码。在和服务器建立连接之后，客户端进入一个循环，反复从标准输入读取文本行，发送文本行给服务器，从服务器读取回送的行，并输出结果到标准输出。当 fgets 在标准输入上遇到 EOF 时，或者因为用户在键盘上键入 Ctrl+D，或者因为在一个重定向的输入文件中用尽了所有的文本行时，循环就终止。

code/netp/echoclient.c

```
1   #include "csapp.h"
2
3   int main(int argc, char **argv)
4   {
5       int clientfd;
6       char *host, *port, buf[MAXLINE];
7       rio_t rio;
8
9       if (argc != 3) {
10          fprintf(stderr, "usage: %s <host> <port>\n", argv[0]);
11          exit(0);
12      }
13      host = argv[1];
14      port = argv[2];
15
16      clientfd = Open_clientfd(host, port);
17      Rio_readinitb(&rio, clientfd);
18
19      while (Fgets(buf, MAXLINE, stdin) != NULL) {
20          Rio_writen(clientfd, buf, strlen(buf));
21          Rio_readlineb(&rio, buf, MAXLINE);
22          Fputs(buf, stdout);
23      }
24      Close(clientfd);
25      exit(0);
26  }
```

code/netp/echoclient.c

图 11-20　echo 客户端的主程序

循环终止之后，客户端关闭描述符。这会导致发送一个 EOF 通知到服务器，当服务器从它的 reo_readlineb 函数收到一个为零的返回码时，就会检测到这个结果。在关闭它的描述符后，客户端就终止了。既然客户端内核在一个进程终止时会自动关闭所有打开的描述符，第 24 行的 close 就没有必要了。不过，显式地关闭已经打开的任何描述符是一个良好的编程习惯。

图 11-21 展示了 echo 服务器的主程序。在打开监听描述符后，它进入一个无限循环。每次循环都等待一个来自客户端的连接请求，输出已连接客户端的域名和 IP 地址，并调用 echo 函数为这些客户端服务。在 echo 程序返回后，主程序关闭已连接描述符。一旦客户端和服务器关闭了它们各自的描述符，连接也就终止了。

第 9 行的 clientaddr 变量是一个套接字地址结构，被传递给 accept。在 accept 返回之前，会在 clientaddr 中填上连接另一端客户端的套接字地址。注意，我们将 clientaddr 声明为 struct sockaddr_storage 类型，而不是 struct sockaddr_in 类型。根据定义，sockaddr_storage 结构足够大能够装下任何类型的套接字地址，以保持代码的协议无关性。