800页 我真的想去si

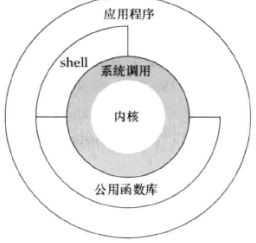
不多说了，我去看书了==。==

UNIX环境高级编程

第一章：UNIX基础知识

引言：所有操作系统都为它们所运行的程序提供服务，典型的包括执行新程序，获得时间，分配存储，打开文件等

UNIX体系结构：操作系统定义为一种控制计算机硬件资源，提供程序运行环境的软件，通常将这种软件称为内核。内核的接口被称作系统调用，公用函数库构建在系统调用接口上，应用程序即可使用公用函数库，也可使用系统调用，shell是一个特殊的应用程序，为运行其他应用程序提供了一个接口。广义上的操作系统包括内核和一些使得计算机能够发挥作用，并使计算机具有自己的特性，其他软件包括系统实用程序、应用程序、shell、公共函数库等。



登录：登录名，shell（有许多不同的种类），所有的linux系统都使用GNU shell

文件系统：为层次结构，起点为根（root “/”）,

文件名：字母，数字，句点，短横线，下划线，创建新目录时会自动创建两个新文件名. ..指向当前目录和父目录

路径名：斜线分隔的一个或多个文件名组成的序列构成路径，以斜线开头的路径名称为绝对路径名，否则为相对路径名

工作目录：每个进程都有工作目录

起始目录：登录时，工作目录设为起始目录

文件描述符：通常是一个小的非负整数，内核用来标识一个特定进程正在访问的文件，当内核打开一个现有文件或创建一个新文件，都返回这个，读写文件可以使用这个

标准输入、输出、错误：如果不做特殊处理，这三个描述符都链接向终端，大多数shell都提供一种办法让描述符重新定位到文件，如ls > file.list

不带缓冲的I.O：open、read、write、lseek以及close、

标准IO：

程序：存储在磁盘上某个目录中的可执行文件。内核使用exec函数将程序入读内存执行程序

进程：程序的执行实例

进程ID：一个非负整数，进程的唯一标识符

进程控制：fork exec waitpid

线程：通常一个进程只有一个控制线程，某一时刻执行的一组机器指令。一个进程内的所有线程共享同一地址空间、文件描述符、栈以及进程相关的属性。因为它们能访问同一存储区，所以要各线程访问共享数据时需要采取同步措施避免不一致性

线程ID：只在它所属的进程内起作用。

用户ID：一个数值向系统标识各个不同的用户。系统管理员在确定一个用户的登录名的同时，确定其用户ID，用户不能更改用户ID

组ID：口令文件登录项包括用户的组ID，它是一个数值。

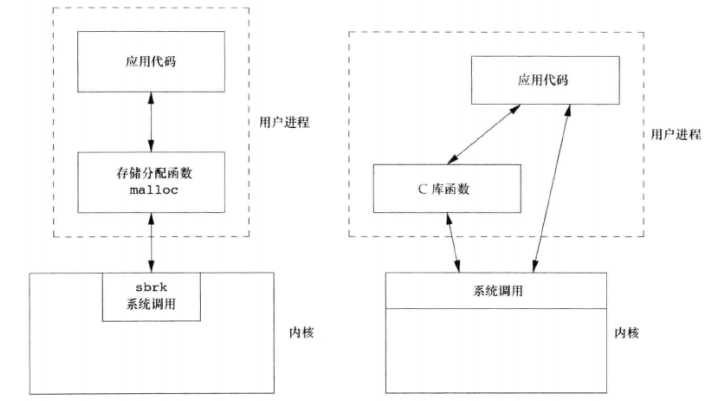
附属组ID：除了在口令文件中对一个登录名指定一个组ID外，大多数UNIX系统版本还允许一个用户属于另外一些组

信号：信号通知进程发生了某种情况。例如若某一进程执行除法操作，除数为0，则将名为SIGFPE的信号发送给该进程。进程有一下三种处理信号的方式1、忽略哦2、按照默认方式处理3、提供一个函数信号发生时调用这个函数。

UNIX使用过两种不同的时间值：1、日历时间2、进程时间。 度量进程执行时间有三个，时钟时间，用户CPU时间，系统CPU时间

系统调用：指良好定义，数量有限，直接进入内核的入口点

库函数：UNIX使用的技术是为每个系统调用在标准C库中设置一个具有同样名字的函数。



系统调用通常提供一种最小接口，而库函数通常提供比较复杂的功能。

第二章：UNIX标准及实现（回头看或许更好吧）

UNIX版本很多，因此制定了标准：ISO C标准 IEEE POSIX标准 Single UNIX Specification标准 FIPS标准

系统实现：SVR4 4.4BSD FreeBSD Linux Mac OS X Solaris

标准和实现的关系：FreeBSD Linux Mac OS X Solaris都提供UNIX编程环境。

限制：IOS C定义的限制都在头文件<limits.h>中 如整型的最大到多少，可同时打开标准I/O流的最小个数

POSIX限制定义了数值限制，最大最小值，运行时可增加的值，运行时不变值，其他不变值，路径名可变值

三个函数 sysconf pathconf fpathconf 来获取UNIX系统的各种预设的限制

限制分为三种 （1）编译时限制（头文件）（2）与文件或目录无关的运行时限制（sysconf函数）（3）与文件或目录有关的运行时限制（pathconf和fpathconf函数）

选项：和限制一样有三种看选项的方法（1）编译时选项定义在<unistd.h>中（2）与文件或目录无关的运行时选项用sysconf函数来判断（3）与文件或目录有关的运行时选项通过调用pathconf或fpathconf函数来判断

下面这部分讲的是unix的文件系统，挺深入的，重点不在函数细节，在于设计的思想和方法

第三章：文件I/O

终于开始讨论UNIX系统了，=。=

可用的文件I/O函数，包括打开文件、读文件、写文件。用到五个函数open、read、write、lseek以及close

不同缓冲长度对read和write有影响

本章的函数被称为不带缓冲的I/O，术语不带缓冲指每个read和write都调用内核中的一个系统调用。

涉及多个进程间共享资源，原子操作的概念就非常重要

文件描述符：

对于内核所有打开文件都通过文件描述符引用，当打开一个文件或创建新文件，内核向进程返回一个文件描述符，读写文件，用返回的文件描述符作为参数给read和write

文件描述符 0 1 2分别与标准输出，标准输入，标准错误关联。长替换为复航常量STDIN\_FILENO,STDOUT\_FILENO和STDERR\_FILENO

文件描述符变化范围为0~OPEN\_MAX-1

函数open和openat：可打开或创建一个文件

openat让线程可以使用相对路径名打开目录中的文件，而不再只能打开当前工作目录，避免TOCTTOU错误，

creat可以创建一个新文件，缺点是以致谢方式打开创建的文件，

close关闭一个打开文件。加上文件描述符

lseek显式的为一个打开文件设置偏移量

read从打开文件中读取数据

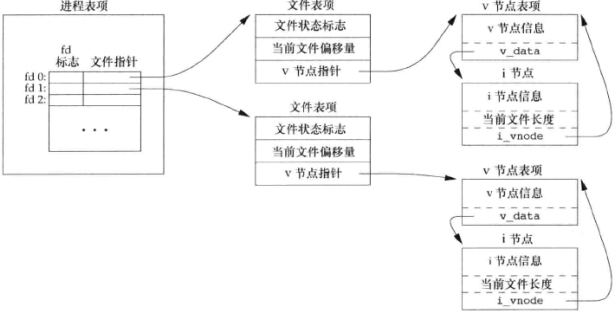
write向打开文件写数据

IO效率，预读

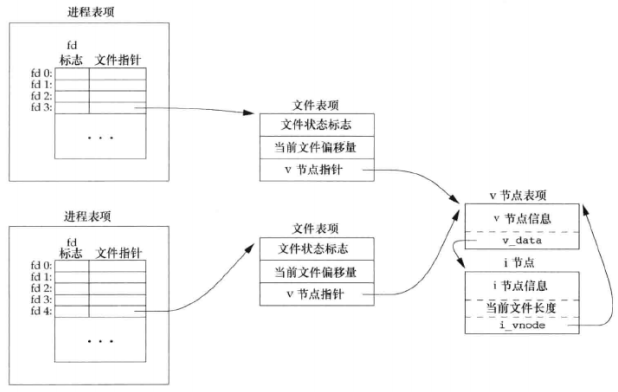
文件共享：

内核使用三种数据数据结构表示打开文件

如下图：



当两个独立进程各自打开了同一文件，



多个进程读同一文件每个进程有它自己的文件表项，其中也有他自己当前的文件偏移量。但是写同一文件，就可能产生预想不到的效果。

原子操作：追加到一个文件，原子性地定位并执行I/O，

（太细了，这本书真的太细了，关于函数部分内容不要太纠结了，太细节了，还是要宏观把握知识，看完这本书至少要知道unix这个操作系统的大致外貌，关于操作系统的四大组件他是怎么处理的。）

一般而言，原子操作指的是由多步组成的一个操作。如果该操作原子地执行，要么执行完所有步骤，要么一步也不执行，不可能只执行所有步骤的子集

ioctl函数：终端I/O使用的最多

第四章：文件和目录

引言：本章描述文件系统的其他特征和文件的性质。

Stat函数返回命名文件有关的信息结构。Fstat获得已在描述符fd上打开文件的有关信息。Lstat类似stat，但当命名的文件是一个符号链接时，lstat返回该符号链接的有关信息，而不是由该符号链接引用的文件的信息

Fstatat函数为一个相对于当前打开目录的路径名返回文件统计信息。

文件类型：普通文件和目录

普通文件：最常用的，包含了某种形式的数据，至于是文本还是二进制，对于UNIX内核并无区别。

目录文件：

块特殊文件：

字符特殊文件：

FIFO：用于进程间通信，有时也称为命名管道

套接字：用于进程间的网络通信

符号链接：这种类型的文件指向另一个文件

文件类型信息包含在stat结构的st\_mode成员中。

设置用户ID和设置组ID，一个进程相关联的ID有6个或更多，文件也有所属的用户ID，组ID，有的有权限去，有的没有权限

文件访问权限有九种

新文件和目录的所有权，新文件的用户ID设置为进程的有效用户ID。组ID可以是进程的有效组ID，所在目录的组ID

函数chmod，fchmod，fchmodat这三个函数可以更改现有文件的访问权限

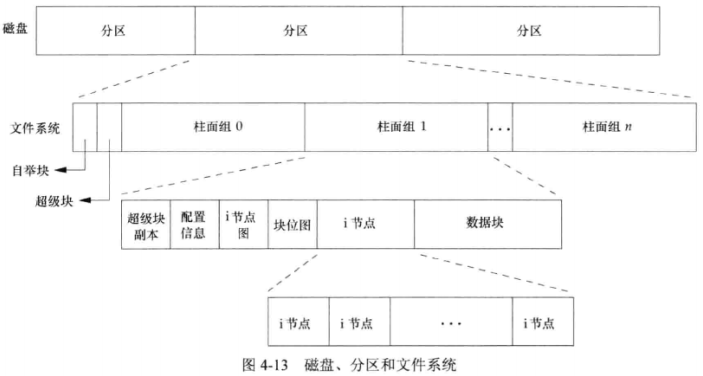
chown fchown fchownat 和 lchown可用于更改文件的用户ID和组ID。

文件长度stat结构成员st\_size表示以字节为单位的文件的长度

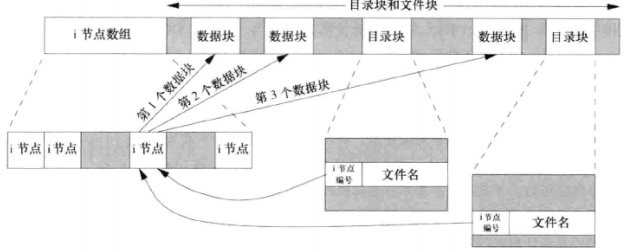
文件空洞：偏移量超过尾端并写入了数据

文件截断：文件尾端截去数据缩短文件

文件系统基本结构：正在使用的文件系统有多种实现，传统的基于BSD的UNIX文件系统（称为UFS），本节讨论这个，UFS以Berkeley快速文件系统为基础，本文讨论该文件系统



一个磁盘每个分区可以包含一个文件系统，i节点和数据块放大后



对于文件，有指向相同节点的硬链接，当链接计数减少到0时，才可以删除这个文件。

LINK\_MAX制定一个文件链接数的最大值

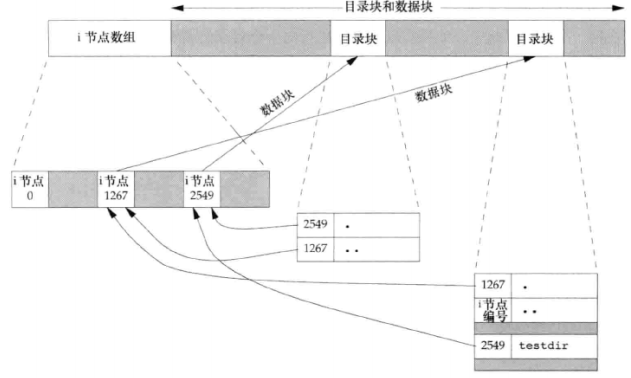
另外一种链接类型称为符号链接，实际内容包含该符号链接指向的文件的名字。软链接

i节点包含了文件类型，访问权限位，长度，指向文件数据块的指针等

目录项中的i节点编号指向同一文件系统中的相应i节点，一个目录项不能指向另一个文件系统的i节点

不更换文件系统情况下为一个文件重命名，该文件的实际内容并未移动，秩序构造一个指向现有i节点的新目录项

新建一个目录，那么文件系统中会发生的变化



创建指向现有文件的链接的方法：link和linkat函数，删除现有目录项，可以用unlink函数

文件或目录重命名可以用rename函数或者renameat函数

符号链接是对一个文件的间接指针，硬链接直接指向文件的i节点。对符号链接以及它指向何种对象并无任何系统限制，任何用户都可以创建指向目录的符号链接，符号链接一般用于将一个文件或目录结构移动到系统中的另一位置

创建符号链接symlink 或symlinkat

读取符号链接readlink和readlinkat

文件时间：修改时间，状态更改时间区别

更改文件的访问和修改时间futimens和utimensat

函数mkdir mkdirat创建目录，. ..自动创建 rmdir删除空目录

读目录：一系列的内容读目录

函数chdir fchdir可以改变当前工作目录， getcwd可以获得当前的完整工作目录，每个进程都有一个当前工作目录，

设备特殊文件，st\_dev和st\_rdev这两个字段经常引起混淆，

第五章 标准IO库

标准IO库围绕流进行，

流定向决定读写字符是单字节还是多字节的

freopen清楚一个流的定向，fwide设置一个流的定向

每个进程预定义三个流 标准输入 输出 错误

缓冲：标准IO 提供缓冲为了尽可能减少read和write调用的次数

三种 全缓冲 行缓冲 不带缓冲

默认下错误不带缓冲，指向终端设备的流是行缓冲，否则是全缓冲

打开流有三个函数 fopen freopen fdopen 返回文件指针 参数路径，不要文件描述符

fclose关闭一个打开的流

读和写流：三种 每次一个字符 一行 直接IO

一个字符 getc fgetc getchar

出错和到尾端分别用 ferror和feof来判断

ugetc可以把字符以相反的顺序压送回流中

输出 putc fputc putchar

每次一行IO

fgets gets

fputs puts

二进制IO（直接IO）

fread fwrite 适用于读写对象，可以设定一次读写的字节数用sizeof

实现细节是标准IO都要借助第三章的IO函数

tmpnam和tmpfile提供了两个函数帮助创建临时文件

内存流：fmemopen允许调用者提供缓冲区用于内存流

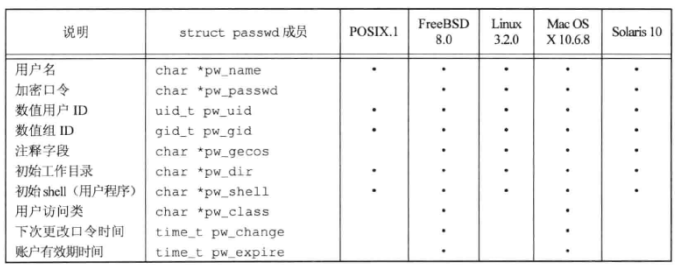
标准io有替代软件

第六章：系统数据文件和信息

UNIX正常运作需要大量与系统有关的数据文件，例如，口令文件和组文件

原本都是ASCII文本文件，使用标准IO库读这些文件，对于较大的系统，非ASCII格式存放这些文件，但仍向使用其他文件格式的应用程序提供接口

口令文件包含如下的字段：



口令文件是/etc/passwd，是一个ASCII文件。每一行 包含上面的各个字段

Linux有四个用户 root squid nobody finger

阴影口令：单向加密算法处理过的用户口令副本

UNIX组文件有四个字段，组名，加密口令，数值组ID，指向各用户名指针的数组

附属组ID：可以属于口令文件记录项中组ID所对应的组，还可以属于另外16个组

UNIX还记录有其他系统数据文件，口令文件和组文件。日常中，有记录各网络服务器提供服务的数据文件，有一个记录协议信息的数据文件。

登录账户记录，utmp文件记录当前登录到系统的各个用户；wtmp文件跟踪各个登录和注销事件

系统标识：uname返回主机和操作系统有关的信息

时间和日期：time函数返回当前时间和日期

一个系统可以有多个系统时钟

clock\_gettime函数可用于获取指定时钟的时间

第七章：进程环境（好玩的要来了）

本章涉及到main函数的调用过程

main函数：int main(int argc,char \*argv[]);argc是命令行的参数的数目，argv是指向参数的各个指针所构成的数组。

当内核执行C程序时，在调用main之前会先调用一个特殊的启动例程，可执行程序文件将此例程指定为程序的起始地址—这是由连接编译器设置的。

8中方式让进程终止，正常 main返回 调用exit \_exit或\_Exit 最后一个线程从其启动例程返回 从最后一个线程调用pthread\_exit。异常有调用abort 接收一个信号 最后一个线程对取消请求作出响应

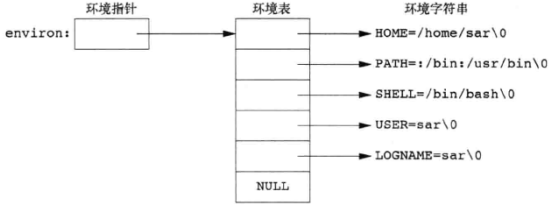
exit(0) 等价于 return(0)

atexit函数可以为进程登记函数让exit来自动调用登记的函数，登记一次就会调用一次，调用的顺序和登记的顺序相反

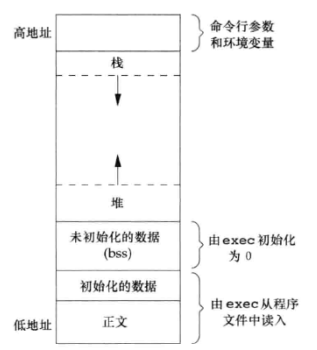
调用main之前调用的exec进程会将命令行参数传递给该新程序。

每个程序都有一张环境表，和参数表一样，环境表也是一个字符指针数组，environ作为环境表的指针，称为环境指针

历史上 有 int main（int argc,char \*argv[],char \*envp[]）;



C程序的存储空间布局：正文段，初始化数据段，未初始化数据段，栈，堆



共享库：使得可执行文件中不再需要包含公用的库函数，而只需要在所有进程都可引用的存储区中保存这种库例程的一个副本

存储空间分配：malloc calloc realloc free

环境变量：通过getenv来获取环境变量值，返回一个指向环境变量字符串的指针，可以用putenv setenv unsetenv来设置环境变量或者增加环境变量

函数setjmp和longjmp：由于C中的goto语句不能跨越函数，执行这种类型跳转功能的是这两个函数

调用函数时用到的栈：地址由高到低，先调用的函数在栈的底部，后调用的函数在栈的顶部

如果深层嵌套的函数中出现了问题，那么想要返回栈底的函数，就要一层层的返回，这样就会很麻烦，解决这个问题的办法就是用非局部goto，即setjmp和longjmp函数，用希望返回的位置用setjmp设一个值，在需要返回的位置用longjmp函数返回，longjmp还能返回一个值，给setjmp函数

随着longjum回到setjmp那一行，自动变量、寄存器变量，易失变量并不一定回滚

函数gerlimit和setrlimit可以查询和更改进程的资源限制，更改的规则是软限制值需要小于等于硬限制值，任何一个进程都可以修改软限制值和降低硬限制值，只有超级用户进程可以修改硬限制值

第八章：进程控制

本章介绍UNIX系统的进程控制，包括创建新进程、执行程序、进程终止，还将说明进程属性的各种ID，如何用进程控制原语影响它们，解释器文件、system函数、进程会计机制

进程标识，进程ID，非负整型，唯一性，可复用，新建进程不使用最近终止进程所使用的ID。

ID为0的通常是调度进程，为内核的一部分，别名交换进程，

ID 1通常是init进程，在自举过程结束时由内核调用，该进程绝不会停止，是普通用户进程，通常读取系统有关的初始化文件，并将系统引导到一个状态，以超级用户特权运行。

除了进程ID 每个进程还有 父进程ID 实际用户ID 有效用户ID 实际组ID 有效组ID

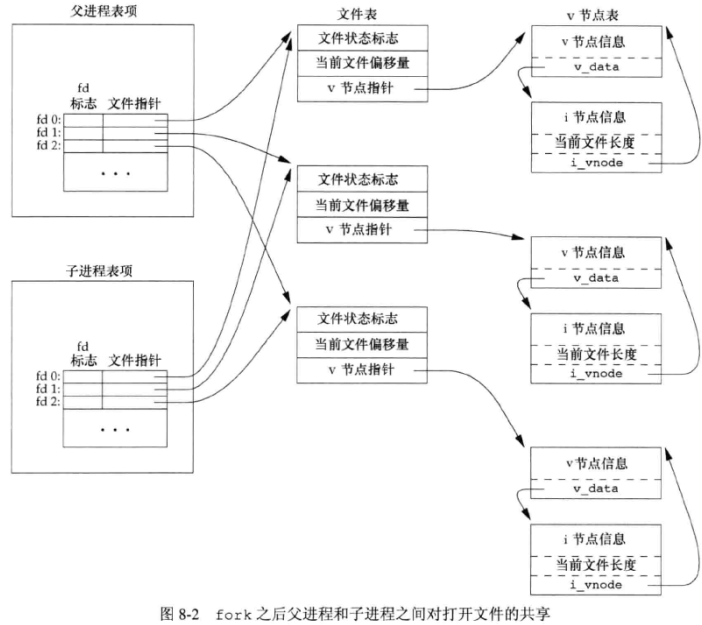
一个现有进程可以调用fork函数创建一个新进程。被称为子进程，调用一次返回两次，返回给子进程0，父进程子进程id。

一个父进程的子进程有多个，并且没有办法可以获得所有子进程ID 一个子进程的父进程只有一个，用getppid可以获得。

子进程 父进程继续执行fork调用之后的指令。子进程是父进程的副本。子进程获得父进程数据空间、堆、栈的副本。子进程拥有副本，父进程和子进程并不共享这些存储空间、父进程和子进程共享正文段。

文件共享：父进程和子进程每个相同的打开描述符共享一个文件表项，重要的一点是他们还共享文件的偏移量，例如他们如果向同一个文件写入内容，那么当父进程写入后，子进程再写入时会更新偏移量到父进程写到的地方。

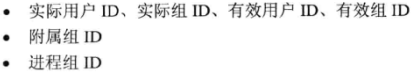
共享文件如下图：

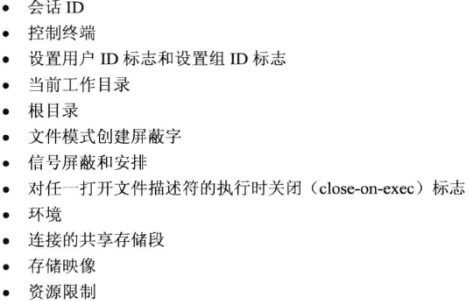


如果父进程和子进程写同一描述符指向的文件，但又没有任何形式的同步，那么它们的输出就会相互混合。

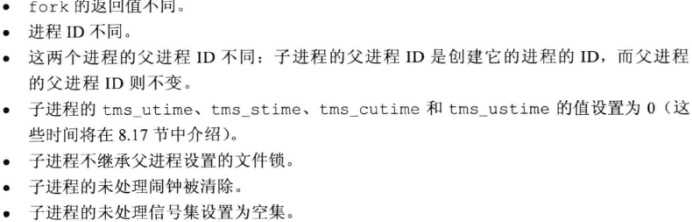
fork后处理文件描述符通常有以下两种情况，1、父进程等待子进程完成，这种情况，父进程无须对描述符做任何处理，当子进程中止后，偏移量自然已经做了更新。2、父进程和子进程各自执行不同的程序段。Fork之后，父进程和子进程各自关闭它们不需使用的文件描述符，这样就不会干扰对方使用的文件描述符。网络服务进程经常使用这种方法。

父进程还有很多其他属性由子进程继承，如：





父进程和子进程之间的区别如下：



fork失败的主要原因：1、系统已经有了太多进程 2、该实际用户ID的进程总数超过过了系统限制 CHILD\_MAX规定了每个用户ID 在任意时刻可拥有的最大进程数。

Fork有两种用法：1、一个父进程希望复制自己，使父进程和子进程同时执行不同的代码段。网络服务进程中很常见，父进程等待客户端的服务请求。当这种请求到达时，父进程调用fork 使子进程处理此请求，自己则继续等待下一个服务请求；2、一个进程要执行一个不同的程序，在shell比较常见，子进程从fork返回后立即调用exec

vfork函数：调用序列和返回值与fork相同，但是vfork用于创建一个新进程，而该新进程的目的是exec一个新程序，所以也不会引用父进程地址空间，vfork保证子进程先运行，在它调用exec或exit之后父进程才可能被调用运行，

函数exit会关闭所有标准流，\_exit不会冲洗标准流

对父进程已经终止的所有进程，他们的父进程都改变为init进程。这些进程由init进程收养。

对子进程在父进程之前终止，那么父进程调用wait或者waitpid时，可以得到子进程ID、终止状态、该进程使用的CPU时间总量。一个已经终止的，父进程尚未善后处理的进程被称为僵死进程。由init收养的进程在终止后悔init会自动调用wait不让其变为僵死进程。

wait和waitpid，如果所有子进程都还在运行，则阻塞父进程。如果一个子进程已经终止，正等待父进程获取终止状态，则获得该子进程的终止状态立即返回。如果没有任何子进程，则立即出错返回。

在一个进程终止前，wait使其调用者阻塞，而waitpid有一选项，可使调用者不阻塞，waitpid并不等待在其调用之后的第一个终止子进程，它有若干个选项，可以控制它所等待的进程。

返回的终止状态是整型，通过宏来查看结果 有 WIFEXITED WIFSIGNALEN WIFSTOPPED WIFCONTINUED

Waitpid可等待一个特定的进程，而wait则返回任一终止子进程的状态。提供了wait的非阻塞版本，通过WUNTRACED和WCONTINUED选项支持作业控制

两次fork避免僵死进程，把第二次fork的进程交给了init进程

waitid允许一个进程指定要等待的子进程，但它单独使用两个单独的参数表示要等待的子进程所属的类型

wait3 wait4比之前的wait函数要多一个返回终止进程及其所有子进程使用的资源概况 的参数

竞争条件：当多个进程都企图对某种共享数据进行处理，而最后的结果又取决于进程运行的顺序 父进程等待子进程终止需要调用wait函数中的一个，如果一个进程要等待其父进程终止，可以使用下列形式的循环



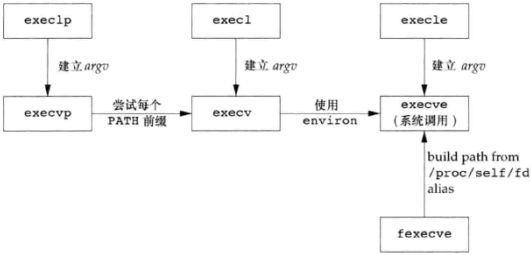
这种形式称为轮询，浪费了CPU时间，每隔1s就要唤起进程进行条件测试，为避免竞争条件和轮询，多个进程之间之间需要某种信号发送和接受的办法，在UNIX中使用信号机制，以及各种形式的进程间通信（IPC）

假定宏 TELLWAIT TELLPARENT TELLCHILD WAITPARENT WAITCHILD用来同步

函数exec用于在fork创建新的子进程后调用以执行另一个程序。调用exec并不创建新进程，前后的进程ID并未改变。exec只是用磁盘上的一个新程序替换了当前进程的正文段、数据段、堆段和栈段，新程序则从其main函数开始执行

在exec前后实际用户ID和组ID保持不变

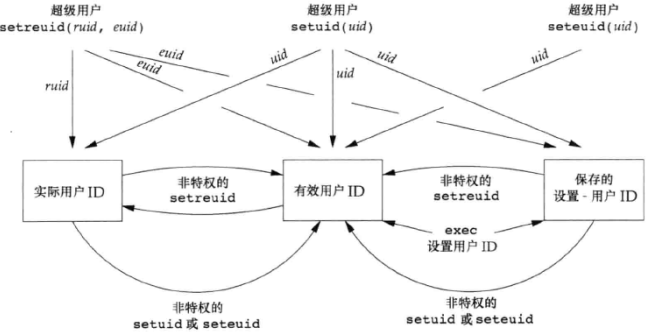
七个exec函数之间的关系：只有execve是内核的系统调用，另外6个是库函数，最终都要经过系统调用。



更改用户ID 和 更改组ID，特权以及访问控制是基于用户ID和组ID的。设计应用时，总是试图用最小特权。

setuid和setgid可以设置实际用户ID和有效用户ID以及设置实际组ID和有效组。

设置不同用户ID的各函数



组ID

解释器文件：所有现今的UNIX系统都支持解释器文件！内核使调用exec函数的进程实际执行的是该解释器文件第一行中pathname所指定的文件。

程序中执行一个命令字符串可以用system函数 ，参数是一个字符指针，如果参数为空指针，当命令处理程序可用时，返回非0值。这一特征可用来判断一个操作系统上是否支持system函数。

因为system在其视线实现中调用了fork exec和waitpid，因此有3种返回值。Fork失败或者waitpid返回除EINRT之外的出错，system返回-1，并设置errno指示错误类型，如果exec失败，则其返回值如同shell执行exit(127)；否则三个函数都成功，那么system返回shell的终止状态 system(“ data > file ”);

进程会计：大多数UNIX系统提供了一个选项进行进程会计处理。每当进程结束内核就写一个会计记录。典型的会计记录包含总量较小的二进制数据，一般包括命令名，所使用的的CPU时间总量，用户ID和组ID，启动时间等。

会计记录的说句都由内核保存在进程表中，并在一个新进程被创建时初始化。进程终止时写一个会计记录。因此：

1. 不能获取永远不终止的进程的会计记录，如init或者内核守护进程
2. 在会计文件中记录的顺序对应于进程终止的顺序，而不是它们启动的顺序。会计记录对应于进程而不是程序。如果一个进程顺序执行了三个程序，只会写一个会计记录。记录中的命令名对应于程序C，但CPU时间是三个程序之和。

用户标识：任一进程都可以得到其实际用户ID和有效用户ID及组ID。而运行该程序用户的登录名可以调用getpwuid或者getlogin函数获取。

如果调用此函数的进程没有连接到用户登录时所用的终端，则函数会失败。这种进程一般称为守护进程（daemon）

进程调度：

以nice值为标志，nice值越小，优先级越高，你越友好，你的调度优先级就越低。系统默认NZERO，范围在0-（2\*NZERO）-1

进程可以通过nice函数获取或者更改它的nice值。使用这个函数，进程只能影响自己的nice值，不能影响其他进程的nice值。

getpriority函数可以像nice一样获取进程的nice值，还可以获取一组相关进程的nice值。参数which有三个值可选 宏PRIO\_PROCESS表进程，PRIO\_PGRP表示进程组，PRIO\_USER表示用户ID

setpriority函数可用于为进程，进程组，特定用户ID的所有进程设置优先级。

进程时间：

可以度量的三个时间：墙上时钟时间，用户CPU时间、系统CPU时间，任何一个进程都可以调用times函数获得他自己以及已终止子进程的上述值。所有返回值都已经转换为秒数。

第九章：进程关系

本章详细说明进程组，会话的概念。还将介绍登录shell和所有从登录shell启动的进程之间的关系

BSD终端登录：系统自举，内核创建进程ID为1的进程，也就是init进程，该进程使系统进入多用户模式，读取文件/ec/ttys，对每一个允许登录的终端设备，init调用一次fork，它所生成的子进程则exec getty程序，getty对终端设备调用open函数，以读、写方式将终端打开。然后getty输出login: 之类的信息，等待用户键入用户名，之后它调用login程序。如果用户正确登录，login将完成一系列工作，然后登录用户的登录shell开始运行，父进程是init进程，登录shell读取启动文件并执行，当执行完后，用户得到shell提示符，并能键入命令

Linux终端登录：类似于BSD终端登录

BSD网络登录：有一个inetd进程，等待大多数网络连接。作为系统启动的一部分，init调用一个shell，使其执行shell脚本/etc/rc。由此脚本启动一个守护进程inetd。，一旦shell脚本终止，inetd父进程就变成了 init，inted等待TCP/IP连接请求到达主机，而当一个连接请求到达时，它执行一次fork，然后生成的子进程exec适当的程序。

Linux网络登录：除了有些版本使用的守护进程为xinetd代替inetd进程之外，与BSD网络登录相同。

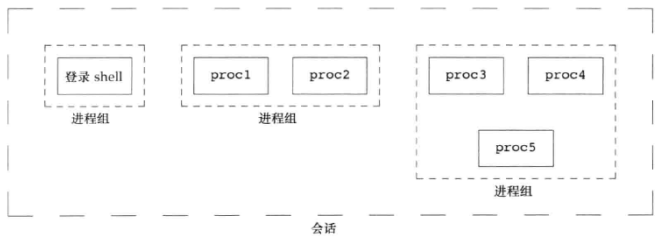
进程组：每个进程除了有一进程ID之外，还属于一个进程组。进程组是一个或多个进程的集合，同一进程组中的各进程接收来自同一终端的各种信号。每个进程组有一个唯一的进程组ID。进程组ID类似于进程ID，是一个正整数，函数getpgrp返回进程的进程组ID

进程组组长：与进程组id相同，可以创建一个进程组，创建该组中的进程，然后终止，只要在某个进程组中有一个进程存在，则该进程组就存在，这与其组长进程是否终止无关。某个进程组中的最后一个进程可以终止，也可以转移到另一个进程组。

进程调用函数setpgid可以加入现有的进程组或者创建一个新进程组。

一个进程只能为它自己或它的子进程设置进程组ID，在它的子进程调用了exec后，它就不再更改该子进程的进程组ID

会话是一个或多个进程组的集合，例如下图：



进程调用setsid函数建立一个新会话。

父进程已经终止的进程称为孤儿进程，由init收养，而整个进程组也可能成为孤儿，孤儿进程组则是一个进程组中的每个成员的父进程要么是该组的一个成员，要么不是该组所属会话的成员。另一种描述是：一个进程组不是孤儿的条件是该组中有一个进程，其父进程在属于同一会话的另一个组中。如果进程组不是孤儿进程组，那么在属于同一会话的另一进程组中的父进程就有机会重启该组中停止的进程。

第十章：信号

信号是软件中断

信号提供了一种处理异步事件的方法

每个信号都有一个名字，以SIG开头，如SIGABRT,SIGALRM两个分别是夭折信号和闹钟信号。

在头文件<signal.h>中，信号名被定义为正整数常量。

不存在编号为0的信号，kill对信号编号0由特殊的应用

产生信号的条件：1、用户按终端键 2、硬件异常如无效内存引用，除数为0等 3、进程调用kill函数可以把任一信号发送给另一个进程，发送和接受的进程所有者必须相同，发送信号进程所有者必须是超级用户4、用户可用kill命令将信号发送给其他进程，命令只是kill函数的接口5、某种软件条件发生，也应该通知有关进程产生信号

当信号出现时，可以告诉内核以三种方式之一处理：1、忽略2、捕捉信号3、执行系统默认动作（大多数系统的默认动作是终止该进程）

默认进程终止可以采用终止+core表示在进程当前工作目录的core文件中复制了该进程的内存映像，大多数UNIX系统调试程序使用core文件检查进程终止状态

一定条件不产生core 有id但是用户或zu不是当前程序文件的所有者，没有写当前工作目录的权限，文件存在用户没有写权限，文件太大

UNIX信号机制最简单接口是signal函数，第一个参数是信号，第二个参数是信号处理方式

在UNIX中，kill并不能恰当的作为杀死，在后台中使用kill命令和kill函数只是将一个信号发送给一个进程或者一个一个进程组，信号是否终止进程取决于该信号的类型。

当执行一个程序时，所有信号的状态都是系统默认或者忽略，通常所有信号都被设置为他们的默认动作，除非调用exec的进程忽略该信号。 而当进程调用fork，其子进程继承父进程的信号处理方式，因为子进程在开始时复制了父进程内存映像。

系统在执行一个低速系统调用而阻塞期间捕捉到一个信号，则该系统调用就被中断而不再继续执行

系统因为收到信号而产生中断，处理完信号后再返回中断处继续执行，中断处的函数需要是可重入的和不可重入的，如a)已知它们使用静态数据结构 b) 他们调用malloc或free c)他们是标准I/O函数 三种情况它们是不可重入的，在信号处理程序中调用非可重入的函数，结果将是不可预期的

定义讨论信号的术语：当造成信号的事件发生时，为进程***产生***一个信号，事件可以是硬件异常、软件条件、终端产生的信号、调用kill函数等。当对信号采取了某种动作，我们说对进程***递送***了一个信号，信号产生和递送之间的时间间隔内，信号是***未决***的。

进程解除对某个信号的阻塞之前，这种信号发生了多次，系统递送这个信号一次或多次，如果递送信号多次，则称这些信号进行了排队

如果有多个信号递送给一个进程，递送的顺序是未规定的，建议先递送与进程当前状态相关的信号

kill函数向进程或进程组发送信号，raise允许进程向自身发送信号，kill中pid参数有四种选择

alarm函数设置一个定时器，某个时刻定时器超时然后产生SIGALRM信号。忽略此信号则终止调用该alarm函数的进程

pause函数使进程挂起，直到捕捉到一个信号。

信号集：一个可以表示多个信号的数据类型， 数据类型定义为sigset\_t，共有五个处理信号集的函数，如果实现的信号数目少于一个整型量所包含的位数，则可用一位代表一个信号的方法实现信号集。

函数sigprocmask可以检测或者更改一个进程的信号屏蔽字

函数sigpending返回一个信号集，对于调用进程而言，其中信号都是阻塞不能递送的，因而也一定是当前未决的。

函数sigaction的功能是检查或修改于指定信号相关联的处理动作。

函数sigsetjmp和siglongjmp：非局部转移的setjmp和longjmp，在信号处理程序中调用longjmp函数有可能会捕捉到一个信号，进入处理程序，此时该信号自动被加入到信号屏蔽字中，阻止后来的此信号中断处理程序，如果用longjmp跳出信号处理程序，此进程的信号屏蔽字会发生什么呢？因此，为了避免这种情况，定义了两个新函数，在信号处理程序中进行非局部转移应该用sigsetjmp和siglongjmp这两个函数。

函数sigsuspend可以在一个原子操作中先恢复信号屏蔽字，然后使进程休眠，进程的信号屏蔽字设置为由sigmask指向的值，接收到恢复了屏蔽字的信号之前，这个进程会被挂起，直到捕捉到一个信号而且从该信号处理程序返回。另一个应用是等待信号处理程序设置一个全局变量，还可以用信号实现父子进程之间的同步，使用两个用户定义的信号来实现进程同步的五个函数，如果希望等待信号时去休眠，则使用sigsuspend很合适。

函数abort的功能是使程序异常终止。该函数将SIGABRT信号发送给调用进程，调用abort将向主机环境递送一个未成功终止的通知，abort不会理会进程对此信号的忽略和阻塞，捕捉此信号并从信号处理程序返回，abort仍不会返回其调用者

函数

函数system，比第八章的版本增加了信号处理，忽略SIGINT和SIGQUIT，阻塞SIGCHLD。因为执行system时父进程要等待子进程完成，所以阻塞SIGCHLD，而用system运行另一个程序时，也不应使父、子进程两者都捕捉终端产生的两个信号：中断和退出，父进程此时处于等待子进程的状态，这两个信号应该只发送给正在运行的程序-子进程

clock\_nanosleep可以使用相对于特定时钟的延迟时间来挂其调用线程

使用排队信号需要的操作：使用sigaction函数安装信号处理程序时指定SA\_SIGINFO标志。如果没有给出这个标志，信号会延迟，但信号是否进入队列要取决于具体实现；在sigaction结构的sa\_sigaction成员中提供信号处理程序。使用sigqueue函数发送信号。信号也不能无限排队，到达相应的限制以后，sigqueue就会失败。

作业控制信号 SIGCHLD SIGCONT SIGSTOP SIGTSTP SIGTTIN SIGTTOU。除SIGCHLD以外，大多数应用程序并不处理这些信号，

信号名和信号编号的映射：有的是用字符指针数组，指向信号名，下标为信号编号；psignal函数可以打印与信号编号对应的字符串；如果在sigaction信号处理程序中有siginfo结构，可以使用psiginfo函数打印信号信息，如果只需要信号的字符描述部分，可以使用strsignal函数，sig2str 和 str2sig可以将信号编号映射为信号名，另一个反之

第十一章 线程

本章将进一步深入理解进程，了解如何使用多个控制线程在单进程环境中执行多个任务。一个进程中所有线程都可以访问该进程的组成部件，如文件描述符和内存

不管什么情况下，只要单个资源在多个用户间共享，就必须处理一致性问题。本章将讨论目前可用的同步机制，防止多个线程在共享资源时出现不一致的问题。

线程概念：典型的UNIX进程可以看成只有一个控制线程：一个进程在某一个时刻只能做一件事情，有了多个控制线程之后，在程序设计时就可以把进程设计成在某一时刻能够做不止一件事，每个线程处理各自独立的任务。

为每个时间类型分配单独的处理线程，可以简化处理异步事件的代码。每个线程在进行事件处理时可以采用同步编程模式；

每个进程必须使用操作系统提供的复杂机制才能实现内存和文件描述符的共享，而多个线程自动地可以访问相同的存储地址空间和文件描述符

有些问题可以分解从而提高整个程序的吞吐量。只有一个控制线程的情况下，一个单线程进程要完成多个任务，只需要把这些任务串行化。有多个控制线程时，相互独立的任务的处理就可以交叉进行，只需要为每个任务分配一个单独的线程；

交互的程序同样可以通过使用多线程来改善响应时间，多线程可以把程序中处理用户输入输出的部分和其他部分分开；

处理器的数量并不影响程序结构，所以不管处理器的个数多少，程序都可以通过使用线程得以简化。

每个线程都包含有表示执行环境所必须的信息，其中包括进程中标识线程的线程ID、一组寄存器值、栈、调度优先级和策略、信号屏蔽字、errno变量以及线程私有数据，一个进程的所有信息对该进程的所有线程都是共享的，包括可执行程序的代码，程序的全局内存堆内存、栈以及文件描述符。

线程标识：每个线程都有一个线程ID，进程ID在整个系统中是唯一的，但线程ID不同，线程ID只有在它所属的进程上下文中才有意义。

线程id是用pthread\_t数据类型来表示的，实现的时候可以用一个结构来代表pthread\_t数据类型。

比较两个线程ID使用一个函数pthread\_equal

线程可以通过调用prhread\_self函数获得自身的线程ID

传统UNIX模型中，每个进程只有一个控制线程，创建多个控制线程之前，程序的行为与传统的进程并没有什么区别。新的线程可以通过调用pthread\_create函数创建

线程创建时并不能保证哪个线程会先运行，是新创建的线程，还是调用线程，新创建的线程可以访问进程的地址空间，并且继承调用线程的浮点环境和信号屏蔽字，但该线程的挂起信号集会被清除。

pthread函数在调用失败时通常会返回错误码，它们并不想其他的POSIX函数一样设置errno

线程终止：如果进程中任意线程调用了exit、\_Exit或者\_exit，那整个进程就会终止。如果默认的动作是终止进程，那么发送到线程的信号就会终止整个进程

线程可以有三种方式退出：线程可以简单地从启动例程中返回，返回值是线程的退出码；线程可以被同一进程中的其他线程取消；线程调用pthread\_exit

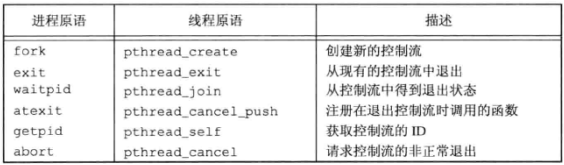
函数pthread\_join使调用线程等待指定线程终止，期间将自身挂起，当指定线程通过pthread\_exit退出或者简单地从启动例程中返回时，进程中的其他线程可以通过调用pthread\_join获得该线程的退出状态

函数pthread\_exit和pthread\_join两个函数可以在线程间交换信息，exit把相应的参数返回，join可以取得这个参数

函数pthread\_cancel可以通过调用来实现取消同一进程中的其他线程，但是并不等待线程终止，只是提出请求

线程可以安排他们退出时调用的函数，这样的函数称为线程清理处理程序 一个线程可以建立多个清理处理程序，处理程序记录在栈中，他们的执行顺序与他们注册时相反，使用函数pthread\_cleanup\_push pthread\_cleanup\_pop，如果线程是通过它的启动例程中返回而终止的话，它的清理处理程序就不会被调用

进程函数和线程函数的相似之处：

  
默认情况下，线程的终止状态会保存直到对该线程调用pthread\_join。如果线程已经被分离，线程的底层存储资源可以在线程终止时立即被收回。可以调用prhread\_detach分离线程

线程同步：当每个控制线程共享相同的内存时。需要确保每个线程看到一致的数据视图。当一个线程可以修改的变量，其他线程也可以读取或者修改的时候，我们就需要对这些线程进行同步。

锁：为线程之间进行同步所设立的机制，要访问线程的共同变量，需要先获得锁

互斥量：可以使用pthread的互斥接口来保护数据，互斥量本质是一把锁。互斥变量用pthread\_metex\_t数据类型表示，pthread\_mutex\_t数据类型表示的互斥变量使用之前，需要先初始化，要么静态分配互斥量，要么调用pthread\_mutex\_init函数进行初始化。如果动态分配互斥量，释放内存前要调用pthread\_mutex\_destory

对互斥量加锁：使用pthread\_mutex\_lock，如果已经上锁，调用线程将阻塞直到互斥量被解锁。如果不希望被阻塞，它可以使用pthread\_mutex\_trylock尝试对互斥量进行加锁，如果成功锁住就返回0，如果失败返回EBUSY。不会阻塞线程。

对互斥量解锁：调用pthread\_mutex\_unlock

避免死锁：如果一个线程试图对同一个互斥量加锁两次，其自身就会陷入死锁。

解决死锁的办法之一：如果程序需要对多个互斥量加锁，给互斥量排序，依次加锁。

解决死锁的一个实例：使用一个存储互斥量指针的哈希表，相同的用链表解决，先获得哈希表，再获得互斥量

函数pthread\_mutex\_timedlock互斥量原语允许绑定线程阻塞时间，在当线程试图获取一个已加锁的互斥量时。可以避免取得不到的互斥量时发生死锁

读写锁：与互斥量类似，但是有更高的并行性，互斥量只有加锁和不加锁两种状态，一次只有一个线程可以对其加锁。读写锁有3个状态：读模式下加锁状态，写模式下加锁状态，不加锁状态。 一次只有一个线程可以占有写模式的读写锁，但是多个线程可以同时占有读模式的读写锁

写加锁状态：此时任何试图对这个锁加锁的进程都会被阻塞

读加锁状态：所有以读模式对它进行加锁的线程都可以得到访问权，任何希望以写模式对此所进行加锁的线程都会阻塞，直到所有线程释放它们的读锁为止。当一个线程试图以写模式获取锁时，读写锁通常会阻塞随后的读模式锁请求。避免读模式锁长期占用

读写锁也叫做共享互斥锁，读写锁是读模式锁住时，就可以说是以共享模式锁住的。写模式锁住时，就可以说是互斥模式锁住的。

带有超时的读写锁：和上面相同，为了避免获取读写锁而永久的陷入阻塞状态。

条件变量：线程可用的另一种同步机制，给多个线程提供了一个会合的场所。由互斥量保护，线程在改变条件之前必须先锁住互斥量。其他线程在获得互斥量之前不会察觉这种改变，因为互斥量必须在锁定后才能计算条件。

条件变量有一个互斥量进行保护，先把锁住的互斥量传给函数然后函数吧调用的线程放到等待队列上，对互斥量解锁。有两个函数可以用于通知线程条件已经满足，pthread\_cond\_signal函数至少能唤起一个等待该条件的线程，而pthread\_cond\_breadcast函数则能唤醒等待该条件的所有进程。

自旋锁：与互斥量类似，但它不是通过休眠使进程阻塞，而是在获取锁之前一直处于忙等。锁被持有的时间短，而且线程不希望在重新调度上花费太多的成本。

屏障：用户协调多个线程并行工作的同步机制。允许每个线程等待，直到所有的合作线程都到达某一点，然后从该点继续执行。允许任意数量的线程等待，直到所有的线程完成处理工作，而线程不需要退出。所有线程达到屏障后可以接着工作，pthread\_join就是一种屏障，允许一个线程等待，直到另一个线程退出。

第12章：线程控制

本章讲解控制线程行为方面的详细内容，介绍线程属性和同步原语属性。还将介绍同一进程中的多个线程之间如何保持数据的私有性。最后讨论基于进程的系统调用如何与线程进行交互。

线程限制：线程的几个属性的限制，主要影响移植性，有线程退出时操作系统试图销毁线程特定数据的最大次数，进程可以创建的键的最大数目，一个线程的栈可用的最小字节数，进程可以创建的最大线程数。

线程属性：pthread接口允许通过设置每个对象关联的不同属性来细调线程和同步对象的行为。创建线程时用到的pthread\_attr\_t类型指针里面包含着线程的许多属性，如可以用pthread\_detach分离线程，如果在创建时就对线程的终止状态不感兴趣，可以修改pthread\_attr\_t结构中的detachstate线程属性。让线程一开始就属于分离状态。

线程栈属性管理：对支持线程栈属性的系统的进行线程栈属性的管理，可以使用pthread\_attr\_getstack和pthread\_attr\_setstack

同步属性：就像线程具有属性一样，线程的同步对象也有属性，互斥量属性，读写锁属性，条件变量属性和屏障属性

互斥量属性：用pthread\_mutexattr\_t结构表示。值得注意的是进程共享属性，健壮属性以及类型属性。

读写锁属性：支持唯一属性进程共享属性

条件变量属性：进程共享属性和时钟属性

屏障属性：进程共享属性

线程安全：如果函数在相同的时间点可以被多个线程安全地调用，则该函数是***线程安全***的。异步信号安全的

线程特定数据：线程私有数据，存储和查询某个特定线程相关数据的一种机制。不用担心和其他线程的同步访问问题。这样设计有两个原因，有时候需要维护基于每线程的数据；提供让基于进程的接口适应多线程环境的机制。

取消选项：可取消状态和可取消类型两属性影响着线程在响应pthread\_cancel函数调用时所呈现的行为，但并没有包含在pthread\_attr\_t结构中。

线程和信号：进程中处理信号已经很复杂，线程中处理信号会更加复杂。每个线程都有自己的信号屏蔽字，但是信号的处理是进程中所有线程共享的。

线程与fork：线程调用fork时，就为子进程创建整个进程地址空间的副本。子进程内部只存在一个线程，它是由父进程中调用fork的线程的副本构成的。如果父进程中的线程占有锁，子进程同样占有锁，但问题是子进程并不包含占有锁的线程的副本，因此不知道它占有了哪些锁，需要释放哪些锁。所以子进程在fork返回以后，如果不是紧接着调用exec，就需要清理锁状态。如果调用exec，旧的空间就被抛弃，因此锁的状态就无关紧要。

线程和IO：pread和pwrite函数在多线程下非常有用，因为所有线程共享相同的文件描述符。可以使用pwrite来解决并发线程对同一文件进行写操作的问题。

第十三章 守护进程

守护进程是生存期长的一种进程。常在系统引导装入时启动，仅在系统关闭时终止。

在后台运行，没有控制终端。

特征：大多数守护进程都以超级用户特权运行，所有的守护进程都没有控制终端。没有控制终端可能是守护进程调用了setsid的结果，大多数用户层守护进程都是进程组的组长进程以及会话的首进程，而且是这些进程组和会话中的唯一进程。应当引起注意的是用户层守护进程的父进程是init进程。

编写规则：首先用调用umask将文件模式创建屏蔽字设置为一个已知值；调用fork，然后使父进程exit，如果守护进程是作为一条简单的shell命令启动的，那父进程终止会让shell认为这条命令执行完毕。虽然子进程继承了父进程的进程组ID，但获得了一个新进程ID，这保证了子进程不是一个进程组的组长进程。调用setsid创建一个新会话，然后执行一些操作，使调用进程，成为新会话的首进程，成为新进程组的组长进程，没有控制终端。将当前工作目录更改为根目录。关闭不再需要的文件描述符。某些守护进程打开/dev/null使其具有文件描述符0、1和2，这样，任何一个试图读标准输入，写标准输出或标准错误的库例程都不会产生任何效果。

出错记录：一个集中的守护进程出错记录设施，如BSD的syslog设施。该设施的接口是syslog函数。

单实例守护进程：为了正常运作，某些守护进程实现为在任一时刻只允许该守护进程的一个副本。例如排他的访问一个设备可以用文件和记录锁机制。

守护进程的惯例：若守护进程使用锁文件，通常存储在/var/run目录中；若守护进程支持配置选项，那配置文件通常放在/etc目录中。守护进程可用命令行启动，但它们是由系统初始化脚本之一启动的；若一个守护进程有一个配置文件，那么当该守护进程启动时会读取该文件，在此之后一般就不会再查看它。

客户进程-服务器进程模型：守护进程常常作为服务器进程。服务器进程等待客户进程与其联系，提出某种类型的服务要求。服务器进程中调用fork然后exec另一个程序来向客户进程提供服务是很常见的。服务器进程通常管理着多个文件描述符，而子进程对所有不需要的文件描述符设置执行关闭标志是一个很好地行为

第十四章：高级IO

本章涉及到很多概念和函数，15-17章会用到很多。

非阻塞IO：

系统调用被分为两类：“低速”系统调用和其他。低速系统调用是可能会使进程永远阻塞的一类系统调用。

虽然读写磁盘文件会暂时阻塞调用者，但并不能将与磁盘IO有关的系统调用视为低速

非阻塞IO使我们可以发出open、read、write这样的IO操作，并使这些操作不会永远阻塞。如果这种操作不能完成，则调用立即出错返回，表示该操作如继续执行将阻塞

对于一个给定的描述符，有两种为其指定非阻塞IO的方法，1、如果调用open获得描述符，可指定O\_NONBLOCK标志2、对于已经打开的一个描述符，可调用fcntl，由该函数打开O\_ONOBLOCK文件状态标志

用非阻塞IO可以在出现错误时直接返回，许多write调用中可能只有一部分能输出数据，其余都返回错误，但是不会因此阻塞。

记录锁（字节范围锁）：

当第一个进程正在读或修改文件的某个部分时，使用记录锁可以阻止其他进程修改同一文件区。

早期：flock （锁整个文件） fcntl 扩展的 lockf可以支持锁文件的一部分。

fcntl记录锁：参数中可以选择锁的属性，命令属性等、F\_GETLK测试能否建立一把锁，然后F\_SETLK或F\_SETLKW企图建立那把锁，这两个不是一个原子操作，因此必须处理由F\_SETLK返回的可能的出错。

文件尾端加锁

建议性锁和强制性锁：

IO多路转接：

telnet进程有两个输入，两个输出。解决阻塞IO读取输入时不知道哪个输入有数据而发生阻塞的问题可以使用两种方法解决，1、使用新的进程，两个进程分别处理两个输入，这样不用担心阻塞。2、使用一个进程，但使用非阻塞IO读取数据，将两个输入符都设置为非阻塞的，对第一个描述符发一个read，如果该输入有数据，则读数据并处理它。如果没有则立即返回。对第二个描述符作同样的处理。等待一段时间，再尝试从第一个描述符读，称为轮询。但是浪费CPU资源3、异步IO。利用这种技术，进程告诉内核：当描述符准备好可以进行IO时，用一个信号通知它。但是可移植性是一个问题，每个进程而言只有一个信号，对操作多个文件描述符时会无法分辨哪一个文件描述符有数据。

较好的解决上述有两个文件描述符需要一个进程进行读取，进程阻塞在没有数据的文件描述符的问题的较好的技术是使用IO多路转接。为使用这种技术，先构造一张我们感兴趣的描述符的列表，然后调用一个函数，直到这些描述符中的一个已准备好进行IO，该函数才返回。

poll，pselect和select这三个函数使我们能够执行IO多路转接。当函数返回时，进程会被告知那些描述符已准备好可以进行I/O

select函数使我们可以执行IO多路转接。

传给select的参数告诉内核：我们关心的描述符；对于每个描述符我们所关心的条件；愿意等待多长时间；

从select返回时，内核告诉我们：已准备好的描述符的总数量；对于读写或异常这3个条件中的每一个，哪些描述符是已准备好。

对于这种返回信息，就可以调用相应的IO函数，并且确知该函数不会堵塞。

还有变体pselect，支持纳秒而非微秒，超时值被声明为const，保证调用pselect不会改变此值。pselect可以使用可选信号屏蔽字，调用pselect时，以原子操作的方式安装信号屏蔽字，返回时，恢复以前的信号屏蔽字。

poll函数：

类似于select，但是程序接口有所不同，poll不为每个条件构造一个描述符集，而是构造一个pollfd结构的数组，每个数组元素制定一个描述符编号以及我们的对该描述符感兴趣的条件。

poll函数结构中，数组元素中的events成员告诉内核我们关心描述符的哪些事件。revents成员由内核设置，用于说明每个描述符发生了哪些事件。

异步IO：解决了局限性，带来了其他很多的复杂操作

System V异步IO：异步IO信号是SIGPOLL，调用ioctl指定产生SIGPOLL信号的条件，还应为该信号建立信号处理程序

BSD异步IO：异步IO信号是SIGIO和SIGURG的组合。

POSIX异步IO：

函数readv和writev：用于在一次函数调用中读写多个非连续缓冲区，同时也将这两个函数称为散布读和聚集写。第二个参数是iovec结构数组的一个指针。比多次调用write和read来读取数据更加的方便。

函数readn和writen：读写一个管道，网络设备或终端时，需要考虑这两个函数，功能分别是读写指定的N字节数据，并处理返回值可能小于要求值的情况。按需多次调用read和write直到读写了N字节数据。

存储映射IO：能将一个磁盘文件映射到存储空间中的一个缓冲区上。从缓冲区读数据，就相当于读文件中的相应字节。将数据存入缓冲区，相应字节就自动写入文件。就可以不适用read和write的情况下执行IO

第十五章：进程间通信

第八章说明了用打开的文件在进程之间交换信息，本章将说明进程通信的其他技术（IPC InterPrecess Communication）

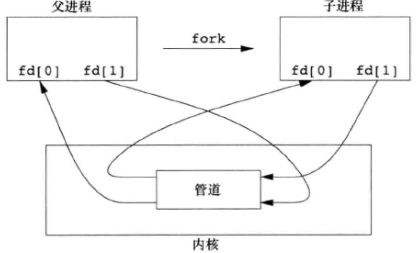
IPC种类：管道 FIFO 消息队列 信号量 共享存储 套接字（网络IPC）

管道：

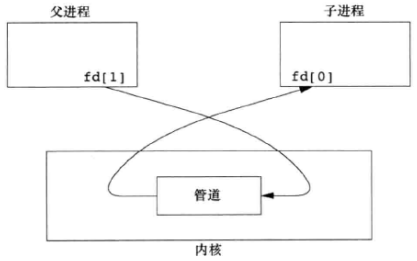
半双工或全双工，只能在具有公共祖先的两个进程之间使用

pipe(int fd[2]) fd[0]为读而打开，fd[1]为写而打开，fd[1]的输出是fd[0]的输入

通常，进程先调用pipe，接着调用fork，从而创建从父进程到子进程的IPC通道，反之亦然



fork之后做什么取决于我们想要的数据流的方向，比如我们要从父进程到子进程的管道，父进程关闭读端，子进程关闭写端



读一个写端被关闭的管道时，所有数据读取后，read返回0表示文件结束

写一个读端被关闭的管道时，产生信号SIGPIPE，如果忽略该信号或者捕捉信号并从处理程序返回，则write返回-1

函数popen和pclose：实现操作 创建一个管道，fork一个子进程，关闭未使用的管道端，执行一个shell运行命令，然后等待命令终止

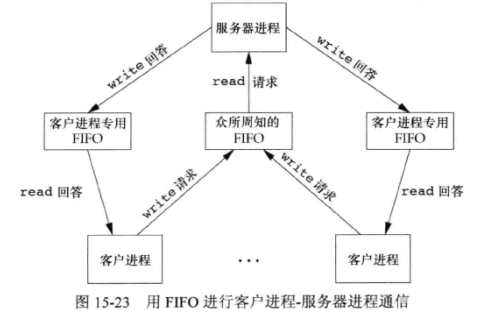
协同进程：一个过滤进程既产生某个过滤程序的输入，又读取该过滤程序的输出，它就变成了协同进程。

FIFO：也被称为命名管道，与未命名管道相比，可以在不相关的进程交换数据，不需要有共同的祖先。

FIFO的两个用途：

Shell命令使用FIFO将数据从一条管道传送到另一条时，无须创建中间临时文件

客户进程-服务器进程应用程序中，FIFO用作汇聚点，在客户进程和服务器进程两者之间传递数据



这种模型的问题是服务器进程不能判断一个客户进程是否崩溃终止，客户进程专用FIFO会遗留在文件系统中。

XSI IPC ：消息队列，信号量，共享存储器

标识符和键：XSI IPC结构都用一个非负整数的标识符加以引用。作为身份ID，与文件描述符不同，IPC标识符不是小的整数。标识符是IPC对象内部名，卫视多个合作进程能够在同一IPC上汇聚，因此需要外部名：键。无论何时创建IPC结构，都应该指定一个键。

消息队列：消息链接表，存储内核中，消息队列标识符标识。msgget msgsnd msgrcv三个函数，msgget负责创建一个新队列或打开一个现有队列，msgsnd将新消息添加到队列尾端，msgrcv用于从队列中取消息，现在消息队列速度没什么优势，还很麻烦，因此不要再使用了

信号量：不同于交换数据的其他IPC机构，他是一个计数器，用于为多个进程提供对共享数据对象的访问。也就是pv操作中的那个信号量。信号量值的测试及减1操作应是原子操作。

信号量的复杂性：信号量并非是单个非负值，而必须定义为好友一个或多个信号量值的集合，创建信号量时，要指定集合中信号量的初值，信号量创建应该独立于它的初始化。因为不能原子的创建一个信号量集合。即使没有进程正在使用各种形式的XSI IPC，它们依然是存在的。

和互斥量比较，信号量和文件记录锁效率没有互斥量高

共享存储：允许两个或多个进程共享一个给定的存储区，数据不需要再客户进程和服务器进程之间负值，最快的一种IPC，服务器进程在将数据放入共享存储区，则它做完这一操作之前，客户进程不应当去取这些数据，信号量用于同步共享存储访问，shmget获得共享存储标识符，shmctl函数对共享存储段执行多种操作，shmat将其连接到它的地址空间中。

第十六章 网络IPC 套接字

上一章是同一台电脑之间的通信，本章是不同电脑之间的通信

套接字接口：同样的接口既可以用于计算机间通信，也可以用于计算机内通信，套接字接口可以采用许多不同的网络协议进行通信，本章讨论基于TCP IP协议栈

套接字：使用socket函数创建，三个参数分别是域，套接字类型，网络协议

套接字也是一种文件描述符：

调用socket和调用open相似，均可获得用于IO的文件描述符。当不再需要该文件描述符时，调用close来关闭对文件和套接字的访问，并且释放该描述符以便重新使用。

虽然socket本质上是一个文件描述符，但不是所有参数为文件描述符的函数都可以接受套接字描述符。

套接字通信是双向的，可以采用shutdown函数来禁止一个套接字的IO ，SHUT\_RD SHUT\_WR 两个选项。

寻址：网络地址+端口号

地址格式：为了统一，地址会被强制转换成一个通用的地址结构sockaddr，IPv4因特网中，套接字地址用结构sockaddr\_in表示

地址查询：通过调用gethostent，可以找到给定计算机系统的主机信息

使用bind函数来关联地址和套接字：对于服务器，需要给一个接受客户端请求的服务器套接字关联上一个众所周知的地址。最简单的方法就是服务器保留一个地址并且注册在/etc/services或者某个名字服务中。

建立连接：如果要处理一个面向连接的网络服务，开始交换数据之前，需要在客户端进程套接字和服务器进程套接字之间之间建立连接，使用connect函数建立，成功连接需要服务器等待连接队列有足够的空间，因此，应用程序必须能够处理connect返回的错误。尤其是一个服务器运行在一个负载很重的系统中时。

connect出错的指数补偿算法：如果调用connect失败，进程会休眠一小段时间，然后进入下次循环进行尝试。循环休眠时间指数级增大，最大两分钟

如果connect失败，可迁移的应用程序需要关闭套接字。如果重试，必须打开一个新的套接字。

如果套接字描述符处于非阻塞模式，那么连接不能马上建立时，connect会返回-1并将errno设置为特殊的错误码EINPROGRESS。应用程序可以使用poll或者select来判断文件描述符何时来写，如果可写，连接完成。

服务器调用listen函数来选考它愿意接受连接请求。

listen函数：参数backlog提供了一个提示，提示系统该进程要入队的未完成请求数量。实际值由系统决定，上限由<sys/socket.h>中的SOMAXCONN指定。一旦队列满，系统就会拒绝多余的连接请求。

服务器调用了listen，套接字就能接受连接请求，使用accept函数获得连接请求并建立连接。

函数accept所返回的文件描述符是套接字描述符，该描述符连接到调用connect的客户端。

数据传输：

既然一个套接字端点表示为一个文件描述符，那么只要建立连接，就可以使用read和write来通过套接字通信，进行进程间交换数据

传输数据的函数，3个发送，3个接收。

send sendto sendmsg recv recvfrom recvmsg

采用文件描述符来访问套接字是非常有意义的，它允许程序对联网环境的网络访问一无所知。

套接字选项：

提供了两个套接字选项接口来控制套接字行为。一个接口用来设置选项，另一个接口可以查询选项的状态。

setsockopt 和 getsockopt

带外数据：

允许更高优先级的数据传输 ，带外数据先行传输，即使传输队列已经有数据。TCP支持带外数据，并将其称为紧急数据，TCP仅仅支持一个字节的紧急数据。

接收带外数据会产生信号，可以用fcntl来指定让哪个进程来处理这个信号

非阻塞和异步IO：

通常，recv函数没有数据可用时会阻塞等待。同样，当套接字输出队列没有足够空间来发送消息时，send函数会阻塞。

在非阻塞模式下，行为会发生改变。函数不会阻塞而是会失败。将errno设置为EWOULDBLOCK或者EAGAIN。当这种情况发生时，可以使用poll或select来判断能否接受或者传输数据。

异步IO使用信号SIGIO来实现，两个步骤，1，建立套接字所有权，这样信号可以被传递到合适的进程。2，通知套接字当IO操作不会阻塞时发信号。

第十七章 高级进程间通信

UNIX系统提供的各种IPC，包括管道 套接字

本章介绍一种高级IPC，UNIX域套接字机制。这种形式的IPC可以在统一计算机系统上运行的两个进程之间传送打开文件描述符。服务进程可以使他们的打开文件描述符与指定的名字相关联，同一系统上运行的客户进程可以使用这些名字与服务器进程汇聚。还会了解到操作系统如何为每一个客户进程提供一个独用的IPC通道。

UNIX域套接字：

用以同一台计算机上运行的进程之间的通信，相比于使用因特网域套接字效率更高。UNIX域套接字仅仅复制数据，不执行协议处理。不需要添加或删除网络报头，无需计算校验和，不要产生序列号，无需发送确认报文。

提供流 和 数据报两种接口，socketpair函数创建无命名的、相互连接的UNIX域套接字。

一对相互连接的UNIX域套接字可以起到全双工管道的作用：两端对读和写开放。我们将其称为fd管道，以便和普通半双工管道区分

域套接字可以和消息队列结合从而让消息队列也可以使用poll和select机制，因为套接字是文件描述符

虽然socketpair函数能创建一对相互连接的套接字，但是每个套接字都没有名字，这意味着无关进程不能使用它们。但是套接字可以改变地址，通过一定的设置，因为套接字也是一个文件描述符，因此绑定地址也就是给文件描述符设置一个文件能让别的进程通过目录找到它，操作是通过bind函数将一个文件描述符和套接字进行绑定，别的进程就能用这个文件描述符来访问套接字

UNIX域套接字建立唯一链接的方法：

serv\_listen函数将调用socket创建一个UNIX域套接字，然后将一个众所周知的指定的路径填入套接字的sockaddr\_un 结构，调用这个listen函数来通知内核该进程作为服务器进程等待客户进程的连接请求。并返回套接字绑定的这个文件描述符

当收到客户进程连接请求后，服务器进程调用serv\_accept函数，然后服务器进程会阻塞，等待一个客户进程调用cli\_conn。从accept返回时，返回值是连接到客户进程的崭新的描述符。

客户进程调用cli-conn函数对连到服务器进程的连接进行初始化

传送文件描述符：

它使服务器进程能够处理打开一个文件所要做的一些操作，以及向调用进程返回一个描述符，该描述符可以用于以后的所有IO函数。涉及打开文件和设备的所有细节对客户进程是透明的。

三个函数，服务器进程想将一个描述符传送给另一个进程时，可以调用send\_fd或send\_err。等待接受描述符的进程调用recv\_fd

Open服务器进程：

使用文件描述符传送技术开发一个open服务器进程，一个由一个进程执行以打开一个或多个文件。服务器进程不是将文件内容送回调用进程，而是送回一个打开文件描述符。该服务器进程对任何类型的文件如设备或套接字都能起作用。客户进程和服务器进程使用IPC交换最小量的信息，从客户进程到服务器进程传送文件名和打开模式，而从服务器进程到客户进程返回描述符。文件内容不通过IPC交换。

通过域套接字可以实现一个全双工的管道。

第18章：终端IO

两种不同的工作模式

规范模式输入处理。以行为单位进行处理

非规范模式输入处理。输入字符不装配成行

有对终端进行操作的接口函数，还有获得和设置终端属性的方法。

还有终端的选项标志，此处有4个终端标志字段。每个有很多选项。可以通过与屏蔽字进行与运算就能得到

而且上面的标志都可以进行检查和更改

波特率函数：现在指“位/秒”可以进行设置

行控制函数：函数对输出进行传递和发送。

终端标识：大多数UNIX系统版本中，控制终端名字是/dev/tty，ctermid可以确定终端的名字

第19章：伪终端

伪终端是指对于一个应用程序它看上去像一个终端。但是事实上不是一个终端。

通常，一个进程打开伪终端主设备，然后调用fork，子进程简历建立一个新的会话，打开一个相应的伪终端从设备。将其文件描述符复制到标准输入输出错误，然后调用exec。伪终端从设备成为子进程的控制终端。

对于伪终端从设备上的用户进程来说，其标准输入输出错误都是终端设备，都可以用18章中的终端IO函数，无意义的除外

任何写到伪终端主设备的都会作为从设备的输入，反之亦然。

典型用途：

可以构造提供网络登录的服务器

窗口系统通常提供一个终端模拟器

Script程序

Expect程序

运行协同程序

观看长时间运行程序的输出

pty\_fork函数：用fork调用打开主设备和从设备，创建作为会话首进程的子进程并使其具有控制终端。

第20章：数据库函数库

本章介绍如何开发一个多用户数据库的C函数库，调用这个函数库提供的C语言函数，其他程序可以获取和存储数据库中的记录。

本章开发的函数库具有多用户并发机制，支持多个进程对数据库的并发更新，提供了并发控制。

接口：

db\_open 打开数据库。返回一个代表数据库的句柄（一个不透明指针），建立索引和数据文件

db\_close 关闭数据库。关闭索引和数据文件，并释放数据库使用过程中你分配到的所有用于内部缓冲区的存储空间。

db\_store 存入新纪录。提供键和数据，

db\_fetch 获取记录。通过指定键

db\_delete 删除记录。

db\_rewind 回滚到数据库第一条记录

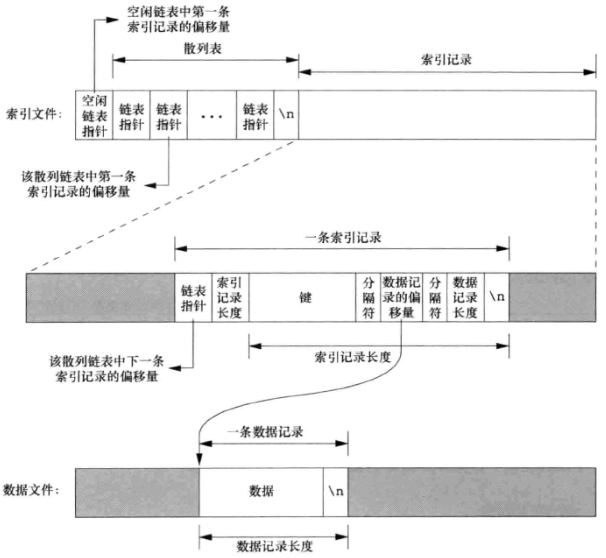
db\_nextrec 将参数指针复制到存储区域开始的内存位置，和db-rewind配合可以逐条记录的访问数据库

实现概述：

访问数据库的函数库通常使用两个文件来存储信息：一个索引文件和一个数据文件。

索引文件：包括实际的索引值（键）和一个指向数据文件中对应数据记录的指针。为提高按键查询的速度和效率，散列表和B+树是两种常用的技术。此处使用固定大小散列表，链表法解决散列冲突。键和索引以null结尾的字符串形成存储。不适用二进制存储，这样以空间换移植性

db\_store要求每个键只有一条对应的记录。数据库实现的基本结构：



索引文件由三部分构成：空闲链表指针、散列表、索引记录。给定一个键，要在数据库中寻找一条记录时，db\_fetch根据该键计算散列值，由此散列值可以确定一条散列链。沿着这条散列链，可以找到所有具有这一散列值的索引记录。当遇到一个索引记录的链表指针字段为0时，表示到达了散列链的末尾。

索引文件中键的顺序和数据文件中对应数据记录的顺序与调用db\_store的顺序一样。

使用固定大小散列表作为索引是为了当每个散列链都不太长时，这个方法能保证快速的访问。我们的目的是能够快速的查找任一键，而又不使用太复杂的数据结构。如B树或动态散列表。B树能够用已排序的键的顺序来遍历数据库，而动态散列表可以保证仅用两次磁盘存取就能找到数据记录。

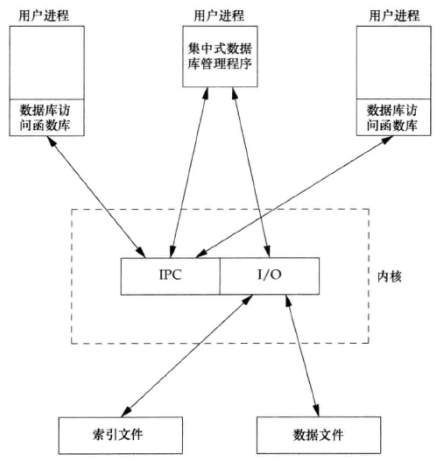
集中式与非集中式：

集中式：一个进程作为数据库管理者，所有数据库访问工作由它完成，其他进程通过IPC机制与此中心进程进行联系。

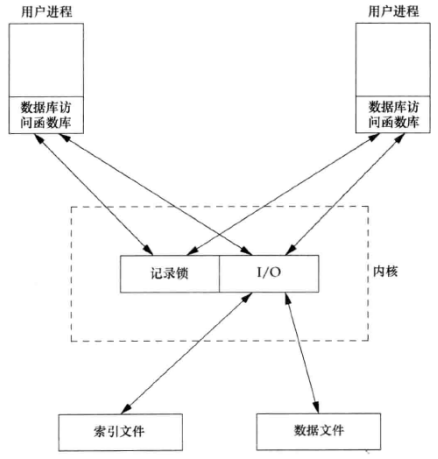
非集中式：每个库函数使用要求的并发控制，然后发起自己的IO函数调用

一般非集中式要比集中式快一点。

集中式：

.

非集中式：



集中式优点是能够根据需要来对操作模式进行调整。例如，可以通过中心进程给不同的进程赋予不同的优先级。另一个优点是恢复要比非集中式方法容易。集中式方法的另一个优点是，恢复要比非集中式方法容易。所以如果杀死了数据库进程，只需要该处查看以识别出需要解决的未完成事务，然后将数据库恢复到一致状态。

非集中式一般使用字节范围记录锁机制来实现并发控制。

并发：

对索引文件和数据文件两个文件的加锁

粗粒度锁：将这两个文件中的一个作为整个数据库的锁，并要求调用者在对数据库进行操作前必须获得这个锁。这种锁的问题在于它限制了并发，当一个进程向一条散列链中添加一条记录时，其他进程无法访问另一条散列链上的记录。其中db\_fetch和db\_nextrec要求具有读锁，db\_delete，db\_store和db\_open要求具有写锁。

细粒度锁：具有更高的并发性，一个读进程或写进程在操作一条记录前必须先获得此记录所在散列链的读锁或者写锁。一条散列链允许同时有多个读进程，但只能有一个写进程。 使用更精细的字节范围锁。