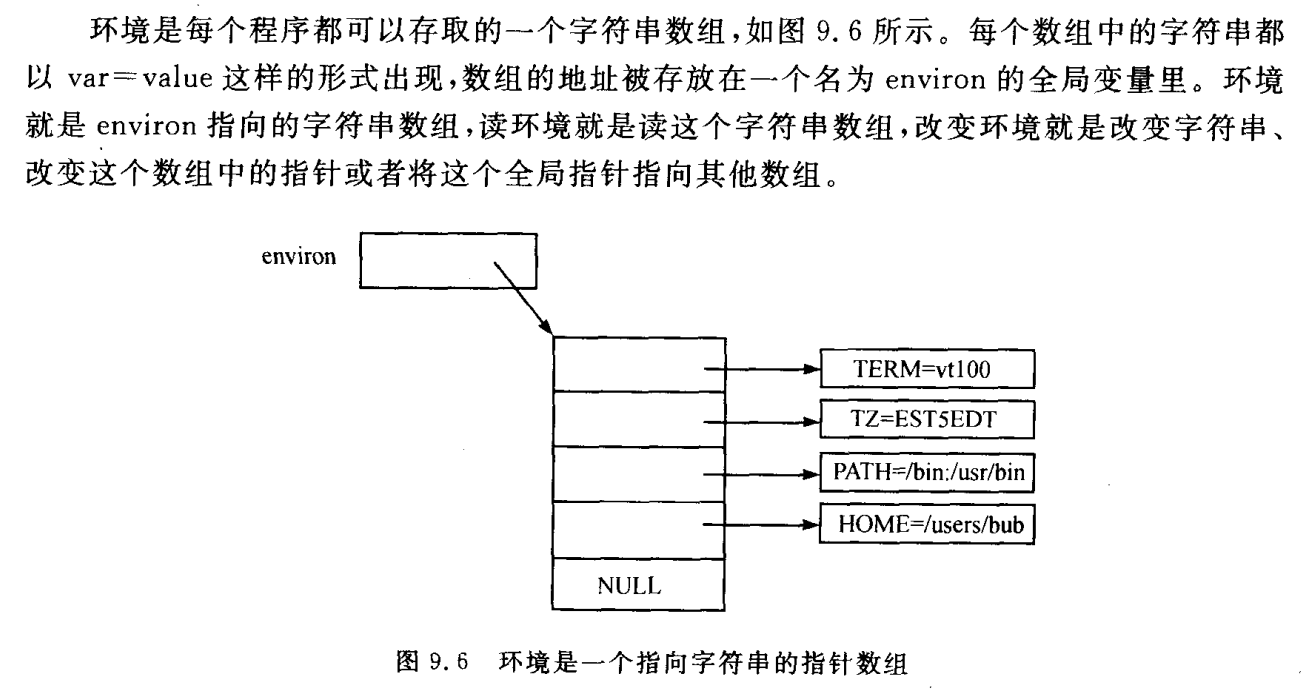
僵尸进程只是包含退出状态，但是仍然占据着进程id，所以过多的僵尸进程会导致无法创建进程。当父进程结束之后，僵尸进程变成孤儿进程，由init进程销毁。

孤儿进程：父进程结束，子进程还没有结束。

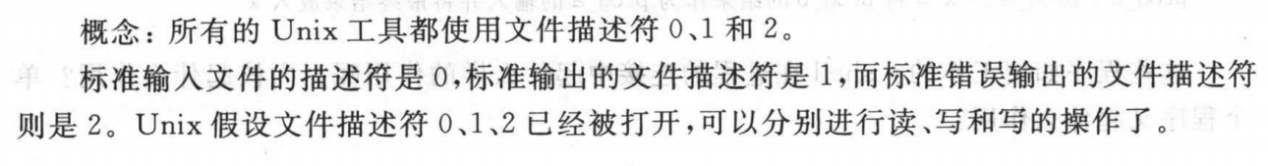
27、unix/Linux里面的程序都遵循以0退出表明成功的惯例。

28、在调用exec相关的函数的时候，environ将指针和指向的数组会复制到新的程序的数据空间。

29、什么是环境？



1. gch文件是预编译头，就是把头文件事先翻译成一种二进制的格式，供后续的编译过程使用。
2. 看图

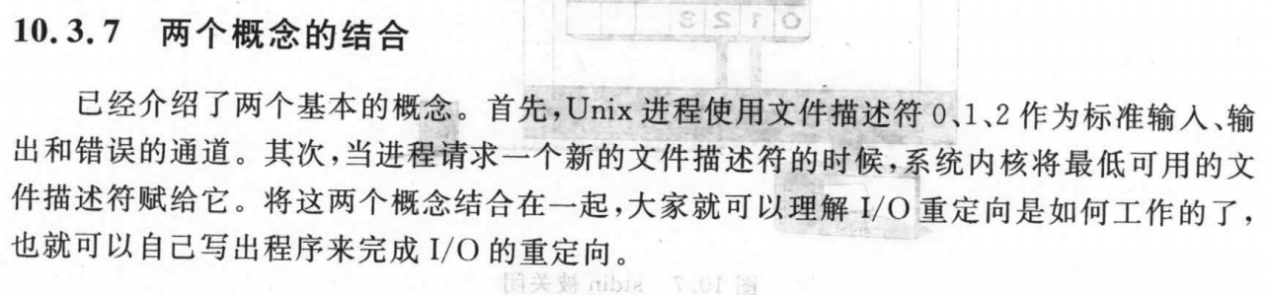


32、重定向IO的是shell而不是程序

33、概念2:最低可用文件描述符原则（Lowest-Available-fd）

什么是文件描述符？它是一个数组的索引号。每个进程都有它打开的一组文件，这些打开的文件被保持在一个数组中，文件描述符即为某文件在此数组的索引。

1. 看图



35、我从上面的归纳总结得到，stdin，stdout，stderr是FILE \*类型，称为标准输入输出流，上述三个与每个进程里的文件描述符数组里的0，1，2相对应，而Linux里的工具使用的都是文件描述符0、1、2，这更验证了我上述所说的情况，即改变文件描述符里面的内容，标准输入输出流就会跟着改变。

36、fork之后，父进程打开的进程描述符都被复制到子进程里面。在子进程里面执行exec，文件描述符也和运行exec之前一样。如果子进程改变了它的进程描述符，则不会影响父进程的。因为打开的文件并非是程序的代码也不是数据，它们属于进程的属性，因此调用exec并不改变它们。

37、管道是没有边界的，当所有的写数据端被关闭的时候，读的时候会返回0，这个时候可以判断为结束。此外，如果没有数据，read端是会阻塞的，当然可以设置不阻塞。多个进程对数据read的时候要注意数据不完整性。管道是进程间进行单向传输的。

38、这里提到了协同进程，这里的说法是这样的，不同于shell这种用户每输入一个命令就创建一个新的进程，协同进程两个程序都持续运行，当其中一个程序完成工作后，把控制权传给另一个程序。

39、在文件/etc/services中定义了众所周知的服务端口列表。

40、host byte order(主机字节序) 和 network byte order（网络字节序）的相关概念。

网络字节序是大端字节序，这是标准化的。然而，我们每个主机有可能是小端，也有可能是大端。主机字节序就是自己主机内部数据在内存的存储方式，可以测试看看是大端还是小端。

41、fgets读取结束是EOF或者新的一行（也就是\n标志），同时read等待输入的时候如无输入就会阻塞。

42、很多服务器程序以d结尾，这个代表了：daemon，精灵，守护进程的意思。查阅一下守护进程。（看了一下，守护进程比较难懂）终于搞懂了，什么是守护进程?守护进程的特点是独立于控制终端，不能直接与用户交互，不受用户登陆注销的影响，在后台一直运行着。守护进程的关键是setsid函数，这个函数产生一个新的会话期，并且是会话期里的进程组组长，且没有控制终端。

43、到目前为止，接触了socket编程。domain包括：unix域内编程，tcp协议，udp协议。在此总结一下：

1. unix是创建一个socket文件进行通信的，这个可以查看bind里面的example

2. tcp协议的服务端需要socket，bind，listen，accept，write/read，close

3. udp协议的只需要socket,bind，然后使用recvfrom，sendto等函数，甚至客户端不需要connect也可以。

4. 其实总结起来，对于udp，tcp而言，都是要绑定一个地址，然后获取发送数据的地址以此进行通信。

5. 一些重要的地址结构需要留意一下，比如struct sockaddr\_storage,不过这个我找不到它声明的地方

44、当子进程退出或者被终止的时候，内核会发送SIGCHLD给父进程。

45、在这里总结一下。关于EINTR错误的情况。在一些可能阻塞的系统调用里头，当处于阻塞的情况下，捕获某个信号且处理函数返回的时候，这个系统调用被中断，且errno被设置为EINTR。（引入<errno.h>方可看到errno）。一开始，我做实验，发现accept没有被中断，仔细查找得知，原来我使用的是signal方法，这个方法在健壮性方面不如sigaction，signal方法隐含了设置restart标志，所以accept会再次重启。所以，当我改成sigaction的时候，就会出现EINTR的错误。

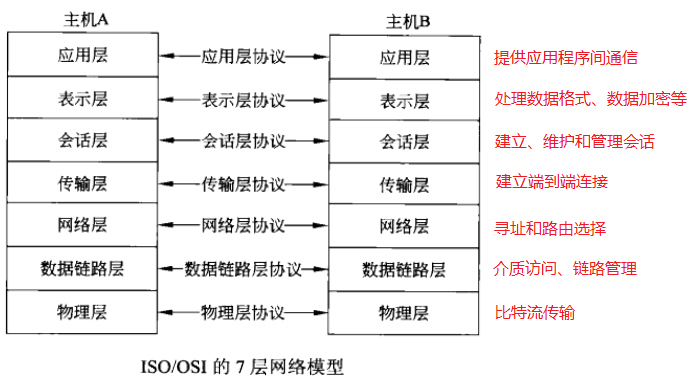
另外，一般重启的方法是：判断errno然后手动重启（即再次调用相同的方法）。但是，对于一些比如connect不能循环处理，要使用select或poll，不过我没有试过。另外，上述的实验给了我一个提示，设置信号restart属性，不过这个只能针对一部分函数。

46、这里有个地方跟书本上的有出入。书本上说多次信号来会导致阻塞，会导致信号丢失。不过我做了实验，发现相同的信号多次来临也不会丢失。

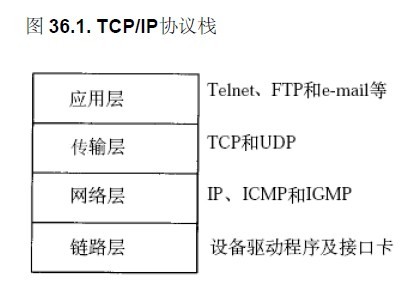
47、stat函数传递的参数，可以是先定义指针，也可以是先定义结构体stat，然后再传递参数。神奇。

48、一些关于tcp/ip基础的总结与归纳

1.OSI，开放系统互联模型，共七层，



1. tcp/ip协议栈分为四层，分别是如下：应用层，运输层，网络层，链路层

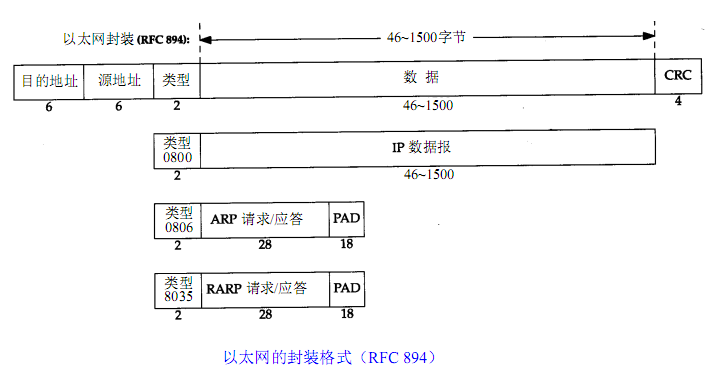


3. 传输层及其以下的机制由内核提供，应用层由用户进程提供，应用程序对通讯数据的含义进行解释，而传输层及其以下处理通讯的细节，将数据从一台计算机通过一定的路径发送到另一台计算机。

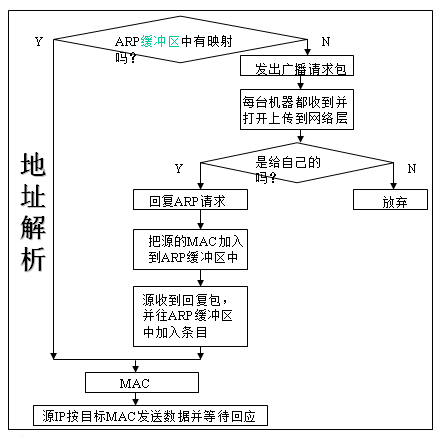
4. 不同的协议层对数据包有不同的称谓，在传输层叫做段（segment），在网络层叫做数据包（packet），在链路层叫做帧（frame）。

5. 在链路层，对数据帧的长度都有限制，将这个限制称作最大传输单元（MTU，Maximum Transmission Unit）。如果IP层有一个数据报要传，而且数据的长度比链路层的MTU还大，那么IP层就要进行分片（Fragmentation），把数据报分成若干片，这样每一片都小于MTU。

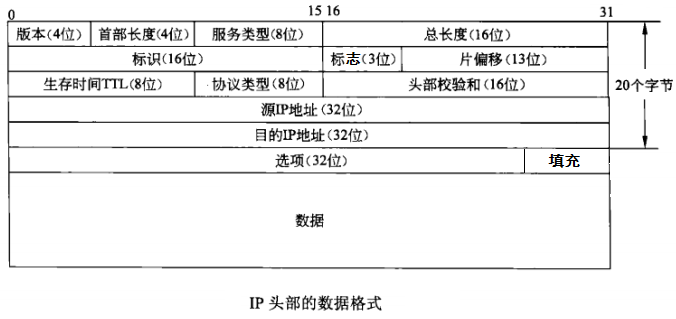
6. 以太网的帧格式如下，



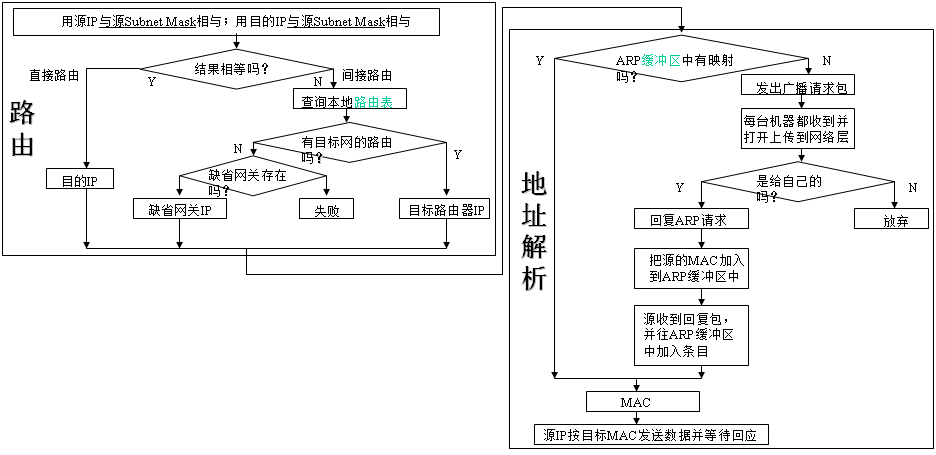
1. ARP地址解析协议，简单来说，通过已知的ip进行广播，获取MAC地址。

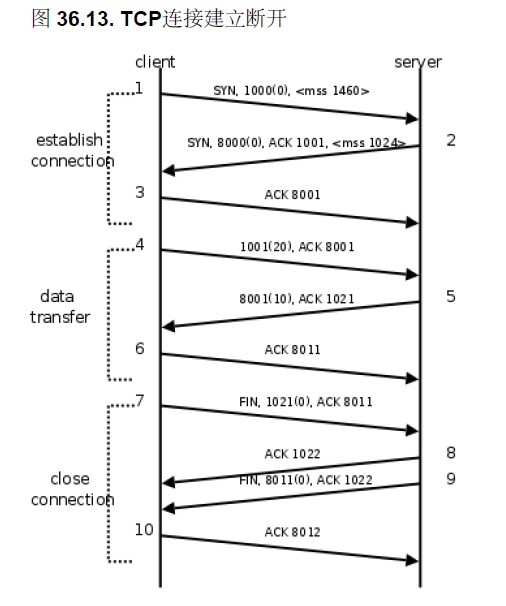
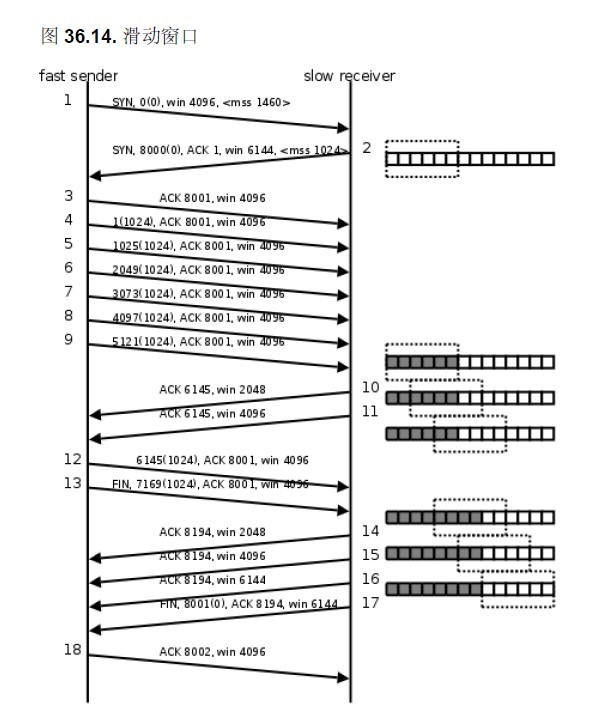


1. ip头部数据格式，----------



1. 这里要注意分片的问题。每一片都有自己的IP 头部，IP头部中的标识是一样的，但是片偏移不同（以8字节为单位）。除了最后一片，分片要求其他片除去IP头部的大小必须是8字节的整数倍。除了第一片有tcp/udp头部，其他片都没有。分片完成后，每一片独自成为一个数据包（跟数据报概念不同，参见这里），可以走不同的路由，最后到达目的地的时候IP层根据它们各自IP头部的信息重新组成一个IP数据报。分片是有风险的，因为一旦某一片丢失，就需要重传这个IP数据报，因为IP层本身并没有超时重传的机制，可靠性需要TCP层来保证（一些UDP协议的可靠性由应用程序保证），一旦一个TCP段中的某一片丢失，TCP协议层会超时重传。此外，分片可以发生在源主机或者中间的路由，如果发生在中间的路由，源主机根本不知道是怎样分片，所以要尽量避免分片。
2. 还有一点需要注意的是，路由的处理过程----



1. 根据我的理解是，先比较原IP地址和目的IP地址是否在同一个网络上，如果不，则查询本地的路由表，本地路由表有目标网的路由，则跳转到那个网络，否则走默认网关。
2. 
3. 
4. tcp协议最重要的两点是三次握手&四次挥手和滑动窗口（为了解决发送端发送数据过快，接收端处理数据过慢的情形）。此外，从这个例子还可以看出，发送端是一K一K地发送数据，而接收端的应用程序可以两K两K地提走数据，当然也有可能一次提走3K或6K数据，或者一次只提走几个字节的数据，也就是说，应用程序所看到的数据是一个整体，或说是一个流（stream），一条消息有多少字节对应用程序是不可见的，因此TCP协议是面向流的协议，这也是容易出现粘包问题的原因。而UDP是面向消息的协议，每个UDP段都是一条消息，应用程序必须以消息为单位提取数据，不能一次提取任意字节的数据，这一点和TCP是很不同的。这个从编程的内容也可以看出。
5. ip头部总长度字段表示是首部和数据报的长度和，UDP头部总长度也是如此。

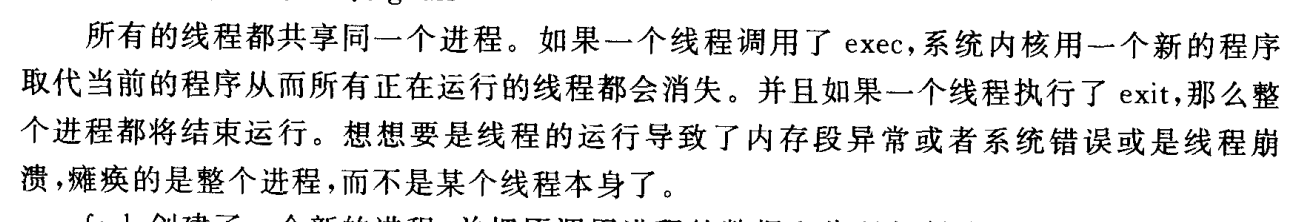
49、流与数据报的区别：本质区别就是传输层协议里的TCP协议与IP协议的区别。

50、可以通过发送编号为0的信号确定某个进程是否存在。“signal 0″ is kind of like a moral equivalent of “ping”.Using “kill -0 NNN” in a shell script is a good way to tell if PID “NNN” is alive or not:signal 0 is just used to check process is exists or not.如果进程不存在，内核将不会发送信号，而是返回错误并设置errno为ESRCH。

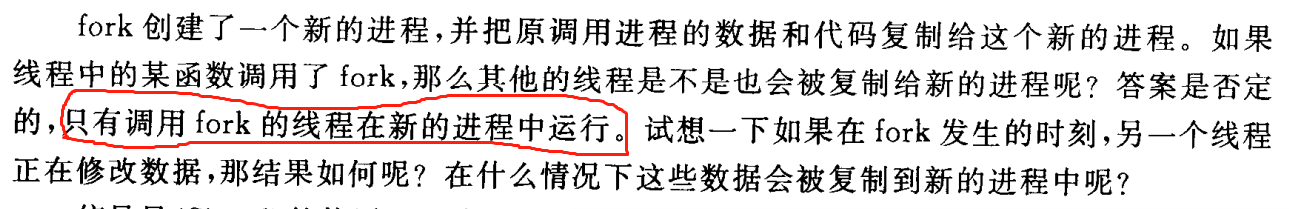
51、unix域的socket编程里面，通过文件名进行进程间通信的话，在服务端想要得到客户端的“地址”，就要在客户端bind绑定一个新的文件名，否则，服务端accept获取到的是NULL（相对于流传输而言）。

52、进程和线程。（这个必须要好好总结一下）把进程和线程的概念和它们的区别讲清楚。

先说一下，进程这个概念是明显的，清晰的，这与程序在内存里运行紧密相关。线程则比较难以解释。但是，我们回归本质，从CPU的角度看待这个问题。由于CPU的速度太快，于是，它会轮流执行每一个进程。当这个进程结束或者时间片到的时候，它就会被切换出去，此时，进程自身要保存自己的信息，以便下次轮到CPU的时候恢复之前的上下文环境。即：先加载程序A的上下文，然后开始执行A，保存程序A的上下文，调入下一个要执行的程序B的程序上下文，然后开始执行B,保存程序B的上下文。。。。对于CPU而言，进程的存活时间为CPU加载上下文+CPU执行+CPU保存上下文直到这个进程结束。那线程在CPU看来是怎么样的呢？我们知道，每一个进程的程序都有许多程序段组成，当进程A得到CPU宠幸的时候，进程A的上下文不变的时候，分享了CPU的时间段，类似于进程。也就是说，对于CPU而言，都是CPU时间段的描述，只不过颗粒大小不一样而已。



1. 看图



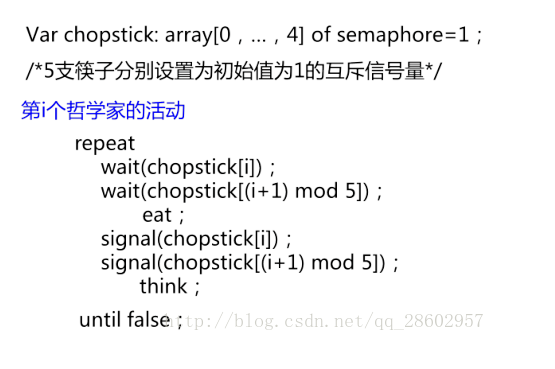
这个可以做个实验。另外，pthread\_join函数只需要执行一次，即可回收线程。如果在线程里fork之后一个进程里sleep多秒后，你会发现，pthread\_join函数会一直等待，直到另外一个进程也结束随着线程一起join入主线程。

54、信号量。

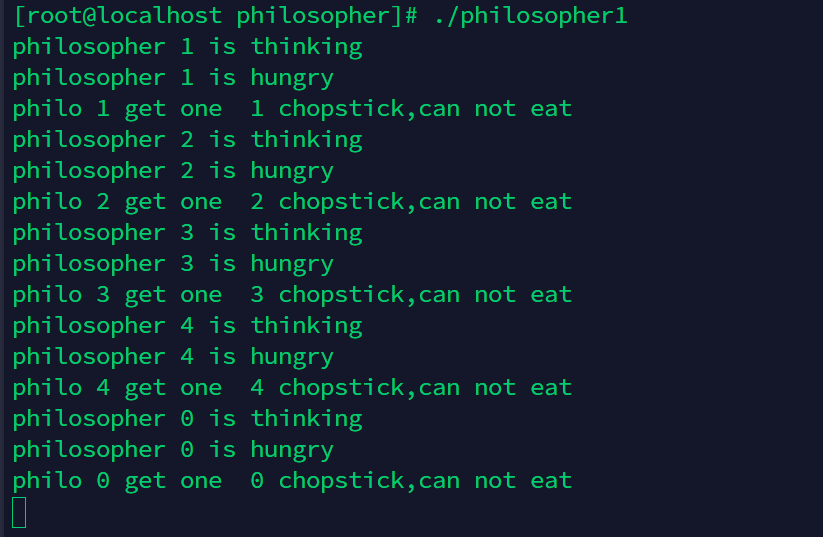
55、哲学家进餐问题。

哲学家进餐问题：如果五位哲学家同时饥饿各自拿起左边的筷子，那么，五个代表筷子的信号量为0，当他们试图拿起右边的筷子的时候，就会出现死锁的问题。为了重现死锁，我在每一个线程获取第一根筷子的时候，睡眠三秒，就会出现死锁。

伪代码：



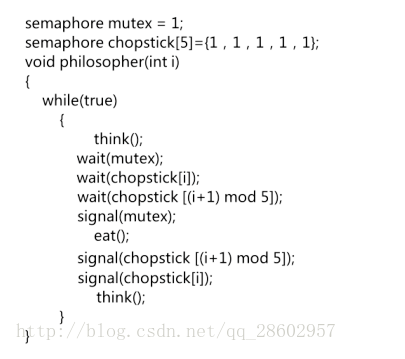
可能的结果：

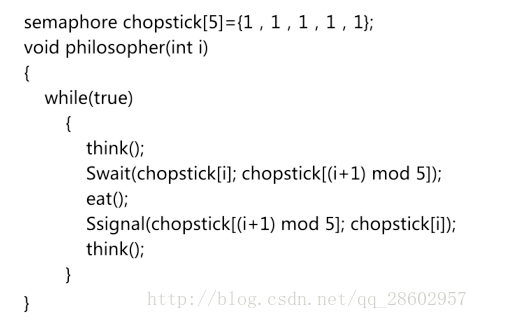


解决的方法：

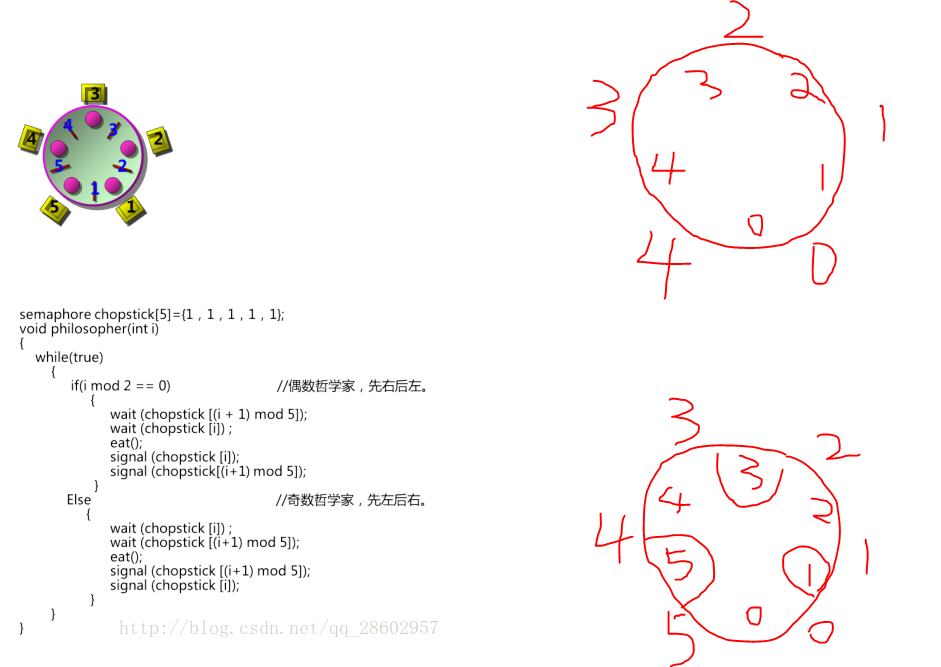
1、仅当哲学家的左右手筷子都拿起时才允许进餐。这个实现的思路，看网上有两种，第一种是在获取左右筷子之前使用信号量进行保护。但是，这个方法有一个问题，就是效率低下，一些互不相关的哲学家需要拿起筷子的时候也被阻塞。第二种是利用 AND 型信号量机制实现。但是，我暂时找不到相关的AND信号量。

相关伪代码：

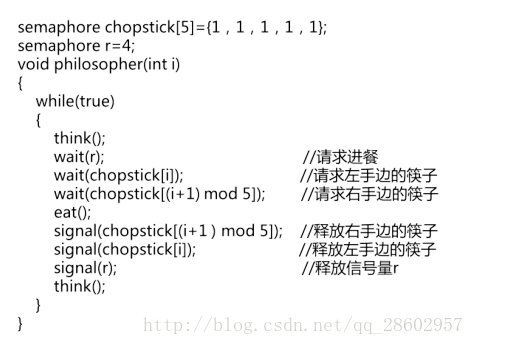




2.规定奇数号哲学家先拿左筷子再拿右筷子，而偶数号哲学家相反。一开始我以为这个答案只适合奇数的科学家，后来仔细想想，偶数的哲学家同样适合。仔细想想，这个方法为什么不会导致死锁呢？我们反过来推导，什么情况下会导致死锁？当每个人手握一个筷子，资源形成一个环的时候，就会形成死锁。现在因为奇偶的规定，所以导致N/2个人需要等待，N/2个人获得锁，这种情况下，资源是不会形成闭合状态。

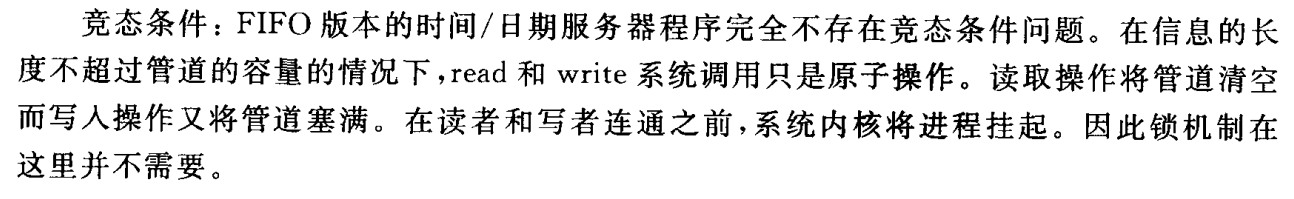


1. 至多只允许四位哲学家同时去拿左筷子，最终能保证至少有一位哲学家能进餐，并在用完后释放两只筷子供他人使用。这种方法能够让资源得到更有效的利用。



以上。

1. 对于命名管道而言，在信息长度不超过管道容量的情况下（PEPE\_BUF），不存在竞争问题。这是因为，如果另一端没有准备好，就会阻塞，在管道容量的情况下，是一个原子操作，不会发生诸如数据重叠，读不到的情况。

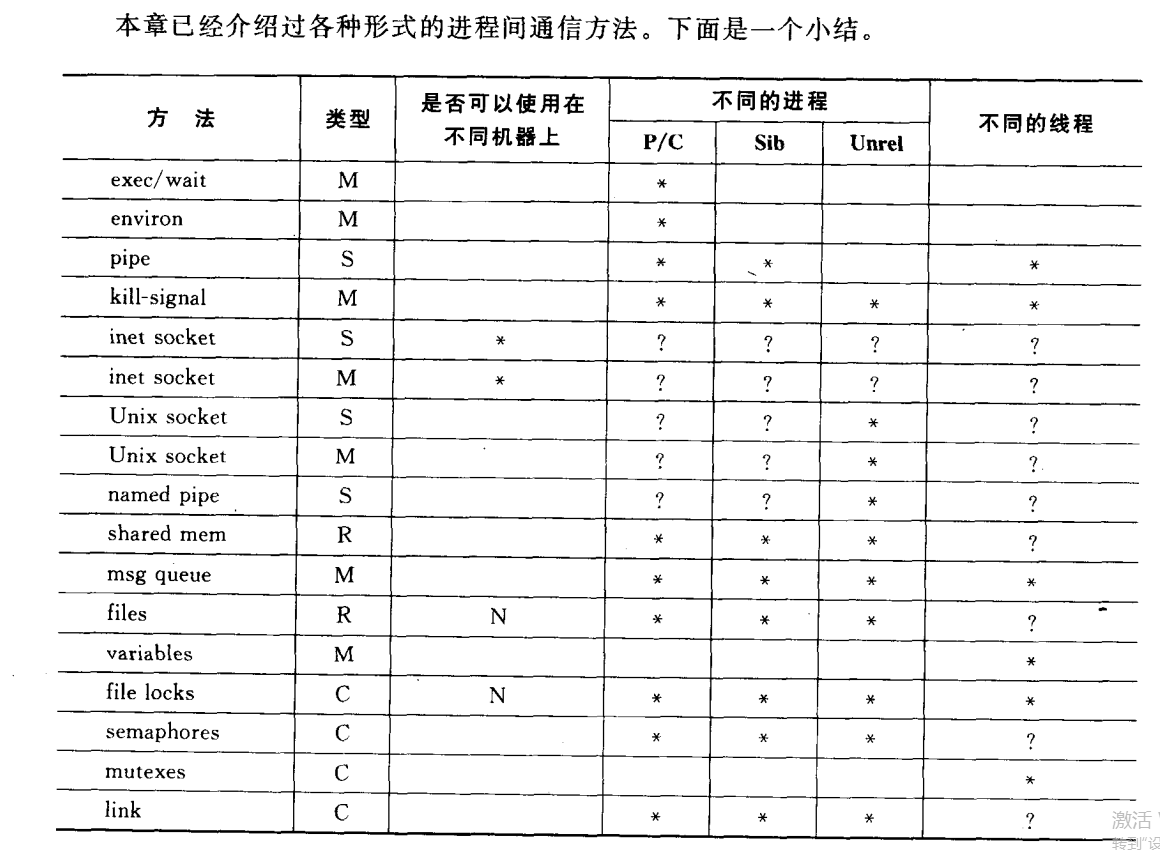


57、对于文件锁，针对应用出现了读锁和写锁。此外，还有劝告锁和强制锁。一开始搞不清这些锁类型的关系，但是，查看了资料得知，劝告锁和强制锁都可以使用读锁和写锁，系统默认是劝告锁，对于强制锁而言，需要使用mount -o mand还要设置组权限。这两种的区别是，对于劝告锁，如果一个另外的进程不加锁往文件写的话，也是可以写进去的。强制锁则不会。

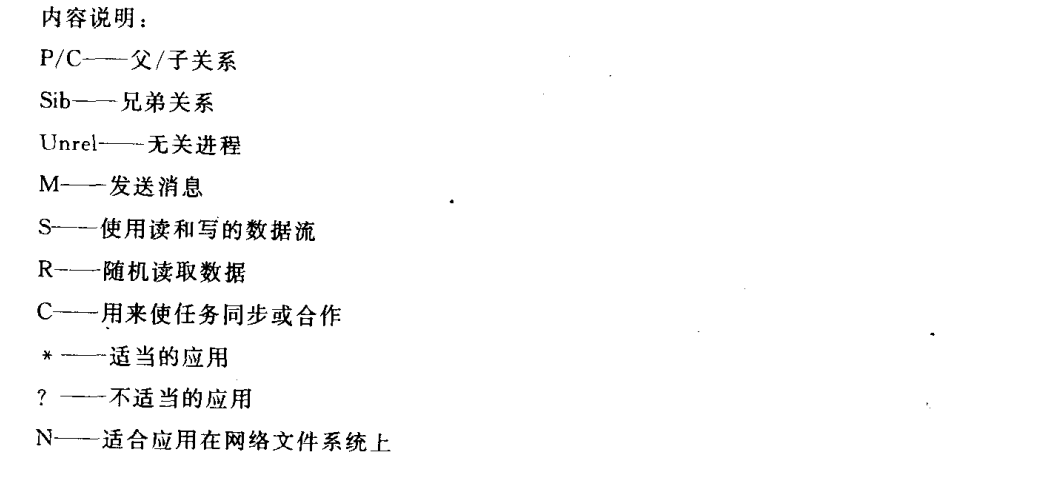
58、对于信号量而言，semget函数可以创建一定个数的信号量集合。但是，对于semop函数而言，第二第三个参数表明参加对信号量集完成的一组操作。它可以是一组信号量里面的一个出现多次，而这些操作是当作一个原子性的操作来完成的。

59、消息队列的时候，要注意如果存入的内容不能超过消息队列大小，否则会有阻塞的情况。（验证类型的时候出现）。

60、进程间通信的方式



其中，表格里面的含义是：



61、终于看完这本书了，说一下感受。先讲一下心情，看完了倒没有多激动，曾经有一段时间甚至无法静下心来看书，只能说外界影响的事情是比较多的，方方面面的烦恼。这么久以来，我认为看书最好的状态是内心平静，毫无波澜，休息充足。太亢奋不行，容易一带而过，太累了容易看不进。还有一种反面情况，心不在焉，内心安静不下来。希望以后有很多很多内心平和的时间去看书，去累积知识，希望以后烦恼少一点。作罢。谈一下这本书的收获吧。书籍非常不错，尤其是入门而言，由浅入深，循序渐进，通过一些需求或者例子引出相关函数。再由相关问题和例子的不完善程度引出下一章的知识点。每一章最后有一个总结，方便对这一章总结。而且看了一下，书籍是十多年前（2004年出版）的，这么多年来，相关的系统调用都没有很大的改变，完全不影响代码的书写，这一点非常美好。不得不说，这本书是被埋没的好书，我已经搜索不到正版书籍了，只能淘宝买个影印版，非常遗憾了。未来的计划是先回顾一下这本书的相关代码，然后继续深入学习Unix/Linux相关知识，继而回去研究PHP的扩展。希望接下来一切顺利。写于2019年12月25日。陈伟辉写。