计 算 机 科 学 丛 书

UNIX 环境高级编程

Advanced Programming in the UNIX Environment

(美) W. Richard Stevens 著

尤晋元 等译

机械工业出版社
China Machine Press

Addison-Wesley

计算机科学丛书

UNIX 环境高级编程

（美） W.Richard Stevens 著

尤晋元 等译

机械工业出版社
China Machine Press

本书全面介绍了UNIX系统的程序设计界面——系统调用界面和标准C库提供的许多函数。

本书的前15章着重于理论知识的阐述，主要内容包括UNIX文件和目录、进程环境、进程控制、进程间通信以及各种I/O。在此基础上，分别按章介绍了多个应用实例，包括如何创建数据库函数库，PostScript 打印机驱动程序，调制解调器拨号器及在伪终端上运行其他程序的程序等。

本书内容丰富权威，概念清晰精辟，一直以来被誉为UNIX编程的“圣经”，对于所有UNIX程序员——无论是初学者还是专家级人士——都是一本无价的参考书籍。

本书版权登记号：图字：01-1999-2858

图书在版编目（CIP）数据

UNIX环境高级编程 /（美）史迪文斯（Stevens，W.R.）著；尤晋元等译. - 北京：机械工业出版社，2000.2
(计算机科学丛书)
书名原文：Advance Programming in the UNIX Environment
ISBN 7-111-07579-X

Ⅰ.U… Ⅱ.①史… ②尤… Ⅲ.UNIX操作系统-程序设计 Ⅳ.TP316.81

中国版本图书馆CIP数据核字(1999)第55299号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）
责任编辑：陈 谊
北京牛山世兴印刷厂印刷 · 新华书店北京发行所发行
2000年2月第1版 · 2001年2月第4次印刷
787mm × 1092mm 1/16 · 34.5印张
印数：16 001-19 000册
定价：55.00元

译 者 序

在UNIX环境下进行编程的程序设计人员需要学习并掌握以下两方面的内容和技术：一方面是要清楚地了解UNIX程序设计界面所提供的各种服务，以及正确应用它们的基本技术；另一个方面是在开发较复杂的软件时，提高综合应用各种系统服务的能力，并找到疑难问题的解决方法。关于UNIX设计与实现的参考文献通常侧重于内核的结构设计，以及各层次的数据结构和算法设计，很难满足UNIX程序设计人员的上述要求。而UNIX的手册性资料虽然篇幅很大，但往往偏重于对界面服务逐一进行说明，缺少深入的技术性分析和较复杂应用技术的介绍。我致力于UNIX操作系统领域研究数十年，也曾力图在此方面作一些有益的工作。1993年在美国访问期间，我的几位学生及朋友向我大力推荐此书，称它是“在UNIX环境下进行程序设计的有关人员必读且经常需查阅的首选参考书”。阅读完此书后，我认为这是一本非常值得向国内读者推荐的好书。此书具有如下主要特点：

(1) 内容丰富实用。书中包含了在UNIX环境下进行程序设计所需的各方面内容和技术。既包含了UNIX手册资料中的关键部分，又包含了综合应用系统调用和C函数库中各函数的技术。既能满足UNIX环境下一般程序设计人员的要求，又能满足要进一步提高应用技巧，解决疑难问题的高级程序设计人员的要求。

(2) 提供了大量应用实例。书中既有说明单个系统调用和库函数使用方法的小程序，也有综合应用它们的较大程序，特别是第16章～第19章，各包含了一个规模较大的程序。这些程序的源代码总计约为10 000行，全部用ANSI C编写。

(3) 为了说明系统调用和库函数的应用技术及其可能发生的各种问题，在必要时对UNIX的内部数据结构和算法进行了说明，这种理论与应用实践的结合，非常有助于读者提高程序设计的水平。

本书第1章～第15章由尤晋元翻译，第16章由刘炀翻译，第17章、第18章由俞茂元翻译，第19章由王鲲翻译，杨怡玲翻译了习题和习题解答。全书由尤晋元统稿。在本书成稿过程中还得到上海交通大学计算机系系统软件研究室的多名青年教师、博士生和硕士生以及张茹玲同志的帮助，在此一并感谢。还要特别感谢机械工业出版社华章公司在本书的策划、编辑及出版方面所做的努力。

1999年11月

译 者 简 介

尤晋元 上海交通大学计算机科学及工程系教授、博士生导师、系主任，中国计算机学会理事，上海计算机学会软件工程专业委员会主任。在科研工作方面，主要从事于操作系统、分布对象计算、移动计算、构件/构架技术和Java技术方面的研究。在教学方面，长期承担操作系统及分布计算等课程的教学工作，主编和翻译了多本操作系统教材和参考书，包括《UNIX操作系统教程》、《UNIX高级编程技术》、《操作系统：设计与实现》等。

前　言

本书说明了UNIX系统的程序设计界面——系统调用界面和标准C库提供的很多函数。这些对编写运行于UNIX系统中的程序非常有帮助。

与大多数操作系统一样，UNIX对程序运行提供了大量的服务——打开文件、读文件、启动一个新程序、分配存储区以及获得当前时间等。这些被称之为系统调用界面（system call interface）。另外，标准C库提供了大量广泛用于C程序中的函数（格式化输出变量的值、比较两个字符串等）。

系统调用界面和库函数可参见《UNIX程序员手册》第2、3部分。本书不是这些内容的重复。手册中没有给出实例及基本原理，而这些则正是本书所要弥补的。

UNIX标准

80年代出现了各种版本的UNIX，80年代后期在此基础上制定了数个国际标准，包括：C程序设计语言的ANSI标准、IEEE POSIX标准族（还在继续制定）、X/Open可移植性指南。

本书也介绍了这些标准，但是并不只是说明标准本身，而是着重说明它们与受到广泛重视的一些实现之间的关系，主要指SVR4以及即将发布的4.4BSD。这就提供了一种对现实世界的说明，而这正是标准本身以及仅描述标准的文献所缺少的。

本书的组织

本书分为6个层次：

(1) 对UNIX程序设计概念和术语的简要描述（第1章），以及对各种UNIX标准化工作和UNIX实现的讨论（第2章）。

(2) 不带缓存的I/O（第3章），文件和目录（第4章），标准I/O库（第5章）以及系统数据文件（第6章）。

(3) 进程——UNIX进程的环境（第7章），进程控制（第8章），进程之间的关系（第9章）以及信号（第10章）。

(4) 终端I/O（第11章），高级I/O（第12章）以及精灵进程（第13章）。

(5) IPC——进程间通信（第14、15章）。

(6) 实例——一个数据库的函数库（第16章），与PostScript 打印机的通信（第17章），调制解调器拨号程序（第18章）以及使用伪终端（第19章）。

如果对C语言较熟悉并具有某些应用UNIX的经验，那么对学习本书将非常有益，但是并不要求读者具有UNIX编程经验。本书面向的主要读者是：熟悉UNIX或其他某个操作系统，希望了解大多数UNIX系统提供的各种服务的详细情况的程序员。

本书中的实例

本书包含了大量实例——大约10 000行源代码。所有实例都用ANSI C语言编写。在阅读本书时，建议准备一本你所使用的UNIX系统的《UNIX程序员手册》，在细节方面有时需要参考

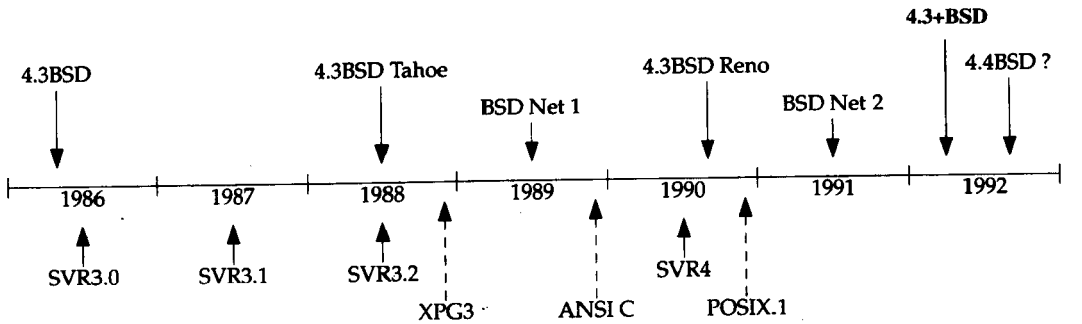
该手册。

几乎对于每一个函数和系统调用，本书都用一个小的完整的程序进行了演示。这可以让读者清楚地了解它们的用法，包括参数、返回值等。有些小程序还不足以说明库函数和系统调用的复杂功能和应用技巧，所以书中还包含了一些较大的实例（见第16～19章）。

所有实例的源代码文件都可在因特网上用匿名ftp从published/books/stevens.advprog.tar.Z下的ftp.uu.net下载。读者可以在自己的机器上修改并运行这些源代码。

用于测试实例的系统

不幸的是，所有的操作系统都在不断变更，UNIX也不例外。下图给出了系统V和4.xBSD最近的进展情况。



4.xBSD是由加州大学伯克利分校计算机系统研究小组开发的。该研究小组还发布了BSD Net1和BSD Net2版，其公开的源代码源自4.xBSD系统。SVRx 表示AT&T的系统V第x版。XPG3指X/Open可移植性指南的第3次发行本。ANSI C是C语言的ANSI标准。POSIX.1是类UNIX系统的IEEE和ISO界面标准。2.2和2.3节将对这些标准和版本之间的差别作更多的说明。

> 4.3+BSD表示介于BSD Net2和4.4BSD之间的UNIX系统。
>
> 在本书写作时，4.4BSD尚未发行，所以不能称一个系统为4.4BSD。为了用一个简单的名字来引用该系统，故使用4.3+BSD。

本书中的大多数实例曾在4种UNIX系统上运行过，它们是：

(1) U.H公司（UHC）的UNIX系统V/386 R4.2（vanilla SVR4），运行于Intel 80386处理机上。

(2) 加州大学伯克利分校计算机科学系计算机系统研究小组的4.3+BSD，运行于HP工作站上。

(3) 伯克利软件设计公司的BSD/386（是BSD Net2的导出版），运行于Intel 80386处理机上。该系统与4.3+BSD几乎相同。

(4) Sun 公司的SunOS 4.1.1和4.1.2（该系统与伯克利系统有很深的渊源，但也包含了许多系统V的特征），运行于SPARC工作站SLC上。

本书还提供了许多时间测试，及用于测试的实际系统。

W.Richard Stevens
于亚利桑那州塔克森市
1992年4月

目　录

第1章 UNIX基础知识

1.1 引言

所有操作系统都向它们运行的程序提供服务。典型的服务有执行新程序、打开文件、读文件、分配存储区、获得当前时间等等，本书集中阐述了UNIX操作系统各种版本所提供的服务。

以严格的步进方式、不超前引用尚未说明过的术语的方式来说明UNIX几乎是不可能的(可能也会是令人厌烦的)。本章从程序设计人员的角度快速浏览UNIX，并对书中引用的一些术语和概念进行简要的说明并给出实例。在以后各章中，将对这些概念作更详细的说明。本章也对不熟悉UNIX的程序设计人员简要介绍了UNIX提供的各种服务。

1.2 登录

1.2.1 登录名

登录UNIX系统时，先键入登录名，然后键入口令。系统在其口令文件，通常是/etc/passwd文件中查看登录名。口令文件中的登录项由7个以冒号分隔的字段组成：登录名，加密口令，数字用户ID(224)，数字组ID(20)，注释字段，起始目录(/home/stevens)，以及shell程序(/bin/ksh)。

很多比较新的系统已将加密口令移到另一个文件中。第6章将说明这种文件以及存取它们的函数。

1.2.2 shell

登录后，系统先显示一些典型的系统信息，然后就可以向shell程序键入命令。shell是一个命令行解释器，它读取用户输入，然后执行命令，用户通常用终端，有时则通过文件(称为shell脚本)向shell进行输入。常用的shell有：

- Bourne shell, /bin/sh
- C shell, /bin/csh
- KornShell, /bin/ksh

系统从口令文件中登录项的最后一个字段中了解到应该执行哪一个shell。

自V7以来，Bourne shell得到了广泛应用，几乎每一个现有的UNIX系统都提供Bourne shell。C shell是在伯克利开发的，所有BSD版本都提供这种shell。另外，AT&T的系统V/386 R3.2和SVR4也提供C shell（下一章将对这些不同的UNIX版本作更多说明)。KornShell是Bourne shell的后继者，它由SVR4提供。KornShell在大多数UNIX系统上运行，但在SVR4之前，通常它需要另行购买，所以没有其他两种shell流行。

本书将使用很多shell实例，以执行已开发的程序，其中将应用Bourne shell和KornShell都具有的功能。

Bourne shell是Steve Bourne在贝尔实验室中开发的，其控制流结构使人想起Algol 68。C shell是在伯克利由Bill Joy完成的，其基础是第6版shell（不是Bourne shell）。其控制结构很像C语言，它支持一些Bourne shell没有提供的功能，如作业控制，历史机制和命令行编辑。KornShell是David Korn在贝尔实验室中开发的，它兼容Bourne shell，并且也包含了使C shell非常流行的一些功能，如作业控制、命令行编译等。

本书将使用这种形式的注释来描述历史，并对不同的UNIX实现进行比较。当我们了解了历史缘由后，采用某种特定实现技术的原因将变得清晰起来。

1.3 文件和目录

1.3.1 文件系统

UNIX文件系统是目录和文件的一种层次安排，目录的起点称为根(root)，其名字是一个字符/。

目录（directory）是一个包含目录项的文件，在逻辑上，可以认为每个目录项都包含一个文件名，同时还包含说明该文件属性的信息。文件属性是：文件类型，文件长度，文件所有者，文件的许可权（例如，其他用户能否能访问该文件），文件最后的修改时间等。stat和fstat函数返回一个包含所有文件属性的信息结构。第4章将详细说明文件的各种属性。

1.3.2 文件名

目录中的各个名字称为文件名（filename）。不能出现在文件名中的字符只有两个，斜线(/)和空操作符（null）。斜线分隔构成路径名(在下面说明)的各文件名，空操作符则终止一个路径名。尽管如此，好的习惯是只使用印刷字符的一个子集作为文件名字符(只使用子集的理由是：如果在文件名中使用了某些shell特殊字符，则必须使用shell的引号机制来引用文件名)。

当创建一个新目录时，自动创建了两个文件名：.(称为点)和..(称为点-点)。点引用当前目录，点-点则引用父目录。在最高层次的根目录中，点-点与点相同。

某些UNIX文件系统限制文件名的最大长度为14个字符，BSD版本则将这种限制扩展为255个字符。

1.3.3 路径名

0个或多个以斜线分隔的文件名序列(可以任选地以斜线开头)构成路径名（pathname），以斜线开头的路径名称为绝对路径名（absolute pathname），否则称为相对路径名（relative pathname）。

实例

不难列出一个目录中所有文件的名字，程序1-1是ls(1)命令的主要实现部分

程序1-1 列出一个目录中的所有文件

```
#include    <sys/types.h>
#include    <dirent.h>
```

```
#include    "ourhdr.h"

int
main(int argc, char *argv[])
{
    DIR             *dp;
    struct dirent   *dirp;

    if (argc != 2)
        err_quit("a single argument (the directory name) is required");

    if ( (dp = opendir(argv[1])) == NULL)
        err_sys("can't open %s", argv[1]);

    while ( (dirp = readdir(dp)) != NULL)
        printf("%s\n", dirp->d_name);

    closedir(dp);
    exit(0);
}
```

ls(1)这种表示方法是UNIX的惯用方法，用以引用UNIX手册集中的一个特定项。它引用第一部分中的ls项，各部分通常用数字1至8表示，在每个部分中的各项则按字母顺序排列。假定你有一份所使用的UNIX系统的手册。

> 早期的UNIX系统把8个部分都集中在一本手册中，现在的趋势是把这些部分分别安排在不同的手册中：有用户专用手册，程序员专用手册，系统管理员专用的手册等等。
>
> 某些UNIX系统把一个给定部分中的手册页又用一个大写字母进一步分成若干小部分，例如，AT&T〔1990e〕中的所有标准I/O函数都被指明在3S部分中，例如fopen(3S)。
>
> 某些UNIX系统，例如以Xenix为基础的系统，不是采用数字将手册分成若干部分，而是用C表示命令(第1部分)，S表示服务(通常是第2、3部分)等等。

如果你有联机手册，则可用下面的命令查看ls命令手册页：

```
man 1 ls
```

程序1-1只打印一个目录中各个文件的名字，不显示其他信息，如若该源文件名为myls.c，则可以用下面的命令对其进行编译，编译的结果送入系统默认名为a.out的可执行文件名：

```
cc myls.c
```

某种样本输出是：

```
$ a.out /dev
.
..
MAKEDEV
console
tty
mem
kmem
null
```

此处略去多行

```
printer
$ a.out /var/spool/mqueue
can't open /var/spool/mqueue:Permission denied
$ a.out /dev/tty
can't open /dev/tty:Not a directory
```

本书将以这种方式表示输入的命令以及其输出：输入的字符以粗体表示，程序输出则以另一种字体表示。如果欲对输出添加注释，则以中文宋体表示，输入之前的美元符号($)是shell打印的提示符，本书将shell提示符显示为$。

注意，列出的目录项不是以字母顺序排列的，ls命令则一般按字母顺序列出目录项。

在这20行的程序中，有很多细节需要考虑：

• 首先，其中包含了一个头文件ourhdr.h。本书中几乎每一个程序都包含此头文件。它包含了某些标准系统头文件，定义了许多常数及函数原型，这些都将用于本书的各个实例中，附录B列出了常用头文件。

• main函数的说明使用了ANSI C标准所支持的新风格（下一章将对ANSI C作更多说明）。

• 取命令行的第1个参数argv〔1〕作为列出的目录名。第7章将说明main函数如何被调用，程序如何存取命令行参数和环境变量。

• 因为各种不同UNIX系统的目录项的实际格式是不一样的，所以使用函数opendir, readdir和closedir处理目录。

• opendir函数返回指向DIR结构的指针，并将该指针传向readdir函数。我们并不关心DIR结构中包含了什么。然后，在循环中调用readdir来读每个目录项。它返回一个指向dirent结构的指针，而当目录中已无目录项可读时则返回null指针。在dirent结构中取出的只是每个目录项的名字(d_name)。使用该名字，此后就可调用stat函数(见4.2节)以决定该文件的所有属性。

• 调用了两个自编的函数来对错误进行处理：err_sys和err_quit。从上面的输出中可以看到，err_sys函数打印一条消息（“Permission denied(许可权拒绝)”或“Not a directory(不是一个目录)”），说明遇到了什么类型的错误。这两个出错处理函数在附录B中说明，1.7节将更多地叙述出错处理。这两个出错处理函数在附录B中说明1.7节将更详细地叙述出错处理。

• 当程序将结束时，它以参数0调用函数exit。函数exit终止程序。按惯例，参数0的意思是正常结束，参数值1～255则表示出错。8.5节将说明一个程序(例如shell或我们所编写的程序)如何获得它所执行的另一个程序的exit状态。

1.3.4 工作目录

每个进程都有一个工作目录(working directory，有时称为当前工作目录(current working directory))。所有相对路径名都从工作目录开始解释。进程可以用chdir函数更改其工作目录。

例如，相对路径名doc/memo/joe指的是文件joe，它在目录memo中，而memo又在目录doc中，doc则应是工作目录中的一个目录项。从该路径名可以看出，doc和memo都应当是目录，但是却不清楚joe是文件还是目录。路径名/urs/lib/lint是一个绝对路径名，它指的是文件(或目录) lint，而lint在目录lib中，lib则在目录usr中，usr则在根目录中。

1.3.5 起始目录

登录时，工作目录设置为起始目录(home directory)，该起始目录从口令文件（见1.2节）中

的登录项中取得。

1.4 输入和输出

1.4.1 文件描述符

文字描述符是一个小的非负整数，内核用以标识一个特定进程正在存访的文件。当内核打开一个现存文件或创建一个新文件时，它就返回一个文件描述符。当读、写文件时，就可使用它。

1.4.2 标准输入、标准输出和标准出错

按惯例，每当运行一个新程序时，所有的shell都为其打开三个文件描述符：标准输入、标准输出以及标准出错。如果像简单命令ls那样没有做什么特殊处理，则这三个描述符都连向终端。大多数shell都提供一种方法，使任何一个或所有这三个描述符都能重新定向到某一个文件，例如：

```
ls > file.list
```

执行ls命令，其标准输出重新定向到名为file.list的文件上。

1.4.3 不用缓存的I/O

函数open、read、write、lseek以及close提供了不用缓存的I/O。这些函数都用文件描述符进行工作。

实例

如果愿意从标准输入读，并写向标准输出，则程序1-2可用于复制任一UNIX文件。

程序1-2 将标准输入复制到标准输出

```
#include    "ourhdr.h"

#define BUFFSIZE    8192

int
main(void)
{
    int     n;
    char    buf[BUFFSIZE];

    while ( (n = read(STDIN_FILENO, buf, BUFFSIZE)) > 0)
        if (write(STDOUT_FILENO, buf, n) != n)
            err_sys("write error");

    if (n < 0)
        err_sys("read error");

    exit(0);
}
```

头文件<unistd.h>(ourhdr.h中包含了此头文件)及两个常数STDIN_FILENO和STDOUT_FILENO是POSIX标准的一部分（下一章将对此作更多的说明）。很多UNIX系统服务的函数原型，例如我们调用的read和write都在此头文件中。函数原型也是ANSI C标准的一部分，本章的

稍后部分将对此作更多说明。

两个常数STDIN_FILENO和STDOUT_FILENO定义在<unistd.h>头文件中，它们指定了标准输入和标准输出的文件描述符。它们的典型值是0和1，但是为了可移植性，我们将使用这些新名字。

3.9节将详细讨论BUFFSIZE常数，说明各种不同的值将如何影响程序的效率。但是不管该常数的值如何，此程序总能复制任一UNIX文件。

read函数返回读得的字节数，此值用作要写的字节数。当到达文件的尾端时，read返回0，程序停止执行。如果发生了一个读错误，read返回－1。出错时大多数系统函数返回－1。

如果编译该程序，其结果送入标准的a.out文件，并以下列方式执行它:

```
a.out > data
```

那么，标准输入是终端，标准输出则重新定向至文件data，标准出错也是终端。如果此输出文件并不存在，则shell创建它。

第3章将更详细地说明不用缓存的I/O函数。

1.4.4 标准I/O

标准I/O函数提供一种对不用缓存的I/O函数的带缓存的界面。使用标准I/O可无需担心如何选取最佳的缓存长度，例如程序1-2中的BUFFSIZE常数。另一个使用标准I/O函数的优点与处理输入行有关(常常发生在UNIX的应用中)。例如，fgets函数读一完整的行，而另一方面，read函数读指定字节数。

我们最熟悉的标准I/O函数是printf。在调用printf的程序中，总是包括<stdio.h>(通常包括在ourhdr.h中)，因为此头文件包括了所有标准I/O函数的原型。

实例

程序1-3的功能类似于调用read和write的前一个程序1-2，5.8节将对程序1-3作更详细的说明。它将标准输入复制到标准输出，于是也就能复制任一UNIX文件。

程序1-3 用标准I/O将标准输入复制到标准输出

```
#include    "ourhdr.h"

int
main(void)
{
    int     c;

    while ( (c = getc(stdin)) != EOF)
        if (putc(c, stdout) == EOF)
            err_sys("output error");

    if (ferror(stdin))
        err_sys("input error");

    exit(0);
}
```

函数getc一次读1个字符，然后putc将此字符写到标准输出。读到输入的最后1个字节时，getc返回常数EOF。标准输入、输出常数stdin和stdout定义在头文件<stdio.h>中，它们分别表示标准输入和标准输出文件。