国外计算机科学教材系列

UNIX 进程间通信

（第二版）

Interprocess Communications in UNIX

Second Edition

[美] John Shapley Gray 著
张 宁 等译

PH PTR

電子工業出版社
Publishing House of Electronics Industry
URL:http://www.phei.com.cn

国外计算机科学教材系列

UNIX 进程间通信
（第二版）

Interprocess Communications in UNIX
Second Edition

[美] John Shapley Gray 著

张 宁 等译

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内容简介

本书从操作系统的角度对UNIX进程间通信的基本原理进行了全面剖析。该书从进程的基本概念入手，阐述了进程环境、进程的使用、原语通信、管道、消息队列、信号量、共享内存、远程过程调用、套接字、线程等基本理论，内容丰富实用、讲解清楚、通俗易懂，并通过大量应用实例对所涉及的相关技术要点加以解释，是一本讲解UNIX操作系统的优秀书籍，适合于高等院校计算机专业作为UNIX操作系统的教材或参考读物之用，也可作为计算机开发人员学习UNIX进程间通信的参考书。

图书在版编目(CIP)数据

UNIX进程间通信（第二版）/（美）格雷（Gray, J. S.）著；张宁等译. —北京：电子工业出版社，2001.3
书名原文：Interprocess Communications in UNIX, Second Edition
国外计算机科学教材系列
ISBN 7-5053-6571-1

Ⅰ.U... Ⅱ.①格...②张... Ⅲ.UNIX操作系统－教材 Ⅳ.TP316.81

中国版本图书馆CIP数据核字（2001）第12849号

丛 书 名：国外计算机科学教材系列
书　　名：**UNIX进程间通信（第二版）**
原 书 名：Interprocess Communications in UNIX, Second Edition
著　　者：[美] John Shapley Gray
译　　者：张　宁　等
责任编辑：赵红燕
排版制作：今日电子公司制作部
印 刷 者：北京东光印刷厂
出版发行：电子工业出版社 URL:http://www.phei.com.cn
　　北京市海淀区万寿路173信箱　邮编：100036　电话：68279077
经　　销：各地新华书店
开　　本：787 × 1092　1/16　　印张：23.5　字数：580千字
版　　次：2001年3月第1版　2001年3月第1次印刷
书　　号：ISBN 7-5053-6571-1
　　TP·3635
定　　价：36.00元
版权贸易合同登记号　图字：01-2000-3485

出 版 说 明

随着21世纪的到来，计算机技术的发展更加迅猛，在各行各业的应用更加广泛，越来越多的高等院校增设了有关计算机科学的课程内容，或对现有计算机课程设置进行了适当调整，以紧跟前沿技术。在这个教学体系和学科结构变革的大环境下，对适合不同院系、不同专业、不同层次的教材的需求量与日俱增。此时，如果能够借鉴、学习国外一流大学的先进教学体系，引进具有先进性、实用性和权威性的国外一流大学计算机教材，汲取其精华，必能更好地促进中国高等院校教学的全面改革。

美国Prentice Hall出版公司是享誉世界的高校教材出版商，自1913年成立以来，一直致力于教材的出版，所出版的计算机教材为美国众多大学采用，其中有不少是专业领域中的经典名著，已翻译成多种文字在世界各地的大学中使用，成为全人类的共同财富。许多蜚声世界的教授、学者都是该公司的资深作者，如道格拉斯·科默（Douglas E. Comer）、威廉·斯大林（William Stallings）等。早在1997年，电子工业出版社就从Prentice Hall引进了一套计算机英文版专业教材，并将其翻译出版，同时定名为《国外计算机科学教材系列》（下称：第一轮教材）。截至2000年12月，该系列教材已出版23种，深受读者欢迎，被许多大学选为高年级学生和研究生教材或参考书。

4年过去了，已出版的教材中多数已经有了后续版本。因此，我们开始设计新一轮教材（第二轮教材）的出版，成立了由我国计算机界著名专家和教授组成的“教材出版委员会”，并结合第一轮教材的使用情况和师生反馈意见，组织了第二轮《国外计算机科学教材系列》出版工作。

第二轮教材的出版原则为：

1. 引进Prentice Hall出版公司2000年和2001年推出的新版教材，作为替换版本。
2. 在著名高校教授的建议下，除了从Prentice Hall新选了一些教材之外，还从McGraw-Hill和Addison Wesley Longman等著名专业教材出版社、麻省理工学院出版社和剑桥大学出版社等著名大学出版社引进了一些经典教材，作为增补版本。
3. 对于第一轮中无新版本的优秀教材，我们将其作为延用版本，直接进入第二轮使用。
4. 对于第一轮中翻译质量较好且无新版本的教材，我们将其进行了修订，也作为延用版本，进入第二轮使用。

这次推出的教材覆盖学科范围广、领域宽、层次多，既有本科专业课程教材，也有研究生课程教材，以适应不同院系、不同专业、不同层次的师生对教材的需求。广大师生可自由选择和自由组合使用。

按照计划，本轮教材规划出版37种，其中替换版本8种，新增版本14种，延用版本15种。教材内容涉及的学科方向包括网络与通信、操作系统、计算机组织与结构、算法与数据结构、数据库与信息处理、编程语言、图形图像与多媒体、软件工程等。本轮教材计划于2001年7月前全部出版。教材的使用年限平均为3年。我们还将陆续推出一些教材的参考课件，希望能为授课老师提供帮助。

为了保证本轮教材的选题质量和翻译质量，我们约请了清华大学、北京大学、北京航空航天大学、复旦大学、上海交通大学、南京大学、浙江大学、哈尔滨工业大学、华中科技大学、西安交通

大学、国防科学技术大学、解放军理工大学等著名高校的教授和骨干教师参与了本轮教材的选题、翻译和审校工作。他们中既有讲授同类教材的骨干教师和博士，也有积累了几十年教学经验的教授和博士生导师。

在本轮教材的选题、翻译和编辑加工过程中，为提高教材质量，我们做了大量细致的工作，包括：

1. 对于新选题和新版本进行了全面论证。
2. 对于延用版本，认真审查了前一版本教材，修改了其中的印刷错误。
3. 对于译者和编辑的选择，达到了专业对口。
4. 对于从英文原书中发现的错误，我们通过与作者联络、从网上下载勘误表等方式，一一做了修改。
5. 对于翻译、审校、编辑、排版、印刷质量进行了严格的审查把关。

通过这些工作，保证了本轮教材的质量较前一轮有明显的提高。相信读者一定能够从字里行间体会到我们的这些努力。

今后，我们将继续加强与各高校教师的密切联系，为广大师生引进更多的国外优秀教材和参考书，为我国计算机科学教学体系与国际教学体系的接轨做出努力。

由于我们对国际计算机科学、我国高校计算机教育的发展存在认识上的不足，在选题、翻译、出版等方面的工作中还有许多有待提高之处，恳请广大师生和读者提出批评和建议。

电子工业出版社

2001 年春

教材出版委员会

译 者 序

UNIX是一个大规模和复杂的系统，伴随着用户的要求正在不断地进行演变。大多数精致复杂的程序都涉及某种形式的IPC，也就是进程间通信（Interprocess Communication）。在UNIX系统平台上利用IPC进行应用程序设计是一种可以充分利用系统资源、提高程序运行效率的良好方法。从UNIX的发展来看，应用程序经历了从利用单个庞大的程序到使用某种形式的IPC进行彼此间通信来完成用户任务的演变。尤其是进入20世纪90年代后，线程和多线程程序设计技术已成为UNIX的主流。本书在介绍进程间通信基本技术的同时，特别强调了线程的使用以及与线程间通信有关的技术，本书作者在第一版的基础上专门补充了一章。全书覆盖了与UNIX进程间通信所涉及的有关进程和线程的大部分内容。

尽管国内已出版了大量的有关UNIX的书籍，但专门介绍进程间通信的书籍还比较少。译者在读完此书后，感觉这是一本非常值得向读者推荐的好书，它内容丰富实用、讲解清楚、通俗易懂，并通过大量应用实例对所涉及的相关技术要点加以解释，对一般程序设计人员提高在UNIX环境下开发应用程序的技巧有很大帮助。本书也适合于具有一定UNIX基础的读者进一步了解和学习UNIX进程间通信机制之用。

本书由北京大学计算机系张宁副教授主持翻译和统一审校，计算机系的杨洁、胡钢、高旭、陈原、晏齐、饶文靖、陈鸿、程世军、段卫华、付强同志参加了部分翻译工作。高旭同志进行了全书译稿的修改和整理工作。正是由于他们的不懈努力和工作，使得此书的译稿得以顺利完成，在此表示由衷的感谢。由于译者水平有限，译文中难免有不妥之处，敬请广大读者批评指正。就中文译本中的技术问题，读者可直接与译者本人联系，电子邮件地址为nzhang@ailab.pku.edu.cn。

译者

2001年3月于燕园

序　　言

在写此书的第一版时，我已经注意到UNIX是一个大规模和复杂的系统，并且为满足用户的要求正在不断地演变。这里想强调的是：帮助读者掌握UNIX中进程间通信的基本概念一直是本书的目标。自第一版出版发行以来，UNIX已经发生了明显的变化，到20世纪90年代中期，线程和多线程程序设计技术已成为UNIX的主流。为强调线程的使用以及与线程间通信有关的技术，我在原有第一版的基础上补充了一章，此章内容充实，但不具有涵盖性，因为有关线程的论题是相当复杂的。

这里假设读者具有C程序设计的经验，而且即使不很专业，也已在一个基本的UNIX环境下工作过，熟悉UNIX中的一些文本编辑器，如vi或pico（可从Washington大学获得）。本书对特定的UNIX系统调用和预定义函数库做了广泛的扩充，建议读者阅读实际系统的联机使用手册中有关系统和库函数调用的说明。附录A中给出了有关UNIX联机使用手册的格式和使用说明。

书中所有的引用和例子均在Sun SPARC 10 Model 41的Sun Microsystems的Solaris2.5系统上运行通过，每一个例子均为可独立运行的程序。除标明以外，命令行例子均是基于C Shell的。用Sun Microsystems的C++编译器CC（V4.2）编译所有的源程序代码。Solaris原来用于AT&T UNIX，在移植到Solaris之前，Sun Microsystems的平台上运行的是基于BSD版的SunOS，许多例子（和一些习题）已经在SunOS 4.1.3上以及PC平台上的Slackware Linux上通过。由Linus Torvalds研制成功的Linux是一个基于类UNIX的免费软件，并由BSD管理，同时它的编程环境适合AT&T。通常情况下只需少许修正便可生成清晰的、可执行的代码。在上述的方法中均要用到GNU C编译器，即gcc。在任一UNIX的环境设置中应当具有对IPC（进程间通信）的支持，以帮助用户学习在本书中所讲述的有关信号量、消息队列和共享内存的知识。

Solaris默认支持IPC。然而，在某些版本的UNIX（如SunOS和Linux）中，需要重新修改系统配置文件并重新编译系统内核部分以支持IPC。除非你目前是系统管理员，否则，需要在系统管理员的帮助下完成UNIX系统内核部分的重编译工作。为做好有关线程方面的工作，必须提供一个线程库。幸运的是大多数新的UNIX版本有这样的线程库。

有关这方面的工作还有待于进一步完善。我非常欢迎读者提出、建议、更正错误以及给出习题的解答。读者可通过电子邮件与我联系：gray@hartford.edu，并可通过匿名FTP方式获得本书中所有例子程序的源代码：/ftp/pub/ptr/professional_computer_science.w-022/gray/interprocess_communications2。

John Shapley Gray

目　　录

第1章　程序和进程

1.1　导言

对所有的操作系统而言，进程是一个基本的概念。从某种抽象角度讲，进程由一个执行（运行）程序、它的当前值、状态信息以及通过操作系统管理此进程执行情况的资源组成。进程是一个动态的实体。在基于UNIX的操作系统中，在任一时间点上，多个进程并发地执行。从每一个进程来看，它们似乎均具有独立地访问、控制所有的系统资源的能力。这两个论点是模糊的。大多数UNIX操作系统可运行在支持多个活动进程的具有一个单一处理单元的平台上。然而，实际上在每一个时间点上只有一个进程在实际运行。通过快速地改变当前执行的进程，UNIX操作系统显现出多个进程的并发执行。操作系统中进程间的这种多元化执行方式称为多道程序设计（multiprogramming）或多任务处理（multitasking），可支持真正并发处理的具有多处理单元的系统称为多处理系统（multiprocessing）。

正如上面所指出的，进程部分由一个程序的执行过程组成。一个程序是由一组指令和相关数据组成的一个非活动的、静态的实体。如果一个程序被激活多次，就会生成多个进程。我们可以将一个程序看作两个基本的格式：

- **源程序**：一个源程序由某个特定程序设计语言（如C语言）的一系列有效语句组成，它以ASCII文本形式存储。为讨论方便起见，我们将考虑只包含ASCII码中32～127间字符的纯文本形式，这样的源程序文件可以在显示器或打印机上输出。在大多数条件下，源程序的访问权限为不可执行的。程序1.1给出了一个用C语言写出的源程序的例子。
- **可执行程序**：一个可执行程序是通过编译、汇编等变换程序形成的可被操作系统执行的一组代码，在大多数情况下它不是纯文本形式，所以不能在终端上显示或由打印机打印输出。

程序1.1　一个C语言源程序

```
/*
显示Hello World三次
*/
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
char            *cptr = "Hello World\n";          /* 设置静态字符串 */
char             buffer1[25];

main(void){
  void             showit(char *);                /* 函数原型       */
  int              i = 0;                         /* 累加变量       */
```

```
    strcpy(buffer1, "A demonstration\n");          /* 库函数        */
    write(1, buffer1, strlen(buffer1)+1);          /* 系统调用      */
    for (; i < 3; ++i)
      showit(cptr);                                /* 函数调用      */
}
void
showit(char *p){
    char          *buffer2;
    if ((buffer2 = (char *) malloc((unsigned) (strlen(p)+1))) != NULL){
      strcpy(buffer2, p);                          /* 拷贝字符串    */
      printf("%s", buffer2);                       /* 显示字符串    */
      free(buffer2);                               /* 释放空间      */
    } else {
      printf("Allocation error.\n");
      exit(1);
    }
}
```

1.2 库函数

任意复杂的程序编写均需要使用库函数。一个函数是一个声明和语句的汇集，它执行一个特定的动作，并且/或返回一个值。函数或由用户自己定义，或以前已经定义并可为用户所调用。以前定义的函数以目标码的形式存储在库文件中，作为生成可执行程序时的中间步骤的目标码的格式是一种特定文件格式。像可执行文件一样，目标码文件也不能在终端上显示或在打印机上打印输出。存储在库文件中的函数称为库函数或运行库例程。

在大多数UNIX系统中，库文件的标准位置在目录/usr/lib下。辅助的库文件也可在/usr/local/lib目录下找到。传统上，约定库文件以三个字母lib为前缀，再加上一个扩展.a。可用于维护库文件的UNIX档案实用程序ar检查库文件的内容①。例如，命令：

```
% ar t /usr/lib/libc.a | pr -4 -t
```

将以管道形式把标准C库的内容（libc.a）输出到pr实用程序，并以一个四列的形式在终端上显示。此库中的目标码默认均由C程序编译而成，因此，在一个C程序中，对printf的调用均是针对/usr/lib/libc.a中的printf函数的。

如果你的系统支持命令行中的-k选项，则可通过下面的命令从windex数据库获得一个单列表的大纲形式的输出：

```
% man -k \ (3
```

正如所显示的，此命令要求man（manual）程序对联机手册第3节中每一项以单行形式概括地显示其内容。第3节（以及它所属的几个小节）包含了联机手册中相关页的子程序和库函数说明。命令行中的符号“\”用来避开圆括号，以使shell命令在使用man命令时扫过圆括号，否则会产生

① 对以“a”开头的所有命令行选项，文档实用程序是许多系统实用程序的例外之一。

由于shell命令要解释圆括号而产生的语法错误。如果你的系统指出还没有生成windex数据库，则可用命令：

```
% catman -w
```

生成它，并赋给你相应的访问权限。

也可通过查看联机手册第3节中Intro页获得对库函数更为丰富的了解。在大多数系统中，命令：

```
% man -s3 Intro
```

将返回Intro页。这里-s标志用来告知man显示适当的页。某些版本的man命令省略了小节选项“s”（即，-3表示第3节）。其他的联机手册信息的组织可参看附录A。

1.3　系统调用

在程序中使用的一些前面定义的函数实际上属于系统调用。在以某种格式汇集库函数时，系统调用请求UNIX操作系统代表唤醒的进程直接进行某些工作。由操作系统执行的代码处于系统内核（kernel，即中央控制程序）中，系统调用充当一个对这些码的高/中级别的语言接口。为保证系统内核的完整性，执行系统调用的进程必须暂时地由用户模式（拥有用户访问权限）转换为系统模式（拥有系统访问权限），在某些情况下这一转换会带来一定的系统开销，从而使得对同一任务而言系统调用在执行效率上低于库函数调用。许多库函数（如处理输入和输出）是满缓冲的，因此当执行特定的任务时允许系统具有某些控制权利。

联机手册的第2节包含了一些系统调用。输出联机手册内容信息的方法类似于前面所使用的命令，区别在于小节选项用2而不用3。非常重要的一点是某些库函数已被嵌入到系统调用中，例如库函数调用scanf和printf使用系统调用read和write。

库函数调用和系统调用的关系如图1.1所示。

图中的箭头表示可能的通信路径。如图所示，可执行程序可以直接地利用系统调用以要求系统内核去执行一个特定的功能。或者也可以理解为可执行程序可以唤醒一个库函数以达到执行系统调用的目的。

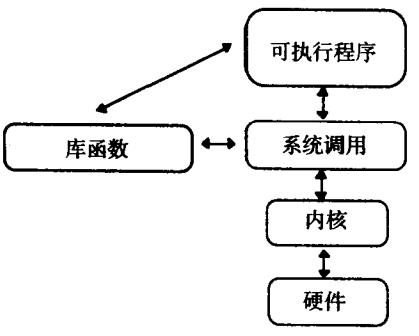

图1.1　硬件和软件层次

1.4 链接目标码

库文件中的码由根据需要对源程序进行编译后生成的目标码组成。当用C进行程序设计时，对额外的用于系统调用的目标码，以及没有包含在标准的C库中的库函数可以在编译时指定。这一点可以通过使用编译选项 -l，后跟没有前缀的库的名字以及扩展.a 加以实现。例如，C 编译命令：

```
% cc prog.c -lm
```

说明在用 cc 进行编译程序时，将源程序 prog.c 编译后的目标码与数学库 libm.a 进行链接。注意库函数经常需要在源程序中加上头文件说明，这些头文件包含了函数原型、宏定义、常量定义等说明。若没有适当的头文件说明，在编译时程序就会出错。另外，如果在源程序中加上了适当的头文件说明，但是忘记链接包含目标码的相关的库，则也不会得到正确的编译结果，如此造成的错误经常会以如下方式给出出错信息："Undefined symbol first referenced in file...ld: fatal: Symbol referencing errors. No output written to a.out. Compilation failed.(在文件中发现没有定义的符号……致命错误：符号引用错误。没有输出到 a.out。编译失败。)。"

在联机手册的大纲节（参见附录 A）中列出了需要的头文件。当有多个头文件说明时，在源程序中列出的顺序应当与联机手册中给出的顺序一致。头文件在源程序中说明的顺序非常重要，因为有时一个特定的头文件会与前面引用的头文件相关，这一相关关系可在<sys/types.h>中查到。<sys/types.h>中说明的 types.h 放在系统子目录 sys 下。

习题 1.1

使用 ar 命令检查标准 C 库（/usr/lib/libc.a）的内容。在标准的 C 库中收藏了多少与 printf 相关的函数？

习题 1.2

任意库函数 / 系统调用发生在一个以上的库吗？如果是，举出一例并解释为什么会出现此类情况。

1.5 管理失败

在大多数情况下[①]，如果一个系统调用或库函数调用失败，则会返回一个值 -1，并给一个外部变量 errno 赋一个值以说明实际的错误是什么。所有的错误码均可在头文件<sys/errno.h>中找到已定义的常量。令唤醒的程序检查一个系统调用或函数调用的返回值以确定其是否成功是一个良好的习惯。如果唤醒失败，程序将采取适当的行为，其常见情况是显示一个简短的出错信息，并 exit（终止）此程序。可用库函数 perror 产生一个出错信息。

对每一个系统调用和库函数调用的详细说明通过一个概述表给出。此概述表示了一个压缩的 UNIX 联机操作手册信息。一个典型的概述表的格式如图 1.2 所示。

① 此类型是必要的，因为返回一个整数值的系统调用通常返回 -1 以表示失败，而对那些返回一个指向字符的指针的系统调用，通常返回的是 NULL 指针。然而，就像系统调用是由互不相关的、具有不同想法的程序员所编写的一样，偶尔也会遇到不满足此约束的返回值。

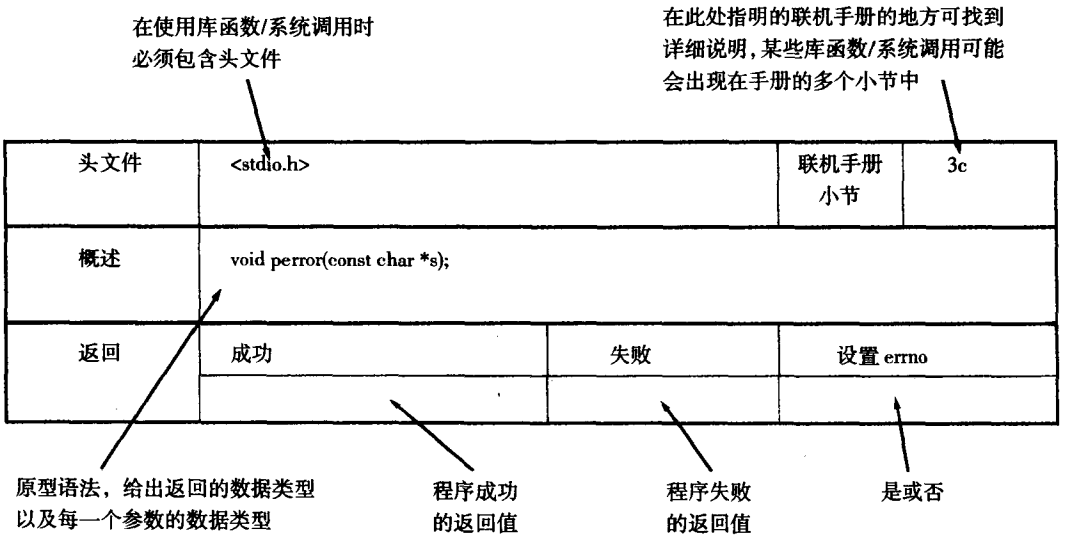

图 1.2　概述表的格式与说明

正如perror的概述表指出的，如果使用perror，则必须在源程序的开头说明头文件<stdio.h>。注意如前所示，在概述表中并不涉及头文件<sys/errno.h>。只有在需要引用指定的错误码时，才包含<sys/errno.h>的说明。库函数perror只有一个指向一个字符串常量的指针作为参数（即const char *）。另外，perror没有返回值，如果失败，它不修改errno。

通过使用系统调用perror和errno而提供某些错误检查的程序如程序1.2所示。

程序 1.2　使用 errno 和 perror

```
/*    检查errno及perror的使用  */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
extern int      errno;
void
main(int argc, char *argv[]) {
  int  n_char = 0, buffer[10];    /* 将n_char 初始化为0 -- errno默认为0。*/
  printf("n_char = %d \t errno = %d \n", n_char, errno);    /* 显示输出 */
  n_char = write(1, "Enter a word ", 14);
  /* 通过系统调用read从标准输入设备获取10个字符 */
  n_char = read(0, buffer, 10);
printf("\nn_char = %d \t errno = %d \n", n_char, errno);
if (n_char =  = -1) {            /* 若read失败..... */
    perror(argv[0]);
    exit(1);
  }
  n_char = write(1, buffer, n_char);   /* 显示所读的字符 */
}
```