北京科海培训中心

UNIX 系统程序员进阶

林新观 编著

-81
/1

清华大学出版社

北京科海培训中心

UNIX 系统程序员进阶

林新观　编著

清华大学出版社

(京)新登字 158 号

内 容 提 要

本书翔实地叙述了 UNIX 系统中至关重要的系统调用和库函数的使用技巧，重点地描述了 C 程序在 UNIX 系统各基本领域中的应用。全书共分为 14 章，内容包括：UNIX 系统的 I/O 机制；设备 I/O 的控制；在系统程序中操作文件和目录的方法；在程序中获取有关用户和时间等系统信息；在程序中处理信号和中断的方法；如何产生进程和常驻进程的方法；如何在程序中实现任务控制；内部进程之间的通信与网络通信的方法；仿真终端的使用等。

本书面向 UNIX 系统程序员和开发人员，同时本书也可以作为高等院校计算机系师生的教材及参考书。

版权所有，盗版必究。

本书封面贴有清华大学出版社激光防伪标签，无标签者不得进入各书店。

书　名：UNIX 系统程序员进阶
作　者：林新观
出版者：清华大学出版社(北京清华大学校内，邮编 100084)
印刷者：北京市科普印刷厂印刷
发　行：新华书店总店北京科技发行所
开　本：16 印张：9.625 字数：231 千字
版　次：1996 年 3 月第 1 版　1996 年 3 月第 1 次印刷
印　数：00001～5000
书　号：ISBN 7-302-02210-0/TP・1065
定　价：19.00 元

目 录

此为试读,需要完整PDF请访问: www.ertongbook.com

第1章 导 引

在过去几年里，UNIX 操作系统已遍及于工作站，而使用了 UNIX 的个人电脑，则变得更为方便和强有力。有关 UNIX 系统以及 C 语言的使用已经出版了许多书，然而，把 C 程序与 UNIX 操作系统有机地结合在一起的书却不多。这样，那些希望在 UNIX 系统下写系统程序的人就必须经常阅读由操作系统提供的不很充分的文档，或者查阅已经实现的系统工具的源代码。虽然这样做可以了解更为复杂的内容，但对于那些希望事半功倍的系统程序员来说，却不是一个最好的办法。

本书试图给编写系统程序的人提供一条简捷的途径，以学会如何成为一个职业(Professional)系统程序员。书中详细讨论了大多数系统调用和库函数的使用技巧，有效地描述了 UNIX 操作系统各个基本方面 C 程序的使用方法。

请注意，本书不是侧重于介绍 C 程序，更不是一本 C 程序员的高级指南。本书是为那些不但希望成为一个有经验的 C 程序工作者，而且更希望成为一个有经验的 UNIX 系统程序工作者的人而写的。因此，对于那些在大学计算机系、研究所和软件中心等科研单位工作的，希望在个人电脑或工作站上的 UNIX 系统中实现他们自己的系统工具或系统函数，或者为了工作需要而必须修改系统程序的人们来说，本书一定会使他们有所裨益。

下面先介绍几个书中用到的概念和术语，然后说明本书描述的重点。

1.1 系统调用和库函数

在讨论由 UNIX 系统提供的库函数与系统调用之前，首先要解释一下这两者之间的区别。这些术语经常会被用错，即使是对那些应该知道的人们来说也是如此。

就像其名字所暗示的那样，一个系统调用是指一个需要操作系统代表用户程序来执行某些任务的请求。例如：read 是一个系统调用，它请求操作系统用存储在一个磁盘设备(或其他设备)上的数据去填充一个缓冲区。如果任何人在他们想执行任务的时候都能随便访问设备，那将引起一场大的灾难。所以，这种服务必须请求操作系统来做，它(经常是透明的)记录所有处理每个设备的请求。

而一个库函数，并不经常需要操作系统来执行它的任务。库函数的一个实例是 sin 函数，它计算一个角度的正弦值。由于这个计算只需要简单地对一个有限序列求和，所以并不需要操作系统干预。

1.2 UNIX 的 版本

本书介绍的重点是基于加州大学伯克利分校的 UNIX 版本和 AT&T 的 UNIX 系统 V 版本。虽然系统 V 通常扮演了“标准”UNIX 的角色，但是我们更侧重于伯克利的 UNIX 环境，这是因为：

(1) 在美国，大多数大学和政府的计算中心都使用 BSD UNIX，而不是系统 V。形成这种局面的原因有好几个，但是其中有两个也许是最主要的：一是 BSD UNIX 提供的网络功能直到最近还不能在系统 V 上完全达到；二是 BSD UNIX 更适合于作为一个研究环境，比如提供了快速文件系统和较为先进的虚拟内存处理，以及大量的程序语言。

(2) 在我国 SUN 工作站占有率是非常大，而 SUN 系统(一个最大的可以安装 UNIX 系统的工作站)使用了基于 BSD 的操作系统。虽然 SUN 出于兼容的需要已开始向系统 V 偏移，但是与其他系统相比，SUN 的系统仍然更像 BSD UNIX。一些其他的厂商，像 IBM 等，也提供 BSD UNIX 版本给他们在大学里的工作站客户。

(3) 老的 UNIX 的变迁，如来自 Bell 实验室的第 7 版和早期的 Xenix 和 Venix 版本，也更接近于 BSD 程序环境而非系统 V 的环境。

当然，系统 V 也同样被相当广泛地使用。出于这个原因，系统 V 与 BSD 之间有较大区别的地方都将在书中描述。这一点特别体现在第 5 章“设备输入/输出控制”和 第 11 章“内部进程通信”中。在这些章节中，提供了对 BSD UNIX 和系统 V 两者环境的完整描述以及实例。

1.3 错 误 处 理

对于标准输入/输出库中所有的函数，当错误出现时都会返回一个事先定义好的常量 EOF 或 NULL。其他的库函数在错误出现时通常返回 −1 或 0 (主要取决于它们返回的值是什么类型)，但有一些函数可以返回不同的值以指出几种不同错误中的某一种。与库函数不同的是，系统调用在指出错误出现的表达方式上是相同的。每一个系统调用在错误出现时返回 −1，并且在完全成功时大多数返回 0 (除非需要它们返回一些其他的整数值)。此外，外部整数变量 errno 被置为一个准确指出出现什么错误的数值。这些错误的“值”被定义在引用文件 errno.h 中，并且用库函数 perror 可以很容易打印出这些错误值。

在当前的 UNIX 版本中，如系统 V 和 4.3 BSD，标准的输入/输出(stdio)函数都设定一个适当的 errno，以便 perror 可结合它们一起使用。遗憾的是，stdio 的早期版本没有设置适当的 errno。所以，perror 不能与这些早期的函数一起使用。

错误的处理是重要的。一个好的程序是不会在一个未曾预料的错误中死去的。程序应该通过检查那些由于执行失败而会带来问题的系统调用和库函数的返回码，以便应付这种不可避免的意外的局面出现。

接下来我们介绍一下本书的结构，它是以一种 "自底向上" 的方式来组织的，在书中首先描述方法，接着介绍执行简单任务的程序实例，然后是在前面描述过的基础上实现一些较为复杂的程序实例。

本书从 UNIX 系统程序工作者的角度出发，结合实例讨论了下面的内容：

- 高级的和低级的输入/输出机制
- 操作文件和目录的方法
- 设备(特别文件) 输入/输出的控制
- 获取关于用户的信息

此为试读，需要完整PDF请访问：www.ertongbook.com

- 从系统中取得时间以及对各类事件记时的方法
- 信号处理与中断机制
- 建立进程和执行程序
- 任务控制
- 内部进程通信的方法
- 使用 TCP/IP 进行网络通信(客户/服务器方式的互联网)
- 文件系统的内部结构
- 仿真终端的使用
- 读取核心数据结构

第 2 章 标准输入/输出库

一个学习 C 语言如何使用的程序员，通常就是学会如何使用标准输入/输出库(stdio)中的库函数来执行输入和输出。这些库函数执行所谓的高级输入和输出。也就是说，这些库函数为程序员提供了三个方面的主要功能：

- 自动开辟缓冲区。即使一次读或写的数据只有几个字节，库函数仍然在大到由数千个字节组成的“块”中执行实际输入或输出（缓冲区的大小通常由定义在引用文件 stdio.h 中的常量 BUFSIZE 确定）。库函数似乎在读或写很小的单元，但数据被实际存放在一个缓冲区中。这个缓冲区在内部开辟给库函数使用，并且对于程序员来说是不可见的。
- 自动执行输入和输出转换。例如，在使用 printf 函数打印一个整数(带有符号 %d)时，这个整数的字符表达式被实际打印出来。同样，在使用 scanf 时，一个整数的字符表达式被转换为它的数据值。
- 输入和输出被自动格式化。也就是说，可以使用字段宽度并且以任何需要的格式来打印数据和字符串。

本章提供了最为普通的、使用标准输入/输出库中库函数的一个综述。

2.1 文件指针

在标准输入/输出库中，一个文件被称为一串字符流，并且被一个指向类型为 FILE 的目标指针所描述，该指针被称为文件指针。FILE 数据类型被定义在引用文件 stdio.h 中，在使用任何 stdio 函数之前，它应该被定义进去。文件指针 stdin、stdout 和 stderr 是预先定义好的，它们分别对应于标准输入(键盘)、标准输出(终端屏幕)和标准错误输出。

大部分 stdio 函数都需要一个文件指针来对应传递给它们的一串打开的字符流。然而，当从标准输入中读或往标准输出中写时，stdio 提供了“简便”的函数来使用这些字符流中的某一个，而无需它们都被指定。表 2-1 示出了这些函数以及和它们等价的函数。

表 2-1 用于标准输入和输出的简便函数

“简便”函数	等价函数
getchar( )	fgetc(stdin), getc(stdin)
gets(buf)	fgets(buf, BUFSIZ, stdin)
printf(args)	fprintf(stdout, args)
putchar(c)	fputc(c, stdout), putc(c, stdout)
puts(buf)	fputs(buf, stdout)
scanf(args)	fsacnf(stdin, args)

2.2　打开和建立文件

要读或写一个文件，必须首先打开这个文件。函数 fopen 就是用来打开文件的。fopen 使用两个参数：一个是字符串，包含要打开的文件名；另一个也是字符串，描述文件应如何打开。该函数返回一个指针，指向类型为 FILE 的打开的字符流；或者返回常量 NULL，表示文件不能被打开。

fopen 的第 2 个参数可能的取值有下面几个：

r　文件将只为读取而打开。如果想成功打开，那么这文件必须事先已经存在。

w　文件将只为写入而打开。如果文件不存在，它将被作为一个空文件建立。如果文件已经存在，则其内容将被破坏。

a　文件将只为写入而打开。如果文件不存在，它将被作为一个空文件建立。如果文件已经存在，则其内容将不被破坏，任何写入文件的数据都将接着写在该文件尾部，而不是覆盖已经存在的内容。

另外，对于较新的标准输入/输出库的版本，可在上述字母后面添加一个 +，使得被打开的文件不但可用于读而且可用于写。需要注意的是，指定 r+ 需要文件已经存在，同时不破坏文件中的内容，当指定 w+ 或 a+ 时，如果文件不存在，就建立它。

2.3　关 闭 文 件

函数 fclose 用来关闭一个打开的字符流。fclose 使用一个参数，即对应于将被关闭的字符流的指针。在调用时，该函数刷新字符流使用的缓冲区，并执行一些其他的内部清除功能。在调用成功时，返回 0；出现错误时，返回常量 EOF。

2.4　读和写文件

标准输入/输出库提供了一些从文件中读写数据的途径。

2.4.1　函数 getc 和 putc

最简单的读和写数据的途径，就是一次处理一个字符(字节)。这可以使用函数 getc 和 putc 来实现。getc 接受一个参数：一个对应于一串为读而打开的字符流的文件指针。该函数返回从这字符流中读取的下一个字符，或者是一个常量 EOF(当已读到文件结尾时)。putc 接受两个参数：一个是要被写入的字符；另一个是对应于一串为写而打开的字符流的文件指针。该函数把字符写入字符流中，在成功时，返回 0；在出现错误时，返回 EOF。

应当注意，虽然 getc 和 putc 一次处理一个字符，但是 stdio 库在上述函数每次被调用时，并不实际去调系统函数来从硬盘中读写。取而代之的是，库为这些字符在内部开辟了一个缓冲区，并且只为每数千个字符调用一次系统调用。这样，即使处理非常大的文件，仍然是非常有效的。

例 2-1 为把一个文件的内容追加到另一个中去的一个小程序。其中，第 1 个参数指明了其内容要被复制到其他文件中去的文件名，而第 2 个参数指明了要被追加内容的文件名。如果要被追加内容的文件不存在，它将被建立。

例 2-1 aFILEchar——逐个字符地把一个文件追加到另一个文件中。

```
#include <stdio.h>
main(argc, argv)
int argc;
char * * argv;
{
    int c;
    FILE * source, * target;
    if (argc ! = 3) { /* 检查参数个数 */
     fprintf(stderr, "Usage: %s source—file target—file\n", * argv);
     exit(1);
    }
   if ((source = fopen(argv[1], "r")) == NULL) { /* 打开源文件用于读
*/
      perror(argv[1]);
      exit(1);
   }
  if ((target = fopen(argv[2], "a")) == NULL) {/* 打开目标文件用于追加
*/
    perror(argv[2]);
    exit(1);
  }
  while ((c = getc(source)) ! = EOF) /* 逐个字符地读源文件，并写入目标文件
*/
  putc(c, target);
  fclose(source);
  fclose(target);
  exit(0);
  }
```

为了简明地强调本章讨论的内容，例 2-1(以及下面的例子)违反了一个重要的 UNIX 协定：任何程序都要有意义，程序应该操作两个有实际意义的文件，或是在它的标准输入和输出上工作。文本格式化程序 tbl，eqn，nroff 和 troff 都是好的程序例子。给出一列文件名，这些程序将打开文件并处理它们中的数据。然而，如果没有给出文件名，这些程序将从它们的标准输入中读取数据。这使得程序可作为过滤器来使用，这样它们就能被个别调用，

或者作为管道线的一部分。

2.4.2　函数 fgets 和 fputs

标准输入/输出库提供的读写文件的另一个途径，是允许程序一次处理一行数据。一行被定义为由一个换行字符终止的一串由 0 和其他字符组成的字符串。函数 fgets 接受 3 个参数：一个指针指向要被填充的字符缓冲区；一个整数指出这缓冲区的大小；一个文件指针对应一个为读而打开的字符流。在调用成功时，返回一个指向被填满的缓冲区的指针；或返回常量 NULL(在文件到达结尾时)。缓冲区将被一行字符填入，包括换行字符，并且用一个空字符来终止。fputs 接受两个参数：一个指针指向一个以空字符终止的字符串；一个文件指针对应一个为写而打开的字符流。在成功调用后，它返回 0；当错误出现时，返回常量 EOF。

例 2-2 为把一个文件追加到另一个文件中去的另一种写法，该程序每次操作一行。常量 BUFSIZ 被定义在引用文件 stdio.h 中，并且针对系统的优化设定一个长度。除非需要一个特别的长度，否则无论何时你使用 stdio 工作时，它都会是一个合适的值。

例 2-2　aFILEline —— 逐行地把一个文件追加到另一个文件中。

```
#include <stdio.h>

main(argc, argv)
int argc;
char * *argv;
{
    FILE *source, *target;
    char line[BUFSIZ];

    if (argc ! = 3) { /* 检查参数个数 */
        fprintf(stderr, "Usage: %s source—file target—file\n", *argv);}
          exit(1);
    }

    if ((source = fopen(argv[1], "r")) == NULL) { /* 打开文件用于读 */
        perror(argv[1]);}
        exit(1);
    }

    if ((target = fopen(argv[2], "a")) == NULL) { /* 打开文件用于追加
*/
        perror(argv[2]);}
        exit(1);
    }

    while (fgets(line, BUFSIZ, source) ! = NULL)  /* 逐行读源文件，并写入
目标文件 */
```

```
        fputs(line, target);
    fclose(source);
    fclose(target);
    exit(0);
}
```

2.4.3 函数 fread 和 fwrite

标准输入/输出库还提供了一种方法，在读写数据时不用去区分它是一个字符还是一行。当工作的文件对象不仅包含正文，同时也包含任意的二进制数据时，这一点通常是非常吸引人的。函数 fread 接受 4 个参数：一个指针指向一个由一些数据组成的数组(字符，整数，结构等等)；一个整数指明数组元素字节的大小；一个整数指明待读的数组元素个数；一个文件指针指向为读而打开的一串字符流。它返回实际读入的数组元素个数，或返回 0(对应于文件的结尾)。函数 fwrite 也接受 4 个参数，同上面 fread 描述的一样。它返回实际写入的数组元素个数，或返回 0(对应于错误出现)。

使用诸如 fread 和 fwrite 这样的函数，优点主要是能够把一个结构强置在输入或输出字符流中，而这种途径 stdio 自身并不提供。例如：如果一个文件包含有 1000 个二进制浮点数，那么读这些数据的最简单的途径就是像下面这段程序所显示的那样：

```
FILE *fPOINT;
float nFLOAT[1000];

......
fread(nFLOAT, sizeof(float), 1000, fPOINT);
......
```

例 2-3 仍然是文件追加程序的又一个版本，这个版本一次复制一整块缓冲区的数据。

例 2-3 aFILEbuf —— 每次整块缓冲区地把一个文件追加到另一个文件中。

```
#include <stdio.h>

main(argc, argv)
int argc;
char **argv;
{
    int n;
    FILE *source, *target;
    char buf[BUFSIZ];

    if (argc != 3) { /* 检查参数个数 */
     fprintf(stderr, "Usage: %s source-file target-file\n", *argv);
     exit(1);
     }
```

此为试读，需要完整PDF请访问：www.ertongbook.com