UNIX 网络程序设计

甘登岱　李广东　廖彬山　高峰霞　编

```
    int     i, iphdrlen, triptime;
struct ip      * ip;
register struct icmp    icp;
long    icp;
long    * lp;
struct timeval    tv;
ciar    tv;
ciar    * pr__type ();
from->sin__addr, s__addr=ntoil(from->sin__addr, s__addr);
if (timing)
      gettimeofday (&tv, (struct timezone  * ) 0);
ip=(struct ip  * ) buf;
iphdrlen=ip->ip__hl<<2;
if (cc  <iphdrlen+ICMP+MINLEN) {
      if (verbose)
         printntf ("packet too short (%d bytes)from %s\n", cc,
                  inet__ntoa (ntohl (from->sin__addr, s__addr) );
         return;
   }
   cc-=iphdrlen;
   icp=(struct icmp  * ) (buf+iphdrlen);
   if (icp->icmp__type!  =  ICMP__ECHOREPLY) {
```

航空工业出版社

UNIX 网络程序设计

甘登岱　李广东　廖彬山　高峰霞　编

1993

(京) 新登字 161 号

内 容 提 要

UNIX 是当今计算机的主流操作系统之一，UNIX 网络环境提供了多种连网手段，具有远程命令执行、远程登录、多种进程通信和丰富的接口功能。

本书详细介绍了 UNIX 网络程序设计，内容包括 UNIX 通信协议及两种 UNIX 通用版本 4.3BSD 和 System V 的网络系统接口，并给出几个实用例程。全书共分十八章：概述、UNIX 模型、进程通信、网络基本术语和概念、通信协议简介、Berkeley 套接字、系统 V 传送层接口、网络库例程、网络安全性、时间和日期、PING 例程、小型文件传输协议 TFTP、行打印机假脱机系统、远程命令执行、远程注册、远程磁带驱动器访问、性能、远程过程调用。

本书资料新颖、内容翔实，可供大专院校计算机软件与应用专业的师生，以及计算机工程技术人员阅读参考。

UNIX 网络程序设计

甘登岱 李广东 廖彬山 高峰霞 编

航空工业出版社 版发行

(北京市安定门外小关东里 4 号)

—邮政编码 100029—

全国各地新华书店经售

北京地质印刷厂印刷

1993年 3 月第1版 1993年 3 月第1次印刷

开本 787×1092毫米 1/ 6 印张 15

印数 1—3500 字数 370 千字

ISBN 7-80046-462-8 / TP · 031

定价 3.50 元

目　录

第一章　概　述

1.1 网络历史

计算机网络的广泛使用使计算机的应用产生了一次革命。计算机网无所不在，从飞机订票系统，到电子邮件，再到电子简报等等。计算网蓬勃发展的诸多原因如下：

- 80年代个人计算机和工作站的增长使得对网络的需求增加。
- 过去由于计算机网比较昂贵，所以只局限于大学、政府研究机构以及大公司使用。随着技术的日益成熟，计算机网的价格逐渐降低，因而使之在越来越多的地方找到了它的用户。
- 现在许多计算机厂商把网络软件作为基本操作系统的一部分，网络软件不再被认为是只有少数客户需要的附加器。
- 我们正处于信息时代，而信息的传播离不开计算机网络。

过去的计算机系统都是独立的。每个计算机都是自封闭的，其所有外设和软件都是用来完成某一特定的工作。如果要完成某一特定的工作，比如要打印输出，那么就得把打印机联到系统上。如果需要大量的磁盘空间，就得把磁盘机联到系统上。计算机共享信息和资源的出现改变了上述状况。

信息共享可以是电子邮件和文件传递。资源共享包含访问其它系统的外设。20 年前这种类型的共享是通过交换磁带、穿孔卡及行打列表实现的。当今计算机通过网络连接起来。一个简单的网络可以是由调制解调器连接起来的两台计算机。而复杂的网络和 TCP／IP Internet 却可连接 150，000 多个系统。连接网络的方式随我们要求网络做的事情的不同而多种多样。一些网络的典型应用如下：

- 与其它计算机上的用户交换电子邮件，当今人们通过电子邮件通信已经很平常了。
- 系统之间交换文件。通过网络传递文件比邮寄磁盘和磁带方便快速，而且容易。
- 共享外部设备。其中包括共享行式打印机和磁带等。外设共享的趋势很大部分是因为个人计算机和工作站市场的扩大，而外设的价值常超出计算机本身的价格。在一个有许多个人计算机和工作站的机关中，共享外设是可以理解的。
- 在别的机器上运行程序。在许多情况下，某一特定的程序更适于在别的机器下运行。比如，在一个分时系统上或一具有良好程序开发工具的工作站上编辑和调试程序无疑是最佳选择，而另一系统可能更适合于运行此程序。不同的程序需要不同的环境。比如，有的需要并行环境而有的需要海量存贮环境。
- 远程注册。若两台计算机通过网络连接在一起，便可在一台机器上注册进入另一台机器。具有远程注册的网很容易地把计算机连接起来了，而不必把网上的每个终端节点完全连接。

在以后的各章中，我们将更加详尽地讨论以上各种功能。

1.2 OSI 模型

首先介绍网络层次模型 OSI(国际标准组织的开放互连模型)。这是一个 7 层的模型。

OSI 模型为描述网络提供了一个详细标准。当今大多数计算机网络都是用 OSI 模型描述。

由于传输层是提供可靠数据传送的最低层，因而传输层是比较重要的。它为以上各层提供了一个可以认为是无差错的传输路由。顺序控制、错误检测、重传以及流控制一般都由传输层及其以下各层来实现。

1.3 进程

计算机网络的基本实体便是进程。一个进程是一个正在运行的程序。当我们说两台计算机正在通信，这便意味着运行于两台机器上的两个进程正在通信。

第二章将详尽地描述 Unix 的进程。第三章将描述 Unix 进程间的通信。

在第六和第七章中将把进程间通信扩展到网络中不同系统上。

1.4 简化模型

本书主要关心 OSI 模型的最顶上三层即对话层、表示层和应用层以及这三层与传输层的接口。通常把此三层结合为一层且称之为进程层。另外，还把最下两层结合为一层称之为数据链层。我们将使用一简化的 OSI 模型如图 1.1 所示。

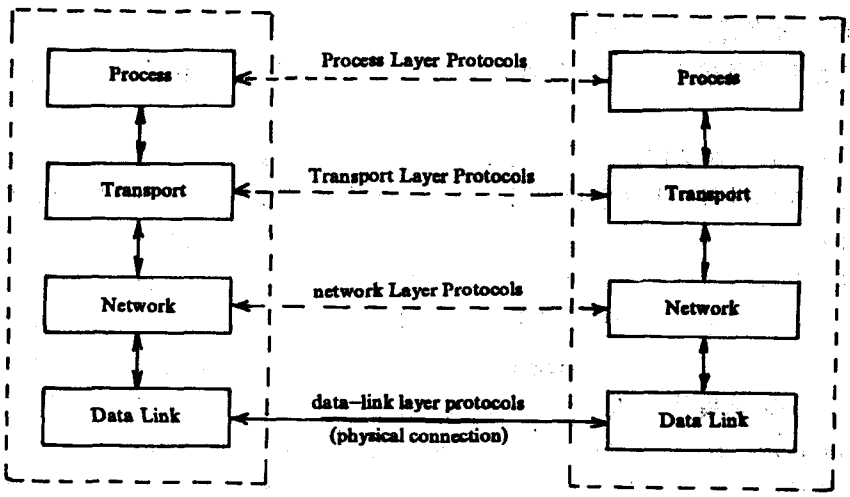

图 1.1 简化的 4 层模型

图 1.1 说明了网中互相连接的两个系统。其实这两个系统真正相连的为数据链层。虽

然表面上看这两个进程在互相通信，但实际上数据是从进程层流向传输层，经网络层到达数据链层，然后经过物理线路到另外一个系统的数据链层，再由上通过网络层，传输层，最后传到另一进程那儿。

层的概念导致了每层上协议的定义。每一层都存在协议，最底层也不例外。我们把图1.1中四个水平线表示的层的协议称为同级协议。同级协议用于相同层之间。第五章将讨论数据链、网络和传输层的各种协议，之后各章将讨论特定的应用程序，提供进程层上各种面向应用的协议。

由于数据总是在一给定层及其层以上一层或下一层之间流动。因此层之间的接口对网络程序设计是很重要的。第七章将详细描述传递层与以上层的接口。

图1.1所示模型的另外一个特点是从最低层到传输层都包含在主机操作系统中。

1.5 客户–服务器模型

网络应用的标准模型为客户–服务器模型。服务器是这样一个进程，它一直等待着客户进程的请求以便为客户进程做一些事情。典型的情况如下：

- 服务器进程在某一计算机系统上开始执行。它初始化本身，然后进入睡眠状态以等待客户进程的请求。
- 在本系统或与网络服务器相连的其它系统上，某一客户进程开始执行。通过网给客户进程把请求发送给服务器进程要求服务。服务器进程能够提供如下形式的服务：
  - 把当天的时间返回给客户进程。
  - 为客户进程存取文件。
  - 允许客户注册进入服务器系统。
  - 为客户在服务器系统上执行命令。
  - 服务器进程为客户进程服务完后，便进入睡眠状态等待着另一次客户请求。

进一步讲，我们可以把服务器进程分为两类：

1. 若服务器进程能够在已知的短时间内处理完请求，那么便亲自处理它。称这些服务器为重复服务器(iterative servers)。告知时间便是以这种方式完成的。

2. 当服务器进程事先不知道要化多长时间完成服务请求，服务器便以并发的方式处理它，称这种服务器为并发服务器(concurrent severs)。并发服务器激活别的进程完成服务请求，而本身转入睡眠状态等着另一个请求。很自然，这种类型的服务器需要在多进程环境下进行。大多数处理文件信息的客户请求(如打印文件，访问文件等等)皆以这种方式处理。

1.6 Unix 网的历史

第一个 Unix 网络应用程序为 UUCP。它由 Unix 的拷贝程序加上相关的各种命令组成。UUCP 大约于 1976 年开发出来，于 1978 年随着 Unix 版本 7 对外发行。UUCP 为一面向批处理的系统。它主要用于以电话线或直接相连的系统之间。它的主要用途是传送

软件和电子邮件。第五章将更详细地描述 UUCP。有趣的是作为最早的 unix 网络应用程序的 UUCP 至今仍广泛使用。

程序 CU 也是 1978 年随着 unix 版本 7 发行的。此后几乎每个 unix 系统都包含它。虽然严格说来 CU 不是网络软件包的一部分，但它提供了注册进入别的系统的功能。重复的 CU 随着 4.2BSD 出台，更名为 tip。

1978 年 Eric Schmidt 在伯克利为 unix 开发了一网络应用程序作为其硕士论文的一部分。此网被称为 Berknet，随着 PDP-11 上的 unix 版本 7 对外发行。它提供了文件传送，电子邮件和远程打印的功能，被广泛地用于伯克利校园。此系统通过 9600 波特率的 RS-232 线直接相连。这个软件被移植到配有 4.XBSD 的 VAX 机上，然后逐渐被快速的基于以太的局部网代替。

1980 年 9 月 Bolt、Berrnek 和 Newman 与美国国防部的 DARPA 鉴订合同在伯克利的配有 Unix 的 VAX 机上开发了一执行 TCP／IP 协议的系统。1981 年秋天，一个在 4.1BSD 上的版本转交给伯克利。随后伯克利把它综合到 4.2BSD 上。4.2BSD 于 1983 年 8 月对外发行并得到很快的发展。这主要归功于基于以太技术的局部网的发展。4.2BSD 也允许计算机同 80 年代初期蓬勃发展的 ARPANET 相连。

此外，这段时期网络在 AT&T 也得到发展，然而除 uncp 外，研究结果只局限于 AT&T 之内。这与伯克利广泛公布自己的网络发展成果形成鲜明对比。Fritz、Hefner 和 Raetigh 描述了一基于 HYPER 通道的变速局部网。其发展开始于 1980 年左右。它能够把运行不同版本 unix 的不同类型的机器如 VAX780，PDP-11／70，IBM370 及 AT&T3BZOS 等连接起来。这个网面向批处理，支持文件传输，远程命令执行，远程打印及电子邮件。

80 年代中期 System V 上各种各样的网络软件涌现出来。这些软件一般都支持 TCP／IP 协议且通常由开发硬件接口的厂商开发。

第二章　UNIX 模型

2.1 简介

本章首先介绍一些基本的定义，概括地描述一下 Unix 的输入和输出。然后考虑两个重要的概念：信号和进程控制。在 2.6 节开发了一个守护进程(daemon process)的框架。以后各章中将要用到守护进程和开发它所需的许多概念。

2.2 基本定义

以下定义一些基本的名词来描述 Unix 进程。本部分为其余各章提供了词汇表。

C 程序设计语言

本书例子都用 C 语言描述。1989 年制定出了 ANSI 的 C 语言标准。此标准是由 Kernighan 和 Ritchie 共同制定。1988 年便出现了符合 ANSI 标准的 C 编译。由 Kernighan 和 Ritchie 较早制定的 C 语言版本仍有重要的地位。

标准 C 库

每一种 C 编译都提供了频繁使用函数的库。Unix 系统一般把库函数放在 /lib/libc.a 中。Unix 程序员常常意识不到这个库的存在。因为 Unix 的 CC 命令自动把库包含进去。

直到 ANSI C 的出现，才有了标准的库函数定义。

本书中的标准 C 库指的是 Unix 中的 C，因而有些函数可能在 ANSI 标准中无定义。这是因为标准的 Unix C 库包含两类函数，一类是任何 C 编译都提供的(如标准 I/O 库)，一类是 Unix 系统调用(如 read，write，ioctl，pipe 等)。别的操作系统也尽可能模仿 Unix 的系统调用。

Unix 版本

现在，存在着五花八门的 Unix 版本。其中最主要的有如下三种：

1. AT&T Unix System V

System V 的第一版于 1983 年公布。System V 为进程间通信提供了若干设施，如消息队列、信号量以及共享内存。System V 有各种各样的版本。其发展过程如下：

· 1984 年 8 月 System V Release2.0 公布。这一版本为上一版的升级。

· 1984 年 11 月 System V Release2.0 增强版公布。这个版本增加的重要功能有：咨询文件、记录锁定以及请求调度。

· 1986 年 System V Release3.0 公布。它提供的功能有：RFS(远程文件系统)、流、TLI(传送层接口)、TDI(传输供应接口)、共享库以及强制性文件和记录锁定。

· 1987 年 System V Release3.1 公布。这只是一般的升级。

· 1988 年中期 System V Release3.2 版公布。这个版本支持 Intel 80386 处理器。这个版本在二进程代码一级与 Xenix 程序兼容。

· 1989 年末期 System V Release4.0 版公布。这个版本是由 AT&T System V 和 Sun Microsystems Sunos 结合而成。同时它也具有 Microsoft Xenix System V 的特点。并且提供了一个与 ANSI X3jll 国际标准相一致的 C 编译。

从网络的观点来看，随着 Release 3.0 而引入的流、TLI 和 TDI 等概念使得开发网络应用程序更容易。由 Release 2.0 增强版提供的文件和记录锁定的技术简化了网络应用程序的代码量。

2. 伯克利的软件分布系统(BSD)

加州大学伯克利分校的计算机系统研究小组于 1980 年 8 月公布了 4.3BSD 的虚拟 VAX-11 版。而在它之前有 1982—1983 年公布的 4.1BSD 和 4.2BSD，它们是首先得到广泛推广的带有大量网络支持软件的 Unix 系统。因为 BSD 系统对 DARPA 网络协议(TCP／IP)以及各厂商的以太网硬件完全支持，因而促进许多局域网的建立。除了 DEC 公司以外许多计算机厂商都把伯克利系统装在他们的硬件平台上。此外，许多厂商以 4.3BSD 为出发点，增加了许多具有 System V 特点的命令。

1988 年 6 月，4.3BSD 的改进版"4.3BSD Taboe"公布。这个版本为一过渡性的版本，主要是为了有经验的用户测试和评价。"Tahoe"指的是由 Computer Console 公司生产的一种 Cpu。大家关心的是这个版本对网络的支持情况。

3. Microsoft Xenix System V

Microsoft Xenix System V 类似于 AT&T Unix System V Release2.0。Xenix 系统适用于不同的硬件环境，不过它一般在 8086、80286 以及 80386 上运行。

Unix 标准化

当今，有许多单位致力于 Unix 标准化的工作。其中 IEEE 的 1003.1-1988 标准定义了一个基于 Unix 的标准操作系统接口和环境。这个标准是第一个标准，我们称它为 POSIX。此标准致力于源代码级上的应用程序可移植性。此标准主要定义 C 语言与操作系统的接口。

Xopen 是 1984 年由国际计算机销售商组织制定的。他们出版了 37 卷的指导可移植性的手册。此手册提供了内核接口及其它许多 Unix 实用程序。手册的第七卷描述了 Xopen 的传送层接口(XTI)。XTI 类似于 System V Release 3 的传送层接口(TLI)。

内核

内核就是操作系统。内核包括文件系统、存储管理、CPU 调度以及设备 I／O。内核一般直接同底层硬件打交道，但也有一些 Unix 内核通过别的操作系统控制硬件。

程序

程序为一可执行文件。它一般由链接器创建并且存放在磁盘上。Unix 系统中通过 exec 系统调用加载程序执行。

进程

进程是正在运行的某个程序的实例。Unix 系统中进程只能通过 fork 调用创建。一个多任务操作系统能够同时运行多个进程。比如同时有十个用户运行相同的文本编辑程序。进程由用户上下文、正文段、数据段、堆、栈和内核上下文组成。

· 进程的用户上下文包含了用户模式下进程可访问的空间。

· 进程的正文段包含了真正可执行的机器指令。许多操作系统为了避免进程修改自

己的指令，把正文段设为只读。这样一个程序的许多实例可以共享一个正文段。当要执行一个程序时，只有当程序的副本没有在运行或操作系统不支持正文共享时，正文段才从磁盘读入内存。

· 数据段包含程序数据。此段可分为以下三个部分

(1) 初始化的只读数据。它是由程序来初始化的数据元素，进程运行时只读。这区域用于需要初始化却不能更改的一些数据项目比如文字串。支持只读数据区的操作系统不多。

(2) 初始化读写数据。这样的数据由程序初始化并且在进程运行时可以修改。

(3) 非初始化数据。这样的数据在进程执行前不用初始化并且统统置零。进程运行时这些数据可以被修改。Unix 系统调用不把数据段 bss 初始化。这样做的好处在于系统无需在磁盘上为此区域分配空间，因为操作系统在程序执行之前便把此区域初始化为零。除了能节省磁盘空间外，也能减少把程序读入内存的时间。

· 堆用来把数据空间动态地分配给进程。

· 包含栈的进程在运行时，栈动态变化。这些栈包含了函数调用的返回地址及函数所需要的其它数据元素。

进程运行时，为了让堆和栈动态增长，堆和栈之间空出了一些保留空间。

下面用一个程序来说明问题。

```
int debug=1;
char *program;
main(argc, argv)
int argc;
char *argv[ ];
{
    int i;
    char *ptr, *malloc ( );
    programe=argv[0];
    printf("argc=%d\n", arge);
    for (i=1; i<argc; i++) {
        ptr=malloc(strlen(argv[i])+1);
        strcpy(ptr, argv[i]);
        if(debug)
            printf("%s\n", ptr);
        }
}
```

程序中，串"argc=%d\n"和"%s\n"是只读数据。整型变量 debug 为初始化读写函量。字符指针 progname 为非初始化读写变量。我们把变量 i 和 ptr 称为自动变量。main 函数执行时，它们被放在栈上。最后 malloc 函数在堆上分配空间(标准 C 库中的 malloc 函数动态地分配空间，它的参数为所需分配的字节数，返回一个指向分配区域的指针)。包含函数 main，printf，stvcpy 和 malloc 的机器指令皆在正文段。

· 进程的内核上下文只能由内核来维护和访问。此区域包含内核跟踪进程运行以及挂起和重新唤醒进程所需的信息。该区域主要包含与进程有关的机器寄存器值，进程每部

分的物理位置和大小。进程运行时不能访问该区域。进程的内核上下文存放在进程表(process table)中。这个表是一个内核数据结构。每一个进程在这个表中都有一个入口点。

系统调用

Unix 提供了 60 至 200 个能直接获得内核服务的入口。把这些入口称为系统调用。进程与内核之间传送参数和结果的方式以及激活系统调用的机器指令随机器的不同而不同。比如，在 VAX 机上机器执行 chmk 指令激活系统调用。指令的低 16 位用来区分系统调用，而通用寄存器及条件码用来传递参数和结果。不管怎样，C 程序员用不着知道上述细节。标准的 C 库提供了 C 与每个系统调用的接口。这样对程序员来说系统调用与一般的函数一样。

由于 4.3BSD 把一些 System V 的系统调用当作函数调用。因而系统调用和一般的函数调用的界限是模糊不清的。比如，System V time 调用为一系统调用，而在 4.3BSD 下被当作 gettimeofday 函数。

许多系统调用返回−1 作为错误码。返回大于或等于零的数表示正确返回。Unix 系统调用的 C 语言接口提供了一全局整型变量 errno，若系统调用出错，则 errno 表示错误号。文件<errno.h>包含了这些错误号的名称和错误情况。只有发生错误后才检查 errno 的值，成功的系统调用并不把 errno 的值复位为零。

有些 C 库的系统调用和函数返回一个结构来表明是否成功。

比如系统调用 stat 和 fstat 返回一指向结构的指针。

C 起始函数

一个 C 程序一般是从 main 函数开始执行。如下述一完整的 C 程序所示。

```
main ()
{
  printf ( "hello world\n");
}
```

许多 C 编译安排了一个特殊的起始函数。起始函数首先处理完各种初始化工作，然后调用函数 main。当 main 函数返回或退出时，C 起始函数提供的 exit 功能被激活。

一个函数或者通过运行到程序的最后隐式地返回(如上面那个例子一样)或者通过执行 return 语句显式地返回。exit 函数一般要清除标准 I／O 缓冲区，然后通过调用_exit 终止进程。

可以在上面那个例子程序的后面加上 return(0)或 exit(0)。否则的话程序的出口状态不可预测。

参数表

任何执行的程序都要接收参数表。参数表是一个指向字符串的指针数组。参数表的长度不应超过 5120 或 10240 个字节。

一个程序的典型执行方式是把命令行键入终端。例如，当在 Unix shell 下键入

```
echo hello world
```

程序 echo 接收到 echo，hello 和 world 等三个参数后开始执行。

C 起始函数把参数表传给 C 程序的 main 函数。main 函数可为如下形式:

```
main (argc, argv)
int argc;
char * argv [ ];
  {
}
```

第一个参数表示参数表字符串的个数，此值大于等于 1。第二个参数为指向字符串的指针数组。第二个参数也可定义为

```
char * * argv;
```

数组 argv 的每一个指针都指向参数字符。第一个参数确定数组的入口数。大多数 unix 系统在数组 argv[argc]中包含空指针。

参数表放在进程的数据空间中。进程可以随意地使用这些参数。当一个程序被激活时，Unix 通常把参数放在初始栈中。需要的话 main 函数可以缺省此参数。打印"hello wurld"的程序便是这样。

环境表

程序运行时也接收环境变量表，象接收参数表一样。环境变量也传给 main 函数。由于环境变量为指针数组其元数个数未知，因而数组的最后必是一空指针。环境表可以作为程序的第三个参数来访问如下所示。

```
main (argc, argv, envp)
int argc;
char * argv [ ];
char * ehvp [ ];
{
}
```

环境字符串的形式通常为

```
variable = string
```

下列 C 程序打印出所有的环境字符串

```
main (argc, argv, envp)
int argc;
char  * argv [ ];
char  * envp [ ];
{
    int i;
    for (i=0; envp[i] ! =(char * )0; i++)
        printf("%s\n", envp [i]);
    exit (0);
}
```

输出结果如下:

```
HOME = / usrl / stevens
SHELL = / bin / csh
TERM = att630
USER = stevens
PATH = / usrl / stevens / bin:  / usr / local / bin:  / usr / ucb:  / bin:  / usr / bin
```

```
EXINIT=set optimize redraw shell=/bin/csh
```

C 起始函数提供了外部变量 environ。它也能用来访问环境表。此变量的定义如下:

```
extern char * * environ;
```

与上述程序等价的另一个程序如下:

```
main(argc, argv)
int argc;
char * argv [ ];
{
    int   i;
    extern char * * environ;
    for (i=0; environ[i]! =(char * )0; i++)
        printf("%s\n", environ[i]);
    exit(0);
}
```

P[105]由于有了这个变量，不只是 main 函数可以访问环境表，因而用不着把它的值传来传去。用 C 库中的 getenv 函数，程序便可获得环境表中的某一特定变量。下面程序打印出环境变量 HOME 的值。

```
main( )
{
    char * ptr, * getenv( );
    if ((ptr=tetnv("HOME"))= =(char * )0)
            printf ("HOME is not defined\n"__;
    else
            printf("HOEM=%s\n", ptr);
    exit(0);
}
```

若环境表中定义了这个变量，getenv 便返回指向此变量的指针，否则返回空指针。象参数表一样，环境数组指针也放在进程的数据区。需要的话进程可以修改它，但修改环境对父进程无影响。当修改过环境的进程终止时，此进程便把它占有的内存空间释放。终止进程只把它的 exit 函数的 8 位参数传给父进程。不管怎样，一个修改环境的进程将会影响任何由它创建的子进程。

进程号(PID)

每个进程都有一个唯一的进程号(PID)。PID 是一个从 0 到 3000 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。进程可以通过系统调用 getpid 获得它的进程号。此系统调用的形式如下:

```
int getpid( );
```

进程号为 1 的进程是一被称作 init 的特殊进程。进程号为 0 的进程也是一特殊的内核进程，它或者是对换进程(Swapper)或者是调度进程(Schednler)。当用到 unix 的虚拟存贮管理时，进程号为 2 的进程是一被称作页守护(page daemon)的内核进程。除了这些特殊的进程外，其它进程皆无特殊的含义。

父进程号