UNIX技术丛书

UNIX技术
——网络应用篇

沈芝慎 著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
URL: http://www.phei.com.cn

UNIX 技术丛书

UNIX 技术 — 网络应用篇

沈芝慎 著

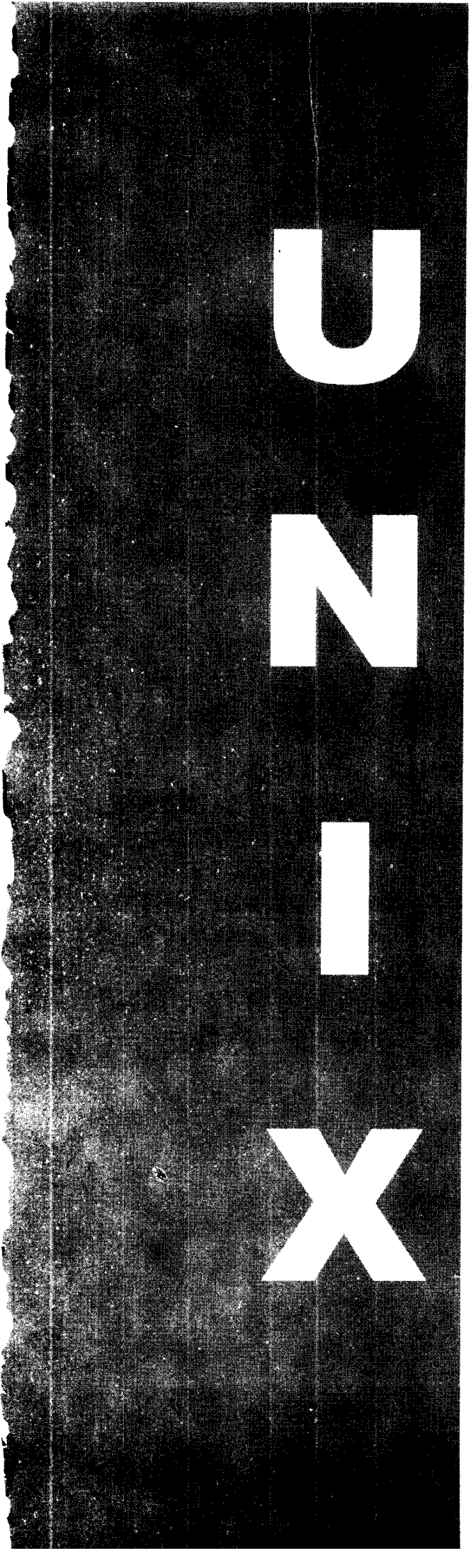
電子工業出版社
Publishing House of Electronics Industry

内 容 简 介

本书作者凭借实际参与网络规划建立的经验，有系统地带领您了解网络的基本原理，学习网络上电子函件、电子新闻、RPC、NFC、Web浏览器等常见软件及工具的结构与功能。本书第一章和第二章为网络基本知识介绍，主要介绍与网络有关的 UNIX 指令、网络常用名词以及网络结构；第三章至第七章介绍了网络中各种应用软件通讯协议，剖析它们的结构、传输方法与建立程序；第八、第九章为网络程序编写，以实例说明如何编写网络软件程序。本书注重网络的实际应用，期待您在枯燥的网络理论后，能轻松自在地漫游于网络世界中。

丛 书 名：UNIX 技术丛书
书　　名：UNIX 技术——网络应用篇
著　　者：沈芝慎
责任编辑：杨丽娟
特约编辑：刘彬
印 刷 者：北京大中印刷厂
出版发行：电子工业出版社、发行
北京市海淀区万寿路 173 信箱　邮编 100036　发行部电话 68214070
URL：http：/www.phei.com.cn
经　　销：各地新华书店经销
开　　本：787 x 1092 1/16　印张：17.75　字数：455 千字
版　　次：1998 年 4 月第 1 版　1998 年 4 月第 1 次印刷
书　　号：ISBN 7-5053-4696-2 / TP·2255
定　　价：24.00
著作权合同登记号　图字：01-97-1879

目　录



0

绪论

本章就阅读本书所需的知识背景及本书的结构作一扼要提示。

谁适合读这本书

1. 有 UNIX 使用经验，并且对网络有兴趣的读者。
2. UNIX 的系统管理员，需要配置网络软件并且定时维护。
3. 网络软件开发人员，计划在 UNIX 上开发网络软件。
4. 对网络已有基本概念，但是希望更进一步了解的读者。

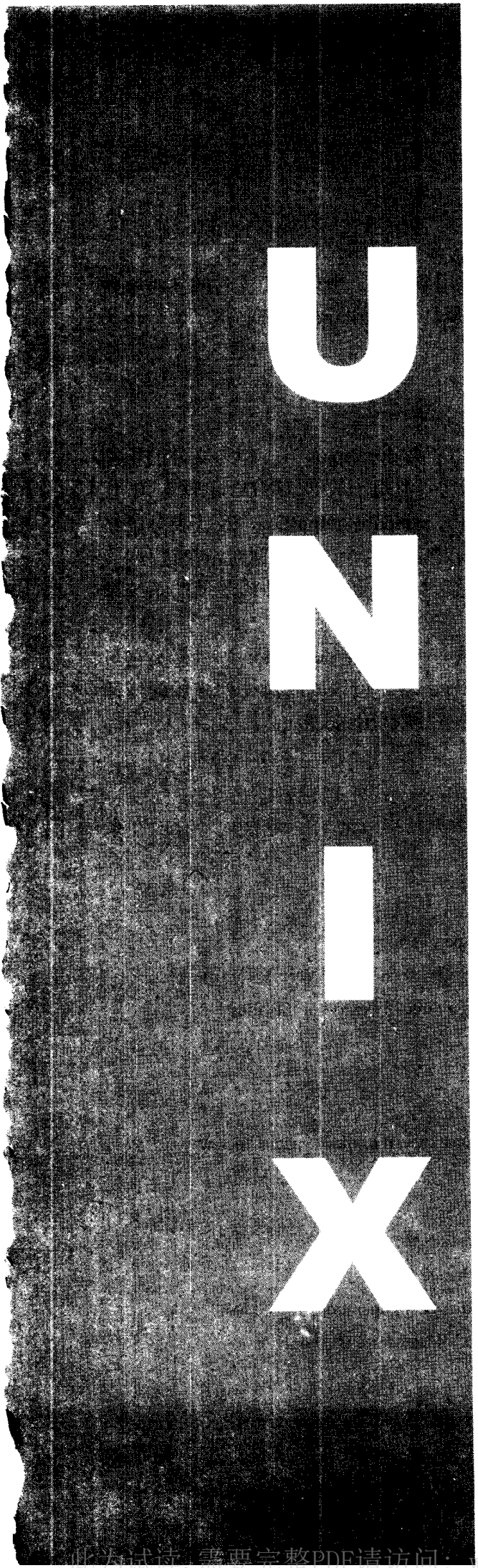
读者需要的背景知识

1. 由于这本书介绍 UNIX 上的网络，所以读者应有 UNIX 的使用经验，并且对基本指令、文件系统与程序控制有所了解。
2. 有 C 语言的编写经验。
3. 曾使用过如 ftp 、 e-mail 等的网络软件。

本书的结构

本书依性质可分为三部份：

1. **第一章～第二章：网络基础介绍** 这部份介绍与网络有关的 UNIX 指令、网络常用的名词，以及网络结构。第一章包括文件 I/O 、信号等指令，这些指令虽不是网络特有的指令，但在开发网络软件时经常使用。第二章介绍 OSI 模型的七层结构、 TCP/IP 通讯协议和 SNA 网络结构，是本书中理论性较强的部份。
2. **第三章～第七章：网络应用介绍** 介绍网络中各种应用软件通讯协议。本书不解释如何使用它们，而是剖析它们的结构、传输方法及配置程序。这些应用软件包括 r 系列指令、域名系统、电子邮递系统、网络新闻与 UUCP 。
3. **第八章～第九章：网络程序编写** 介绍如何编写网络软件程序。第八章为 socket 程序库，第九章为 RPC （ Remote Procedure call ），并都附有实际例子说明。



1

UNIX 网络基本概论

这一章是进入本书正题——网络前的热身活动，我们以很短的篇幅将文件、信号、进程控制的概念作一快速复习，如果读者对上述的内容仍有不清楚之处，请参阅本系列丛书中的基础篇。

文件

网络程序与其他的 UNIX 程序一样，都常常用到文件，所以对于文件的各种系统调用，如打开、关闭、读写，或者文件内的各种属性（ attribute ）都必须了解，因为它们都是网络程序中很重要的一环。

File Descriptor

每打开一个文件，都会传回一个文件描述词（ File descriptor ），作为文件的识别号。文件描述词是一个正整数，从 0 开始向上递增，但是一般的 UNIX 已规定 0 代表标准输入（ standard input ）， 1 代表标准输出（ standard output ）， 2 代表标准错误（ standard error ）。较早的 UNIX 只允许打开最多 20 个文件（针对一个进程）。但目前的系统几乎已无此限制。文件描述词由系统设定，会产生文件描述词的系统调用有 open 、 creat 、 dup 、 pipe 与 fcntl 。

在稍后讨论 socket 程序库时会出现插座描述词（ socket descriptor ）。其实它与文件描述词极为类似，有些系统调用如 read 、 write 等，可以同时对文件描述词或插座描述词进行数据读写。插座描述词也是整数，由系统调用 socket() 产生。

文件特性（ attribute ）

一个文件包括很多特性，都记录在它特有的数据结构 stat 中，这些内容存放在系统的头文件 <sys/stat.h> 中：

```
struct stat {
ushort      st_mode;      /*file type and file access permissions*/
ino_t       st_ino;       /* i-node number */
dev_t       st_dev;       /* ID of device containing a directory entry for this file */
short       st_nilink;    /* number of links */
ushort      st_uid;       /* user ID */
ushort      st_gid;       /* group ID */
dev_t       st_rdev;      /* ID of device, for character special or block special files */
off_t       st_size;      /* file size in bytes */
time_t      st_atime;     /* time of last file access */
time_t      st_mtime;     /* time of last file modification */
time_t      st_ctime;     /* time of last file status change */
long        st_blksize;   /* optimal block size for filesystem operation; 4.3BSD only */
long        st_blocks;    /* actual number of blocks allocated; 4.3BSD only */
};
```

stat 中又有许多种数据类型不是一般的 C 语言数据类型，而是另外定义的数据形态，它们定义在系统文件 <sys/types.h>中。第一栏位 st_mode 定义文件形态，又可分

成下列数种：

```
#define     S_IFMT      0170000     /* type of file */
#define     S_IFREG     0100000     /* regular */
#define     S_IFDIR     0040000     /* directory */
#define     S_IFCHR     0020000     /* character special */
#define     S_IFBLK     0060000     /* block special */
#define     S_IFLNK     0120000     /* symbolic link-BSD only*/
#define     S_IFSOCK    0140000     /* socket - BSD only */
#define     S_IFIFO     0010000     /* fifo - System V only */
```

要取得文件的属性，可以用下面几个程序，由于参数中传入 stat 结构，所以在使用时要放入头文件：

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

int stat(path, buf)
char    *path;
struct stat *buf;

int lstat(path, buf)
char    *path;
struct stat *buf;

int fstat(fd, buf)
int   fd;
stract stat *buf;
```

这三个系统调用都可以取得文件的状态信息，但功能不同：

stat() 以文件名称（文件路径）为参数，**对此文件不需要读、写或执行的权限，但是必须能寻找到此文件。**

lstat() 的功能与 stat() 相同，**但是涉及到链接**（symbolic link）**文件**，lstat() 会传回这个链接（link）本身的状态信息。对这个链接文件仍可使用 stat()，但会传回链接所指向的文件状态信息。

fstat() 的参数不是文件名称或路径，而是文件描述词。

我们用一个实际的程序来看 stat() 如何得到文件的状态信息，程序的输入参数是一个文件名称，并列出（输出）出这个文件形态。

```
/*
 *  Purpose: Sample program of stat - to print the type of file
```

```
 */
#include <sys/types.h>
#include <sys/stat.h>
main(argc,argv)
int     argc;
int     *argv

struct stat fileinfo;
    char        *str;

        printf("File %s:", argv[1]);
        if (stat(argv[1],&fileinfo) < 0) {
                printf("stat error");
                exit(-1);
        }
        switch (fileinfo.st_mode & S_IFMT) {
        case S_IFDIR:
                str = "directory";
                break;
        case S_IFCHR:
                str = "character special";
                break;
        case S_IFBLK:
                str = "block special";
                break;
        case S_IFREG:
                str = "regular";
                break;
        case S_IFLNK:
                str = "symbolic link";
                break;
        case S_IFSOCK:
                str = "socket";
                break;
        case S_IFIFO:
                str = "fifo";
                break;
        default:
                str = "unknown file type";
```

```
            break;
        }
        printf("%s\n",str)"
        exit(0);
```

这个程序的输出结果如下：

```
/dev/tty: character special    或者
/usr/bin: directory
```

文件存取权限

文件可以读取、写入或执行，但是会针对文件所有人（owner）、所属组（group）或其他人（other）而有不同的权限，文件的权限可以用一个八进位的数值来设定或表示文件的权限：

八进制数值	意义
04000	执行时设定用户识别号（user id）
02000	执行时设定所属组识别号（group id）
01000	执行后储存文字影象（"sticky bit"）
00400	文件拥有者可读取
00200	文件拥有者可写入
00100	文件拥有者可执行
00040	文件所属组可读取
00020	文件所属组可写入
00010	文件所属组可执行
00004	其他人可读取
00002	其他人可写入
00001	其他人可执行

文件程序

本节介绍与文件相关的系统调用，包括文件的产生、打开，数据输入输出等各种程序，这些程序都用文件描述词作为文件的识别号：

```
#include    <fcntl.h>
int   open(char *pathname, int oflag   [,int mode]);
```

open 打开一个文件，较早的 UNIX 版本 open 只有前面两个参数，而且参数 oflag 的设定也比较简单：

0 (open for reading)

1 (open for writing)

2 (open for reading and writing)

现在的 UNIX （ BSD 4.3 版本以及 System Ⅴ）增加了第三个参数，对 oflag 的设定比较丰富：

O_RDONLY	打开的文件只可读取。
O_WRONLY	文件只可写入。
O_RDWR	文件可读取或写入。
O_NDELAY	文件打开、读或写数据时不可延迟（ block ）
O_APPEND	每次写入时，接续在原有数据之后，而不破坏原有数据。
O_CREAT	如果文件不存在，产生新文件。
O_TRUNC	如果文件存在，将此文件内容削减使其长度为零。
O_EXCL	如果文件已存在却又设定 O_CREAT，则报告错误。

这几个 oflag 互相可用 bitwise OR ，同时可以有多个设定，例如 O_CREAT|O_WRONLY 可以产生新文件，并且只可写入。第三个参数 mode 只在 oflag 设定为 O_CREAT 时才使用。程序 open() 传回 file descriptor ，如果打开不成功则传回 -1 。

```
int creat (char *pathname, int mode);
```

程序 creat() 产生一个新文件，并传回文件描述词，不成功则传回-1 。如果文件已经存在，再执行 creat() 会使文件长度被削减成为零。

```
int read (int filedes, char *buf, unsigned int nbytes);
```

此程序由目标文件（ filedes 所指的）读出 nbytes 的数据，存放在 buf 中，并且传回实际读取的数据数量，因此它可能等于 nbytes ，也可能小于 nbytes 。如果传回 0 ，表示已读到文件尾；如果读取失败会传回-1 。

```
int write (int filedes, char *buf, unsigned int nbytes);
```

write() 对目标文件（由 filedes 所指的）写入 nbytes 数量的数据，这些数据原本存放在 buf 中，实际写入的数据量会被传回（可能等于或小于 nbytes ）。如果写入失败会传回 -1 。

在稍后的 socket 程序库中也会介绍 read 与 write 除了对文件描述词读写之外，也可以对插座描述词进行读写，使用时只要把第一个参数换成插座描述词即可（也是整数），其他没有任何不同。由于参数 nbytes 是一个 unsigned int （ 16 位），所以每次读写长度不得超过 65536 个位，因此，在程序中需要循环（ loop)反复地作 read()

或 write() 的动作，一直到遇到文件终点（ EOF ）为止。

```
int close (int filedes);
```

关闭文件，参数则是文件描述词。要注意的是每个打开的文件都必须关闭。

```
long lseek(int filedes, long offset, int pos);
```

每个文件打开之后，都会有一个位置指针时时指向文件正在处理的某个字符，下次读写时会从这一位置开始进行。文件刚打开或产生时，此指针都指向文件头，每次 read() 或 write() 时会移动指针，但是除了 read 或 write，也可以用 lseek() 直接移动指针到指定的位置。

lseek 的第一个参数是文件描述词，第二及第三个参数的设定方法及意义如下：

pos=0 文件指针放置在自文件头开始 offset 位的位置，所以 offset 必须为正值。

pos=1 文件指针指向距离目前位置 offset 位处， offset 可设为正或负数。

pos=2 文件指针指向文件长度加上 offset 数值的位置， offset 可设为正或负值。

基本的系统进程

在网络软件上常会用到一些 UNIX 很基本的系统进程，它们虽然不是网络专有的系统程序，但也都不可缺少，下面一一予以介绍：

进程识别号（ process id ）

每个进程都有一个独特的识别号，称为 process id，简称 pid。 pid 在进程新产生时由系统设定成正整数，范围约在 0 到 30000 之间，但是有些数值已被指定为系统内定的进程识别号，例如， 0 是 swapper 或 scheduler 的进程识别号， 1 是 init 的识别号， 2 是 pagedaemon 的识别号等等。

一个新产生的进程也可以得到它自己的识别号，只需调用 getpid 即可，不用传入参数， getpid() 会传回识别号：

```
int getpid();
```

父进程识别号（ parent process id ）

每个进程都有父进程，在 fork 一节中有更详尽解释。进程也因此可得到它的父进程识别号，直接调用 getppid() 即可。

```
int getppid();
```