实验2 套接字编程

# 实验目的

Linux网络套接字编程是linux编程的核心内容，Linux网络套接字编程练习是理解Linux网络原理的重要手段，也是熟练运行Linux网络套接字实现网络编程的必要方式。本实验要求用C语言编写和调试应用Linux网络套接字编程的实现程序。以达到理解和运用Linux有关网络编程基础知识的目的。编写两个程序一个客户端一个服务端，使得两者间可以通信，并且服务端将客户端发送的字符串反转。

# 实验内容

实现一个服务端一个客户段，建立TCP/IP socket链接，然后服务端监听客户段发送的消息，并将其反转后转发给客户端即可。

# 实验步骤

* 明白Linux中建立TCP/IP　socket等方法
* 编写服务端程序,设定监听的IP和端口号,建立连接,当读取到客户段发送来的字符串时反转后发送
* 便写客户段程序,设定监听的IP和端口号,建立连接,接受用户输入的字符串,发送到服务端后将服务端发送来的字符串显示

## 任务分析

### socket

* Linux中服务端建立链接需要如下步骤: socket()确定协议类型、设定结构体 sockaddr\_in 实现ip和端口的设定、bind()绑定、listen()设定监听数以及accept()建立的数据流并通过这个流来进行数据的操作，注意这里服务端的socket有两个，一个是监听的，一个是与客户端建立的socket
* Linux中客户段建立链接需要如下步骤： socket()确定协议类型、设定结构体 sockaddr\_in 实现ip和端口的设定、connect()建立链接，客户端之后一个socket
* 数据流的读取与写入通过 read 和 write 函数实现
* 注意网络字节序和本机字节序的区别，并在适当的地方进行转换

## 概要设计、详细设计

### 服务端

### 

### 客户端

## 

## 调试分析

* 分别在两个终端编译运行客户端和服务端的程序
* 在客户端输入任意字符串
* 客户端应该显示服务端进行反转后的字符串
* 当输入 CLOSE 时两边退出

## 测试结果

* 服务端：

x31415@iZ2zea1nnstgr3h5ccfovoZ:~/linux/socket/server$ ./server  
client<127.0.0.1:55700>: asdfjdslkaf;  
client<127.0.0.1:55700>: sdlfjsdlkjaflk;asdf  
client<127.0.0.1:55700>: jdsakljfl;ksdafjlds  
client<127.0.0.1:55700>: jaklsjflkaskasdjlds  
client<127.0.0.1:55700>: abcdefglkajkasdjlds  
client<127.0.0.1:55700>: CLOSEbalkajkasdjlds

* 客户端

x31415@iZ2zea1nnstgr3h5ccfovoZ:~/linux/socket/client$ ./client   
asdfjdslkaf;  
server<127.0.0.1:2333>: ;faklsdjfdsa  
sdlfjsdlkjaflk;asdf  
server<127.0.0.1:2333>: fdsa;klfajkldsjflds  
jdsakljfl;ksdafj  
server<127.0.0.1:2333>: jfadsk;lfjlkasdj  
jaklsjflkas  
server<127.0.0.1:2333>: saklfjslkaj  
abcdefg  
server<127.0.0.1:2333>: gfedcba  
CLOSE  
server<127.0.0.1:2333>: CLOSE

## 使用说明

去服务端和客户端文件夹下进行编译源代码，使用 make 命令即可，然后分别在两个终端中运行程序，在客户端输入字符串，将回显服务端反转后的字符串，表明两者建立链接成功，输入 CLOSE 结束运行、断开连接。

# 实验总结

首先要清楚Linux下的socket使用方法，客户端和服务端的使用方法不同，所以要清楚理解，同时，服务端在建立链接的同时会创建2个socket其中一个用来监听，另一个用来建立于客户端的链接。还要注意网络字节序和本地字节序的区别，并在适当的位置进行转换。

# 代码

## 客户端

#include<stdio.h>  
#include<stdlib.h>  
#include<string.h>  
#include<errno.h>  
  
// socket(), bind()  
#include<sys/types.h>  
#include<sys/socket.h>  
  
// sockaddr\_in  
#include<arpa/inet.h>  
  
// read() write()  
#include<unistd.h>  
  
#define SERVER\_PORT 2333  
#define SERVER\_ADDR "127.0.0.1"  
void error\_print(const char\* e){  
 perror("Error: ");  
 perror(e);  
 strerror(errno);  
 exit(EXIT\_FAILURE);  
 return;  
}  
  
int main(int avgc, char\*\* avgs){  
  
 struct sockaddr\_in server\_addr;  
 server\_addr.sin\_family = AF\_INET;  
 server\_addr.sin\_port = htons(SERVER\_PORT);  
 if(!~inet\_pton(AF\_INET, SERVER\_ADDR, &server\_addr.sin\_addr.s\_addr))error\_print("server ip...");  
  
 int cfd = socket(AF\_INET, SOCK\_STREAM, 0);  
 if(!~cfd)error\_print("client socket...");   
  
 if(!~connect(cfd, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)))error\_print("connect...");  
  
 char buf[BUFSIZ];  
 int ret;  
  
 char server\_ip[BUFSIZ];  
 while(1){  
 scanf("%s", buf);  
 write(cfd, buf, strlen(buf));  
 ret = read(cfd, buf, sizeof(buf));  
 printf("server<%s:%d>: %s\n",  
 inet\_ntop(AF\_INET, &server\_addr.sin\_addr.s\_addr, server\_ip, sizeof(server\_ip)),  
 ntohs(server\_addr.sin\_port),   
 buf);  
 if(!strcmp(buf, "CLOSE"))break;  
 }  
  
 close(cfd);  
  
 return 0;  
}

## 服务端

#include<stdio.h>  
#include<stdlib.h>  
#include<string.h>  
#include<errno.h>  
  
// socket(), bind()  
#include<sys/types.h>  
#include<sys/socket.h>  
  
// sockaddr\_in  
#include<arpa/inet.h>  
  
// read() write()  
#include <unistd.h>  
  
  
#define SERVER\_PORT 2333  
  
void error\_print(const char\* e){  
 perror("Error: ");  
 perror(e);  
 strerror(errno);  
 exit(EXIT\_FAILURE);  
}  
void swap(char \*a, char \*b){  
 char t = \*a;  
 \*a = \*b;  
 \*b = t;  
 return;  
}  
int main(int avgc, char\*\* avgs){  
  
 // 服务器端 socket 建立  
 // ipv4, 流式数据包，TCP  
 int lfd = socket(AF\_INET, SOCK\_STREAM, 0);  
 if(!~lfd)error\_print("server socket...");  
  
 // 将监听的ip和端口与socket绑定  
 struct sockaddr\_in server\_addr;  
 server\_addr.sin\_family = AF\_INET; // ipv4  
 server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY); // 监听本机所有网卡的所有ip  
 server\_addr.sin\_port = htons(SERVER\_PORT); // 端口  
  
 if(!~bind(lfd, (struct sockaddr\*)&server\_addr, sizeof(server\_addr))){  
 error\_print("server bind...");  
 }  
  
 // 设定监听的上限，默认为128 listen  
 listen(lfd, 10);  
  
 // 监听 accept  
 struct sockaddr\_in client\_addr; // 与客户端相连的socket  
 socklen\_t addrlen = sizeof(client\_addr); // 传入传出  
 int cfd = accept(lfd, (struct sockaddr\*)&client\_addr, &addrlen);  
 if(!~cfd)error\_print("server accept...");  
  
 char buf[BUFSIZ];  
 int ret = 0;  
  
 char client\_ip[BUFSIZ];  
 while(1){  
 ret = read(cfd, buf, sizeof(buf));  
 fprintf(stdout, "client<%s:%d>: %s\n",   
 inet\_ntop(AF\_INET, &client\_addr.sin\_addr.s\_addr, client\_ip, sizeof(client\_ip)),  
 ntohs(client\_addr.sin\_port),  
 buf);  
 fflush(stdout);  
 // printf("client<%s:%d>: %s",   
 // inet\_ntop(AF\_INET, &client\_addr.sin\_addr.s\_addr, client\_ip, sizeof(client\_ip)),  
 // ntohs(client\_addr.sin\_port),  
 // buf);  
 if(!strncmp(buf, "CLOSE", ret - 1))break;  
 for(int i = 0; i < (ret >> 1); ++i)swap(&buf[i], &buf[ret - i - 1]);  
 write(cfd, buf, ret);  
 }  
   
 // 关闭连接  
 close(cfd);  
 close(lfd);  
  
 return 0;  
}