\*\*\*\*\*\*\*\*\*\*\*《朱有鹏老师嵌入式linux核心课程》 \*\*\*\*\*\*\*\*\*\*\*

《3.linux应用编程和网络编程-第1部分-3.1.linux中的文件IO》

--------------------------------------------------------

欢迎访问《朱老师物联网大讲堂》官方网站

www.zhulaoshi.org

以获取本课程更多信息

--------------------------------------------------------

第一部分、章节目录

3.1.1.应用编程框架介绍

3.1.2.文件操作的主要接口API

3.1.3.一个简单的文件读写实例

3.1.4.open函数的flag详解1

3.1.5.open函数的flag详解2

3.1.6.文件读写的一些细节

3.1.7.linux系统如何管理文件

3.1.8.lseek详解

3.1.9.多次打开同一文件与O\_APPEND

3.1.10.文件共享的实现方式

3.1.11.文件描述符的复制1

3.1.12.文件描述符的复制2

3.1.13.fcntl函数介绍

3.1.14.标准IO库介绍

第二部分、章节介绍

3.1.1.应用编程框架介绍

本节讲述linux应用编程课程的整体设计框架和设计思路，然后讲了本课程文件IO的含义和课程内容。

3.1.2.文件操作的主要接口API

本节首先介绍操作系统API的概念，然后介绍文件IO的概念以及常用的API，最后讲了文件描述符。

3.1.3.一个简单的文件读写实例

本节从零开始手写一个简单的读写文件实例程序，顺带介绍使用man手册获取函数原型的技巧，实现向文件中写入内容和从文件中读取内容

3.1.4.open函数的flag详解1

本节讲解open函数的flag参数，主要是读写权限flag和判断文件是否存在的flag，并且同时以实例程序来验证理论。

3.1.5.open函数的flag详解2

本节接上节继续讲解open的flag，主要是创建文件标志、非阻塞标志和同步标志，并且顺便讲了阻塞和非阻塞的概念。

3.1.6.文件读写的一些细节

本节讲解文件读写中一些细节，譬如errno的设置和perror函数、read/write函数中的count参数，最后简单探讨了io效率问题，引出了标准IO

3.1.7.linux系统如何管理文件

本节简单探讨linux系统对文件的管理方法，重点讲了inode、vnode等概念，文件在硬盘上静态存储和在内存中被打开时的映像方式等。

3.1.8.lseek详解

本节讲解lseek函数，该函数用来移动文件的操作指针。利用这一特性我们可以计算文件长度、生成空洞文件等。

3.1.9.多次打开同一文件与O\_APPEND

本节探讨一个文件被多次打开时的情况，这种情况一般在文件共享中发生。重点讲了O\_APPEND标志在其中的作用。

3.1.10.文件共享的实现方式

本节正式提出文件共享的概念，并且分析了三种最长出现的文件共享的方式，以及他们的区别和联系

3.1.11.文件描述符的复制1

本节引入文件描述符的复制，重点讲了dup函数，并且通过实例说明如何使用dup函数复制文件描述符。

3.1.12.文件描述符的复制2

本节分析dup函数的缺陷，并且引入dup2函数来解决。

3.1.13.fcntl函数介绍

本节介绍文件管理函数fcntl，该函数功能很强大，以后会经常用到，这里只是做初步的介绍和示例其基本用法。

3.1.14.标准IO库介绍

本节引入标准IO库函数，并且使用常用标准IO库函数写一个简单的文件操作示例程序。

第三部分、随堂记录

3.1.1.应用编程框架介绍

3.1.1.1、什么是应用编程

(1)整个嵌入式linux核心课程包括5个点，按照学习顺序依次是：裸机、C高级、uboot和系统移植、linux应用编程和网络编程、驱动。

(2)典型的嵌入式产品就是基于嵌入式linux操作系统来工作的。典型的嵌入式产品的研发过程就是；第一步让linux系统在硬件上跑起来（系统移植工作），第二步基于linux系统来开发应用程序实现产品功能。

(3)基于linux去做应用编程，其实就是通过调用linux的系统API来实现应用需要完成的任务。

3.1.1.2、本课程大纲规划

3.1.1.3、课程设计思路

(1)通过本课程9个小课程的学习，学会如何使用linux系统提供的API（和C库函数）来实现一定的功能，通过学习对应用层的编程有所掌握来配合后面驱动的学习。

(2)如果希望深入学习linux应用尤其是网络编程知识，可以去看一些专门介绍这一块的书。

3.1.1.4、什么是文件IO

(1)IO就是input/output，输入/输出。文件IO的意思就是读写文件。

3.1.2.文件操作的主要接口API

3.1.2.1、什么是操作系统API

(1)API是一些函数，这些函数是由linux系统提供支持的，由应用层程序来使用。

(2)应用层程序通过调用API来调用操作系统中的各种功能，来干活。

(3)学习一个操作系统，其实就是学习使用这个操作系统的API。

(1)今天我们要使用linux系统来读写文件，手段就是学习linux系统API中和文件IO有关的几个。

3.1.2.2、linux常用文件IO接口

(1)open、close、write、read、lseek

3.1.2.3、文件操作的一般步骤

(1)在linux系统中要操作一个文件，一般是先open打开一个文件，得到一个文件描述符，然后对文件进行读写操作（或其他操作），最后close关闭文件即可

(2)强调一点：我们对文件进行操作时，一定要先打开文件，打开成功后才能去操作（如果打开本身失败，后面就不用操作了）；最后读写完成之后一定要close关闭文件，否则可能会造成文件损坏。

(3)文件平时是存在块设备中的文件系统中的，我们把这种文件叫静态文件。当我们去open打开一个文件时，linux内核做的操作包括：内核在进程中建立了一个打开文件的数据结构，记录下我们打开的这个文件；内核在内存中申请一段内存，并且将静态文件的内容从块设备中读取到内存中特定地址管理存放（叫动态文件）。

(4)打开文件后，以后对这个文件的读写操作，都是针对内存中这一份动态文件的，而并不是针对静态文件的。当我们对动态文件进行读写后，此时内存中的动态文件和块设备中的静态文件就不同步了，当我们close关闭动态文件时，close内部内核将内存中的动态文件的内容去更新（同步）块设备中的静态文件。

(5)常见的一些现象：

第一个：打开一个大文件时比较慢

第二个：我们写了一半的文件，如果没有点保存直接关机/断电，重启后文件内容丢失。

(6)为什么要这么设计？

以为块设备本身有读写限制（回忆NnadFlash、SD等块设备的读写特征），本身对块设备进行操作非常不灵活。而内存可以按字节为单位来操作，而且可以随机操作（内存就叫RAM，random），很灵活。所以内核设计文件操作时就这么设计了。

3.1.2.4、重要概念：文件描述符

(1)文件描述符其实实质是一个数字，这个数字在一个进程中表示一个特定的含义，当我们open打开一个文件时，操作系统在内存中构建了一些数据结构来表示这个动态文件，然后返回给应用程序一个数字作为文件描述符，这个数字就和我们内存中维护这个动态文件的这些数据结构挂钩绑定上了，以后我们应用程序如果要操作这一个动态文件，只需要用这个文件描述符进行区分。

(2)一句话讲清楚文件描述符：文件描述符就是用来区分一个程序打开的多个文件的。

(3)文件描述符的作用域就是当前进程，出了当前进程这个文件描述符就没有意义了

3.1.3.一个简单的文件读写实例

3.1.3.1、打开文件与关闭文件

(1)linux中的文件描述符fd的合法范围是0或者一个正正数，不可能是一个负数。

(2)open返回的fd程序必须记录好，以后向这个文件的所有操作都要靠这个fd去对应这个文件，最后关闭文件时也需要fd去指定关闭这个文件。如果在我们关闭文件前fd丢掉了那就惨了，这个文件没法关闭了也没法读写了。

3.1.3.2、实时查man手册

(1)当我们写应用程序时，很多API原型都不可能记得，所以要实时查询，用man手册

(2)man 1 xx查linux shell命令，man 2 xxx查API， man 3 xxx查库函数

3.1.3.3、读取文件内容

(1)ssize\_t read(int fd, void \*buf, size\_t count);

fd表示要读取哪个文件，fd一般由前面的open返回得到

buf是应用程序自己提供的一段内存缓冲区，用来存储读出的内容

count是我们要读取的字节数

返回值ssize\_t类型是linux内核用typedef重定义的一个类型（其实就是int），返回值表示成功读取的字节数。

3.1.3.4、向文件中写入

(1)写入用write系统调用，write的原型和理解方法和read相似

(2)注意const在buf前面的作用，结合C语言高级专题中的输入型参数和输出型参数一节来理解。

(3)注意buf的指针类型为void，结合C语言高级专题中void类型含义的讲解

(4)刚才先写入12字节，然后读出结果读出是0（但是读出成功了），这个问题的答案后面章节会讲，大家先思考一下。

3.1.4.open函数的flag详解1

3.1.4.1、读写权限：O\_RDONLY O\_WRONLY O\_RDWR

(1)linux中文件有读写权限，我们在open打开文件时也可以附带一定的权限说明（譬如O\_RDONLY就表示以只读方式打开，O\_WRONLY表示以只写方式打开，O\_RDWR表示以可读可写方式打开）

(2)当我们附带了权限后，打开的文件就只能按照这种权限来操作。

3.1.4.2、打开存在并有内容的文件时：O\_APPEND、O\_TRUNC

(1)思考一个问题：当我们打开一个已经存在并且内部有内容的文件时会怎么样？

可能结果1：新内容会替代原来的内容（原来的内容就不见了，丢了）

可能结果2：新内容添加在前面，原来的内容继续在后面

可能结果3：新内容附加在后面，原来的内容还在前面

可能结果4：不读不写的时候，原来的文件中的内容保持不变

(2)O\_TRUNC属性去打开文件时，如果这个文件中本来是有内容的，则原来的内容会被丢弃。这就对应上面的结果1

(3)O\_APPEND属性去打开文件时，如果这个文件中本来是有内容的，则新写入的内容会接续到原来内容的后面，对应结果3

(4)默认不使用O\_APPEND和O\_TRUNC属性时就是结果4

(5)如果O\_APPEND和O\_TRUNC同时出现会怎么样？

3.1.4.3、exit、\_exit、\_Exit退出进程

(1)当我们程序在前面步骤操作失败导致后面的操作都没有可能进行下去时，应该在前面的错误监测中结束整个程序，不应该继续让程序运行下去了。

(2)我们如何退出程序？

第一种；在main用return，一般原则是程序正常终止return 0，如果程序异常终止则return -1。

第一种：正式终止进程（程序）应该使用exit或者\_exit或者\_Exit之一。

3.1.5.open函数的flag详解2

3.1.5.1、打开不存在的文件时：O\_CREAT、O\_EXCL

(1)思考：当我们去打开一个并不存在的文件时会怎样？当我们open打开一个文件时如果这个文件名不存在则会打开文件错误。

(2)vi或者windows下的notepad++，都可以直接打开一个尚未存在的文件。

(3)open的flag O\_CREAT就是为了应对这种打开一个并不存在的文件的。O\_CREAT就表示我们当前打开的文件并不存在，我们是要去创建并且打开它。

(4)思考：当我们open使用了O\_CREAT，但是文件已经存在的情况下会怎样？经过实验验证发现结果是报错。

(5)结论：open中加入O\_CREAT后，不管原来这个文件存在与否都能打开成功，如果原来这个文件不存在则创建一个空的新文件，如果原来这个文件存在则会重新创建这个文件，原来的内容会被消除掉（有点类似于先删除原来的文件再创建一个新的）

(6)这样可能带来一个问题？我们本来是想去创建一个新文件的，但是把文件名搞错了弄成了一个老文件名，结果老文件就被意外修改了。我们希望的效果是：如果我CREAT要创建的是一个已经存在的名字的文件，则给我报错，不要去创建。

(7)这个效果就要靠O\_EXCL标志和O\_CREAT标志来结合使用。当这连个标志一起的时候，则没有文件时创建文件，有这个文件时会报错提醒我们。

(8)open函数在使用O\_CREAT标志去创建文件时，可以使用第三个参数mode来指定要创建的文件的权限。mode使用4个数字来指定权限的，其中后面三个很重要，对应我们要创建的这个文件的权限标志。譬如一般创建一个可读可写不可执行的文件就用0666

3.1.5.2、O\_NONBLOCK

(1)阻塞与非阻塞。如果一个函数是阻塞式的，则我们调用这个函数时当前进程有可能被卡住（阻塞住，实质是这个函数内部要完成的事情条件不具备，当前没法做，要等待条件成熟），函数被阻塞住了就不能立刻返回；如果一个函数是非阻塞式的那么我们调用这个函数后一定会立即返回，但是函数有没有完成任务不一定。

(2)阻塞和非阻塞是两种不同的设计思路，并没有好坏。总的来说，阻塞式的结果有保障但是时间没保障；非阻塞式的时间有保障但是结果没保障。

(3)操作系统提供的API和由API封装而成的库函数，有很多本身就是被设计为阻塞式或者非阻塞式的，所以我们应用程度调用这些函数的时候心里得非常清楚。

(4)我们打开一个文件默认就是阻塞式的，如果你希望以非阻塞的方式打开文件，则flag中要加O\_NONBLOCK标志。

(2)只用于设备文件，而不用于普通文件。

3.1.5.3、O\_SYNC

(1)write阻塞等待底层完成写入才返回到应用层。

(2)无O\_SYNC时write只是将内容写入底层缓冲区即可返回，然后底层（操作系统中负责实现open、write这些操作的那些代码，也包含OS中读写硬盘等底层硬件的代码）在合适的时候会将buf中的内容一次性的同步到硬盘中。这种设计是为了提升硬件操作的性能和销量，提升硬件寿命；但是有时候我们希望硬件不好等待，直接将我们的内容写入硬盘中，这时候就可以用O\_SYNC标志。

3.1.6.文件读写的一些细节

3.1.6.1、errno和perror

(1)errno就是error number，意思就是错误号码。linux系统中对各种常见错误做了个编号，当函数执行错误时，函数会返回一个特定的errno编号来告诉我们这个函数到底哪里错了。

(2)errno是由OS来维护的一个全局变量，任何OS内部函数都可以通过设置errno来告诉上层调用者究竟刚才发生了一个什么错误。

(3)errno本身实质是一个int类型的数字，每个数字编号对应一种错误。当我们只看errno时只能得到一个错误编号数字（譬如-37），不适应于人看。

(4)linux系统提供了一个函数perror（意思print error），perror函数内部会读取errno并且将这个不好认的数字直接给转成对应的错误信息字符串，然后print打印出来。

3.1.6.2、read和write的count

(1)count和返回值的关系。count参数表示我们想要写或者读的字节数，返回值表示实际完成的要写或者读的字节数。实现的有可能等于想要读写的，也有可能小于（说明没完成任务）

(2)count再和阻塞非阻塞结合起来，就会更加复杂。如果一个函数是阻塞式的，则我们要读取30个，结果暂时只有20个时就会被阻塞住，等待剩余的10个可以读。

(3)有时候我们写正式程序时，我们要读取或者写入的是一个很庞大的文件（譬如文件有2MB），我们不可能把count设置为2\*1024\*1024，而应该去把count设置为一个合适的数字（譬如2048、4096），然后通过多次读取来实现全部读完。

3.1.6.3、文件IO效率和标准IO

(1)文件IO就指的是我们当前在讲的open、close、write、read等API函数构成的一套用来读写文件的体系，这套体系可以很好的完成文件读写，但是效率并不是最高的。

(2)应用层C语言库函数提供了一些用来做文件读写的函数列表，叫标准IO。标准IO由一系列的C库函数构成（fopen、fclose、fwrite、fread），这些标准IO函数其实是由文件IO封装而来的（fopen内部其实调用的还是open，fwrite内部还是通过write来完成文件写入的）。标准IO加了封装之后主要是为了在应用层添加一个缓冲机制，这样我们通过fwrite写入的内容不是直接进入内核中的buf，而是先进入应用层标准IO库自己维护的buf中，然后标准IO库自己根据操作系统单次write的最佳count来选择好的时机来完成write到内核中的buf（内核中的buf再根据硬盘的特性来选择好的实际去最终写入硬盘中）。

3.1.7.linux系统如何管理文件

3.1.7.1、硬盘中的静态文件和inode（i节点）

(1)文件平时都在存放在硬盘中的，硬盘中存储的文件以一种固定的形式存放的，我们叫静态文件。

(2)一块硬盘中可以分为两大区域：一个是硬盘内容管理表项，另一个是真正存储内容的区域。操作系统访问硬盘时是先去读取硬盘内容管理表，从中找到我们要访问的那个文件的扇区级别的信息，然后再用这个信息去查询真正存储内容的区域，最后得到我们要的文件。

(3)操作系统最初拿到的信息是文件名，最终得到的是文件内容。第一步就是去查询硬盘内容管理表，这个管理表中以文件为单位记录了各个文件的各种信息，每一个文件有一个信息列表（我们叫inode，i节点，其实质是一个结构体，这个结构体有很多元素，每个元素记录了这个文件的一些信息，其中就包括文件名、文件在硬盘上对应的扇区号、块号那些东西·····）

强调：硬盘管理的时候是以文件为单位的，每个文件一个inode，每个inode有一个数字编号，对应一个结构体，结构体中记录了各种信息。

(4)联系平时实践，大家格式化硬盘（U盘）时发现有：快速格式化和底层格式化。快速格式化非常快，格式化一个32GB的U盘只要1秒钟，普通格式化格式化速度慢。这两个的差异？其实快速格式化就是只删除了U盘中的硬盘内容管理表（其实就是inode），真正存储的内容没有动。这种格式化的内容是有可能被找回的。

3.1.7.2、内存中被打开的文件和vnode（v节点）

(1)一个程序的运行就是一个进程，我们在程序中打开的文件就属于某个进程。每个进程都有一个数据结构用来记录这个进程的所有信息（叫进程信息表），表中有一个指针会指向一个文件管理表，文件管理表中记录了当前进程打开的所有文件及其相关信息。文件管理表中用来索引各个打开的文件的index就是文件描述符fd，我们最终找到的就是一个已经被打开的文件的管理结构体vnode

(2)一个vnode中就记录了一个被打开的文件的各种信息，而且我们只要知道这个文件的fd，就可以很容易的找到这个文件的vnode进而对这个文件进行各种操作。

3.1.7.3、文件与流的概念

(1)流（stream）对应自然界的水流。文件操作中，文件类似是一个大包裹，里面装了一堆字符，但是文件被读出/写入时都只能一个字符一个字符的进行，而不能一股脑儿的读写，那么一个文件中N多的个字符被挨个一次读出/写入时，这些字符就构成了一个字符流。

(2)流这个概念是动态的，不是静态的。

(3)编程中提到流这个概念，一般都是IO相关的。所以经常叫IO流。文件操作时就构成了一个IO流。

3.1.8.lseek详解

3.1.8.1、lseek函数介绍

(1)文件指针：当我们要对一个文件进行读写时，一定需要先打开这个文件，所以我们读写的所有文件都是动态文件。动态文件在内存中的形态就是文件流的形式。

(2)文件流很长，里面有很多个字节。那我们当前正在操作的是哪个位置？GUI模式下的软件用光标来标识这个当前正在操作的位置，这是给人看的。

(3)在动态文件中，我们会通过文件指针来表征这个正在操作的位置。所谓文件指针，就是我们文件管理表这个结构体里面的一个指针。所以文件指针其实是vnode中的一个元素。这个指针表示当前我们正在操作文件流的哪个位置。这个指针不能被直接访问，linux系统用lseek函数来访问这个文件指针。

(4)当我们打开一个空文件时，默认情况下文件指针指向文件流的开始。所以这时候去write时写入就是从文件开头开始的。write和read函数本身自带移动文件指针的功能，所以当我write了n个字节后，文件指针会自动向后移动n位。如果需要人为的随意更改文件指针，那就只能通过lseek函数了

(5)read和write函数都是从当前文件指针处开始操作的，所以当我们用lseek显式的将文件指针移动后，那么再去read/write时就是从移动过后的位置开始的。

(6)回顾前面一节中我们从空文件，先write写了12字节，然后read时是空的（但是此时我们打开文件后发现12字节确实写进来了）。

3.1.8.2、用lseek计算文件长度

(1)linux中并没有一个函数可以直接返回一个文件的长度。但是我们做项目时经常会需要知道一个文件的长度，怎么办？自己利用lseek来写一个函数得到文件长度即可。

3.1.8.3、用lseek构建空洞文件

(1)空洞文件就是这个文件中有一段是空的。

(2)普通文件中间是不能有空的，因为我们write时文件指针是依次从前到后去移动的，不可能绕过前面直接到后面。

(3)我们打开一个文件后，用lseek往后跳过一段，再write写入一段，就会构成一个空洞文件。

(4)空洞文件方法对多线程共同操作文件是及其有用的。有时候我们创建一个很大的文件，如果从头开始依次构建时间很长。有一种思路就是将文件分为多段，然后多线程来操作每个线程负责其中一段的写入。

3.1.9.多次打开同一文件与O\_APPEND

3.1.9.1、重复打开同一文件读取

(1)一个进程中两次打开同一个文件，然后分别读取，看结果会怎么样

(2)结果无非2种情况：一种是fd1和fd2分别读，第二种是接续读。经过实验验证，证明了结果是fd1和fd2分别读。

(3)分别读说明：我们使用open两次打开同一个文件时，fd1和fd2所对应的文件指针是不同的2个独立的指针。文件指针是包含在动态文件的文件管理表中的，所以可以看出linux系统的进程中不同fd对应的是不同的独立的文件管理表。

3.1.9.2、重复打开同一文件写入

(1)一个进程中2个打开同一个文件，得到fd1和fd2.然后看是分别写还是接续写？

(2)正常情况下我们有时候需要分别写，有时候又需要接续写，所以这两种本身是没有好坏之分的。关键看用户需求

(3)默认情况下应该是：分别写（实验验证过的）

3.1.9.3、加O\_APPEND解决覆盖问题

(1)有时候我们希望接续写而不是分别写？办法就是在open时加O\_APPEND标志即可

3.1.9.4、O\_APPEND的实现原理和其原子操作性说明

(1)O\_APPEND为什么能够将分别写改为接续写？关键的核心的东西是文件指针。分别写的内部原理就是2个fd拥有不同的文件指针，并且彼此只考虑自己的位移。但是O\_APPEND标志可以让write和read函数内部多做一件事情，就是移动自己的文件指针的同时也去把别人的文件指针同时移动。（也就是说即使加了O\_APPEND，fd1和fd2还是各自拥有一个独立的文件指针，但是这两个文件指针关联起来了，一个动了会通知另一个跟着动）

(2)O\_APPEND对文件指针的影响，对文件的读写是原子的。

(3)原子操作的含义是：整个操作一旦开始是不会被打断的，必须直到操作结束其他代码才能得以调度运行，这就叫原子操作。每种操作系统中都有一些机制来实现原子操作，以保证那些需要原子操作的任务可以运行。

3.1.10.文件共享的实现方式

3.1.10.1、什么是文件共享

(1)文件共享就是同一个文件（同一个文件指的是同一个inode，同一个pathname）被多个独立的读写体（几乎可以理解为多个文件描述符）去同时（一个打开尚未关闭的同时另一个去操作）操作。

(2)文件共享的意义有很多：譬如我们可以通过文件共享来实现多线程同时操作同一个大文件，以减少文件读写时间，提升效率。

3.1.10.2、文件共享的3种实现方式

(1)文件共享的核心就是怎么弄出来多个文件描述符指向同一个文件。

(2)常见的有3种文件共享的情况：第一种是同一个进程中多次使用open打开同一个文件，第二种是在不同进程中去分别使用open打开同一个文件（这时候因为两个fd在不同的进程中，所以两个fd的数字可以相同也可以不同），第三种情况是后面要学的，linux系统提供了dup和dup2两个API来让进程复制文件描述符。

(3)我们分析文件共享时的核心关注点在于：分别写/读还是接续写/读

3.1.10.3、再论文件描述符

(1)文件描述符的本质是一个数字，这个数字本质上是进程表中文件描述符表的一个表项，进程通过文件描述符作为index去索引查表得到文件表指针，再间接访问得到这个文件对应的文件表。

(2)文件描述符这个数字是open系统调用内部由操作系统自动分配的，操作系统分配这个fd时也不是随意分配，也是遵照一定的规律的，我们现在就要研究这个规律。

(3)操作系统规定，fd从0开始依次增加。fd也是有最大限制的，在linux的早期版本中（0.11）fd最大是20，所以当时一个进程最多允许打开20个文件。linux中文件描述符表是个数组（不是链表），所以这个文件描述符表其实就是一个数组，fd是index，文件表指针是value

(4)当我们去open时，内核会从文件描述符表中挑选一个最小的未被使用的数字给我们返回。也就是说如果之前fd已经占满了0-9，那么我们下次open得到的一定是10.（但是如果上一个fd得到的是9，下一个不一定是10，这是因为可能前面更小的一个fd已经被close释放掉了）

(5)fd中0、1、2已经默认被系统占用了，因此用户进程得到的最小的fd就是3了。

(6)linux内核占用了0、1、2这三个fd是有用的，当我们运行一个程序得到一个进程时，内部就默认已经打开了3个文件，这三个文件对应的fd就是0、1、2。这三个文件分别叫stdin、stdout、stderr。也就是标准输入、标准输出、标准错误。

(7)标准输入一般对应的是键盘（可以理解为：0这个fd对应的是键盘的设备文件），标准输出一般是LCD显示器（可以理解为：1对应LCD的设备文件）

(8)printf函数其实就是默认输出到标准输出stdout上了。stdio中还有一个函数叫fpirntf，这个函数就可以指定输出到哪个文件描述符中。

3.1.11.文件描述符的复制1

3.1.11.1、dup和dup2函数介绍

3.1.11.2、使用dup进行文件描述符复制

(1)dup系统调用对fd进行复制，会返回一个新的文件描述符（譬如原来的fd是3，返回的就是4）

(2)dup系统调用有一个特点，就是自己不能指定复制后得到的fd的数字是多少，而是由操作系统内部自动分配的，分配的原则遵守fd分配的原则。

(3)dup返回的fd和原来的oldfd都指向oldfd打开的那个动态文件，操作这两个fd实际操作的都是oldfd打开的那个文件。实际上构成了文件共享。

(4)dup返回的fd和原来的oldfd同时向一个文件写入时，结果是分别写还是接续写？

3.1.11.3、使用dup的缺陷分析

(1)dup并不能指定分配的新的文件描述符的数字，dup2系统调用修复了这个缺陷，所以平时项目中实际使用时根据具体情况来决定用dup还是dup2.

3.1.11.4、练习

(1)之前课程讲过0、1、2这三个fd被标准输入、输出、错误通道占用。而且我们可以关闭这三个

(2)我们可以close(1)关闭标准输出，关闭后我们printf输出到标准输出的内容就看不到了

(3)然后我们可以使用dup重新分配得到1这个fd，这时候就把oldfd打开的这个文件和我们1这个标准输出通道给绑定起来了。这就叫标准输出的重定位。

(4)可以看出，我们可以使用close和dup配合进行文件的重定位。

3.1.12.文件描述符的复制2

3.1.12.1、使用dup2进行文件描述符复制

(1)dup2和dup的作用是一样的，都是复制一个新的文件描述符。但是dup2允许用户指定新的文件描述符的数字。

(2)使用方法看man手册函数原型即可。

3.1.12.2、dup2共享文件交叉写入测试

(1)dup2复制的文件描述符，和原来的文件描述符虽然数字不一样，但是这连个指向同一个打开的文件

(2)交叉写入的时候，结果是接续写（实验证明的）。

3.1.12.3、命令行中重定位命令 >

(1)linux中的shell命令执行后，打印结果都是默认进入stdout的（本质上是因为这些命令譬如ls、pwd等都是调用printf进行打印的），所以我们可以在linux的终端shell中直接看到命令执行的结果。

(2)能否想办法把ls、pwd等命令的输出给重定位到一个文件中（譬如2.txt）去，实际上linux终端支持一个重定位的符号>很简单可以做到这点。

(3)这个>的实现原理，其实就是利用open+close+dup，open打开一个文件2.txt，然后close关闭stdout，然后dup将1和2.txt文件关联起来即可。

3.1.13.fcntl函数介绍

3.1.13.1、fcntl的原型和作用

(1)fcntl函数是一个多功能文件管理的工具箱，接收2个参数+1个变参。第一个参数是fd表示要操作哪个文件，第二个参数是cmd表示要进行哪个命令操作。变参是用来传递参数的，要配合cmd来使用。

(2)cmd的样子类似于F\_XXX，不同的cmd具有不同的功能。学习时没必要去把所有的cmd的含义都弄清楚（也记不住），只需要弄明白一个作为案例，搞清楚它怎么看怎么用就行了，其他的是类似的。其他的当我们在使用中碰到了一个fcntl的不认识的cmd时再去查man手册即可。

3.1.13.2、fcntl的常用cmd

(1)F\_DUPFD这个cmd的作用是复制文件描述符（作用类似于dup和dup2），这个命令的功能是从可用的fd数字列表中找一个比arg大或者和arg一样大的数字作为oldfd的一个复制的fd，和dup2有点像但是不同。dup2返回的就是我们指定的那个newfd否则就会出错，但是F\_DUPFD命令返回的是>=arg的最小的那一个数字。

3.1.13.3、使用fcntl模拟dup2

3.1.14.标准IO库介绍

3.1.14.1、标准IO和文件IO有什么区别

(1)看起来使用时都是函数，但是：标准IO是C库函数，而文件IO是linux系统的API

(2)C语言库函数是由API封装而来的。库函数内部也是通过调用API来完成操作的，但是库函数因为多了一层封装，所以比API要更加好用一些。

(3)库函数比API还有一个优势就是：API在不同的操作系统之间是不能通用的，但是C库函数在不同操作系统中几乎是一样的。所以C库函数具有可移植性而API不具有可移植性。

(4)性能上和易用性上看，C库函数一般要好一些。譬如IO，文件IO是不带缓存的，而标准IO是带缓存的，因此标准IO比文件IO性能要更高。

3.1.14.2、常用标准IO函数介绍

(1)常见的标准IO库函数有：fopen、fclose、fwrite、fread、ffulsh、fseek

3.1.14.3、一个简单的标准IO读写文件实例

\*\*\*\*\*\*\*\*\*\*\*《朱有鹏老师嵌入式linux核心课程》 \*\*\*\*\*\*\*\*\*\*\*

《3.linux应用编程和网络编程-第2部分-3.2.文件属性》

--------------------------------------------------------

本课程由朱老师物联网大讲堂推出并提供技术支持，课件可打包下载

网盘地址：http://yunpan.cn/cjVy3RAgfDufK 访问密码 4ad7

--------------------------------------------------------

第一部分、章节目录

3.2.1.linux中各种文件类型

3.2.2.常用文件属性获取

3.2.3.stat函数的应用案例

3.2.4.文件权限管理1

3.2.5.文件权限管理2

3.2.6.读取目录文件

第二部分、章节介绍

3.2.1.linux中各种文件类型

本节介绍linux中各种文件类型，其中重点讲解了文本文件和二进制文件这两种普通文件。

3.2.2.常用文件属性获取

本节介绍stat函数族及其如何获取文件属性结构体，并且用一个简单实例获取了一些文件属性作为验证。

3.2.3.stat函数的应用案例

本节使用stat函数获取文件文件属性后，使用linux系统的宏来获取文件类型和文件读写权限信息。

3.2.4.文件权限管理1

本节重点讲解文件操作时权限检查和判定的规则，然后引入了access函数来检测文件是否具有某种可访问权限

3.2.5.文件权限管理2

本节介绍chmod接口来更改文件权限、chown接口来更改文件属主、umask接口来修改系统默认权限掩码

3.2.6.读取目录文件

本节为实战课，写代码实现读取文件目录，并且将目录中的文件属性和文件名列出来，以后的很多项目中都会用到这个功能。

第三部分、随堂记录

3.2.1.linux中各种文件类型

3.2.1.1、普通文件（- regular file）

(1)文本文件。文件中的内容是由文本构成的，文本指的是ASCII码字符。文件里的内容本质上都是数字（不管什么文件内容本质上都是数字，因为计算机中本身就只有1和0），而文本文件中的数字本身应该被理解为这个数字对应的ASCII码。常见的.c文件, .h文件 .txt文件等都是文本文件。文本文件的好处就是可以被人轻松读懂和编写。所以说文本文件天生就是为人类发明的。

(2)二进制文件。二进制文件中存储的本质上也是数字，只不过这些数字并不是文字的编码数字，而是就是真正的数字。常见的可执行程序文件（gcc编译生成的a.out，arm-linux-gcc编译连接生成的.bin）都是二进制文件。

(3)对比：从本质上来看（就是刨除文件属性和内容的理解）文本文件和二进制文件并没有任何区别。都是一个文件里面存放了数字。区别是理解方式不同，如果把这些数字就当作数字处理则就是二进制文件，如果把这些数字按照某种编码格式去解码成文本字符，则就是文本文件。

(4)我们如何知道一个文件是文件文件还是二进制文件？在linux系统层面是不区分这两个的（譬如之前学过的open、read、write等方法操作文件文件和二进制文件时一点区别都没有），所以我们无法从文件本身准确知道文件属于哪种，我们只能本来就知道这个文件的类型然后用这种类型的用法去用他。有时候会用一些后缀名来人为的标记文件的类型。

(5)使用文本文件时，常规用法就是用文本文件编辑器去打开它、编辑它。常见的文本文件编辑器如vim、gedit、notepad++、SourceInsight等，我们用这些文本文件编辑器去打开文件的时候，编辑器会read读出文件二进制数字内容，然后按照编码格式去解码将其还原成文字展现给我们。如果用文本文件编辑器去打开一个二进制文件会如何？这时候编辑器就以为这个二进制文件还是文本文件然后试图去将其解码成文字，但是解码过程很多数字并不对应有意义的文字所以成了乱码。

(6)反过来用二进制阅读工具去读取文本文件会怎么样？得出的就是文本文字所对应的二进制的编码。

3.2.1.2、目录文件（d directory）

(1)目录就是文件夹，文件夹在linux中也是一种文件，不过是特殊文件。用vi打开一个文件夹就能看到，文件夹其实也是一种特殊文件，里面存的内容包括这个文件的路径，还有文件夹里面的文件列表。

(2)但是文件夹这种文件比较特殊，本身并不适合用普通的方式来读写。linux中是使用特殊的一些API来专门读写文件夹的。

3.2.1.3、字符设备文件（c character）

3.2.1.4、块设备文件（b block）

(1)设备文件对应的是硬件设备，也就是说这个文件虽然在文件系统中存在，但是并不是真正存在于硬盘上的一个文件，而是文件系统虚拟制造出来的（叫虚拟文件系统，如/dev /sys /proc等）

(2)虚拟文件系统中的文件大多数不能或者说不用直接读写的，而是用一些特殊的API产生或者使用的，具体在驱动阶段会详解。

3.2.1.5、管道文件（p pipe）

3.2.1.6、套接字文件（s socket）

3.2.1.7、符号链接文件（l link）

3.2.2.常用文件属性获取

3.2.2.1、stat、fstat、lstat函数简介

(1)每个文件中都附带了这个文件的一些属性（属性信息是存在于文件本身中的，但是它不像文件的内容一样可以被vi打开看到，属性信息只能被专用的API打开看到）

(2)文件属性信息查看的API有三个：stat、fstat、lstat，三个作用一样，参数不同，细节略有不同。

(3)linux命令行下还可以去用stat命令去查看文件属性信息，实际上stat命令内部就是使用stat系统调用来实现的。

(4)stat这个API的作用就是让内核将我们要查找属性的文件的属性信息结构体的值放入我们传递给stat函数的buf中，当stat这个API调用从内核返回的时候buf中就被填充了文件的正确的属性信息，然后我们通过查看buf这种结构体变量的元素就可以得知这个文件的各种属性了。

(5)fstat和stat的区别是：stat是从文件名出发得到文件属性信息结构体，而fstat是从一个已经打开的文件fd出发得到一个文件的属性信息。所以用的时候如果文件没有打开（我们并不想打开文件操作而只是希望得到文件属性）那就用stat，如果文件已经被打开了然后要属性那就用fstat效率会更高（stat是从磁盘去读取文件的，而fstat是从内存读取动态文件的）。

(6)lstat和stat/fstat的差别在于：对于符号链接文件，stat和fstat查阅的是符号链接文件指向的文件的属性，而lstat查阅的是符号链接文件本身的属性。

3.2.2.2、struct stat结构体简介

(1)struct stat是内核定义的一个结构体，在<sys/stat.h>中声明，所以我们可以用。这个结构体中的所有元素加起来就是我们的文件属性信息。

3.2.2.3、写个程序来查看一些常见属性信息

3.2.3.stat函数的应用案例

3.2.3.1、用代码判断文件类型

(1)文件类型就是-、d、l····

(2)文件属性中的文件类型标志在struct stat结构体的mode\_t st\_mode元素中，这个元素其实是一个按位来定义的一个位标志（有点类似于ARM CPU的CPSR寄存器的模式位定义）。这个东西有很多个标志位共同构成，记录了很多信息，如果要查找时按位&操作就知道结果了，但是因为这些位定义不容易记住，因此linux系统给大家事先定义好了很多宏来进行相应操作。

(3)譬如S\_ISREG宏返回值是1表示这个文件是一个普通文件，如果文件不是普通文件则返回值是0.

3.2.3.2、用代码判断文件权限设置

(1)st\_mode中除了记录了文件类型之外，还记录了一个重要信息：文件权限。

(2)linux并没有给文件权限测试提供宏操作，而只是提供了位掩码，所以我们只能用位掩码来自己判断是否具有相应权限。

3.2.4.文件权限管理1

3.2.4.1、st\_mode中记录的文件权限位

(1)st\_mode本质上是一个32位的数（类型就是unsinged int），这个数里的每一个位表示一个含义。

(2)文件类型和文件的权限都记录在st\_mode中。我们用的时候使用专门的掩码去取出相应的位即可得知相应的信息。

3.2.4.2、ls -l打印出的权限列表

(1)123456789一共9位，3个一组。第一组三个表示文件的属主（owner、user）对该文件的可读、可写、可执行权限；第2组3个位表示文件的属主所在的组（group）对该文件的权限；第3组3个位表示其他用户（others）对该文件的权限。

(2)属主就是这个文件属于谁，一般来说文件创建时属主就是创建这个文件的那个用户。但是我们一个文件创建之后还可以用chown命令去修改一个文件的属主，还可以用chgrp命令去修改一个文件所在的组。

3.2.4.3、文件操作时的权限检查规则

(1)一个程序a.out被执行，a.out中试图去操作一个文件1.txt，这时候如何判定a.out是否具有对1.txt的某种操作权限呢？

(2)判定方法是：首先1.txt具有9个权限位，规定了3种人（user、group、others）对该文件的操作权限。所以我们判定1.txt是否能被a.out来操作，关键先搞清楚a.out对1.txt到底算哪种人。准确的说是看a.out被谁执行，也就是当前程序（进程）是哪个用户的进程。

(3)刚才上面说的是我的分析，到底对不对还得验证。

3.2.5.文件权限管理2

3.2.5.1、access函数检查权限设置

(1)文本权限管控其实蛮复杂，一般很难很容易的确定对一个文件是否具有某种权限。设计优秀的软件应该是：在操作某个文件之前先判断当前是否有权限做这个操作，如果有再做如果没有则提供错误信息给用户。

(2)access函数可以测试得到当前执行程序的那个用户在当前那个环境下对目标文件是否具有某种操作权限。

3.2.5.2、chmod/fchmod与权限修改

(1)chmod是一个linux命令，用来修改文件的各种权限属性。chmod命令只有root用户才有权利去执行修改。

(2)chmod命令其实内部是用linux的一个叫chmod的API实现的。

3.2.5.3、chown/fchown/lchown与属主修改

(1)linux中有个chown命令来修改文件属主

(2)chown命令是用chown API实现的

3.2.5.4、umask与文件权限掩码

(1)文件掩码是linux系统中维护的一个全局设置，umask的作用是用来设定我们系统中新创建的文件的默认权限的。

(2)umask命令就是用umask API实现的

3.2.6.读取目录文件

3.2.6.1、opendir与readdir函数

(1)opendir打开一个目录后得到一个DIR类型的指针给readdir使用

(2)readdir函数调用一次就会返回一个struct dirent类型的指针，这个指针指向一个结构体变量，这个结构体变量里面记录了一个目录项（所谓目录项就是目录中的一个子文件）。

(3)readdir调用一次只能读出一个目录项，要想读出目录中所有的目录项必须多次调用readdir函数。readdir函数内部户记住哪个目录项已经被读过了哪个还没读，所以多次调用后不会重复返回已经返回过的目录项。当readdir函数返回NULL时就表示目录中所有的目录项已经读完了。

3.2.6.2、dirent结构体

3.2.6.3、读取目录实战演练

3.2.6.4、可重入函数介绍

(1)有些函数是可重入的有些是不可重入的，具体概念可以去百度。

(2)readdir函数和我们前面接触的一些函数是不同的，首先readdir函数直接返回了一个结构体变量指针，因为readdir内部申请了内存并且给我们返回了地址。多次调用readdir其实readir内部并不会重复申请内存而是使用第一次调用readdir时分配的那个内存。这个设计方法是readdir不可重入的关键。

(3)readdir在多次调用时是有关联的，这个关联也标明readdir函数是不可重入的。

(4)库函数中有一些函数当年刚开始提供时都是不可重入的，后来意识到这种方式不安全，所以重新封装了C库，提供了对应的可重复版本（一般是不可重入版本函数名\_r）

\*\*\*\*\*\*\*\*\*\*\*《朱有鹏老师嵌入式linux核心课程》 \*\*\*\*\*\*\*\*\*\*\*

《3.linux应用编程和网络编程-第3部分-3.3.获取系统信息》

--------------------------------------------------------

本课程由朱老师物联网大讲堂推出并提供技术支持，课件可打包下载

网盘地址：http://yunpan.cn/cjVy3RAgfDufK 访问密码 4ad7

技术交流QQ群：朱老师物联网讲堂1群 397164505

--------------------------------------------------------

第一部分、章节目录

3.3.1.关于时间的概念

3.3.2.linux系统中的时间

3.3.3.时间相关API实战1

3.3.4.时间相关API实战2

3.3.5.linux中使用随机数

3.3.6.proc文件系统介绍

3.3.7.proc文件系统的使用

第二部分、章节介绍

3.3.1.关于时间的概念

本节讲述有关时间的一些概念，包括GMT时间、UTC时间和计算机中与时间有关的部件、工作原理。

3.3.2.linux系统中的时间

本节讲解linux系统中如何记录时间，以及linux API中与时间操作有关的部分。

3.3.3.时间相关API实战1

本节通过实践编写代码测试的方式来逐一测试使用linux中时间相关API函数

3.3.4.时间相关API实战2

本节通过实践编写代码测试的方式来逐一测试使用linux中时间相关API函数

3.3.5.linux中使用随机数

本节讲解随机数和伪随机数的概念，并且讲述linux下获取随机数的API并写代码实际测试。

3.3.6.proc文件系统介绍

本节介绍linux中/proc目录代表的proc虚拟文件系统工作原理和作用，并讲述了proc中常见的几个文件的作用。

3.3.7.proc文件系统的使用

本节讲述proc文件系统的使用方法并写代码进行实践测试。

第三部分、随堂记录

3.3.1.关于时间的概念

3.3.1.1、GMT时间

(1)GMT是格林尼治时间，也就是格林尼治地区的当地之间。

(2)GMT时间的意义？用格林尼治的当地时间作为全球国际时间，用以描述全球性的事件的时间，方便大家记忆。

(3)一般为了方便，一个国家都统一使用一个当地时间。

3.3.1.2、UTC时间

(1)GMT时间是以前使用的，近些年来越来越多的使用UTC时间。

(2)关于北京时间，可以参考：http://www.cnblogs.com/qiuyi21/archive/2008/03/04/1089456.html

3.3.1.3、计算机中与时间有关的部件

(1)点时间和段时间。段时间=点时间-点时间

(2)定时器和实时时钟。定时器（timer）定的时间就是段时间，实时时钟（RTC）就是和点时间有关的一个器件。

3.3.2.linux系统中的时间

3.3.2.1、jiffies的引入

(1)jiffies是linux内核中的一个全局变量，这个变量用来记录以内核的节拍时间为单位时间长度的一个数值。

(2)内核配置的时候定义了一个节拍时间，实际上linux内核的调度系统工作时就是以这个节拍时间为时间片的。

(3)jiffies变量开机时有一个基准值，然后内核每过一个节拍时间jiffies就会加1，然后到了系统的任意一个时间我们当前时间就被jiffies这个变量所标注。

3.3.2.2、linux系统如何记录时间

(1)内核在开机启动的时候会读取RTC硬件获取一个时间作为初始基准时间，这个基准时间对应一个jiffies值（这个基准时间换算成jiffies值的方法是：用这个时间减去1970-01-01 00:00:00 +0000(UTC)，然后把这个时间段换算成jiffies数值），这个jiffies值作为我们开机时的基准jiffies值存在。然后系统运行时每个时钟节拍的末尾都会给jiffies这个全局变量加1，因此操作系统就使用jiffies这个全局变量记录了下来当前的时间。当我们需要当前时间点时，就用jiffies这个时间点去计算（计算方法就是先把这个jiffies值对应的时间段算出来，然后加上1970-01-01 00:00:00 +0000(UTC)即可得到这个时间点）

(2)其实操作系统只在开机时读一次RTC，整个系统运行过程中RTC是无作用的。RTC的真正作用其实是在OS的2次开机之间进行时间的保存。

(3)理解时一定要点时间和段时间结合起来理解。jiffies这个变量记录的其实是段时间（其实就是当前时间和1970-01-01 00:00:00 +0000(UTC)这个时间的差值）

(4)一个时间节拍的时间取决于操作系统的配置，现代linux系统一般是10ms或者1ms。这个时间其实就是调度时间，在内核中用HZ来记录和表示。如果HZ定义成1000难么时钟节拍就是1/HZ，也就是1ms。这些在学习驱动时会用到。

3.3.2.3、linux中时间相关的系统调用

(1)常用的时间相关的API和C库函数有9个：time/ctime/localtime/gmtime/mktime/asctime/strftime/gettimeofday/settimeofday有9个：time/ctime/localtime/gmtime/mktime/asctime/strftime/gettimeofday/settimeofday

(2)time系统调用返回当前时间以秒为单位的距离1970-01-01 00:00:00 +0000(UTC)过去的秒数。这个time内部就是用jiffies换算得到的秒数。其他函数基本都是围绕着time来工作的。

(3)gmtime和localtime会把time得到的秒数变成一个struct tm结构体表示的时间。区别是gmtime得到的是国际时间，而localtime得到的是本地（指的是你运行localtime函数的程序所在的计算机所设置的时区对应的本地时间）时间。mktime用来完成相反方向的转换（struct tm到time\_t）

(4)如果从struct tm出发想得到字符串格式的时间，可以用asctime或者strftime都可以。（如果从time\_t出发想得到字符串格式的时间用ctime即可）

(5)gettimeofday返回的时间是由struct timeval和struct timezone这两个结构体来共同表示的，其中timeval表示时间，而timezone表示时区。settimeofday是用来设置当前的时间和时区的。

(6)总结：不管用哪个系统调用，最终得到的时间本质上都是一个时间（这个时间最终都是从kernel中记录的jiffies中计算得来的），只不过不同的函数返回的时间的格式不同，精度不同。

3.3.3.时间相关API实战1

3.3.3.1、time

(1)time能得到一个当前时间距离标准起点时间1970-01-01 00:00:00 +0000(UTC)过去了多少秒

3.3.3.2、ctime

(1)ctime可以从time\_t出发得到一个容易观察的字符串格式的当前时间。

(2)ctime好处是很简单好用，可以直接得到当前时间的字符串格式，直接打印来看。坏处是ctime的打印时间格式是固定的，没法按照我们的想法去变。

(3)实验结果可以看出ctime函数得到的时间是考虑了计算机中的本地时间的（计算机中的时区设置）

3.3.3.3、gmtime和localtime

(1)gmtime获取的时间中：年份是以1970为基准的差值，月份是0表示1月，小时数是以UTC时间的0时区为标准的小时数（北京是东8区，因此北京时间比这个时间大8）

(2)猜测localtime和gmtime的唯一区别就是localtime以当前计算机中设置的时区为小时的时间基准，其余一样。实践证明我们的猜测是正确的。

3.3.4.时间相关API实战2

3.3.4.1、mktime

(1)从OS中读取时间时用不到mktime的，这个mktime是用来向操作系统设置时间时用的。

3.3.4.2、asctime

(1)asctime得到一个固定格式的字符串格式的当前时间，效果上和ctime一样的。区别是ctime从time\_t出发，而asctime从struct tm出发。

3.3.4.3、strftime

(1)asctime和ctime得到的时间字符串都是固定格式的，没法用户自定义格式

(2)如果需要用户自定义时间的格式，则需要用strftime。

3.3.4.4、gettimeofday和settimeofday

(1)前面讲到的基于time函数的那个系列都是以秒为单位来获取时间的，没有比秒更精确的时间。

(2)有时候我们程序希望得到非常精确的时间（譬如以us为单位），这时候就只能通过gettimeofday来实现了。

3.3.5.linux中使用随机数

3.3.5.1、随机数和伪随机数

(1)随机数是随机出现，没有任何规律的一组数列。

(2)真正的完全随机的数列是不存在的，只是一种理想情况。我们平时要用到随机数时一般只能通过一些算法得到一个伪随机数序列。

(3)我们平时说到随机数，基本都指的是伪随机数。

3.3.5.2、linux中随机数相关API

(1)连续多次调用rand函数可以返回一个伪随机数序列

(2)srand函数用来设置rand获取的伪随机序列的种子

3.3.5.3、实战演示

(1)单纯使用rand重复调用n次，就会得到一个0-RAND\_MAX之间的伪随机数，如果需要调整范围，可以得到随机数序列后再进行计算。

(2)单纯使用rand来得到伪随机数序列有缺陷，每次执行程序得到的伪随机序列是同一个序列，没法得到其他序列

(3)原因是因为rand内部的算法其实是通过一个种子（seed，其实就是一个原始参数，int类型），rand内部默认是使用1作为seed的，种子一定的算法也是一定的，那么每次得到的伪随机序列肯定是同一个。

(4)所以要想每次执行这个程序获取的伪随机序列不同，则每次都要给不同的种子。用srand函数来设置种子。

3.3.5.4、总结和说明

(1)在每次执行程序时，先用srand设置一个不同的种子，然后再多次调用rand获取一个伪随机序列，这样就可以每次都得到一个不同的伪随机序列。

(2)一般常规做法是用time函数的返回值来做srand的参数。

3.3.5.5、在linux系统中获取真正的随机数

(1)linux系统收集系统中的一些随机发生的事件的时间（譬如有人动鼠标，譬如触摸屏的操作和坐标等）作为随机种子去生成随机数序列。

3.3.6.proc文件系统介绍

3.3.6.1、操作系统级别的调试

(1)简单程序单步调试

(2)复杂程序printf打印信息调试

(3)框架体系日志记录信息调试

(4)内核调试的困境

3.3.6.2、proc虚拟文件系统的工作原理

(1)linux内核是一个非常庞大、非常复杂的一个单独的程序，对于这样的一个程序来说调试是非常复杂的。

(2)项kernel这样庞大的项目，给里面添加/更改一个功能是非常麻烦的，因为你这添加的一个功能可能会影响其他已经有的。

(3)早期内核版本中尽管调试很麻烦，但是高手们还可以凭借个人超凡脱俗的能力去驾驭。但是到了2.4左右的版本的时候，这个难度已经非常大了。

(4)为了降低内核调试和学习的难度，内核开发者们在内核中添加了一些属性专门用于调试内核，proc文件系统就是一个尝试。

(5)proc文件系统的思路是：在内核中构建一个虚拟文件系统/proc，内核运行时将内核中一些关键的数据结构以文件的方式呈现在/proc目录中的一些特定文件中，这样相当于将不可见的内核中的数据结构以可视化的方式呈现给内核的开发者。

(6)proc文件系统给了开发者一种调试内核的方法：我们通过实时的观察/proc/xxx文件，来观看内核中特定数据结构的值。在我们添加一个新功能的前后来对比，就可以知道这个新功能产生的影响对还是不对。

(7)proc目录下的文件大小都是0，因为这些文件本身并不存在于硬盘中，他也不是一个真实文件，他只是一个接口，当我们去读取这个文件时，其实内核并不是去硬盘上找这个文件，而是映射为内核内部一个数据结构被读取并且格式化成字符串返回给我们。所以尽管我们看到的还是一个文件内容字符串，和普通文件一样的；但是实际上我们知道这个内容是实时的从内核中数据结构来的，而不是硬盘中来的。

3.3.6.3、常用proc中的文件介绍

(1)/proc/cmdline

(2)/proc/cpuinfo

(3)/proc/devices

(4)/proc/interrupts

3.3.7.proc文件系统的使用

3.3.7.1、cat以手工查看

3.3.7.2、程序中可以文件IO访问

3.3.7.3、在shell程序中用cat命令结合正则表达式来获取并处理内核信息

3.3.7.3、扩展：sys文件系统

(1)sys文件系统本质上和proc文件系统是一样的，都是虚拟文件系统，都在根目录下有个目录（一个是/proc目录，另一个是/sys目录），因此都不是硬盘中的文件，都是内核中的数据结构的可视化接口。

(2)不同的是/proc中的文件只能读，但是/sys中的文件可以读写。读/sys中的文件就是获取内核中数据结构的值，而写入/sys中的文件就是设置内核中的数据结构的元素的值。

(3)历史上刚开始先有/proc文件系统，人们希望通过这种技术来调试内核。实际做出来后确实很有用，所以很多内核开发者都去内核调价代码向/proc目录中写文件，而且刚开始的时候内核管理者对proc目录的使用也没有什么经验也没什么统一规划，后来的结果就是proc里面的东西又多又杂乱。

(4)后来觉得proc中的内容太多太乱缺乏统一规划，于是乎又添加了sys目录。sys文件系统一开始就做了很好的规划和约定，所以后来使用sys目录时有了规矩。

\*\*\*\*\*\*\*\*\*\*\*《朱有鹏老师嵌入式linux核心课程》 \*\*\*\*\*\*\*\*\*\*\*

《3.linux应用编程和网络编程-第4部分-3.4.linux进程全解》

--------------------------------------------------------

本课程由朱老师物联网大讲堂推出并提供技术支持，课件可打包下载

网盘地址：http://yunpan.cn/cjVy3RAgfDufK 访问密码 4ad7

技术交流QQ群：朱老师物联网讲堂1群 397164505

--------------------------------------------------------

第一部分、章节目录

3.4.1.程序的开始和结束

3.4.2.进程环境

3.4.3.进程的正式引入

3.4.4.fork创建子进程

3.4.5.父子进程对文件的操作

3.4.6.进程的诞生和消亡

3.4.7.父进程wait回收子进程

3.4.8.waitpid介绍

3.4.9.exec族函数及实战1

3.4.10.exec族函数及实战2

3.4.11.进程状态和system函数

3.4.12.进程关系

3.4.13.守护进程的引入

3.4.14.编写简单守护进程

3.4.15.使用syslog来记录调试信息

3.4.16.让程序不能被多次运行

3.4.17.linux的进程间通信概述

3.4.18.linux的IPC机制1-管道

3.4.19.SystemV IPC介绍

第二部分、章节介绍

3.4.1.程序的开始和结束

本节讲述一个典型程序的开始运行和结束运行，引入引导代码让大家更容易的理解操作系统是如何运行一个程序的。

3.4.2.进程环境

本节讲解进程的环境变量和虚拟地址空间，这些都是一个进程在系统中运行时的外部环境。

3.4.3.进程的正式引入

本节正式引入进程的概念，讲述了进程的ID以及获取进程ID的函数，多进程调度实现宏观上并行的原理。

3.4.4.fork创建子进程

本节引入子进程，并且讲解并实战演示了fork函数如何创建子进程。

3.4.5.父子进程对文件的操作

本节通过父子进程对文件的操作来说明父子进程的区别和联系。

3.4.6.进程的诞生和消亡

本节讲述进程的诞生和消亡过程，着重讲了僵尸进程和孤儿进程的概念，进程资源回收和状态返回等。

3.4.7.父进程wait回收子进程

本节讲解wait函数回收子进程的信号式异步通信工作原理，并且实战演练了使用wait来回收子进程的过程。

3.4.8.waitpid介绍

本节首先介绍waitpid和wait函数的差别，然后实战演示了waitpid函数的3种常见用法，最后简单讲解了竟态的概念。

3.4.9.exec族函数及实战1

本节开始讲解exec族函数的作用和各个API的差异，并且写代码进行演示。

3.4.10.exec族函数及实战2

本节演示了exec族的p后缀和e后缀函数的用法。

3.4.11.进程状态和system函数

本节详细讲解了linux中进程的5种状态和状态转换图，最后讲了下system函数

3.4.12.进程关系

本节讲解进程的四种关系：无关系、父子关系、进程组和会话。

3.4.13.守护进程的引入

本节引入守护进程的概念，并且介绍了常见的一些系统级服务器守护进程及其作用。

3.4.14.编写简单守护进程

本节实践编程自己实现一个守护进程，让大家熟悉守护进程的创建过程。

3.4.15.使用syslog来记录调试信息

本节讲解如何在程序中使用syslog记录日志信息，这是一种非常常见的程序调试信息输出手段，也是守护进程必用的调试信息输出手段。

3.4.16.让程序不能被多次运行

本节介绍如何让我们的程序只能运行一次，即所谓单例运行的常规实现方法。这个在实际工作中很有用。

3.4.17.linux的进程间通信概述

本节进行linux中进程间通信IPC的概述，指明了后面如何讲解IPC。

3.4.18.linux的IPC机制1-管道

本节详细讲了传统Unix的进程间通信方法：管道和有名管道。

3.4.19.SystemV IPC介绍

本节对SystemV IPC的三种（信号量、消息队列、共享内存）方式进行概括性讲解和对比。

第三部分、随堂记录

3.4.1.程序的开始和结束

3.4.1.1、main函数由谁调用

(1)编译链接时的引导代码。操作系统下的应用程序其实在main执行前也需要先执行一段引导代码才能去执行main，我们写应用程序时不用考虑引导代码的问题，编译连接时（准确说是连接时）由链接器将编译器中事先准备好的引导代码给连接进去和我们的应用程序一起构成最终的可执行程序。

(2)运行时的加载器。加载器是操作系统中的程序，当我们去执行一个程序时（譬如./a.out，譬如代码中用exec族函数来运行）加载器负责将这个程序加载到内存中去执行这个程序。

(3)程序在编译连接时用链接器，运行时用加载器，这两个东西对程序运行原理非常重要。

(4)argc和argv的传参如何实现

3.4.1.2、程序如何结束

(1)正常终止：return、exit、\_exit

(2)非正常终止：自己或他人发信号终止进程

3.4.1.3、atexit注册进程终止处理函数

(1)实验演示

(2)atexit注册多个进程终止处理函数，先注册的后执行（先进后出，和栈一样）

(2)return、exit和\_exit的区别：return和exit效果一样，都是会执行进程终止处理函数，但是用\_exit终止进程时并不执行atexit注册的进程终止处理函数。

3.4.2.进程环境

3.4.2.1、环境变量

(1)export命令查看环境变量

(2)进程环境表介绍.每一个进程中都有一份所有环境变量构成的一个表格，也就是说我们当前进程中可以直接使用这些环境变量。进程环境表其实是一个字符串数组，用environ变量指向它。

(3)程序中通过environ全局变量使用环境变量

(4)我们写的程序中可以无条件直接使用系统中的环境变量，所以一旦程序中用到了环境变量那么程序就和操作系统环境有关了。

(4)获取指定环境变量函数getenv

3.4.2.2、进程运行的虚拟地址空间

(1)操作系统中每个进程在独立地址空间中运行

(2)每个进程的逻辑地址空间均为4GB（32位系统）

(3)0-1G为OS，1-4G为应用

(4)虚拟地址到物理地址空间的映射

(5)意义。进程隔离，提供多进程同时运行

3.4.3.进程的正式引入

3.4.3.1、什么是进程

(1)动态过程而不是静态实物

(2)进程就是程序的一次运行过程，一个静态的可执行程序a.out的一次运行过程（./a.out去运行到结束）就是一个进程。

(3)进程控制块PCB（process control block），内核中专门用来管理一个进程的数据结构。

3.4.3.2、进程ID

(1)getpid、getppid、getuid、geteuid、getgid、getegid

(2)实际用户ID和有效用户ID区别（可百度）

3.4.3.3、多进程调度原理

(1)操作系统同时运行多个进程

(2)宏观上的并行和微观上的串行

(3)实际上现代操作系统最小的调度单元是线程而不是进程

3.4.4.fork创建子进程

3.4.4.1、为什么要创建子进程

(1)每一次程序的运行都需要一个进程

(2)多进程实现宏观上的并行

3.4.4.2、fork的内部原理

(1)进程的分裂生长模式。如果操作系统需要一个新进程来运行一个程序，那么操作系统会用一个现有的进程来复制生成一个新进程。老进程叫父进程，复制生成的新进程叫子进程。

(2)fork的演示

(3)fork函数调用一次会返回2次，返回值等于0的就是子进程，而返回值大于0的就是父进程。

(4)典型的使用fork的方法：使用fork后然后用if判断返回值，并且返回值大于0时就是父进程，等于0时就是子进程。

(5)fork的返回值在子进程中等于0，在父进程中等于本次fork创建的子进程的进程ID。

3.4.4.3、关于子进程

(1)子进程和父进程的关系

(2)子进程有自己独立的PCB

(3)子进程被内核同等调度

3.4.5.父子进程对文件的操作

3.4.5.1、子进程继承父进程中打开的文件

(1)上下文：父进程先open打开一个文件得到fd，然后在fork创建子进程。之后在父子进程中各自write向fd中写入内容

(2)测试结论是：接续写。实际上本质原因是父子进程之间的fd对应的文件指针是彼此关联的（很像O\_APPEND标志后的样子）

(3)实际测试时有时候会看到只有一个，有点像分别写。但是实际不是，原因是

3.4.5.2、父子进程各自独立打开同一文件实现共享

(1)父进程open打开1.txt然后写入，子进程打开1.txt然后写入，结论是：分别写。原因是父子进程分离后才各自打开的1.txt，这时候这两个进程的PCB已经独立了，文件表也独立了，因此2次读写是完全独立的。

(2)open时使用O\_APPEND标志看看会如何？实际测试结果标明O\_APPEND标志可以把父子进程各自独立打开的fd的文件指针给关联起来，实现分别写。

3.4.5.3、总结

(1)父子进程间终究多了一些牵绊

(2)父进程在没有fork之前自己做的事情对子进程有很大影响，但是父进程fork之后在自己的if里做的事情就对子进程没有影响了。本质原因就是因为fork内部实际上已经复制父进程的PCB生成了一个新的子进程，并且fork返回时子进程已经完全和父进程脱离并且独立被OS调度执行。

(2)子进程最终目的是要独立去运行另外的程序

3.4.6.进程的诞生和消亡

3.4.6.1、进程的诞生

(1)进程0和进程1

(2)fork

(3)vfork

3.4.6.2、进程的消亡

(1)正常终止和异常终止

(2)进程在运行时需要消耗系统资源（内存、IO），进程终止时理应完全释放这些资源（如果进程消亡后仍然没有释放相应资源则这些资源就丢失了）

(3)linux系统设计时规定：每一个进程退出时，操作系统会自动回收这个进程涉及到的所有的资源（譬如malloc申请的内容没有free时，当前进程结束时这个内存会被释放，譬如open打开的文件没有close的在程序终止时也会被关闭）。但是操作系统只是回收了这个进程工作时消耗的内存和IO，而并没有回收这个进程本身占用的内存（8KB，主要是task\_struct和栈内存）

(4)因为进程本身的8KB内存操作系统不能回收需要别人来辅助回收，因此我们每个进程都需要一个帮助它收尸的人，这个人就是这个进程的父进程。

3.4.6.3、僵尸进程

(1)子进程先于父进程结束。子进程结束后父进程此时并不一定立即就能帮子进程“收尸”，在这一段（子进程已经结束且父进程尚未帮其收尸）子进程就被成为僵尸进程。

(2)子进程除task\_struct和栈外其余内存空间皆已清理

(3)父进程可以使用wait或waitpid以显式回收子进程的剩余待回收内存资源并且获取子进程退出状态。

(4)父进程也可以不使用wait或者waitpid回收子进程，此时父进程结束时一样会回收子进程的剩余待回收内存资源。（这样设计是为了防止父进程忘记显式调用wait/waitpid来回收子进程从而造成内存泄漏）

3.4.6.4、孤儿进程

(1)父进程先于子进程结束，子进程成为一个孤儿进程。

(2)linux系统规定：所有的孤儿进程都自动成为一个特殊进程（进程1，也就是init进程）的子进程。

3.4.7.父进程wait回收子进程

3.4.7.1、wait的工作原理

(1)子进程结束时，系统向其父进程发送SIGCHILD信号

(2)父进程调用wait函数后阻塞

(3)父进程被SIGCHILD信号唤醒然后去回收僵尸子进程

(4)父子进程之间是异步的，SIGCHILD信号机制就是为了解决父子进程之间的异步通信问题，让父进程可以及时的去回收僵尸子进程。

(5)若父进程没有任何子进程则wait返回错误

3.4.7.2、wait实战编程

(1)wait的参数status。status用来返回子进程结束时的状态，父进程通过wait得到status后就可以知道子进程的一些结束状态信息。

(2)wait的返回值pid\_t，这个返回值就是本次wait回收的子进程的PID。当前进程有可能有多个子进程，wait函数阻塞直到其中一个子进程结束wait就会返回，wait的返回值就可以用来判断到底是哪一个子进程本次被回收了。

对wait做个总结：wait主要是用来回收子进程资源，回收同时还可以得知被回收子进程的pid和退出状态。

(3)fork后wait回收实例

(4)WIFEXITED、WIFSIGNALED、WEXITSTATUS这几个宏用来获取子进程的退出状态。

WIFEXITED宏用来判断子进程是否正常终止（return、exit、\_exit退出）

WIFSIGNALED宏用来判断子进程是否非正常终止（被信号所终止）

WEXITSTATUS宏用来得到正常终止情况下的进程返回值的。

3.4.8.waitpid介绍

3.4.8.1、waitpid和wait差别

(1)基本功能一样，都是用来回收子进程

(2)waitpid可以回收指定PID的子进程

(3)waitpid可以阻塞式或非阻塞式两种工作模式

3.4.8.2、waitpid原型介绍

(1)参数

(2)返回值

3.4.8.3、代码实例

(1)使用waitpid实现wait的效果

ret = waitpid(-1, &status, 0); -1表示不等待某个特定PID的子进程而是回收任意一个子进程，0表示用默认的方式（阻塞式）来进行等待，返回值ret是本次回收的子进程的PID

(2)ret = waitpid(pid, &status, 0); 等待回收PID为pid的这个子进程，如果当前进程并没有一个ID号为pid的子进程，则返回值为-1；如果成功回收了pid这个子进程则返回值为回收的进程的PID

(3)ret = waitpid(pid, &status, WNOHANG);这种表示父进程要非阻塞式的回收子进程。此时如果父进程执行waitpid时子进程已经先结束等待回收则waitpid直接回收成功，返回值是回收的子进程的PID；如果父进程waitpid时子进程尚未结束则父进程立刻返回（非阻塞），但是返回值为0（表示回收不成功）。

3.4.8.4、竟态初步引入

(1)竟态全称是：竞争状态，多进程环境下，多个进程同时抢占系统资源（内存、CPU、文件IO）

(2)竞争状态对OS来说是很危险的，此时OS如果没处理好就会造成结果不确定。

(3)写程序当然不希望程序运行的结果不确定，所以我们写程序时要尽量消灭竞争状态。操作系统给我们提供了一系列的消灭竟态的机制，我们需要做的是在合适的地方使用合适的方法来消灭竟态。

3.4.9.exec族函数及实战1

3.4.9.1、为什么需要exec函数

(1)fork子进程是为了执行新程序(fork创建了子进程后，子进程和父进程同时被OS调度执行，因此子进程可以单独的执行一个程序，这个程序宏观上将会和父进程程序同时进行)

(2)可以直接在子进程的if中写入新程序的代码。这样可以，但是不够灵活，因为我们只能把子进程程序的源代码贴过来执行（必须知道源代码，而且源代码太长了也不好控制），譬如说我们希望子进程来执行ls -la 命令就不行了（没有源代码，只有编译好的可执行程序）

(3)使用exec族运行新的可执行程序（exec族函数可以直接把一个编译好的可执行程序直接加载运行）

(4)我们有了exec族函数后，我们典型的父子进程程序是这样的：子进程需要运行的程序被单独编写、单独编译连接成一个可执行程序（叫hello），（项目是一个多进程项目）主程序为父进程，fork创建了子进程后在子进程中exec来执行hello，达到父子进程分别做不同程序同时（宏观上）运行的效果。

3.4.9.2、exec族的6个函数介绍

(1)execl和execv 这两个函数是最基本的exec，都可以用来执行一个程序，区别是传参的格式不同。execl是把参数列表（本质上是多个字符串，必须以NULL结尾）依次排列而成（l其实就是list的缩写），execv是把参数列表事先放入一个字符串数组中，再把这个字符串数组传给execv函数。

(2)execlp和execvp 这两个函数在上面2个基础上加了p，较上面2个来说，区别是：上面2个执行程序时必须指定可执行程序的全路径（如果exec没有找到path这个文件则直接报错），而加了p的传递的可以是file（也可以是path，只不过兼容了file。加了p的这两个函数会首先去找file，如果找到则执行执行，如果没找到则会去环境变量PATH所指定的目录下去找，如果找到则执行如果没找到则报错）

(3)execle和execvpe 这两个函数较基本exec来说加了e，函数的参数列表中也多了一个字符串数组envp形参，e就是environment环境变量的意思，和基本版本的exec的区别就是：执行可执行程序时会多传一个环境变量的字符串数组给待执行的程序。

3.4.9.3、exec实战1

(1)使用execl运行ls -l -a

(2)使用execv运行ls

(3)使用execl运行自己写的程序

3.4.10.exec族函数及实战2

3.4.10.1、execlp和execvp

(1)加p和不加p的区别是：不加p时需要全部路径+文件名，如果找不到就报错了。加了p之后会多帮我们到PATH所指定的路径下去找一下。

3.4.10.2、execle和execvpe

(1)main函数的原型其实不止是int main(int argc, char \*\*argv)，而可以是

int main(int argc, char \*\*argv, char \*\*env) 第三个参数是一个字符串数组，内容是环境变量。

(2)如果用户在执行这个程序时没有传递第三个参数，则程序会自动从父进程继承一份环境变量（默认的，最早来源于OS中的环境变量）；如果我们exec的时候使用execlp或者execvpe去给传一个envp数组，则程序中的实际环境变量是我们传递的这一份（取代了默认的从父进程继承来的那一份）

3.4.11.进程状态和system函数

3.4.11.1、进程的5种状态

(1)就绪态。这个进程当前所有运行条件就绪，只要得到了CPU时间就能直接运行。

(2)运行态。就绪态时得到了CPU就进入运行态开始运行。

(3)僵尸态。进程已经结束但是父进程还没来得及回收

(4)等待态（浅度睡眠&深度睡眠），进程在等待某种条件，条件成熟后可进入就绪态。等待态下就算你给他CPU调度进程也无法执行。浅度睡眠等待时进程可以被（信号）唤醒，而深度睡眠等待时不能被唤醒只能等待的条件到了才能结束睡眠状态。

(5)暂停态。暂停并不是进程的终止，只是被被人（信号）暂停了，还可以回复的。

3.4.11.2、进程各种状态之间的转换图

3.4.11.3、system函数简介

(1)system函数 = fork+exec

(1)原子操作。原子操作意思就是整个操作一旦开始就会不被打断的执行完。原子操作的好处就是不会被人打断（不会引来竞争状态），坏处是自己单独连续占用CPU时间太长影响系统整体实时性，因此应该尽量避免不必要的原子操作，就算不得不原子操作也应该尽量原子操作的时间缩短。

(2)使用system调用ls

3.4.12.进程关系

(1)无关系

(2)父子进程关系

(3)进程组（group）由若干进程构成一个进程组

(4)会话（session）会话就是进程组的组

3.4.13.守护进程的引入

3.4.13.1、进程查看命令ps

(1)ps -ajx 偏向显示各种有关的ID号

(2)ps -aux 偏向显示进程各种占用资源

3.4.13.2、向进程发送信号指令kill

(1)kill -信号编号 进程ID，向一个进程发送一个信号

(2)kill -9 xxx，将向xxx这个进程发送9号信号，也就是要结束进程

3.4.13.3、何谓守护进程

(1)daemon，表示守护进程，简称为d（进程名后面带d的基本就是守护进程）

(2)长期运行（一般是开机运行直到关机时关闭）

(3)与控制台脱离（普通进程都和运行该进程的控制台相绑定，表现为如果终端被强制关闭了则这个终端中运行的所有进程都被会关闭，背后的问题还在于会话）

(4)服务器（Server），服务器程序就是一个一直在运行的程序，可以给我们提供某种服务（譬如nfs服务器给我们提供nfs通信方式），当我们程序需要这种服务时我们可以调用服务器程序（和服务器程序通信以得到服务器程序的帮助）来进程这种服务操作。服务器程序一般都实现为守护进程。

3.4.13.4、常见守护进程

(1)syslogd，系统日志守护进程，提供syslog功能。

(2)cron，cron进程用来实现操作系统的时间管理，linux中实现定时执行程序的功能就要用到cron。

3.4.14.编写简单守护进程

3.4.14.1、任何一个进程都可以将自己实现成守护进程

3.4.14.2、create\_daemon函数要素

(1)子进程等待父进程退出

(2)子进程使用setsid创建新的会话期，脱离控制台

(3)调用chdir将当前工作目录设置为/

(4)umask设置为0以取消任何文件权限屏蔽

(5)关闭所有文件描述符

(6)将0、1、2定位到/dev/null

3.4.15.使用syslog来记录调试信息

3.4.15.1、openlog、syslog、closelog

3.4.15.2、各种参数

3.4.15.3、编程实战

(1)一般log信息都在操作系统的/var/log/messages这个文件中存储着，但是ubuntu中是在/var/log/syslog文件中的。

3.4.15.4、syslog的工作原理

(1)操作系统中有一个守护进程syslogd（开机运行，关机时才结束），这个守护进程syslogd负责进行日志文件的写入和维护。

(2)syslogd是独立于我们任意一个进程而运行的。我们当前进程和syslogd进程本来是没有任何关系的，但是我们当前进程可以通过调用openlog打开一个和syslogd相连接的通道，然后通过syslog向syslogd发消息，然后由syslogd来将其写入到日志文件系统中。

(3)syslogd其实就是一个日志文件系统的服务器进程，提供日志服务。任何需要写日志的进程都可以通过openlog/syslog/closelog这三个函数来利用syslogd提供的日志服务。这就是操作系统的服务式的设计。

3.4.16.让程序不能被多次运行

3.4.16.1、问题

(1)因为守护进程是长时间运行而不退出，因此./a.out执行一次就有一个进程，执行多次就有多个进程。

(2)这样并不是我们想要的。我们守护进程一般都是服务器，服务器程序只要运行一个就够了，多次同时运行并没有意义甚至会带来错误。

(3)因此我们希望我们的程序具有一个单例运行的功能。意思就是说当我们./a.out去运行程序时，如果当前还没有这个程序的进程运行则运行之，如果之前已经有一个这个程序的进程在运行则本次运行直接退出（提示程序已经在运行）。

3.4.16.2、实现方法：

(1)最常用的一种方法就是：用一个文件的存在与否来做标志。具体做法是程序在执行之初去判断一个特定的文件是否存在，若存在则标明进程已经在运行，若不存在则标明进程没有在运行。然后运行程序时去创建这个文件。当程序结束的时候去删除这个文件即可。

(2)这个特定文件要古怪一点，确保不会凑巧真的在电脑中存在的。

3.4.17.linux的进程间通信概述

3.4.17.1、为什么需要进程间通信

(1)进程间通信（IPC）指的是2个任意进程之间的通信。

(2)同一个进程在一个地址空间中，所以同一个进程的不同模块（不同函数、不同文件）之间都是很简单的（很多时候都是全局变量、也可以通过函数形参实参传递）

(3)2个不同的进程处于不同的地址空间，因此要互相通信很难。

3.4.17.2、什么样的程序设计需要进程间通信

(1)99%的程序是不需要考虑进程间通信的。因为大部分程序都是单进程的（可以多线程）

(2)复杂、大型的程序，因为设计的需要就必须被设计成多进程程序（我们整个程序就设计成多个进程同时工作来完成的模式），常见的如GUI、服务器。

(3)结论：IPC技术在一般中小型程序中用不到，在大型程序中才会用到。

3.4.17.3、linux内核提供多种进程间通信机制

(1)无名管道和有名管道

(2)SystemV IPC：信号量、消息队列、共享内存

(3)Socket域套接字

(4)信号

3.4.17.4、为什么不详细讲IPC

(1)日常使用少，只有大型程序才能用上

(2)更为复杂，属于linux应用编程中难度最大的部分

(3)细节多

(4)面试较少涉及，对找工作帮助不大

(5)建议后续深入学习时再来实际写代码详细探讨

3.4.18.linux的IPC机制1-管道

3.4.18.1、管道（无名管道）

(1)管道通信的原理：内核维护的一块内存，有读端和写端（管道是单向通信的）

(2)管道通信的方法：父进程创建管理后fork子进程，子进程继承父进程的管道fd

(3)管道通信的限制：只能在父子进程间通信、半双工

(4)管道通信的函数：pipe、write、read、close

3.4.18.2、有名管道（fifo）

(1)有名管道的原理：实质也是内核维护的一块内存，表现形式为一个有名字的文件

(2)有名管道的使用方法：固定一个文件名，2个进程分别使用mkfifo创建fifo文件，然后分别open打开获取到fd，然后一个读一个写

(3)管道通信限制：半双工（注意不限父子进程，任意2个进程都可）

(4)管道通信的函数：mkfifo、open、write、read、close

3.4.19.SystemV IPC介绍

3.4.19.1、SystemV IPC的基本特点

(1)系统通过一些专用API来提供SystemV IPC功能

(2)分为：信号量、消息队列、共享内存

(3)其实质也是内核提供的公共内存

3.4.19.2、消息队列

(1)本质上是一个队列，队列可以理解为（内核维护的一个）FIFO

(2)工作时A和B2个进程进行通信，A向队列中放入消息，B从队列中读出消息。

3.4.19.3、信号量

(1)实质就是个计数器（其实就是一个可以用来计数的变量，可以理解为int a）

(2)通过计数值来提供互斥和同步

3.4.19.4、共享内存

(1)大片内存直接映射

(2)类似于LCD显示时的显存用法

3.4.19.5、剩余的2类IPC

(1)信号

(2)Unix域套接字 socket

\*\*\*\*\*\*\*\*\*\*\*《朱有鹏老师嵌入式linux核心课程》 \*\*\*\*\*\*\*\*\*\*\*

《3.linux应用编程和网络编程-第5部分-3.5.linux中的信号》

--------------------------------------------------------

本课程由朱老师物联网大讲堂推出并提供技术支持，课件可打包下载

网盘地址：http://yunpan.cn/cjVy3RAgfDufK 访问密码 4ad7

技术交流QQ群：朱老师物联网讲堂1群 397164505

--------------------------------------------------------

第一部分、章节目录

3.5.1.什么是信号

3.5.2.常见信号介绍

3.5.3.进程对信号的处理

3.5.4.alarm和pause函数

第二部分、章节介绍

3.5.1.什么是信号

本节介绍什么是信号，以及信号由谁发送、由谁处理，怎么处理的问题，目的是站在一定高度上认识信号的作用和意义。

3.5.2.常见信号介绍

本节对常见需要注意的几个信号进行介绍，其他不常用信号可以暂时不理，遇到时再去查文档处理。

3.5.3.进程对信号的处理

本节介绍进程对信号的三种处理方法，signal函数和sigaction函数用来捕获信号时的差异。

3.5.4.alarm和pause函数

本节对alarm函数和pause函数进行讲解和实战编程，这两个函数的实现都和信号有关。

第三部分、随堂记录

3.5.1.什么是信号

3.5.1.1、信号是内容受限的一种异步通信机制

(1)信号的目的：用来通信

(2)信号是异步的（对比硬件中断）

(3)信号本质上是int型数字编号（事先定义好的）

3.5.1.2、信号由谁发出

(1)用户在终端按下按键

(2)硬件异常后由操作系统内核发出信号

(3)用户使用kill命令向其他进程发出信号

(4)某种软件条件满足后也会发出信号，如alarm闹钟时间到会产生SIGALARM信号，向一个读端已经关闭的管道write时会产生SIGPIPE信号

3.5.1.3、信号由谁处理、如何处理

(1)忽略信号

(2)捕获信号（信号绑定了一个函数）

(3)默认处理（当前进程没有明显的管这个信号，默认：忽略或终止进程）

3.5.2.常见信号介绍

(1)SIGINT 2 Ctrl+C时OS送给前台进程组中每个进程

(2)SIGABRT 6 调用abort函数，进程异常终止

(3)SIGPOLL SIGIO 8 指示一个异步IO事件，在高级IO中提及

(4)SIGKILL 9 杀死进程的终极办法

(5)SIGSEGV 11 无效存储访问时OS发出该信号

(6)SIGPIPE 13 涉及管道和socket

(7)SIGALARM 14 涉及alarm函数的实现

(8)SIGTERM 15 kill命令发送的OS默认终止信号

(9)SIGCHLD 17 子进程终止或停止时OS向其父进程发此信号

(10)

SIGUSR1 10 用户自定义信号，作用和意义由应用自己定义

SIGUSR2 12

3.5.3.进程对信号的处理

3.5.3.1、signal函数介绍

3.5.3.2、用signal函数处理SIGINT信号

(1)默认处理

(2)忽略处理

(3)捕获处理

细节：

(1)signal函数绑定一个捕获函数后信号发生后会自动执行绑定的捕获函数，并且把信号编号作为传参传给捕获函数

(2)signal的返回值在出错时为SIG\_ERR，绑定成功时返回旧的捕获函数

(3)

3.5.3.3、signal函数的优点和缺点

(1)优点：简单好用，捕获信号常用

(2)缺点：无法简单直接得知之前设置的对信号的处理方法

3.5.3.4、sigaction函数介绍

(1)2个都是API，但是sigaction比signal更具有可移植性

(2)用法关键是2个sigaction指针

sigaction比signal好的一点：sigaction可以一次得到设置新捕获函数和获取旧的捕获函数（其实还可以单独设置新的捕获或者单独只获取旧的捕获函数），而signal函数不能单独获取旧的捕获函数而必须在设置新的捕获函数的同时才获取旧的捕获函数。

3.5.4.alarm和pause函数

3.5.4.1、alarm函数

(1)内核以API形式提供的闹钟

(2)编程实践

3.5.4.2、pause函数

(1)内核挂起

(2)代码实践

pause函数的作用就是让当前进程暂停运行，交出CPU给其他进程去执行。当当前进程进入pause状态后当前进程会表现为“卡住、阻塞住”，要退出pause状态当前进程需要被信号唤醒。

3.5.4.3、使用alarm和pause来模拟sleep

\*\*\*\*\*\*\*\*\*\*\*《朱有鹏老师嵌入式linux核心课程》 \*\*\*\*\*\*\*\*\*\*\*

《3.linux应用编程和网络编程-第3部分-3.6.高级IO》

--------------------------------------------------------

本课程由朱老师物联网大讲堂推出并提供技术支持，课件可打包下载

网盘地址：http://yunpan.cn/cjVy3RAgfDufK 访问密码 4ad7

技术交流QQ群：朱老师物联网讲堂1群 397164505

--------------------------------------------------------

第一部分、章节目录

3.6.1.非阻塞IO

3.6.2.阻塞式IO的困境

3.6.3.并发式IO的解决方案

3.6.4.IO多路复用原理

3.6.5.IO多路复用实践

3.6.6.异步IO

3.6.7.存储映射IO

第二部分、章节介绍

3.6.1.非阻塞IO

本节讲解什么是非阻塞IO，如何将文件描述符修改为非阻塞式。

3.6.2.阻塞式IO的困境

本节通过实例代码运行结果，让大家看到在并发式IO访问时非阻塞IO遭遇的困境，由此引入非阻塞式IO。

3.6.3.并发式IO的解决方案

本节介绍解决并发式IO问题的三种方法，并且写代码进行讲解非阻塞式IO方法。

3.6.4.IO多路复用原理

本节讲述IO多路复用的实现原理和相关函数select、poll的参数和使用要点。

3.6.5.IO多路复用实践

本节进行代码实践，使用select和poll来实际解决前面的同时读取鼠标和键盘的任务。

3.6.6.异步IO

本节讲解异步IO的实现原理，并且实际写代码用异步IO实现同时读取鼠标和键盘。

3.6.7.存储映射IO

本节以LCD显示和SystemV IPC的共享内存为案例，简单分析了存储映射IO的原理和优势。

第三部分、随堂记录

3.6.1.非阻塞IO

3.6.1.1、阻塞与非阻塞

3.6.1.2、为什么有阻塞式

(1)常见的阻塞：wait、pause、sleep等函数；read或write某些文件时

(2)阻塞式的好处

3.6.1.3、非阻塞

(1)为什么要实现非阻塞

(2)如何实现非阻塞IO访问：O\_NONBLOCK和fcntl

3.6.2.阻塞式IO的困境

3.6.2.1、程序中读取键盘

3.6.2.2、程序中读取鼠标

3.6.2.3、程序中同时读取键盘和鼠标

3.6.2.4、问题分析

3.6.3.并发式IO的解决方案

3.6.3.1、非阻塞式IO

3.6.3.2、多路复用IO

3.6.3.3、异步通知（异步IO）

3.6.4.IO多路复用原理

3.6.4.1、何为IO多路复用

(1)IO multiplexing

(2)用在什么地方？多路非阻塞式IO。

(3)select和poll

(4)外部阻塞式，内部非阻塞式自动轮询多路阻塞式IO

3.6.4.2、select函数介绍

3.6.4.3、poll函数介绍

3.6.5.IO多路复用实践

3.6.5.1、用select函数实现同时读取键盘鼠标

3.6.5.2、用poll函数实现同时读取键盘鼠标

3.6.6.异步IO

3.6.6.1、何为异步IO

(1)几乎可以认为：异步IO就是操作系统用软件实现的一套中断响应系统。

(2)异步IO的工作方法是：我们当前进程注册一个异步IO事件（使用signal注册一个信号SIGIO的处理函数），然后当前进程可以正常处理自己的事情，当异步事件发生后当前进程会收到一个SIGIO信号从而执行绑定的处理函数去处理这个异步事件。

3.6.6.2、涉及的函数：

(1)fcntl（F\_GETFL、F\_SETFL、O\_ASYNC、F\_SETOWN）

(2)signal或者sigaction（SIGIO）

3.6.3.代码实践

3.6.7.存储映射IO

3.6.7.1、mmap函数

3.6.7.2、LCD显示和IPC之共享内存

3.6.7.3、存储映射IO的特点

(1)共享而不是复制，减少内存操作

(2)处理大文件时效率高，小文件不划算

\*\*\*\*\*\*\*\*\*\*\*《朱有鹏老师嵌入式linux核心课程》 \*\*\*\*\*\*\*\*\*\*\*

《3.linux应用编程和网络编程-第7部分-3.7.linux线程全解》

--------------------------------------------------------

本课程由朱老师物联网大讲堂推出并提供技术支持，课件可打包下载

网盘地址：http://yunpan.cn/cjVy3RAgfDufK 访问密码 4ad7

技术交流QQ群：朱老师物联网讲堂1群 397164505

--------------------------------------------------------

第一部分、章节目录

3.7.1.再论进程

3.7.2.线程的引入

3.7.3.线程常见函数

3.7.4.线程同步之信号量1

3.7.5.线程同步之信号量2

3.7.6.线程同步之互斥锁

3.7.7.线程同步之条件变量

第二部分、章节介绍

3.7.1.再论进程

本节再次回顾进程，使用多进程的方式解决同时读取键盘和鼠标的问题，从而让大家体会进程的调度特性带来的IO并发解决方案。

3.7.2.线程的引入

本节使用线程再次重写解决同时读取键盘和鼠标的问题，让大家体会进程和线程的相同之处，然后再讲解线程机制的优势。

3.7.3.线程常见函数

本节对线程编程中常用的一些主要函数进行讲解，其实围绕着讲解线程编程的细节也就出来了。

3.7.4.线程同步之信号量1

本节开始讲解线程同步，主要是线程同步的概念和原因，然后开始讲线程同步的第一种方法信号量。

3.7.5.线程同步之信号量2

本节接上节讲解信号量来做线程同步，并且从零开始写了一个案例来演示信号量如何实现线程同步。

3.7.6.线程同步之互斥锁

本节引入互斥锁，并且使用互斥锁技术来实现线程同步，再进行代码实践。

3.7.7.线程同步之条件变量

本节讲解线程同步的第三种犯法条件变量，并且进行代码编程实践使用条件变量来完成上节课的例子。

第三部分、随堂记录

3.7.1.再论进程

3.7.1.1、多进程实现同时读取键盘和鼠标

3.7.1.2、使用进程技术的优势

(1)CPU时分复用，单核心CPU可以实现宏观上的并行

(2)实现多任务系统需求（多任务的需求是客观的）

3.7.1.3、进程技术的劣势

(1)进程间切换开销大

(2)进程间通信麻烦而且效率低

3.7.1.4、解决方案就是线程技术

(1)线程技术保留了进程技术实现多任务的特性。

(2)线程的改进就是在线程间切换和线程间通信上提升了效率。

(3)多线程在多核心CPU上面更有优势。

3.7.2.线程的引入

3.7.2.1、使用线程技术同时读取键盘和鼠标

3.7.2.2、linux中的线程简介

(1)一种轻量级进程

(2)线程是参与内核调度的最小单元

(3)一个进程中可以有多个线程

3.7.2.3、线程技术的优势

(1)像进程一样可被OS调度

(2)同一进程的多个线程之间很容易高效率通信

(3)在多核心CPU（对称多处理器架构SMP）架构下效率最大化

3.7.3.线程常见函数

3.7.3.1、线程创建与回收

(1)pthread\_create 主线程用来创造子线程的

(2)pthread\_join 主线程用来等待（阻塞）回收子线程

(3)pthread\_detach 主线程用来分离子线程，分离后主线程不必再去回收子线程

3.7.3.2、线程取消

(1)pthread\_cancel 一般都是主线程调用该函数去取消（让它赶紧死）子线程

(2)pthread\_setcancelstate 子线程设置自己是否允许被取消

(3)pthread\_setcanceltype

3.7.3.3、线程函数退出相关

(1)pthread\_exit与return退出

(2)pthread\_cleanup\_push

(3)pthread\_cleanup\_pop

3.7.3.4、获取线程id

(1)pthread\_self

3.7.4\_5.线程同步之信号量12

3.7.4.1、任务：用户从终端输入任意字符然后统计个数显示，输入end则结束

3.7.4.2、使用多线程实现：主线程获取用户输入并判断是否退出，子线程计数

(1)为什么需要多线程实现

(2)问题和困难点是？

(3)理解什么是线程同步

3.7.4.3、信号量的介绍和使用

3.7.6.线程同步之互斥锁

3.7.6.1、什么是互斥锁

(1)互斥锁又叫互斥量（mutex）

(2)相关函数：pthread\_mutex\_init pthread\_mutex\_destroy

pthread\_mutex\_lock pthread\_mutex\_unlock

(3)互斥锁和信号量的关系：可以认为互斥锁是一种特殊的信号量

(4)互斥锁主要用来实现关键段保护

3.7.6.2、用互斥锁来实现上节的代码

注意：man 3 pthread\_mutex\_init时提示找不到函数，说明你没有安装pthread相关的man手册。安装方法：1、虚拟机上网；2、sudo apt-get install manpages-posix-dev

3.7.7.线程同步之条件变量

3.7.7.1、什么是条件变量

3.7.7.2、相关函数

pthread\_cond\_init pthread\_cond\_destroy

pthread\_cond\_wait pthread\_cond\_signal/pthread\_cond\_broadcast

3.7.7.3、使用条件变量来实现上节代码

3.7.7.4、线程同步总结

\*\*\*\*\*\*\*\*\*\*\*《朱有鹏老师嵌入式linux核心课程》 \*\*\*\*\*\*\*\*\*\*\*

《3.linux应用编程和网络编程-第8部分-3.8.网络基础》

--------------------------------------------------------

本课程由朱老师物联网大讲堂推出并提供技术支持，课件可打包下载

网盘地址：http://yunpan.cn/cjVy3RAgfDufK 访问密码 4ad7

技术交流QQ群：朱老师物联网讲堂1群 397164505

--------------------------------------------------------

第一部分、章节目录

3.8.1.网络通信概述

3.8.2.网络通信基础知识1

3.8.3.网络通信基础知识2

3.8.4.网络通信基础知识3

3.8.5.网络通信基础知识4

3.8.6.网络通信基础知识5

第二部分、章节介绍

3.8.1.网络通信概述

本节课从进程间通信开始讲起并引出网络通信，然后分析了网络通信的实质，最后讲了网络编程部分的学习方法和注意事项。

3.8.2.网络通信基础知识1

本节开始讲解网络通信的基础知识，主要是网络通信的发展历史、三大网络和网络通信的传输媒介。

3.8.3.网络通信基础知识2

本节讲了OSI 7层网络模式、网卡、集线器、交换机等网络设备的功能和作用。

3.8.4.网络通信基础知识3

本节讲解路由器的工作原理和作用，以及DNS的概念。

3.8.5.网络通信基础知识4

本节讲解DHCP和NAT这两个网络通信概念，并通过这两个概念让大家对网络通信有更深入理解，方便后续学习。

3.8.6.网络通信基础知识5

本节讲了IP地址的分类，和子网掩码的作用，以及子网的判断。

第三部分、随堂记录

3.8.1.网络通信概述

3.8.1.1、从进程间通信说起：网络域套接字socket，网络通信其实就是位于网络中不同主机上面的2个进程之间的通信。

3.8.1.2、网络通信的层次

(1)硬件部分：网卡

(2)操作系统底层：网卡驱动

(3)操作系统API：socket接口

(4)应用层：低级（直接基于socket接口编程）

(5)应用层：高级（基于网络通信应用框架库）

(6)应用层：更高级（http、网络控件等）

3.8.1.3、本部分学习方法

(1)重点1：掌握网络通信的架构层次和基本原理

(2)重点2：掌握socket及其相关函数的使用

(3)重点3：掌握服务器和客户端程序通信的方法

3.8.2.网络通信基础知识1

3.8.2.1、网络通信的发展历程

(1)单机阶段

(2)局域网阶段

(3)广域网internet阶段

(4)移动互联网阶段

(5)物联网阶段

3.8.2.2、三大网络

(1)电信网、电视网络、互联网

3.8.2.3、网络通信的传输媒介

(1)无线传输：WIFI、蓝牙、zigbee、4G/5G/GPRS等

(2)有线通信：双绞线、同轴电缆、光纤等

3.8.3.网络通信基础知识2

3.8.3.1、OSI 7层网络模型（详见百度介绍）

(1)7层名字和顺序要记住，有时候笔试题目经常遇到。

(2)网络搜索资料，自己看自学，逐步去理解。

3.8.3.2、网卡

(1)计算机上网必备硬件设备，CPU靠网卡来连接外部网络

(2)串转并设备

(3)数据帧封包和拆包

(4)网络数据缓存和速率适配

3.8.3.3、集线器（HUB）

(1)信号中继放大，相当于中继器

(2)组成局域网络，用广播方式工作。

(3)注意集线器是不能用来连接外网的

3.8.3.4、交换机

(1)包含集线器功能，但更高级

(2)交换机中有地址表，数据包查表后直达目的通信口而不是广播

(3)找不到目的口时广播并学习

3.8.4.网络通信基础知识3

3.8.4.1、路由器

(1)路由器是局域网和外部网络通信的出入口

(2)路由器将整个internet划分成一个个的局域网，却又互相联通。

(3)路由器对内管理子网（局域网），可以在路由器中设置子网的网段，设置有线端口的IP地址，设置dhcp功能等，因此局域网的IP地址是路由器决定的。

(4)路由器对外实现联网，联网方式取决于外部网络（如ADSL拨号上网、宽带帐号、局域网等）。这时候路由器又相当于是更高层级网络的其中一个节点而已。

(5)所以路由器相当于有2个网卡，一个对内做网关、一个对外做节点。

(6)路由器的主要功能是为经过路由器的每个数据包寻找一条最佳路径（路由）并转发出去。其实就是局域网内电脑要发到外网的数据包，和外网回复给局域网内电脑的数据包。

(7)路由器技术是网络中最重要技术，决定了网络的稳定性和速度。

3.8.5.1、DNS（Domain Name Service 域名服务）

(1)网络世界的门牌号：IP地址

(2)IP地址的缺点：难记、不直观

(3)IP地址的替代品：域名，譬如www.zhulaoshi.org

(4)DNS服务器就是专门提供域名和IP地址之间的转换的服务的，因此域名要购买的

(5)我们访问一个网站的流程是：先使用IP地址（譬如谷歌的DNS服务器IP地址为8.8.8.8）访问DNS服务器（DNS服务器不能是域名，只能是直接的IP地址），查询我们要访问的域名的IP地址，然后再使用该IP地址访问我们真正要访问的网站。这个过程被浏览器封装屏蔽，其中使用的就是DNS协议。

(6)浏览器需要DNS服务，而QQ这样的客户端却不需要（因为QQ软件编程时已经知道了腾讯的服务器的IP地址，因此可以直接IP方式访问服务器）

3.8.5.网络通信基础知识4

3.8.5.1、DHCP（dynamic host configuration protocl，动态主机配置协议）

(1)每台计算机都需要一个IP地址，且局域网内各电脑IP地址不能重复，否则会地址冲突。

(2)计算机的IP地址可以静态设定，也可以动态分配

(3)动态分配是局域网内的DHCP服务器来协调的，很多设备都能提供DHCP功能，譬如路由器。

(4)动态分配的优势：方便接入和断开、有限的IP地址得到充分利用

3.8.5.2、NAT（network address translation，网络地址转换协议）

(1)IP地址分为公网IP（internet范围内唯一的IP地址）和私网IP（内网IP），局域网内的电脑使用的都是私网IP（常用的就是192.168.1.xx）

(2)网络通信的数据包中包含有目的地址的IP地址

(3)当局域网中的主机要发送数据包给外网时，路由器要负责将数据包头中的局域网主机的内网IP替换为当前局域网的对外外网IP。这个过程就叫NAT。

(4)NAT的作用是缓解IPv4的IP地址不够用问题，但只是类似于打补丁的形式，最终的解决方案还是要靠IPv6。

(5)NAT穿透简介

3.8.6.网络通信基础知识5

3.8.6.1、IP地址分类（IPv4）

(1)IP地址实际是一个32位二进制构成，在网络通信数据包中就是32位二进制，而在人机交互中使用点分十进制方式显示。

(2)IP地址中32位实际包含2部分，分别为：网络地址和主机地址。子网掩码，用来说明网络地址和主机地址各自占多少位。

(3)由网络地址和主机地址分别占多少位的不同，将IP地址分为5类，最常用的有3类

3.8.6.2、三类IP地址

(1)A类。

(2)B类

(3)C类

(4)127.0.0.0用来做回环测试loopback

3.8.6.3、如何判断2个IP地址是否在同一子网内

(1)网络标识 = IP地址 & 子网掩码

(2)2个IP地址的网络标识一样，那么就处于同一网络。

源IP地址：发出数据包的网络的IP地址

目标IP地址：要接收数据包的计算机的IP地址

二进制方式 0xffffffff 0xC0A80166/0x6601A8C0 本质

点分十进制方式 255.255.255.255 192.168.1.102 方便人看的

IP地址 = 网络地址 + 主机地址

网络地址用来表示子网

主机地址是用来表示子网中的具体某一台主机的。

譬如可以8位表示网络，24位表示主机

也可以16位表示网络，16位表示主机

14为表示网络，18位表示主机

子网掩码为255.255.255.0时表示前24位为网络地址，后8位为主机地址

子网掩码为255.255.0.0时表示前16位为网络地址，后16位为主机地址

网络地址决定了这种网络中一定可以有多少个网络，譬如子网掩码为255.255.255.0时表示我们这一种网络一共最多可以有2^24个，每个这种网络中可以有2^8个主机。

如果子网掩码为255.255.0.0时，表示我们这种网络可以有2^16个网络，每个这种网络中最多可以有2^16个主机。

192.168.1.102 & 255.255.255.0 = 192.168.1.0

192.168.1.253 & 255.255.255.0 = 192.168.1.0

192.168.1.4和192.168.12.5，如果子网掩码是255.255.255.0那么不在同一网段，如果子网掩码是255.255.0.0那么就在同一个网段

\*\*\*\*\*\*\*\*\*\*\*《朱有鹏老师嵌入式linux核心课程》 \*\*\*\*\*\*\*\*\*\*\*

《3.linux应用编程和网络编程-第9部分-3.9.linux网络编程实践》

--------------------------------------------------------

本课程由朱老师物联网大讲堂推出并提供技术支持，课件可打包下载

网盘地址：http://yunpan.cn/cjVy3RAgfDufK 访问密码 4ad7

技术交流QQ群：朱老师物联网讲堂1群 397164505

--------------------------------------------------------

第一部分、章节目录

3.9.1.linux网络编程框架

3.9.2.TCP协议的学习1

3.9.3.TCP协议的学习2

3.9.4.socket编程接口介绍

3.9.5.IP地址格式转换函数实践

3.9.6.soekct实践编程1

3.9.7.soekct实践编程2

3.9.8.socket实践编程3

3.9.9.socket编程实践4

第二部分、章节介绍

3.9.1.linux网络编程框架

本节讲述网络编程的框架，分层思想和TCP/IP协议的介绍，BS架构和CS架构的介绍等。

3.9.2.TCP协议的学习1

本节详细介绍TCP协议的特点，其中重点讲述了TCP协议保证实现可靠传输的机制。

3.9.3.TCP协议的学习2

本节接上节继续讲解TCP协议，主要讲了TCP协议建立连接和关闭连接时的握手方法，最后讲了使用TCP来实现的常见应用层协议。

3.9.4.socket编程接口介绍

本节介绍linux API中与网络编程相关的接口函数，后面的实战编程中都要用到这些函数。

3.9.5.IP地址格式转换函数实践

本节通过代码实践来给大家演示IP地址格式转换的几个函数

3.9.6.soekct实践编程1

本节开始编写基于TCP的客户端和服务器连接通信程序

3.9.7.soekct实践编程2

本节接上节继续编写，并且已经实现客户端和服务器的连接。

3.9.8.socket实践编程3

本节实现客户端和服务器之间的任意发送和接收、反复发送接收等功能

3.9.9.socket编程实践4

本节通过定义一个简单的应用层协议，来向大家介绍TCP连接建立后如何通过应用层协议来实现业务逻辑。

第三部分、随堂记录

3.9.1.linux网络编程框架

3.9.1.1、网络是分层的

(1)OSI 7层模型

(2)网络为什么要分层

(3)网络分层的具体表现

3.9.1.2、TCP/IP协议引入

(1)TCP/IP协议是用的最多的网络协议实现

(2)TCP/IP分为4层，对应OSI的7层

(3)我们编程时最关注应用层，了解传输层，网际互联层和网络接入层不用管

3.9.1.3、BS和CS

(1)CS架构介绍（client server，客户端服务器架构）

(2)BS架构介绍（broswer server，浏览器服务器架构）

3.9.2.TCP协议的学习1

3.9.2.1、关于TCP理解的重点

(1)TCP协议工作在传输层，对上服务socket接口，对下调用IP层

(2)TCP协议面向连接，通信前必须先3次握手建立连接关系后才能开始通信。

(3)TCP协议提供可靠传输，不怕丢包、乱序等。

3.9.2.2、TCP如何保证可靠传输

(1)TCP在传输有效信息前要求通信双方必须先握手，建立连接才能通信

(2)TCP的接收方收到数据包后会ack给发送方，若发送方未收到ack会丢包重传

(3)TCP的有效数据内容会附带校验，以防止内容在传递过程中损坏

(4)TCP会根据网络带宽来自动调节适配速率（滑动窗口技术）

(5)发送方会给各分割报文编号，接收方会校验编号，一旦顺序错误即会重传。

3.9.3.TCP协议的学习2

3.9.3.1、TCP的三次握手

(1)建立连接需要三次握手

(2)建立连接的条件：服务器listen时客户端主动发起connect

3.9.3.2、TCP的四次握手

(3)关闭连接需要四次握手

(4)服务器或者客户端都可以主动发起关闭

注：这些握手协议已经封装在TCP协议内部，socket编程接口平时不用管

3.9.3.3、基于TCP通信的服务模式

(1)具有公网IP地址的服务器（或者使用动态IP地址映射技术）

(2)服务器端socket、bind、listen后处于监听状态

(3)客户端socket后，直接connect去发起连接。

(4)服务器收到并同意客户端接入后会建立TCP连接，然后双方开始收发数据，收发时是双向的，而且双方均可发起

(5)双方均可发起关闭连接

3.9.3.4、常见的使用了TCP协议的网络应用

(1)http、ftp

(2)QQ服务器

(3)mail服务器

3.9.4.socket编程接口介绍

3.9.4.1、建立连接

(1)socket。socket函数类似于open，用来打开一个网络连接，如果成功则返回一个网络文件描述符（int类型），之后我们操作这个网络连接都通过这个网络文件描述符。

(2)bind

(3)listen

(4)connect

3.9.4.3、发送和接收

(1)send和write

(2)recv和read

3.9.4.4、辅助性函数

(1)inet\_aton、inet\_addr、inet\_ntoa

(2)inet\_ntop、inet\_pton

3.9.4.5、表示IP地址相关数据结构

(1)都定义在 netinet/in.h

(2)struct sockaddr，这个结构体是网络编程接口中用来表示一个IP地址的，注意这个IP地址是不区分IPv4和IPv6的（或者说是兼容IPv4和IPv6的）

(3)typedef uint32\_t in\_addr\_t; 网络内部用来表示IP地址的类型

(4)struct in\_addr

{

in\_addr\_t s\_addr;

};

(5)struct sockaddr\_in

{

\_\_SOCKADDR\_COMMON (sin\_);

in\_port\_t sin\_port; /\* Port number. \*/

struct in\_addr sin\_addr; /\* Internet address. \*/

/\* Pad to size of `struct sockaddr'. \*/

unsigned char sin\_zero[sizeof (struct sockaddr) -

\_\_SOCKADDR\_COMMON\_SIZE -

sizeof (in\_port\_t) -

sizeof (struct in\_addr)];

};

(6)struct sockaddr 这个结构体是linux的网络编程接口中用来表示IP地址的标准结构体，bind、connect等函数中都需要这个结构体，这个结构体是兼容IPV4和IPV6的。在实际编程中这个结构体会被一个struct sockaddr\_in或者一个struct sockaddr\_in6所填充。

3.9.5.IP地址格式转换函数实践

3.9.5.1、inet\_addr、inet\_ntoa、inet\_aton

3.9.5.2、inet\_pton、inet\_ntop

?

3.9.6\_7.soekct实践编程1\_2

3.9.6.1、服务器端程序编写

(1)socket

(2)bind

(3)listen

(4)accept，返回值是一个fd，accept正确返回就表示我们已经和前来连接我的客户端之间建立了一个TCP连接了，以后我们就要通过这个连接来和客户端进行读写操作，读写操作就需要一个fd，这个fd就由accept来返回了。

注意：socket返回的fd叫做监听fd，是用来监听客户端的，不能用来和任何客户端进行读写；accept返回的fd叫做连接fd，用来和连接那端的客户端程序进行读写。

3.9.6.2、客户端程序编写

(1)socket

(2)connect

概念：端口号，实质就是一个数字编号，用来在我们一台主机中（主机的操作系统中）唯一的标识一个能上网的进程。端口号和IP地址一起会被打包到当前进程发出或者接收到的每一个数据包中。每一个数据包将来在网络上传递的时候，内部都包含了发送方和接收方的信息（就是IP地址和端口号），所以IP地址和端口号这两个往往是打包在一起不分家的。

3.9.8.socket实践编程3

3.9.8.1、客户端发送&服务器接收

3.9.8.2、服务器发送&客户端接收

3.9.8.3、探讨：如何让服务器和客户端好好沟通

(1)客户端和服务器原则上都可以任意的发和收，但是实际上双方必须配合：client发的时候server就收，而server发的时候client就收

(2)必须了解到的一点：client和server之间的通信是异步的，这就是问题的根源

(3)解决方案：依靠应用层协议来解决。说白了就是我们server和client事先做好一系列的通信约定。

3.9.9.socket编程实践4

3.9.9.1、自定义应用层协议第一步：规定发送和接收方法

(1)规定连接建立后由客户端主动向服务器发出1个请求数据包，然后服务器收到数据包后回复客户端一个回应数据包，这就是一个通信回合

(2)整个连接的通信就是由N多个回合组成的。

3.9.9.2、自定义应用层协议第二步：定义数据包格式

3.9.9.3、常用应用层协议：http、ftp······

3.9.9.4、UDP简介