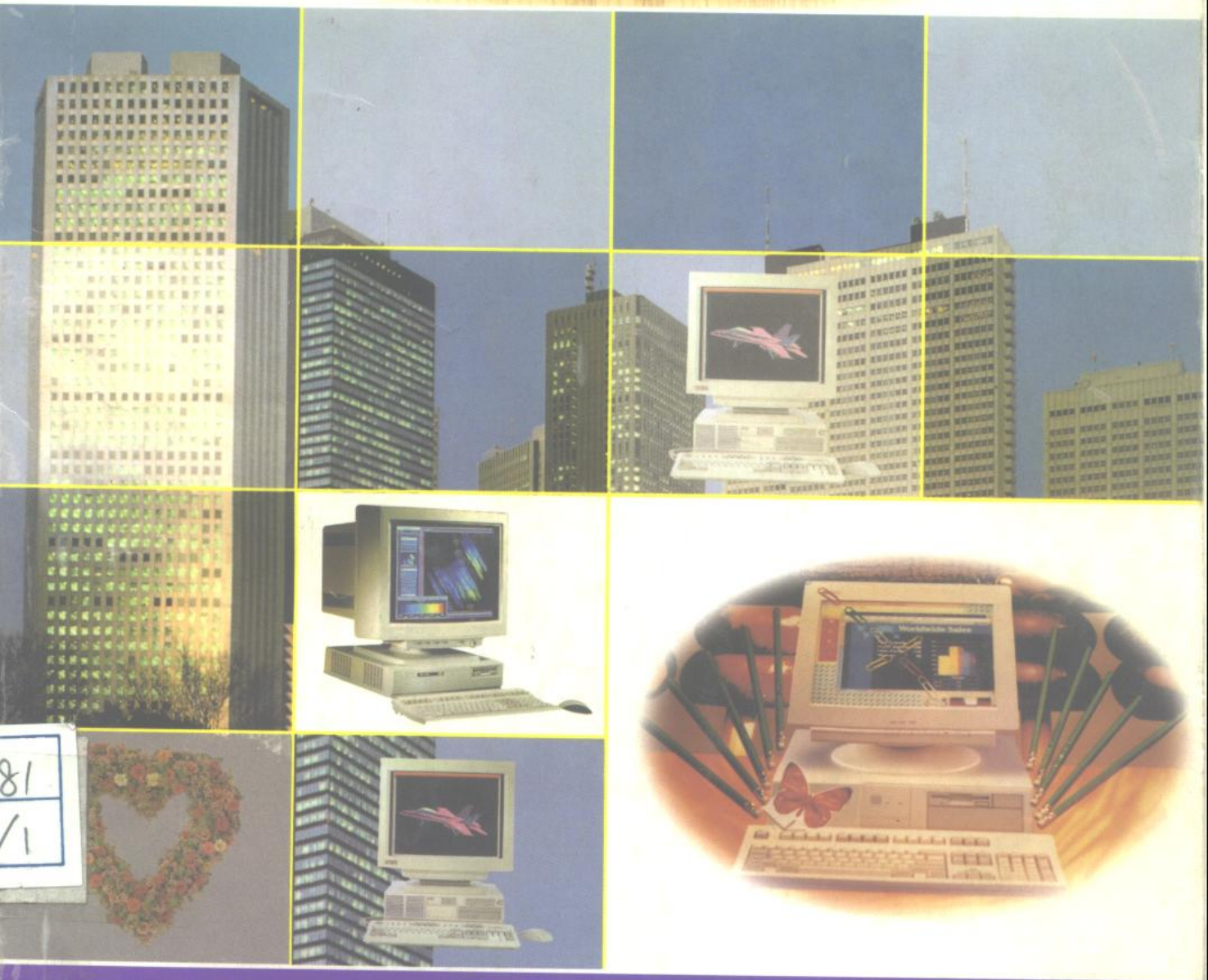
UNIX/XENIX程序设计技术与实例

李增智 解新忠 陈 娇 编著



西 安 交 通 大 学 出 版 社

UNIX/XENIX 程序设计技术与实例

李增智　解新杰　陈妍　编著

西安交通大学出版社

内容提要

本书共 11 章：概述，UNIX 文件操作，文件及目录操作，标准级文件输入输出，进程控制，进程通信，信号，shell 程序设计，标准库函数，网络程序设计，屏幕处理。每章均在介绍程序设计技术的同时，由简入繁，给出各类适合读者编程需求的实例。书后附录了 UNIX 的系统调用及基本库函数，Vi 的使用说明以及 UNIX 常用命令。

本书面向程序设计人员，所举实例，均在机上通过，具有较强的实用性。

(陕)新登字 007 号

UNIX/XENIX 程序设计技术与实例

李增智　解新杰　陈　妍　编著

责任编辑　李勤

*

西安交通大学出版社出版发行

(西安市咸宁西路 28 号　邮政编码 710049)

西安理工大学印刷厂印装

*

开本：787×1092　1/16　印张：14　字数：338 千字

1996 年 5 月第 1 版　1996 年 5 月第 1 次印刷

印数：1——5000

ISBN7—5605—0862—6/TP・139　定价：13.80 元

目　录

第1章

概述

1.1 UNIX简介

1.1.1 UNIX发展

UNIX操作系统最早是由美国贝尔实验室的K. Thompson和D. M. Ritchie于1969年在DEC公司的PDP-7机器上开发的。1970年该系统投入运行。1973年，D. M. Ritchie用C语言改写了原来由汇编语言编写的UNIX。从此以后，UNIX操作系统和应用程序几乎都用C语言书写。因此UNIX系统的可移植性能特别优越，在不同计算机上很容易通过移植来实现实用的操作系统。

经过20多年的不断完善，UNIX操作系统已发展成为当今唯一的全谱系操作系统，并被配置在从巨型机到微型机的各种计算机上，UNIX的优良性、开放性和可移植性得到人们的普遍青睐。

现在存在着五花八门的UNIX版本，主要有三种：

1. AT&T UNIX System V

System V的第1版是由美国电话电报公司于1983年公布的，该系统简单易学，易于维护。System V为进程间通信提供了许多手段。如消息队列，信号量及共享内存。到1989年末期已公布了System V Release 4.0版。

2. BSD(伯克利软件发行)

另一个UNIX分支是由美国加利福尼亚州立大学伯克利分校开发的BSD UNIX操作系统。最初，它是由贝尔实验室的V6派生而来，而后自具特色。目前，该分支已发表了4.3 BSD版本，它主要针对VAX-11机，其功能非常齐全，用户能够以很低的价格买到其源程序，因此得到了广泛地推广，成为目前最有影响的UNIX系统。

3. Microsoft的XENIX System V

它类似于AT&T UNIX System V，主要用于IBM个人计算机。一般在8086，80286，80386处理器上运行。

1.1.2 UNIX系统结构

UNIX是一个操作系统，由一组控制用户程序与最低层的机器功能交互的程序组成。UNIX操作系统控制计算机的资源，使多个用户可以并发访问这些资源，UNIX负责调度、控制与系统相连的外部设备，所有这些都是通过一个分层的软件体系结构来实现的。

UNIX系统的分层体系结构如图1-1。

该分层模型的中心是硬件，包括计算机系统的各个物理成份。紧靠硬件的是UNIX内核，它是整个操作系统的核心，由功能独立的几个部分组成，负责进程管理、存储管理、设备管理和

文件系统管理。它提供硬件和用户间的低层界面。由于内核只占很小的存储空间，能够常驻内存，从而从根本上保证了系统能够以较高的效率运行，使诸进程并行，多用户分时使用 UNIX 系统。中间层为外壳 shell，shell 为用户和操作系统间提供了一个交互式的接口，是 UNIX 系统命令解释程序。外层为应用层，包含有非常丰富的语言处理程序、系统实用程序和开发软件的工具性软件。所有这些程序可通过 shell 命令使用，这些系统软件向用户提供了相当完备的程序设计环境。

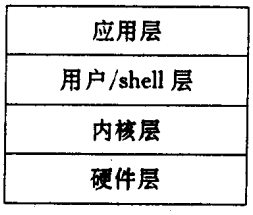


图 1-1　UNIX 的分层体系结构

UNIX 系统的内核向外壳程序提供了充分和强有力的支持，外壳程序则以内核为基础，灵活而恰到好处地运用了内核的支持。这种有机的结合，构成一个整体，实现了非常强大的功能。

1.1.3　UNIX 文件系统

UNIX 文件系统是 UNIX 操作系统的一个重要组成部分。系统和用户文件都存放在物理磁盘介质上的文件系统中。UNIX 文件系统的组织看起来像一棵倒置的树，根在最上面，向下扩展。如图 1-2 所示。

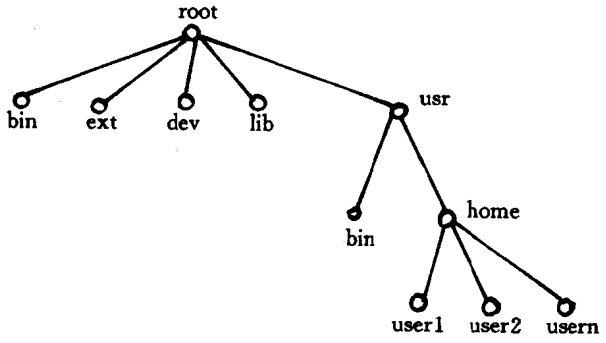


图 1-2　UNIX 文件系统

每个 UNIX 系统都有一个唯一的根目录，即树根。其下有若干子目录及文件，相当于树枝，每个子目录下面还可以有从属的子目录和文件，如此类推。在描述这种树型结构的 UNIX 系统时，用斜线“/”作为目录与目录、目录与文件之间的分隔符。单独的“/”表示根目录。

在这种树形结构的文件系统中，目录或文件的深度不受限制。

UNIX 文件系统是可装卸的。用户可以在磁盘上任意建立文件系统。根文件系统之外的文件系统可以直接或间接地与根文件连接起来使用，这种连接被称为安装。如图 1-3 所示。

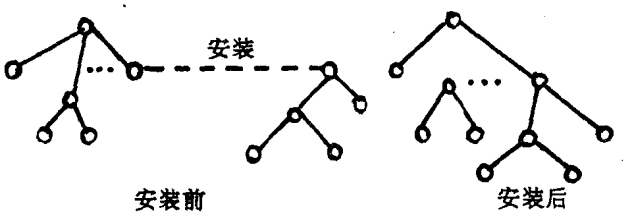


图 1-3 UNIX 文件系统安装示意图

图示将一个完整的文件系统安装到另一个已经存在的文件系统的某个目录下，嫁接成一个整体，需要时还可以卸下来，通过安装，可以使多台磁盘机上的文件系统在逻辑上作为一个文件系统使用，因而可以任意扩大存储空间，具有较大的灵活性和安全性。

需要说明的是：通过安装使得若干文件系统具有联系，并在逻辑上构成一个单独的文件系统，而在实际上，这些文件互相是独立的。

1.1.4 命令程序设计语言——shell

UNIX 的另一个特点是提供了称为外壳 shell 的命令程序设计语言。这种命令程序设计语言，不但具有一般命令语言的功能，用以解释执行命令，而且具有程序设计语言的特点，可以用判断、循环等语句设计实现较高的功能。它作为 UNIX 核心与用户程序之间的界面，向用户提供高性能的访问 UNIX 系统的接口。

UNIX 系统一般有三个常用的 shell：

Bourne	shell：	/bin/sh
korn	shell：	/bin/ksh
C	shell：	/bin/csh

当你注册进入一个 UNIX 系统时，将指定的 shell 投入执行，它就等待接受用户的命令并解释执行之。shell 类型在/etc/passwd 文件中指定，缺省情况为 Bshell。

在某种意义上，shell 可以看作是 UNIX 提供给用户的所有操作系统命令的集合。

1.2 UNIX 系统概念

由于 UNIX 是完全并发的操作系统，它不同于现今人们熟悉的 DOS，因此具有自己独特的概念。以下说明一些基本的 UNIX 概念。

1. 进程

进程是正在执行的某个程序，UNIX 中所有用户工作由进程完成。在 UNIX 系统中，进程只能通过 fork 系统调用创建，一个进程也可以执行另一个程序来代替自己，这个机制非常精巧且高效(详见进程控制一章)。由于 UNIX 是多任务操作系统，因此系统中同时有多个进程运行。

2. 系统调用

UNIX 提供了上百个能直接获得内核服务的入口，这些入口称为系统调用。系统调用是执行特定功能的一组子程序，这些系统调用的功能异常丰富，从与时间和日期相关的，到那些允

许进程共享信息的，还有诸如完成文件读写、设备控制功能等，可以说是面面俱到。

许多系统调用返回“－1”作为错误码，表示系统调用失败，返回大于或等于 0 的数表示正确返回。

3. UNIX 文件

UNIX 文件的主要特征是没有记录的概念，它是一个字节序列，文件中数据的含义完全由使用它的程序决定。UNIX 本身对数据不做任何解释，对文件内容亦不加任何限制，由于文件形式相同，所以能够很自然地将一个程序的输出用作另一个程序的输入，即通过文件直接实现不同程序（甚至可以是用不同语言书写的程序）之间的数据传递。把文件的这种特性和已有的应用程序相结合，可以实现非常多的功能。

4. UNIX 文件类型

在 UNIX 中，把目录及与计算机连接的多种输入/输出设备（终端、打印机、磁带、磁盘等）都作为文件来处理，并对所有的输入输出提供统一的界面。

UNIX 文件有三种类型：

普通文件：普通文件包括可阅读的正文文件或源程序。另外，可执行文件（二进制）也按普通文件存储，它是不可阅读的。

目录文件：在 UNIX 中，存储空间被分成许多块，每一块空间都有一个目录文件，用来保存该块空间所存文件的各种信息。为确保目录系统的完整性，系统又允许读取目录，而不能对其本身进行编辑或修改，用户只能通过在目录中增加文件、删除文件、建立目录或删除目录来使用目录文件。

特殊文件：特殊文件是与硬件设备有关的文件。信息不由该文件保存，而由设备直接提供，它只是操作系统内核与 I/O 设备之间的通道。在系统中，I/O 设备的访问和普通文件的访问一样，每个 I/O 设备至少要有一个特殊文件与之对应。因此，在程序中可以用与读写一般文件相同的方式来读写特殊文件，即通过文件名来进行读写。特殊文件一般在/dev 子目录下，它分为两大类：块特殊文件和字符特殊文件。

5. 主次设备号

每个 I/O 设备都有主次设备号。特殊文件名与 I/O 设备的设备号有关。主设备号是用来区分设备的类别的，如硬盘、软盘、打印机、终端等就属于不同的类别，它们的主设备号不同。次设备号反映了同类设备的不同台号。一般系统中主要设备的主设备号有：

终端	tty
硬盘	hd
软盘	fd
打印机	lp

文件（sys/types. h）中定义的数据类型 dev_t，用来保存这两个设备号。dev_t 是一个短整数（两个字节），主、次设备号各占一个字节。

6. 目录

目录是由内核维护的特殊类型的文件。它只能由内核修改，进程只能读它。目录看似像一个一般文件，目录的内容是文件名和 i-节点号的清单。

7. i-节点号

UNIX 系统中，每个文件都被赋予了一个编号，称为 i-节点号。每个文件实体都对应一个

唯一的 i-节点号。一个文件实体可能有多个文件名，因此，若几个文件 i-节点号相同，则表明这几个文件名对应的是同一个文件实体，即该文件实体有这几个不同的文件名。

8. 路径名

文件系统中从根目录至某个目录或文件的完整的路径标识符，称为该目录或文件的路径名。它实际上反映了文件系统中从根目录开始到指定的目录或文件为止的路径。由于路径名比较冗长，尤其对处于树结构深处的目录或文件而言更是如此，为克服这一弊病，引入了当前目录这一概念。

9. 当前目录

用户正在操作使用的目录称为当前目录，也称当前工作目录。在当前目录中的文件可以直接存取，即通过直接打入文件名就可引用文件。在它下面的所有目录和文件的路径名都可以从它开始来表示，称为相对路径名。

10. 登录目录

在为用户建立登录帐号时分配给用户的文件系统中的一个目录。按照惯例，登录目录与用户的标识符名字相同，这一目录是用户在登录完毕时的当前目录。在登录目录中，用户对自己创建的文件和子目录有全部的存取权限和控制权，系统中用户的登录目录可以由系统管理员指定，一般在/usr 子目录下。用户只能在他们拥有足够权限的地方创建或修改文件或目录。

11. 超级用户

超级用户的注册名一般是 root，超级用户可以自由地访问文件，不受任何权限的影响。一般由系统管理员作为超级用户注册，完成系统所需的维护工作。超级用户的用户号为 0。

12. 进程标识号

UNIX 系统中，每个进程都有 4 个与之相关的标识号，它们分别是：

实际用户号：文件系统中用户号用来记录文件的属主。每个注册的用户都有唯一一个与之相对应的用户号。它是一个整数值，文件/etc/passwd 中有每个注册用户的用户名和用户号。

有效用户号：一般情况下，此值与实际用户号的值相同。

实际用户组号：每个用户还有一个正整数值的用户组号。它一般用来标识 UNIX 系统中同一管理部门或者同一项目组中的所有用户。同一用户组下面一般有很多用户。文件/etc/group 包含了所有同组用户名及与之相对应的用户组号。

有效用户组号：一般情况下，此值与实际用户组号相同。

13. 文件访问权限

在 UNIX 中，用户对文件的访问是有限制的，这主要是因为 UNIX 是多用户操作系统，从安全性角度来考虑设置的。文件的存取控制是按三个级别进行的，分别称为文件主用户、同组用户和其他用户，每个级别的存取权限的类型是相同的，分别为读、写和执行是否允许。

UNIX 内核要经过下列判断才能决定一个用户对一个文件能否访问。

●若进程的有效用户号为 0(超级用户)，允许访问。

●若用户的有效用户号和文件属主的用户号相匹配，而且文件的相应访问位(读、写、执行)被置位，允许访问。

●若用户的有效用户号与文件属主的用户号不匹配，而有效用户组号相匹配，且文件的相应同组用户访问允许位置位，允许访问。

●若用户的有效用户号和有效用户组号均与文件属主的相应标识号不匹配，且文件的相

应其他用户访问允许位置位，允许访问。

●否则，不允许访问。

从上述分析中可知，UNIX 是按用户号、同组用户号、其他用户三个级别依次判断的。

14. 进程号

每个进程都有一个独特的进程号 PID，它是一个大于 0 的整数。每当内核创建一个进程，它便赋予此进程一个进程号，系统用进程号对进程进行唯一的标识。

在 UNIX 系统中，有三个进程号具有特殊的含义：进程号为 0 的进程为调度进程，进程号为 1 的是初始化进程 init，进程号为 2 的进程是页守护进程，负责虚拟存储管理。这三个进程由 UNIX 内核维护。其他进程号都没有特殊的含义。

15. 文件描述符

文件描述符是用来标识为 I/O 而打开的文件的一个小整数。允许值从 0 开始，上限值随系统而异。UNIX 系统中，标准输入、标准输出、标准错误输出分别分配描述符 0，1，2，且每次分配描述符时，都分配系统中可用的最小的文件描述符。UNIX 的这一属性，带来了极大的灵活性，如 I/O 重定向。（详见标准输入输出一节）

1.3 UNIX 目录结构

在文件数目巨大，特别是在多用户的情况下，如不对文件进行分区管理，那么按文件名存取文件的方法就不适用了，因为不同的用户可能使用相同的文件名，这时，按名取文件就不知该取哪一个。因此，UNIX 采用了多层次的目录结构方式，对文件进行分组存放，这样不同层次，不同目录中的文件即使同名也无妨了。

为使读者对 UNIX 整个系统有所了解，下面列出一些在 UNIX 系统中最普通的目录：

/bin：存放一些基本的 UNIX 命令。

/dev：存放设备文件。该目录下，console 指系统的主控终端，tty 表示用户的当前登录终端。在进行程序设计时，如果某些提示信息一定要写在终端上，就可以把它们写到 tty 文件中。这样，无论如何重定位，该信息总会显示在设备上。

/dev/null：可以看作一个垃圾桶。在程序执行时，对于那些不想看到的信息，都可以把它们重定位到该文件中。

/etc：主要存放一些系统管理命令，及相应的一些管理表格。

/usr：含有一些与 UNIX 用户有关的子目录。按照惯例，用户的登录目录也放在该目录下。

/usr/include：该目录下的文件对程序员来说特别重要。这些头文件提供了在用 C 语言进行开发时，所有使用的标准数据结构的说明。在 include 目录中，还有一个子目录 sys，该目录下一般存放与系统调用相关的一些头文件。该目录下的文件可以说是信息金矿。

/usr/lib：该目录下，最主要的是目标库。存放程序中外部引用时要用到的编译过的函数库。像编译时用到的 C 库就在该目录下。

/usr/sys：通常放有重新生成核心时所要使用的管理文件及目标文件。

1.4 UNIX 程序设计环境

UNIX 系统为程序员提供了一个十分吸引人的程序设计环境，并提供了许多支持编程的工具。它不仅提供了 C 编译器和调试程序，而且对其他语言如 Basic、Fortran、Pascal 等都有相应的语言处理器。读者可根据自己的情况选择任一程序设计语言完成自己的要求。本书的实例均以 C 语言程序的形式给出。

为了提高编程效率，在应用程序开发过程中应尽量采用下列技术：

●一个程序只完成一个功能。

●避免不必要的杂乱的程序输出，应尽量使一个程序的输出为另一个程序的输入。

●对现有的工具，应尽量使用或稍作修改后使用，避免从头开始重新设计一个工具。

●应把工作划分成尽可能小的部分，然后逐步增加，直到完全实现要求的功能。这需要在写程序前，结构设计必须合理。

上述四点是程序员应遵守的一般原则，而不是必须如此。之所以这样要求，是从开发效率上来考虑的。遵从上述原则开发的程序一般错误较少，开发期较短，移植较容易。

下面简要地介绍一下 C 语言程序的实现过程。

1. 用 ed 或 Vi 编辑源程序

ed 和 Vi 是 UNIX 系统提供的正文编辑器，其中 ed 为行编辑程序、Vi 为全屏幕编辑程序。所谓全屏幕就是指光标可以移动到屏幕上文本所在的任何地方，进行插入、删除和修改。Vi 提供了比 ed 功能更强的编辑功能，格式为：

ed　文件名　　或　　Vi　文件名

这里的文件名必须有后缀 .C，说明 C 源程序，它既可以是一个新文件，也可以是一个已存在的文件。如果文件已存在，则将此文件装入内存，并显示在屏幕上，供用户修改。如果文件不存在，则创建该文件并进行编辑。

当编辑完成后，一般要保存文件，并退出编辑程序。

附录 B 中详细地介绍了 Vi 的用法。

2. 编译、链接 C 程序

源程序一旦编辑完成，可以用 CC 来对程序进行编译、链接。编译工具由预处理程序，优化程序、汇编程序和装配程序组成。CC 的格式为：

CC　　[option]　　文件名

option 为任选项，最主要的有：

－C：源程序只编译，不链接，它将产生目标文件，目标文件是与文件名相同后缀为 .O 的文件。

－O〈outfile〉：产生一个名为 outfile 的输出文件，这个缺省文件名称是 a.out。

例如： CC　－C　myprog.C

将编译 myprog.C 程序，并产生名为 myprog.o 的目标文件。

CC　－O　myprog　myprog.C

编译并链接 myprog.C 程序，生成可执行文件，文件名为 myprog。

3. 运行程序

一旦程序正确地编译、链接后，就可以执行。在 shell 命令方式下，键入可执行文件的文件名，即可运行该文件。

4. 调试程序

一般说来，大多数程序第一次运行时都不能保证完全正确。如果程序运行失败，必须从程序的执行情况中判断问题之所在。一般有两种选择：一是在程序中加上一些 printf 语句打印出运行时关键变量的内容；另一种方法是使用调试工具来调试。adb 和 sdb 是两种可用的调试工具。

一个程序运行失败可能由于：

●程序无限循环。

●由于内容错误或其它硬件错误，使程序意外地退出。

●虽然程序运行了，但不是希望中的结果。

●当获取所需的资源时（如文件）失败，使程序退出。

一个 C 语言程序必须不断地经过上述四个步骤的重复，直到得到正确的可执行文件。

第2章

UNIX 文件操作

在 UNIX 中，所有的输入、输出都被看作是文件访问，这里的文件已不再是传统意义上的磁盘文件，而且还包括诸如显示终端等外部设备，即使由管道实现的进程间的通信也被处理成文件访问。

和其它许多操作系统不同，UNIX 只有一种类型的文件——顺序文件。每个顺序文件包含无结构的字节序列，读和写一般也是顺序执行的，每次读/写操作后，指针指向下一个字符。UNIX 提供了一个系统调用，它可以允许文件指针指向文件的任何位置，经过它，可以实现文件的随机访问。

要对一个文件进行访问，必须首先打开要访问的文件，然后进行读、写操作，访问结束后，关闭文件。系统中有三个文件一直保持打开，它们是标准输入、标准输出和标准错误输出。

UNIX 系统提供了两个级别的输入输出方式：

●UNIX I/O 系统调用。

●标准 I/O 库。

UNIX 的 I/O 系统调用为内核的直接入口，兼容性较差，对程序员来说，系统调用使用起来更难，因为它们功能较差，也没有格式化输入输出的能力。

标准 I/O 库为进程和内核之间提供了高层的接口，提供了诸如缓冲、逐行输入、格式化输出等功能。这些函数更符合 C 语言的普遍标准，与其它机器上的 C 语言更兼容，更容易移植到其它机器其它操作系统上。对程序员来说，标准级函数也更容易使用，更高级、功能更强。标准级函数有时也称为流式输入输出函数。因为在使用这些函数时，通常把操作的文件看作是一个字符流。

本章主要介绍系统级文件 I/O，标准级文件 I/O 将在第 4 章中进行讨论。

2.1 UNIX 系统级文件 I/O

本节将介绍 UNIX 提供的访问文件的基本操作。每一项基本操作对应于一个系统调用，这些系统调用可以对 UNIX 内核提供的 I/O 功能进行直接访问。UNIX 系统内任何其它的文件访问机制，最终都是通过这些系统调用实现的。

表 2-1 列出用于 UNIX 系统级文件 I/O 操作的系统调用。

2.1.1 文件的打开

在对一个文件进行读写操作前，必须打开此文件。open 系统调用产生一个文件或打开一个已存在的文件供读写。

```
#include <fcntl.h>
```

```
int open(name,mode)
  char *name;
  int mode;
```

name 是要打开的文件名，它是以'\O'结束的字符串。

表 2-1

系统调用	说明
open	打开一文件，用于读或写
creat	建立一个空文件
read	从一文件中读取数据
write	向一个文件写数据
lseek	移动文件指针到指定位置
unlink	删除一个文件

mode 为文件状态标志，UNIX 允许用符号表示参数 mode 这些值在头文件(fcntl.h)中定义，mode 的值是通过下列表中的标志进行"或"运算而构成的。(前三个标志只能用一个)

- ●O-RDONLY　打开文件仅用于读。
- ●O-WRONLY　打开文件仅用于写。
- ●O-RDWR　打开文件用于读写。
- ●O-NDELAY　打开文件，读写中无阻塞。
- ●O-APPEND　每次写在文件末尾。
- ●O-SYNC　每次写都等待文件的数据和文件的状态被物理更新。
- ●O-CREAT　如果文件不存在，则创建文件。如果文件存在，则这个标志没有作用。
- ●O-TRUNC　如果文件已存在，则其长度被截短至 0。
- ●O-EXCL　如果 O-EXCL 标志和 O-CREAT 标志被设置，则在该文件已存在的情况下，open 失败。

一般把 O-PDONLY 定义为 O，O-WRONLY 定义为 1，O-RDWR 定义为 2，这是从程序的兼容性考虑的。

如果 open 系统调用成功，则返回分配的文件描述符的值，否则返回－1。open 成功后，读写文件位置被置成文件起始位置。

2.1.2 文件的建立

UNIX 系统调用 creat 用来创建新文件或重写已存在的文件。它返回一个所建立的文件的文件描述符。

```
int creat (name,code)
  char *name;
  int code;
```