# 一、Linux极速入门必备命令

## Shell 简介

Shell 是运维和系统管理员操作Linux 系统的首选，简单说，它是一个命令解释器.

Shell

命令行

输入命令 解释执行

命令行相关： 行首“$” 或 “#” － 命令行提示符

行中 ## － 视为注释开始

**注意：** 命令行是区分大小写的！

使用命令行补全（Tab） 和通配符可以提高输入效率

通配符共有3个： “\*”，“?”，“[]”

\* － 用于匹配文件名中任意长度的字符串；

? － 只匹配一个字符；

[] － 用于匹配所有出现在方括号内的字符。可以使用短线“-”来指定字符集范围。如：***ls text[1-3]***  或 ***ls test[a-z]***

Linux 下环境下有几种不同Shell ,常用的有 BASH 、TCSH Shell和 Z-Shell 等.BASH 是默认安装和使用的Shell。

## 入门必备命令

### 2.1寻求帮助 - man 命令

作用： Linux 为所有命令和系统调用编写了帮助手册。使用man 命令可以方便地获取某个命令的帮助信息。

用法： man [手册编号] 命令名

man 命令在显示手册页时实际调用less 完成显示，J K可以上下翻动，空格用于向下翻页。Q键退出。

手册总共分为9 节，各部分内容如下：

|  |  |
| --- | --- |
| 目 录 | 内 容 |
| /usr/share/man/man1 | 普通命令和应用程序 |
| /usr/share/man/man2 | 系统调用 |
| /usr/share/man/man3 | 库调用，主要是libc()函数的使用文档 |
| /usr/share/man/man4 | 设备驱动和网络协议 |
| /usr/share/man/man5 | 文件的详细格式信息 |
| /usr/share/man/man6 | 游戏 |
| /usr/share/man/man7 | 文档使用说明 |
| /usr/share/man/man8 | 系统管理命令 |
| /usr/share/man/man9 | 内核源代码或模块的技术指标 |

### 2.2用户间切换SU

作用： 切换到其他用户

用法： su [用户名]

**$ su martin #切换到 martin 这个用户**

**$ su #切换到 特权用户 root**

**注意：** Ubuntu 默认情况下，系统没有合法的root权限，不能直接使用su 命令提升到 root 权限，而必须要用 sudo 来获得 root 权限

### 2.3特权命令 sudo

作用： 提升当前执行命令的权限，以root 身份执行它.

用法： sudo 命令行

**$ sudo su #切换到root 用户**

**$ sudo rm root.txt #切换到 特权用户 root**

## 文件操作常用命令

### 3.1显示当前目录 pwd 和改变目录 cd

**pwd 命令**

作用： 显示当前目录,即工作目录

用法： pwd

**cd命令**

作用： 改变目录位置

用法： cd **. . .[OPTION] . . .[FILE]. . .**

cd 目录路径 － 进入指定的目录中去

cd .. － 返回父目录

cd / － 进入根目录

cd 或 cd ~ － 进入用户主目录

cd ./\* - 进入当前目录下\*表示的子目录

### 3.2 ls 命令

用法： **ls . . .[OPTION] . . .[FILE]. . .**

常用参数：

1. 不带任何参数 列出当前目录下的所有文件和子目录
2. -F 分类显示，方便阅读
3. -a 显示隐含文件
4. -l 查看文件的各种属性

### 3.3列出目录内容： dir 和 vdir

用法： dir **. . .[OPTION] . . .[FILE]. . .**

vidr **. . .[OPTION] . . .[FILE]. . .**

dir 和 ls 差不多，就比ls 功能少

vdir 相当于 ls -l 命令

### 3.4建立目录mkdir

用法： mkdir **. . .[OPTION] . . .[FILE]. . .**

mkdir 一次可以建立一个或 几个目录

常用参数：

不带任何参数 创建相应目录，如果目录的父级目录路径不存在，则创建失败;

-p 创建相应目录，如果目录的父级目录路径不存在，则一起创建;

### 3.5移动、复制和删除

**移动命令 mv**

用法： mv **. . .[OPTION] 源文件 目标文件**

常用参数：

不带任何参数 将源文件移动到目标文件，注意：如果目标文件存在则**替换**;

-i 将源文件移动到目标文件，如果目标文件存在则提示是否**替换**;

-b 将源文件移动到目标文件，如果目标文件存在则不进行覆盖，而是在目标文件后加~

**复制命令 cp**

用法： cp **. . .[OPTION] 源文件 目标文件**

常用参数：

不带任何参数 将源文件复制到目标文件，注意：如果目标文件存在则**替换**;

-i 将源文件复制到目标文件，如果目标文件存在则提示是否**替换**;

-b 将源文件复制到目标文件，如果目标文件存在则不进行覆盖，而是在目标文件后加~

-r 将子目录及其中的文件一起复制到另一个子目录下

**删除命令 rm**

删除命令可以一次永久性删除一个或几个文件(包含目录)

用法： rm **. . .[OPTION]. . . [FILE]. . .**

常用参数：

不带任何参数 删除文件或相应目录，不给予任何提示;

-i 删除文件或相应目录，删除时进行提醒;

-f 强制性删除文件或相应目录;

-r 将子目录及其中的文件一并删除。**（慎用！特别时在root 权限下）**

### 3.6文件链接ln

ln 建立文件链接

用法： ln **. . .[OPTION] 源文件 目标文件**

常用参数：

不带任何参数 创建硬链接，ls -i 查看可以看到两个文件的inode 值一致;

-s 创建软链接,即别名,如果源文件删除，则软链接(别名)也无

法访问。

### 3.7改变文件所有权 chown 和 chgrp

**chown 命令用于改变文件的所有权。**

用法： chown **. . .[OPTION] [OWNER][:[GROUP]] FILE . . .**

常用参数：

不带任何参数 改变单个或多个文件的属主和属组;

-r 改变一个目录及其下所有文件（和子目录）的所有权设置。

**chgrd 用于单独设置文件的属组。**

用法： chown **. . .[GROUP] FILE . . .**

**$ chgrp nogroup text.txt**

### 3.8改变文件权限 chmod

chmod 用于改变一个文件的权限。它以“用户组 +/- 权限”的表达方式来增加/删除相应的权限。具体来说，用户组包括了文件属主(u)、文件属组(g)、其他人(o) 和所有人(a)，而权限则包括读取(r、w、x)

用法： chmod **. . .[OPTION] . . .[FILE]. . .**

**例：**

**$chmod u+x test.txt**

**$chmod u-x test.txt**

**$chmod ug=wr、o=r text.txt**

### 3.9阅读文件的头部和尾部: head 和 tail

用法： head/tail **. . .[OPTION]. . . FILE**

常用参数：

不带任何参数 显示文件的头部/尾部 10 行;

-n 按指定的行数显示文件的头部/尾部 ;

### 3.10查看文本文件: cat 、more 和 less

cat命令用来一次性查看全部文本文件的内容，后跟文件名作为参数.也可以带上 -n 显示每行的行号。如果文件长达几十上百页，不建议使用 cat.

more 命令用来分页查看文本文件。空格翻页；回车向下滚动一行;Q键退出。

less更人性化的文本阅读工具。

### 3.11编辑器: vim

vim 是UNIX 和 Linux 上标配的编辑器，功能十分强大。

用法 ：

vim file

或 vim

1. 编辑保存文件

分插入和命令两种模式。

**插入模式**

操作：输入字符，并可以按光标键移动输入字符位置.

|  |  |
| --- | --- |
| 命令 | 操作 |
| a | 在光标后插入 |
| i | 在光标所在位置插入 |
| o | 在光标所在位置的下一行插入 |
| Esc | 进入命令模式 |
| : | 进入行命令模式 |

模式切换

**命令模式**

执行除输入字符之外的所有操作，包括保存、搜索、移动光标等。

vim 启动默认是在命令模式，如果在编辑模式，需要按 Esc 切换回命令模式.

可以使用光标键控制上下左右移动，或者 h、j、k、l移动。

**保存和退出**

保存文件、退出等需要切换到行命令模式，在插入模式下输入 Esc 再 输入

:w :q 或 :wq 等。

前提：须切换到命令模式。注意组合命令执行的顺序：w ->q ->!

|  |  |
| --- | --- |
| 命令 | 操作 |
| :w | 保存文件 |
| :w filename | 另存为filename |
| :q | 退出Vim |
| :q! | 强行退出，放弃保存 |

**搜索字符串**

前提：须切换到命令模式

/string 用于向下搜索一个字符串

?string 用于向上搜索一个字符串

如果需要启动或关闭大小写敏感,执行：

:set ignorecase

或

:set noignorecase

**替换字符串**

前提：须切换到命令模式。

语法 : :[range]s/pattern/string/[c、e、g、i]

这条命令将 pattern 所代表的字符串替换成 string。开头的range 用于指定替换作用的范围，如“1,10” 表示从第1行到第10行，“1,$”表示从第1行到最后一行，也就是全文。全文也可以用 “%” 表示。

最后的方括号是可选选项，含义如下:

|  |  |
| --- | --- |
| 标 志 | 含 义 |
| c | 每次替换前询问 |
| e | 不显示错误信息 |
| g | 替换一行中的所有匹配项(这个选项通常需要使用) |
| i | 不区分大小写 |

如Windows 环境下的源码经常会有 “^M”的字符，要清除可以使用下面的命令：

:%s/^M$//g

**删除、复制、粘贴**

前提：须切换到命令模式。

|  |  |
| --- | --- |
| 命令 | 操作 |
| x | 删除光标所在位置的字符 |
| dd | 删除光标所在的行， 2 dd 表示删除2行 |
| D | 删除光标所在位置到行尾之间所有的字符 |
| d | 普遍意义上的删除命令，和移动命令配合使用。例如dw 表示删除光标所在位置到下一单词词头之间所有的字符 |
| yy | 复制光标所在的行 |
| y | 普遍意义上的复制命令，和移动命令配合使用。例如yw表示复制光标所在位置到下一个单词词头之间所有的字符 |
| p | 在光标所在位置粘贴最近复制/删除的内容 |

**撤销和重做**

前提：须切换到命令模式。

|  |  |
| --- | --- |
| 命令 | 操作 |
| u | 撤销一次操作 |
| Ctrl+R | 重做被撤销的操作 |

**程序员特有配置**

:syntax on ## 语法高亮

:set autoindent ## 自动缩进

:set shiftwidth=4 ## 设置Tab 键对应的空格数

## 查找和定位常用命令

### 4.1迅速查找文件 - find 命令

作用： 在指定范围内迅速查找到文件。

用法： find [OPTION] [path . . .] [expression]

例：

$find /usr -name test.txt

【OPTION选项】

特殊情况才会使用，建议日后查帮助

常用参数：

-type 查找时指定文件的类型，可使用参数如下表;

-atime n 查找最后一次使用在n天前的文件,n 使用负数表示;

-mtime n 查找最后一次修改在n天前的文件;

-maxdepth n         查找目录的最大深度

-mindepth n         从指定的目录的第几层深度开始查找

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 含义 | 参数 | 含义 |
| b | 块设备文件 | f | 普通文件 |
| c | 字符设备文件 | p | 命名管道 |
| d | 目录文件 | l | 符号链接 |

### 4.2更快速的定位文件 - locate 命令

作用： 火箭般的速度定位文件。

用法： locate [expression] pattern

locate 并不进入子目录进行搜索，它通过检索数据库来确定文件的位置。可以使用 updatedb 来更新检索数据库。

### 4.3查找文件内容 - grep 命令

作用： 在文件中寻找某些信息。

用法： grep [OPTIONS] PATTERN [FILE...] ## pattern 使用基础正则表达式

$grep open ./test.c

# 二、Linux 服务器开发

## C和C++ 编译器: gcc

GNU C Compiler 的缩写，经过十来年发展，意义变成了 GNU Compiler Collection，可同时支持 C、C++、Objective C和Java 等.

### 1.1 编译第一个C/C++程序

只编译执行一个C程序

$ gcc hello.c

$ ./a.out

$Hello world!

默认的a.out 并不友好，gcc 提供 -o 选项指定执行文件的文件名：

$gcc -o hello hello.c ##编译源代码，并把可执行文件命名为 hello

$Hello world!

编译C++程序，我们可以直接用GCC 编译其中的g ++命令，用法同 gcc；当然g++ 和 gcc 都可以用来编译 c 和 c++程序。gcc 编译c++程序需要带上 -lstdc++ 指定使用c++库。

注:安装g++时如果很慢，建议：

使用vim 编辑 /etc/apt/sources.list 文件，在文件的尾部换行加入如下内容：

deb http://mirrors.aliyun.com/ubuntu/ bionic main restricted universe multiverse  
deb http://mirrors.aliyun.com/ubuntu/ bionic-security main restricted universe multiverse  
deb http://mirrors.aliyun.com/ubuntu/ bionic-updates main restricted universe multiverse  
deb http://mirrors.aliyun.com/ubuntu/ bionic-proposed main restricted universe multiverse  
deb http://mirrors.aliyun.com/ubuntu/ bionic-backports main restricted universe multiverse  
deb-src http://mirrors.aliyun.com/ubuntu/ bionic main restricted universe multiverse  
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-security main restricted universe multiverse  
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-updates main restricted universe multiverse  
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-proposed main restricted universe multiverse  
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-backports main restricted universe multiverse

接下来执行: apt-get update 命令，再调用 sudo apt-get install g++

### 1.2 编译常用选项

|  |  |
| --- | --- |
| 选 项 | 功 能 |
| -c | 只激活预处理、编译和汇编,生成.o 目标代码文件 |
| -S | 只激活预处理和编译，生成扩展名为.s的汇编代码文件 |
| -E | 只激活预处理，并将结果输出至标准输出 |
| -g | 为调试程序(如gdb)生成相关信息 |
| -O | 等同-O1,常用的编译优化选项 |
| -Wall | 打开一些很有用的警告选项，建议编译时加此选项。 |

注意：-c 选项在编写大型程序是必须的，多个文件的源代码首先需要编译成目标代码，再链接成执行文件。如果由多个源文件，工程做法建议采用 makefile 。

### 1.3 在Windows让编程效率飞起来

**1.3.1 Ubuntu安装Samba 服务器**

确认安装: dpkg -l | grep samba

安装： sudo apt-get install samba samba-common

卸载： sudo apt-get autoremove samba

**1.3.2 Samba服务器配置**

sudo vi /etc/samba/smb.conf

在文件最后添加

[Share]

comment=This is samba dir

path=/home/martin/

writable=yes

browseable=yes

增加samba 用户

#sudo smbpasswd -a martin

**1.3.3启动和关闭**

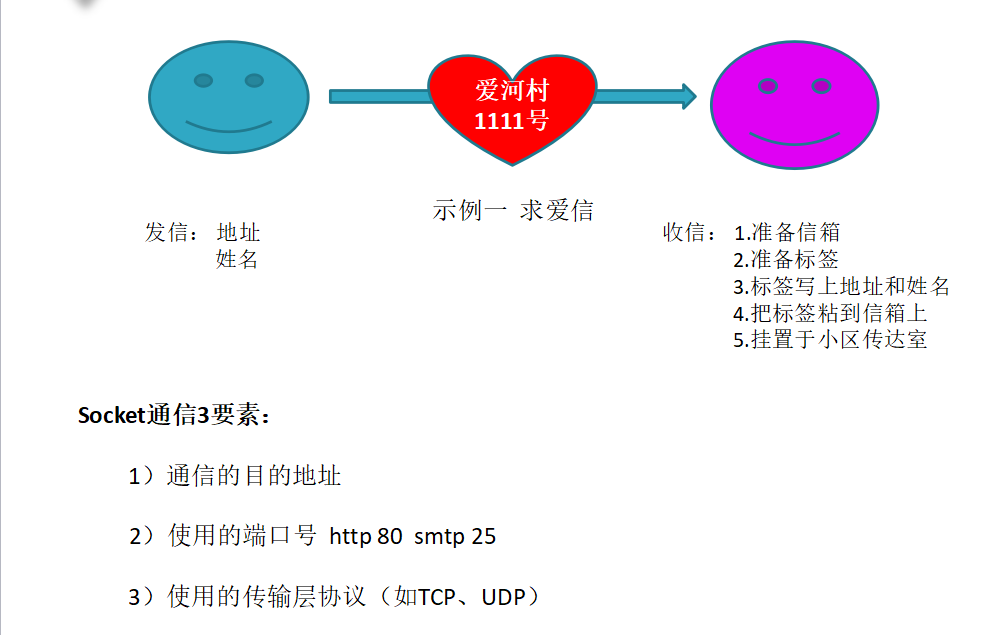
启动Samba服务器: sudo service smbd start  
 关闭Samba服务器： sudo service smbd stop

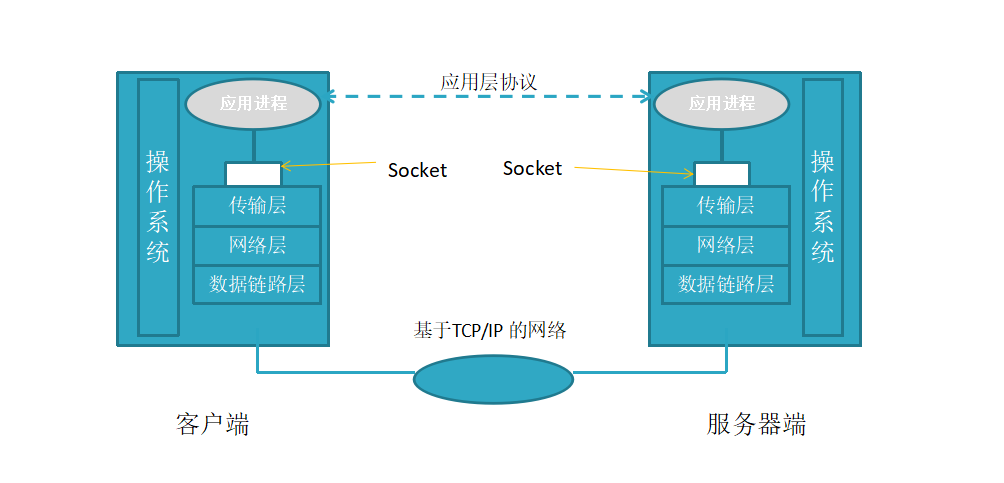
## 网络通信和socket编程

### 2.1项目需求

实现服务器的客户端/服务器程序，客户端通过网络连接到服务器，并发送任意一串英文信息，服务器端接收信息后，将每个字符转换为大写并回送给客户端显示。

### 2.2 Socket通信模型





### 2.3 Socket 编程详解

#### 2.3.1套接字概念

Socket中文意思是“插座”，在Linux环境下，用于表示进程x间网络通信的特殊文件类型。本质为内核借助缓冲区形成的伪文件。

既然是文件，那么理所当然的，我们可以使用文件描述符引用套接字。Linux系统将其封装成文件的目的是为了统一接口，使得读写套接字和读写文件的操作一致。区别是文件主要应用于本地持久化数据的读写，而套接字多应用于网络进程间数据的传递。

在TCP/IP协议中，“IP地址+TCP或UDP端口号”唯一标识网络通讯中的一个进程。“IP地址+端口号”就对应一个socket。欲建立连接的两个进程各自有一个socket来标识，那么这两个socket组成的socket pair就唯一标识一个连接。因此可以用Socket来描述网络连接的一对一关系。

套接字通信原理如下图所示：



套接字通讯原理示意

在网络通信中，套接字一定是成对出现的。一端的发送缓冲区对应对端的接收缓冲区。我们使用同一个文件描述符索发送缓冲区和接收缓冲区。

Socket 通信流程图



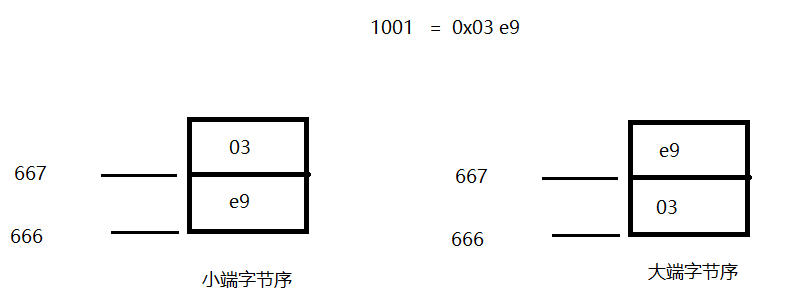
#### 2.3.2 Socket编程基础

##### 2.3.2.1 网络字节序

在计算机世界里，有两种字节序：

大端字节序 - 低地址高字节,高地址低字节

小段字节序 - 低地址低字节,高地址高字节



内存中的多字节数据相对于内存地址有大端和小端之分，磁盘文件中的多字节数据相对于文件中的偏移地址也有大端小端之分。网络数据流同样有大端小端之分，那么如何定义网络数据流的地址呢？发送主机通常将发送缓冲区中的数据按内存地址从低到高的顺序发出，接收主机把从网络上接到的字节依次保存在接收缓冲区中，也是按内存地址从低到高的顺序保存，因此，网络数据流的地址应这样规定：先发出的数据是低地址，后发出的数据是高地址。

TCP/IP协议规定，网络数据流应采用大端字节序，即低地址高字节。

例如端口号是1001（0x3e9），由两个字节保存，采用大端字节序，则低地址是0x03，高地址是0xe9，也就是先发0x03，再发0xe9，这16位在发送主机的缓冲区中也应该是低地址存0x03，高地址存0xe9。但是，如果发送主机是小端字节序的，这16位被解释成0xe903，而不是1001。因此，发送主机把1001填到发送缓冲区之前需要做字节序的转换。同样地，接收主机如果是小端字节序的，接到16位的源端口号也要做字节序的转换。如果主机是大端字节序的，发送和接收都不需要做转换。同理，32位的IP地址也要考虑网络字节序和主机字节序的问题。

为使网络程序具有可移植性，使同样的C代码在大端和小端计算机上编译后都能正常运行，可以调用以下库函数做网络字节序和主机字节序的转换。

#include <arpa/inet.h>

uint32\_t htonl(uint32\_t hostlong);

uint16\_t htons(uint16\_t hostshort);

uint32\_t ntohl(uint32\_t netlong);

uint16\_t ntohs(uint16\_t netshort);

h表示host，n表示network，l表示32位长整数，s表示16位短整数。

如果主机是小端字节序，这些函数将参数做相应的大小端转换然后返回，如果主机是大端字节序，这些函数不做转换，将参数原封不动地返回。

##### 2.3.2.2 sockaddr数据结构

很多网络编程函数诞生早于IPv4协议，那时候都使用的是sockaddr结构体,为了向前兼容，现在sockaddr退化成了（void \*）的作用，传递一个地址给函数，至于这个函数是sockaddr\_in还是其他的，由地址族确定，然后函数内部再强制类型转化为所需的地址类型。



sockaddr数据结构

struct sockaddr {

sa\_family\_t sa\_family; /\* address family, AF\_xxx \*/

char sa\_data[14]; /\* 14 bytes of protocol address \*/

};

struct sockaddr\_in {

sa\_family\_t sin\_family; /\* address family: AF\_INET \*/

in\_port\_t sin\_port; /\* port in network byte order \*/

struct in\_addr sin\_addr; /\* internet address \*/

};

/\* Internet address. \*/

struct in\_addr {

uint32\_t s\_addr; /\* address in network byte order \*/

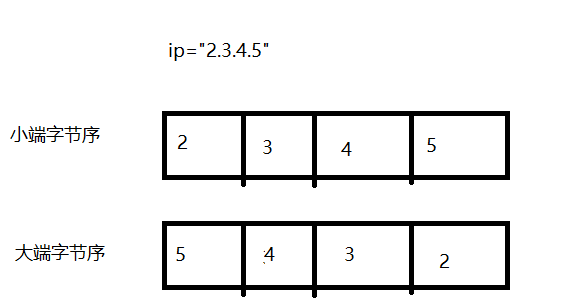
};

IPv4的地址格式定义在netinet/in.h中，IPv4地址用sockaddr\_in结构体表示，包括16位端口号和32位IP地址，但是sock API的实现早于ANSI C标准化，那时还没有void \*类型，因此这些像bind 、accept函数的参数都用struct sockaddr \*类型表示，在传递参数之前要强制类型转换一下，例如：

struct sockaddr\_in servaddr;

bind(listen\_fd, (struct sockaddr \*)&servaddr, sizeof(servaddr)); /\* initialize servaddr \*/

##### 2.3.2.3 IP地址转换函数



#include <arpa/inet.h>

int inet\_pton(int af, const char \*src, void \*dst);

const char \*inet\_ntop(int af, const void \*src, char \*dst, socklen\_t size);

af 取值可选为 AF\_INET 和 AF\_INET6 ，即和 ipv4 和ipv6对应支持IPv4和IPv6。其中inet\_pton和inet\_ntop不仅可以转换IPv4的in\_addr，还可以转换IPv6的in6\_addr。因此函数接口是void \*addrptr。

范例：

|  |
| --- |
| #include <stdio.h>  #include <string.h>  #include <arpa/inet.h>  int main(void){  char ip[]="2.3.4.5";  char server\_ip[64];  struct sockaddr\_in server\_addr;  inet\_pton(AF\_INET, ip, &server\_addr.sin\_addr.s\_addr);  printf("s\_addr : %x\n", server\_addr.sin\_addr.s\_addr);  printf("s\_addr from net to host: %x\n", ntohl(server\_addr.sin\_addr.s\_addr));  inet\_ntop(AF\_INET, &server\_addr.sin\_addr.s\_addr, server\_ip, 64);  printf("server ip : %s\n", server\_ip);  printf("INADDR\_ANY: %d\n", INADDR\_ANY);  server\_addr.sin\_addr.s\_addr = INADDR\_ANY;  inet\_ntop(AF\_INET, &server\_addr.sin\_addr.s\_addr, server\_ip, 64);  printf("INADDR\_ANY ip : %s\n", server\_ip);  return 0;  } |

#### 2.3.3 Socket编程函数

##### 2.3.3.1 socket 函数

#include <sys/types.h> /\* See NOTES \*/

#include <sys/socket.h>

int socket(int domain, int type, int protocol);

domain:

AF\_INET 这是大多数用来产生socket的协议，使用TCP或UDP来传输，用IPv4的地址

AF\_INET6 与上面类似，不过是来用IPv6的地址

AF\_UNIX 本地协议，使用在Unix和Linux系统上，一般都是当客户端和服务器在同一台及其上的时候使用

type:

SOCK\_STREAM 这个协议是按照顺序的、可靠的、数据完整的基于字节流的连接。这是一个使用最多的socket类型，这个socket是使用TCP来进行传输。

SOCK\_DGRAM 这个协议是无连接的、固定长度的传输调用。该协议是不可靠的，使用UDP来进行它的连接。

SOCK\_SEQPACKET该协议是双线路的、可靠的连接，发送固定长度的数据包进行传输。必须把这个包完整的接受才能进行读取。

SOCK\_RAW socket类型提供单一的网络访问，这个socket类型使用ICMP公共协议。（ping、traceroute使用该协议）

SOCK\_RDM 这个类型是很少使用的，在大部分的操作系统上没有实现，它是提供给数据链路层使用，不保证数据包的顺序

protocol:

传0 表示使用默认协议。

返回值：

成功：返回指向新创建的socket的文件描述符，失败：返回-1，设置errno

socket()打开一个网络通讯端口，如果成功的话，就像open()一样返回一个文件描述符，应用程序可以像读写文件一样用read/write在网络上收发数据，如果socket()调用出错则返回-1。对于IPv4，domain参数指定为AF\_INET。对于TCP协议，type参数指定为SOCK\_STREAM，表示面向流的传输协议。如果是UDP协议，则type参数指定为SOCK\_DGRAM，表示面向数据报的传输协议。protocol参数的介绍从略，指定为0即可。

##### 2.3.3.2 bind 函数

#include <sys/types.h> /\* See NOTES \*/

#include <sys/socket.h>

int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

sockfd：

socket文件描述符

addr:

构造出IP地址加端口号

addrlen:

sizeof(addr)长度

返回值：

成功返回0，失败返回-1, 设置errno

服务器程序所监听的网络地址和端口号通常是固定不变的，客户端程序得知服务器程序的地址和端口号后就可以向服务器发起连接，因此服务器需要调用bind绑定一个固定的网络地址和端口号。

bind()的作用是将参数sockfd和addr绑定在一起，使sockfd这个用于网络通讯的文件描述符监听addr所描述的地址和端口号。前面讲过，struct sockaddr \*是一个通用指针类型，addr参数实际上可以接受多种协议的sockaddr结构体，而它们的长度各不相同，所以需要第三个参数addrlen指定结构体的长度。如：

struct sockaddr\_in servaddr;

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

servaddr.sin\_port = htons(6666);

首先将整个结构体清零，然后设置地址类型为AF\_INET，网络地址为INADDR\_ANY，这个宏表示本地的任意IP地址，因为服务器可能有多个网卡，每个网卡也可能绑定多个IP地址，这样设置可以在所有的IP地址上监听，直到与某个客户端建立了连接时才确定下来到底用哪个IP地址，端口号为6666。

##### 2.3.3.3 listen 函数

#include <sys/types.h> /\* See NOTES \*/

#include <sys/socket.h>

int listen(int sockfd, int backlog);

sockfd:

socket文件描述符

backlog:

在Linux 系统中，它是指排队等待建立3次握手队列长度

查看系统默认backlog

cat /proc/sys/net/ipv4/tcp\_max\_syn\_backlog

改变 系统限制的backlog 大小

vim /etc/sysctl.conf  
  
最后添加  
net.core.somaxconn = 1024

net.ipv4.tcp\_max\_syn\_backlog = 1024

保存，然后执行  
sysctl -p

典型的服务器程序可以同时服务于多个客户端，当有客户端发起连接时，服务器调用的accept()返回并接受这个连接，如果有大量的客户端发起连接而服务器来不及处理，尚未accept的客户端就处于连接等待状态，listen()声明sockfd处于监听状态，并且最多允许有backlog个客户端处于连接待状态，如果接收到更多的连接请求就忽略。listen()成功返回0，失败返回-1。

##### 2.3.3.4 accept 函数

#include <sys/types.h> /\* See NOTES \*/

#include <sys/socket.h>

int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);

sockdf:

socket文件描述符

addr:

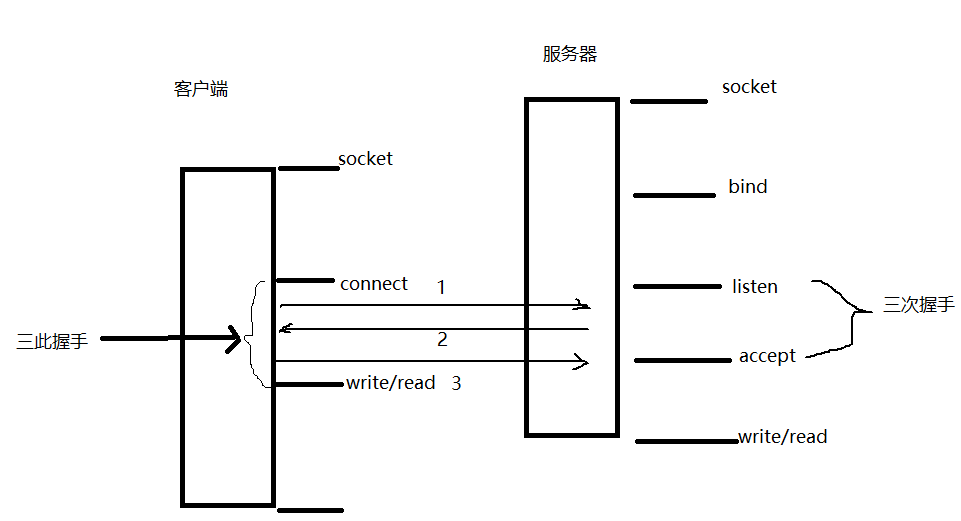
传出参数，返回链接客户端地址信息，含IP地址和端口号

addrlen:

传入传出参数（值-结果）,传入sizeof(addr)大小，函数返回时返回真正接收到地址结构体的大小

返回值：

成功返回一个新的socket文件描述符，用于和客户端通信，失败返回-1，设置errno



三次握手过程

三次握手完成后，服务器调用accept()接受连接，如果服务器调用accept()时还没有客户端的连接请求，就阻塞等待直到有客户端连接上来。addr是一个传出参数，accept()返回时传出客户端的地址和端口号。addrlen参数是一个传入传出参数（value-result argument），传入的是调用者提供的缓冲区addr的长度以避免缓冲区溢出问题，传出的是客户端地址结构体的实际长度（有可能没有占满调用者提供的缓冲区）。如果给addr参数传NULL，表示不关心客户端的地址。

我们的服务器程序结构是这样的：

while (1) {

cliaddr\_len = sizeof(cliaddr);

connfd = accept(listenfd, (struct sockaddr \*)&cliaddr, &cliaddr\_len);

n = read(connfd, buf, MAXLINE);

......

close(connfd);

}

整个是一个while死循环，每次循环处理一个客户端连接。由于cliaddr\_len是传入传出参数，每次调用accept()之前应该重新赋初值。accept()的参数listenfd是先前的监听文件描述符，而accept()的返回值是另外一个文件描述符connfd，之后与客户端之间就通过这个connfd通讯，最后关闭connfd断开连接，而不关闭listenfd，再次回到循环开头listenfd仍然用作accept的参数。accept()成功返回一个文件描述符，出错返回-1。

##### 2.3.3.5 connect函数

#include <sys/types.h> /\* See NOTES \*/

#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

sockdf:

socket文件描述符

addr:

传入参数，指定服务器端地址信息，含IP地址和端口号

addrlen:

传入参数,传入sizeof(addr)大小

返回值：

成功返回0，失败返回-1，设置errno

客户端需要调用connect()连接服务器，connect和bind的参数形式一致，区别在于bind的参数是自己的地址，而connect的参数是对方的地址。connect()成功返回0，出错返回-1。

##### 2.3.3.6 出错处理函数

我们知道，系统函数调用不能保证每次都成功，必须进行出错处理，这样一方面可以保证程序逻辑正常，另一方面可以迅速得到故障信息。

出错处理函数

#include <errno.h>

#include <string.h>

char \*strerror(int errnum); /\* See NOTES \*/

errnum:

传入参数,错误编号的值，一般取 errno 的值

返回值：

错误原因

#include <stdio.h>

#include <errno.h>

void perror(const char \*s); /\* See NOTES \*/

s:

传入参数,自定义的描述

返回值：

无

向标准出错stderr 输出出错原因

### 2.4 Socket 编程范例

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <string.h>

#include <ctype.h>

#include <arpa/inet.h>

#include <errno.h>

#define IP "1.1.1.1"

#define SERVER\_PORT 666

perror\_exit(const char \* des){

//fprintf(stderr, "%s error, reason: %s\n", des, strerror(errno));

perror(des);

exit(1);

}

int main(void){

int sock;//代表信箱

int i, ret;

struct sockaddr\_in server\_addr;

//1.美女创建信箱

sock = socket(AF\_INET, SOCK\_STREAM, 0);

if(sock == -1){

perror\_exit("create socket");

}

//2.清空标签，写上地址和端口号

bzero(&server\_addr, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;//选择协议族IPV4

inet\_pton(AF\_INET, IP, &server\_addr.sin\_addr.s\_addr);

//server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);//监听本地所有IP地址

server\_addr.sin\_port = htons(SERVER\_PORT);//绑定端口号

//实现标签贴到收信得信箱上

ret = bind(sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

if(ret == -1){

perror\_exit("bind");

}

//把信箱挂置到传达室，这样，就可以接收信件了

ret = listen(sock, 128);

if(ret == -1){

perror\_exit("listen");

}

//万事俱备，只等来信

printf("等待客户端的连接\n");

int done =1;

while(done){

struct sockaddr\_in client;

int client\_sock, len;

char client\_ip[64];

char buf[256];

socklen\_t client\_addr\_len;

client\_addr\_len = sizeof(client);

client\_sock = accept(sock, (struct sockaddr \*)&client, &client\_addr\_len);

//打印客服端IP地址和端口号

printf("client ip: %s\t port : %d\n",

inet\_ntop(AF\_INET, &client.sin\_addr.s\_addr,client\_ip,sizeof(client\_ip)),

ntohs(client.sin\_port));

/\*读取客户端发送的数据\*/

len = read(client\_sock, buf, sizeof(buf)-1);

buf[len] = '\0';

printf("recive[%d]: %s\n", len, buf);

//转换成大写

for(i=0; i<len; i++){

/\*if(buf[i]>='a' && buf[i]<='z'){

buf[i] = buf[i] - 32;

}\*/

buf[i] = toupper(buf[i]);

}

len = write(client\_sock, buf, len);

printf("write finished. len: %d\n", len);

close(client\_sock);

}

return 0;

}

### 2.5 项目拓展

将服务器端和客户端进行改造，实现客户端和服务器端完成一段简单的日常对话，以下C代表客户端，S代表服务器端:

C: Do you like C++?  
S: Yes, I do.  
C Why do you like C++?  
S: Because I can use it to write programs.  
S: And you ?

注： 话音未落，服务器发送完 “And you ?”后，调用close 函数故意把与客户端的连接关闭了。。。

接下来，客户端必须重连到服务器发送以下信息，然后双方再友好的关闭连接。  
C: Me too.