FTP实验报告

计23 章晔 2012011321

一、实验目的

在Linux系统上完成一个文件传输协议(FTP)的简单实现，加深对FTP原理和协议细节的理解，利用Socket接口设计实现简单的应用层协议，掌握TCP/IP网络应用程序的基本设计方法和实现技巧。

二、实验原理

(1)FTP

FTP是File Transfer Protocol的简称，即文件传输协议。该协议用于在两台计算机之间传送文件。

FTP会话包含两个通道，一个是空置通道，一个是数据通道。控制通道是和FTP服务器进行沟通的通道，连接服务器，发送指令。数据通道是和服务器进行文件传输的通道。

(2)通信

TCP/IP协议中，网络层的“IP地址”可以唯一标识网络中的主机，而传输层的“协议+端口”可以唯一标识主机中的应用程序（进程）。这样利用三元组（ip地址，协议，端口）就可以标识网络的进程了，网络中的进程通信就可以利用这个标志与其它进程进行交互。

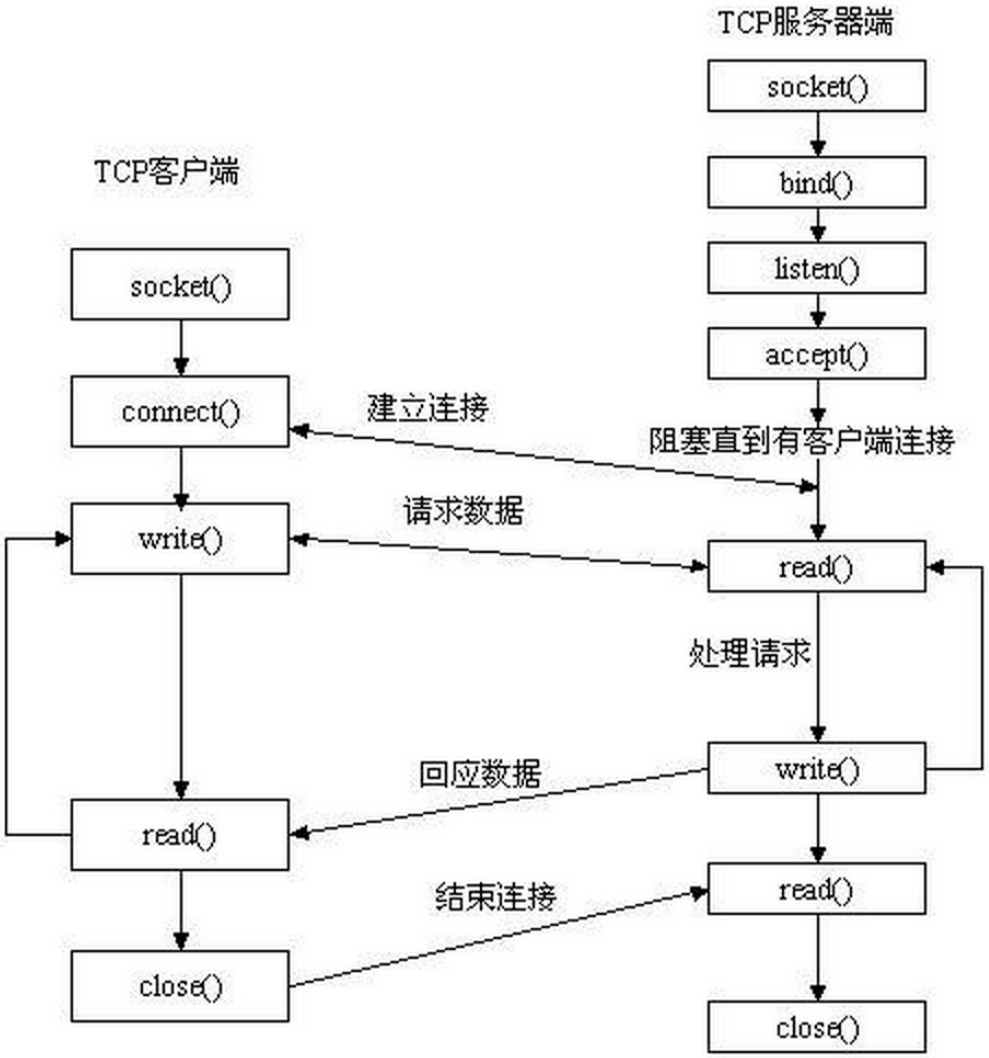
(3)Socket

socket起源于Unix，而Unix/Linux基本哲学之一就是“一切皆文件”，都可以用“打开open–>读写write/read–>关闭close”模式来操作。Socket就是该模式的一个实现。

当应用程序要为因特网通信而创建一个套接字（socket）时，操作系统就返回一个小整数作为描述符（descriptor）来标识这个套接字。然后，应用程序以该描述符作为传递参数，通过调用函数来完成某种操作（例如通过网络传送数据或接收输入的数据）。

在许多操作系统中，套接字描述符和其他I/O描述符是集成在一起的，所以应用程序可以对文件进行套接字I/O或I/O读/写操作。

(4)Socket接口函数



服务器端先初始化Socket，然后与端口绑定(bind)，对端口进行监听(listen)，调用accept阻塞，等待客户端连接。在这时如果有个客户端初始化一个Socket，然后连接服务器(connect)，如果连接成功，这时客户端与服务器端的连接就建立了。客户端发送数据请求，服务器端接收请求并处理请求，然后把回应数据发送给客户端，客户端读取数据，最后关闭连接，一次交互结束。

1、socket函数

syntax:

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

功能说明：

调用成功，返回socket文件描述符；失败，返回－1，并设置errno

参数说明：

domain指明所使用的协议族，通常为PF_INET，表示TCP/IP协议；

type参数指定socket的类型，基本上有三种：数据流套接字、数据报套接字、原始套接字

protocol通常赋值"0"。

两个网络程序之间的一个网络连接包括五种信息：通信协议、本地协议地址、本地主机端口、远端主机地址和远端协议端口。socket数据结构中包含这五种信息。

2、bind函数

syntax:

int bind(int sock_fd, struct sockaddr_in *my_addr, int addrlen);

功能说明：

将套接字和指定的端口相连。成功返回0，否则，返回－1，并置errno.

参数说明：

sock_fd是调用socket函数返回值,

my_addr是一个指向包含有本机IP地址及端口号等信息的sockaddr类型的指针；

struct sockaddr_in结构类型是用来保存socket信息的：

```
struct sockaddr_in {
        short int sin_family;
        unsigned short int sin_port;
        struct in_addr sin_addr;
};
```

addrlen为my_addr的长度。

3、connect函数

syntax:

int connect(int sock_fd, struct sockaddr *serv_addr,int addrlen);

功能说明：

客户端发送服务请求。成功返回0，否则返回－1，并置errno。

参数说明：

sock_fd 是socket函数返回的socket描述符；

serv_addr是包含远端主机IP地址和端口号的指针；

addrlen是结构sockaddr_in的长度。

4、listen函数

syntax:

int listen(int sock_fd， int backlog);

功能说明：

等待指定的端口的出现客户端连接。调用成功返回0，否则，返回－1，并置errno.

参数说明：

sock_fd 是socket()函数返回值；

backlog指定在请求队列中允许的最大请求数。

5、accecpt函数

syntax:

int accept(int sock_fd, struct sockadd_in* addr, int addrlen);

功能说明：

用于接受客户端的服务请求，成功返回新的套接字描述符，失败返回－1，并置errno。

参数说明：

sock_fd是被监听的socket描述符，

addr通常是一个指向sockaddr_in变量的指针，

addrlen是结构sockaddr_in的长度。

6、write函数

syntax:

ssize_t write(int fd,const void *buf,size_t nbytes)

功能说明：

write函数将buf中的nbytes字节内容写入文件描述符fd.成功时返回写的字节数.失败时返回-1. 并设置errno变量.

在网络程序中,当我们向套接字文件描述符写时有俩种可能：

1)write的返回值大于0,表示写了部分或者是全部的数据.

2)返回的值小于0,此时出现了错误.需要根据错误类型来处理.

如果错误为EINTR表示在写的时候出现了中断错误.

如果错误为EPIPE表示网络连接出现了问题。

7、read函数

syntax:

ssize_t read(int fd,void *buf,size_t nbyte)

函数说明：

read函数是负责从fd中读取内容.当读成功时,read返回实际所读的字节数,如果返回的值是0 表示已经读到文件的结束了,小于0表示出现了错误.

如果错误为EINTR说明读是由中断引起的,

如果错误是ECONNREST表示网络连接出了问题。

8、close函数

syntax:

int close(sock_fd);

说明:

当所有的数据操作结束以后，你可以调用close()函数来释放该socket，从而停止在该socket上的任何数据操作。

函数运行成功返回0，否则返回-1

9、IP地址转换

有三个函数将数字点形式表示的字符串IP地址与32位网络字节顺序的二进制形式的IP地址进行转换

（1） unsigned long int inet_addr(const char * cp)：该函数把一个用数字和点表示的IP地址的字符串转换成一个无符号长整型，如：struct sockaddr_in ina

ina.sin_addr.s_addr=inet_addr("202.206.17.101")

该函数成功时：返回转换结果；失败时返回常量INADDR_NONE，该常量=-1，二进制的无符号整数-1相当于255.255.255.255，这是一个广播地址，所以在程序中调用iner_addr（）时，一定要人为地对调用失败进行处理。由于该函数不能处理广播地址，所以在程序中应该使用函数inet_aton（）。

（2）int inet_aton（const char * cp,struct in_addr * inp）：此函数将字符串形式的IP地址转换成二进制形式的IP地址；成功时返回1，否则返回0，转换后的IP地址存储在参数inp中。

（3） char * inet_ntoa（struct in-addr in）：将32位二进制形式的IP地址转换为数字点形式的IP地址，结果在函数返回值中返回，返回的是一个指向字符串的指针。

三、实验内容

实现以下FTP命令：

get：获取远方文件。

put：传给远方文件。

pwd：显示远方目录。

dir：列出远方目录。

cd：更改远方当前目录。

?：显示提供的命令。

quit：退出。

四、实验代码

see ./src

五、思考题

(1)FTP中，为何要建立两个TCP连接来分别传送命令和数据?

答：如果共用端口，那么在传输数据的时候，就不能传输命令了。不但影响传输速度，也会提高出错概率。

(2)比较主动方式和被动方式的主要区别，为何要设计这两种方式?

建立命令通道的方式相同，建立数据通道的方式不同。

PORT（主动）方式：当需要传送数据时，客户端在命令链路上用PORT命令告诉服务器用什么端口接收数据。于是服务器从20端口向客户端的该端口发送连接请求，建立一条数据通道来传送数据。
PASV（被动）方式：当需要传送数据时，服务器打开一个端口，在控制通道上用PASV命令告诉客户端。于是客户端向服务器的该端口发送连接请求，建立一条数据通道来传送数据。
RFC制定ftp pasv模式的主要目的是为了数据传输安全角度出发的，因为PORT方式使用固定20端口进行传输数据，那么作为黑客很容抓取ftp数据，这样一来通过PORT模式传输数据很容易被黑客窃取，因此使用PASV方式较安全。

(3)使用FTP下载大量小文件时，速度会很慢，为什么？怎样改进?
因为下载文件时，即使是多线程，因为共用数据通道，总带宽是一定的。因此下载大量小文件，每个文件的下载速度都很慢。
建立多个数据通道。