基于TCP的服务器端和客户端2

在进行TCP传输数据的时候，要注意TCP的数据传输不存在数据边界，也就是说，我的客户端和服务器端read和write函数的次数是不一定相等的。Ok，这里单纯说理论是很难理解的，我就用新旧代码来进行一下对比，我们来考虑一下这个问题。

首先是没有进行任何修改的client代码：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#define BUF\_SIZE 1024

void error\_handling(char \*message);

int main(int argc,char \*argv[]){

int sock;

struct sockaddr\_in serv\_addr;

char message[BUF\_SIZE];

int str\_len;

if(argc != 3){

printf("Usage : %s <IP> <port>\n",argv[0]);

exit(1);

}

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

if(sock == -1){

error\_handling("socket() error");

}

memset(&serv\_addr,0,sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr = inet\_addr(argv[1]);

serv\_addr.sin\_port = htons(atoi(argv[2]));

if(connect(sock,(struct sockaddr \*)&serv\_addr,sizeof(serv\_addr)) == -1){

error\_handling("connect() error\r\n");

}else{

printf("Connected....");

}

**while(1){**

**printf("Input message(Q to quit):");**

**fgets(message,BUF\_SIZE,stdin);**

**if(!strcmp(message,"q\n") || !strcmp(message,"Q\n"))**

**break;**

**write(sock,message,strlen(message));**

**// sleep(5);**

**str\_len = read(sock,message,BUF\_SIZE-1);**

**message[str\_len] = 0;**

**printf("Message from server : %s \n",message);**

**}**

**close(sock);**

**return 0;**

**}**

void error\_handling(char \*message){

fputs(message,stderr);

fputs("\n",stderr);

exit(1);

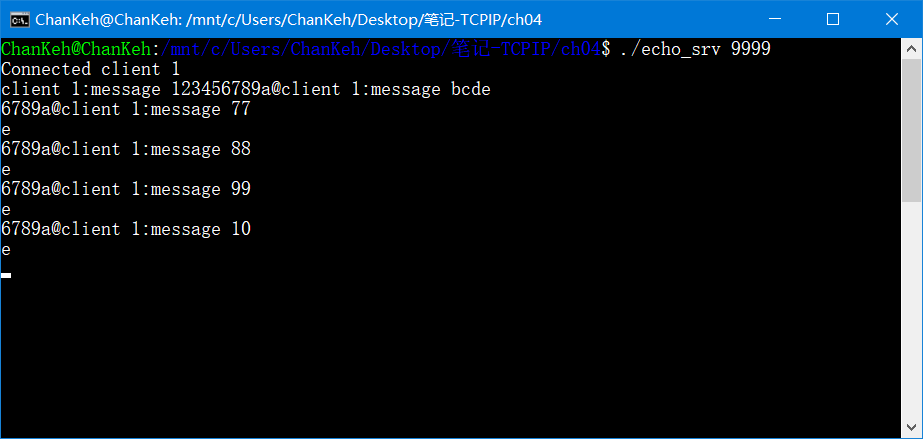
}

请大家看到红色部分的代码，write以后接着进行了read。

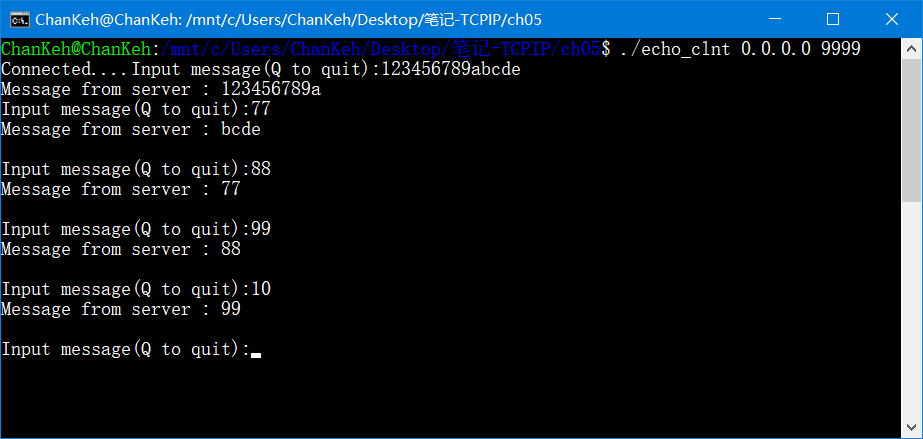
当执行了read以后，我输入的字符串数据就写入到了Scoket的输入缓冲中，而且是一次性全部写入。这时候呢，我们就得考虑一下服务器端的接收缓冲的大小了，尹圣雨老师在源代码里给接收缓冲和输入缓冲分配了同样的大小，都是1024字节，这里我认为不妥，这样的话，对于理解“TCP不存在收发边界”没有什么帮助。于是，这里我将接收缓冲的大小改成了10，更方便进行对比。

当先后运行服务器端和客户端后，我们看到了如下结果：

服务器端：



客户端：



Ok，这里我们可以清楚的看到，由于我的输入字符串长度超过了10，也就是说大于服务器端的接收缓冲大小，因此，字符串被截断，分为多个长度为10的分组（小于10的为最后一个分组）。同样的，服务器端每次都会将一个分组发回给客户端，这样，客户端就会接收到多个长度为10的字符串。所以，这个例子明显的证明了TCP的客户端和服务器端read和write次数不一致（也就是不存在收发边界）。

那么，为了改进这一缺点（使得客户端只接收一个完整的字符串），这里对客户端进行了改进，做法是这样的：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#define BUF\_SIZE 1024

void error\_handling(char \*message);

int main(int argc,char \*argv[]){

int sock;

struct sockaddr\_in serv\_addr;

char message[BUF\_SIZE];

int str\_len,recv\_len,recv\_cnt;

if(argc != 3){

printf("Usage : %s <IP> <port>\n",argv[0]);

exit(1);

}

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

if(sock == -1){

error\_handling("socket() error");

}

memset(&serv\_addr,0,sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr = inet\_addr(argv[1]);

serv\_addr.sin\_port = htons(atoi(argv[2]));

if(connect(sock,(struct sockaddr \*)&serv\_addr,sizeof(serv\_addr)) == -1){

error\_handling("connect() error\r\n");

}else{

printf("Connected....");

}

while(1){

printf("Input message(Q to quit):");

fgets(message,BUF\_SIZE,stdin);

if(!strcmp(message,"q\n") || !strcmp(message,"Q\n"))

break;

str\_len = write(sock,message,strlen(message));

recv\_len = 0;

**while(recv\_len < str\_len){**

**recv\_cnt = read(sock,&message[recv\_len],BUF\_SIZE-1);**

**if(recv\_cnt == -1)**

**error\_handling("read() error!");**

**recv\_len += recv\_cnt;**

**}**

message[str\_len] = 0;

printf("Message from server : %s \n",message);

}

close(sock);

return 0;

}

void error\_handling(char \*message){

fputs(message,stderr);

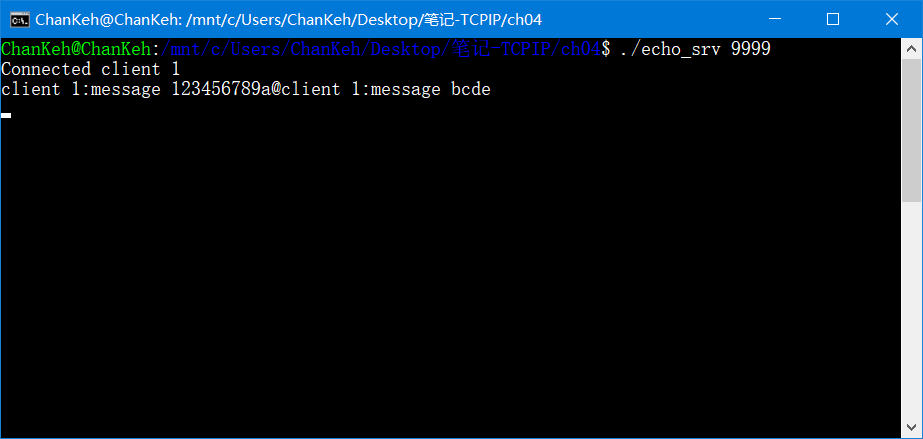
fputs("\n",stderr);

exit(1);

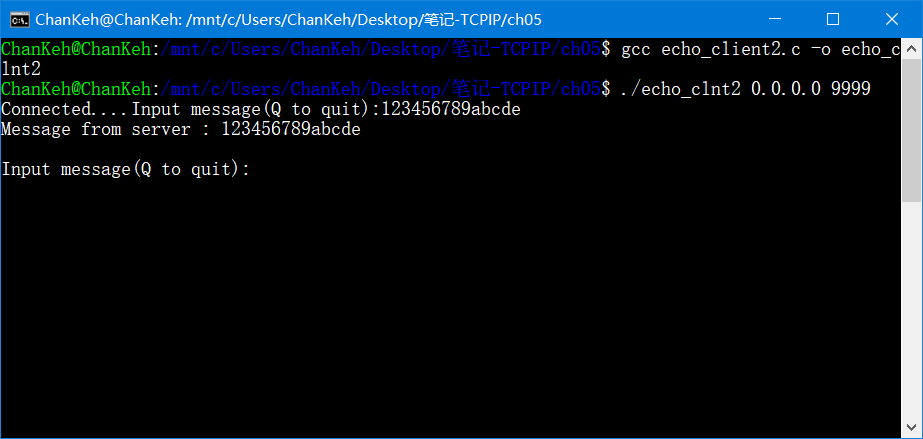
}

各位请看，红色加粗部分，在所有字符串接收完毕之前，我每一次以一定的字节数对服务器端发回的数据进行读取，一直到所有数据读取完毕，我才结束read，然后，将读取的所有数据存储到message这个数组中，最后以一个完整的字符串显示在客户端的console中。

服务器端显示：



客户端显示：



由结果可以看出，我的服务器端由于缓冲区大小仅为10，所以这里分两次对数据进行了读取。（now 数据全部准备完毕，将message可能会分多次写入Socket的发送缓冲区，准备发给客户端）。

这时候，客户端就会通过while循环内的数据，以一定的字节数进行读取，同时每次读取都将Socket的输出缓冲数据存储到客户端的message中，直到都读取完毕。

最后，把完整的message中的数据全部显示出来。也就看到了客户端中的显示结果。

【这里讲解的应该还算清楚吧….】