### C++ 笔记

**内网穿透参考网页：**

<https://blog.csdn.net/yijunziran/article/details/78376352>

**Linux下eclipse**使用技巧：

添加头文件路径方法：

project-->properties-->C/C++ General-->paths and symbols

跳到定义：

F3

Debug时显示Variables查看变量

window -- show view -- other -- Debug -- Exceptions\Variables

1. **socket（服务端+客户端）：**

注意：如果服务端使用accept()监听连接，必须使用accept()返回的套接字进行write()或read()通讯

**服务端配置：**

1. **socket()**

创建套接字，打开一个网络连接

返回值：<0失败，成功的话返回网络文件描述符（socket套接字）

int socket(

int domain, //协议簇tcp协议填AF\_INET

int type, //套接字类型：SOCK\_STREAM (tcp)，SOCK\_DGRAM (udp)

int protocol //0：使用默认协议

);

**2、sockaddr\_in结构体：**

sin\_family=AF\_INET //协议家族：使用TCP/IP填AF\_INET

sin\_addr.s\_addr=htonl(IP) //IP地址

sin\_port=htons(PORT) //端口号

**3、bind()**

服务端套接字**绑定**自己的IP和端口号（就是socket()和sockaddr\_in结构体）

返回值：<0失败

原型：int bind(

int sockfd, //服务端socket套接字

const struct sockaddr \*addr, //服务端或客户端地址结构体

socklen\_t addrlen //服务端或客户端地址信息长度（可以直接

sizeof(addr)）

);

**4、listen()**

监听是否有客户端连接

返回值：<0失败

需要添加头文件：#include "sys/socket.h"

int listen(

int sockfd, //服务端socket套接字

int backlog //允许连接的最大客户端数目

);

**5、accept()**

套接口等待接受一个连接，如果没有连接将**阻塞任务**

返回值：服务器和客户端通讯的套接字

**注意：**如果是监听连接，必须使用accept()返回的套接字进行通讯

SOCKET accept(

int sockfd, //服务端socket套接字

struct sockaddr \*addr, //记录客户端IP，不用的话写NULL

socklen\_t \*addrlen //

);

**完整服务器代码（多进程收发客户端消息）：**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 服务端 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdlib.h>

#include<string.h>

#include<errno.h>

#include "stdio.h"

#include "sys/socket.h"

#include "netinet/in.h"

#include "sys/types.h"

#include "unistd.h" //实现多进程需要包含的头文件

int main(){

int ser\_socket,cli\_socket;

char buf[512],write\_buf[512];

pid\_t ser\_read\_task,cli\_write\_task;

struct sockaddr\_in ser\_addr;

memset(buf,0,sizeof(char)\*512);

ser\_socket=socket(AF\_INET,SOCK\_STREAM,0); //使用TCP协议

if(ser\_socket<0){ //创建失败

perror("socket");

return -1;

}

ser\_addr.sin\_port=htons(8888);

ser\_addr.sin\_addr.s\_addr=htonl(INADDR\_ANY); //?

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

if(bind(ser\_socket, (struct sockaddr\*)&ser\_addr,sizeof(ser\_addr))<0){ //绑定ip和端口

perror("bind");

return -2;

}

if(listen(ser\_socket,10)<0){ //监听，允许10个客户端同时连接

perror("listen");

return -3;

}

printf("====== waiting for client's request\n");

for(;;){

if((cli\_socket=accept(ser\_socket,NULL,NULL))<0){ //允许一个连接

perror("accept");

}

printf("++++++ new client connected...\n");

ser\_read\_task=fork(); //创建一个子进程

if(ser\_read\_task==0){ //子进程

printf("++++++ create read task\n");

close(ser\_socket);

for(;;){

read(cli\_socket,buf,sizeof(buf)); //不断接收客户端发送来的内容

if(strlen(buf)==0){ //客户端关闭了连接

printf("------ client disconnect\n");

break;

}

printf("buf len: %ld receive: %s\n",strlen(buf),buf);

write(cli\_socket,buf,strlen(buf)); //发送读取的内容到客户端

memset(buf,0,sizeof(char)\*512);

}

exit(0); //结束当前子进程

}

cli\_write\_task=fork(); //创建一个子进程

if(cli\_write\_task==0){ //子进程

printf("++++++ create write task\n");

close(ser\_socket);

for(;;){

printf("enter 'q' disconnect please putin: \n");

fgets(write\_buf,512,stdin);

if(strcmp(write\_buf,"q\n")==0){

printf("------ write disconnect\n");

break;

}

write(cli\_socket,write\_buf,strlen(write\_buf));

memset(write\_buf,0,sizeof(char)\*512);

}

exit(0); //结束当前子进程

}

close(cli\_socket);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 服务端 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**客户端配置：**

**1、sockaddr\_in结构体：**

sin\_family=AF\_INET //协议家族：使用TCP/IP填AF\_INET

sin\_addr.s\_addr=inet\_addr(IP) //IP地址

sin\_port=htons(PORT) //端口号

1. **socket()**

创建套接字

打开一个网络连接，返回值：<0失败，成功的话返回网络文件描述符（socket套接字）

int socket(

int domain, //协议簇tcp协议填AF\_INET

int type, //套接字类型：SOCK\_STREAM (tcp)，SOCK\_DGRAM (udp)

int protocol //0：使用默认协议

);

**3、connect()**

客户端通过connect函数与服务端连接进行通信，返回值：<0失败

int connect(

int sockfd, //客户端socket套接字

const struct sockaddr \*addr, //服务端地址信息结构体

socklen\_t addrlen //服务端地址信息结构体大小（sizeof(addr)）

);

1. **fgets()**

接收命令行输入的字符串，并在结尾添加回车（\n）

相比gets()使用fgets()不会发生内存溢出，就算输入的内容超出缓存区，也会自动截断超出部分，但会在接收字符串结尾添加（\n）

注意：fgets()会在接收的字符串结尾添加（\n）

**完整客户端代码：（创建了一个子进程用来接收服务端发来的消息）**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 客户端 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdlib.h>

#include<string.h>

#include<errno.h>

#include "stdio.h"

#include "sys/socket.h"

#include "netinet/in.h"

#include "sys/types.h"

#include "unistd.h" //实现多进程需要包含的头文件

int main(){

int cli\_socket;

char buf[512],read\_buf[512];

pid\_t cli\_write\_task,ser\_read\_task;

struct sockaddr\_in ser\_addr;

memset(buf,0,sizeof(char)\*512);

memset(read\_buf,0,sizeof(char)\*512);

cli\_socket=socket(AF\_INET,SOCK\_STREAM,0); //创建套接字 使用TCP协议

if(cli\_socket<0){ //创建失败

perror("socket");

return -1;

}

ser\_addr.sin\_port=htons(7777);

ser\_addr.sin\_addr.s\_addr=inet\_addr("192.168.1.189"); //?

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

printf("====== start connecting...\n");

if(connect(cli\_socket, (struct sockaddr\*)&ser\_addr,sizeof(ser\_addr))<0){ //连接服务端

perror("connect");

return -2;

}

printf("++++++ connect success!\n");

ser\_read\_task=fork();

if(ser\_read\_task==0){

//close(cli\_socket);

for(;;){

read(cli\_socket,read\_buf,sizeof(read\_buf)); //不断接收服务端发送来的内容

if(strlen(read\_buf)==0){ //客户端关闭了连接

printf("------ client disconnect\n");

break;

}

printf("buf len: %ld receive: %s\n",strlen(read\_buf),read\_buf);

write(cli\_socket,read\_buf,strlen(read\_buf)); //发送读取的内容到服务端

memset(read\_buf,0,sizeof(char)\*512);

}

exit(0); //结束当前子进程

}

// cli\_write\_task=fork();

// if(cli\_write\_task==0){

printf("++++++ create read task\n");

//close(cli\_socket);

for(;;){

printf("enter 'q' disconnect pease putin:\n");

fgets(buf,512,stdin);

if(strcmp(buf,"q\n")==0){

printf("------ disconnect\n");

break;

}

write(cli\_socket,buf,strlen(buf));

memset(buf,0,sizeof(char)\*512);

}

exit(0);

// }

close(cli\_socket);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 客户端 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**2）gdb调试C++代码：**

在编译时要加上-g选项，生成的可执行文件才能用gdb进行源码级调试如下：

gcc **-g** test.c **-o** test //加上**-g**编译

**gdb** test //调试

基本调试指令：

**start** 开始执行程序，停在main函数第一行语句前面等待命令

list（或**l**） 列出源代码，接着上次的位置往下列，每次列10行

next（或**n**） 执行下一行语句

step（或**s**） 执行下一行语句，如果有函数调用则进入到函数中

break（或**b**）行号 对程序的第n行进行打断点

**c** 停留在断点后，输c继续执行

run（或**r**） 运行程序，如果打了断点，停在断点行

print（或**p**） 打印表达式的值，通过表达式可以修改变量的值或者调用函数

backtrace（或bt） 查看各级函数调用及参数

finish 连续运行到当前函数返回为止，然后停下来等待命令

frame（或f） 帧编号 选择栈帧

info（或i） locals 查看当前栈帧局部变量的值

list 行号 列出从第几行开始的源代码

list 函数名 列出某个函数的源代码

quit（或q） 退出gdb调试环境

set var 修改变量的值

1. **fork()**

**注意：**父子进程之间的变量是**位于不同的地址空间**的，**子进程能继承父进程中的变量，但无论如何修改也不会反映到父进程中**

创建子进程，返回值：if=0执行子进程，if>0执行父进程，其他值说明失败

pid\_t child; //子进程句柄

child=fork(); //fork一个子进程

if(child== 0){ //child=0时执行子进程,

printf("this is child process, pid is %d\n",getpid());

}

else if(child> 0){ //child>0时执行父进程

printf("this is father process, pid is %d\n",getpid());

}

else{

printf(“child create failed\n”);

}

**4）setsockopt()**

<http://www.cnblogs.com/qiaoconglovelife/p/5416715.html>

设置套接字选项：

int setsockopt(

int s, //打开的套接字

int level, //套接字等级，一般为SOL\_SOCKET（基本套接字）

int optname, //设置为SO\_REUSEADDR选项，即可重用地址、端口

const void \*optval, //flag（flag=1）

socklen\_t optlen //sizeof(int)

);

1. **recv()**

不论是客户还是服务器应用程序都用recv函数从TCP连接的另一端接收数据

如果没有接收到数据将**阻塞任务**

Int recv(

int fd, //套接字

void \*buf, //接收数据缓存区

int len, //缓存区大小

int flags //一般为0

)

flags：

MSG\_WAITALL：等待所有数据

MSG\_PEEK：查看数据，并不从系统缓冲区移走数据

MSG\_OOB：接受或者发生带外数据

1. **send()**

将buf内的内容发送到文件描述符（套接字）

返回值：成功返回字节数，<0失败

Int send(

int fd, //套接字

void \*buf, //写入缓冲区

int len, //缓冲区长度（要写入的字节数）

int flags //一般为0

);

flags：

MSG\_DONTROUTE：不查找表

1. **read()**

从文件描述符（套接字）中读取内容，没有接收到数据将**阻塞任务**

返回值：成功时返回实际读取的字节数，到0说明读取结束。<0失败

Ssize\_t read(

int fd, //文件描述符（套接字）

void \*buf, //缓存区

size\_t nbyte //缓存区大小（sizeof(buf)）

);

1. **write()**

将buf中的nbytes字节内容写入到文件描述符中（套接字）

返回值：成功返回字节数，<0失败

Ssize\_t write(

int fd, //文件描述符（套接字）

const void \*buf, //写入缓冲区

size\_t nbytes //缓冲区长度（要写入的字节数）

);

**9）inet\_ntoa()**

将网络地址转换成“.”点隔的字符串格式（网络地址->a.b.c.d）

例：

addr=inet\_ntoa(网络ip)

1. **inet\_addr()**

将字符串转化为网络地址（a.b.c.d->网络地址）

例：

addr.s\_addr=inet\_addr("192.168.1.60");

1. **内网穿透实验：**

公网牵线服务器（s）： //下发a的套接字给a和b

#include<stdlib.h>

#include<string.h>

#include<errno.h>

#include "stdio.h"

#include "sys/socket.h"

#include "netinet/in.h"

#include "sys/types.h"

#include "unistd.h" //实现多进程需要包含的头文件

#include <arpa/inet.h>

typedef struct {

struct in\_addr ip;

int port;

} IP;

int main(){

int ser\_socket,cli\_socket[2],i=0;

//SA\_IN server,addr;

IP ip[2];

char buf[512],write\_buf[512];

//pid\_t child\_task[2];

struct sockaddr\_in ser\_addr,addr;

socklen\_t addrlen=sizeof(addr);

memset(buf,0,sizeof(char)\*512);

ser\_socket=socket(AF\_INET,SOCK\_STREAM,0); //使用TCP协议

if(ser\_socket<0){ //创建失败

perror("socket");

return -1;

}

ser\_addr.sin\_port=htons(8888);

ser\_addr.sin\_addr.s\_addr=htonl(INADDR\_ANY); //?

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

if(bind(ser\_socket, (struct sockaddr\*)&ser\_addr,sizeof(ser\_addr))<0){ //绑定ip和端口

perror("bind");

return -2;

}

if(listen(ser\_socket,10)<0){ //监听，允许10个客户端同时连接

perror("listen");

return -3;

}

printf("====== waiting for client's request\n");

for(;;){

if((cli\_socket[i]=accept(ser\_socket,(struct sockaddr\*)&addr,&addrlen))<0){ //允许一个连接

perror("accept");

}

printf("++++++ new client connected...\n");

//memset(buf,0,sizeof(char)\*512);

memcpy(&ip[i].ip,&addr.sin\_addr,sizeof(struct in\_addr)); //获取ip

ip[i].port=addr.sin\_port; //获取port

printf("ip: %s port: %d\n",inet\_ntoa(addr.sin\_addr),ntohs(addr.sin\_port));

i++;

/\*

if(i>=2){

i=0;

//交叉发送客户端IP给客户端

send(cli\_socket[0],&ip[1],sizeof(IP),0);

send(cli\_socket[1],&ip[0],sizeof(IP),0);

//write();

printf("\n------sending\n");

printf("send ip[0]: %s port[0]: %d\n",inet\_ntoa(ip[0].ip),ntohs(ip[0].port));

printf("send ip[1]: %s port[1]: %d\n",inet\_ntoa(ip[1].ip),ntohs(ip[1].port));

close(cli\_socket[0]);

close(cli\_socket[1]);

printf("------ all disconnect\n");

}

\*/

if(i>=2){

i=0;

/\*\*\*\*\*\*发送a客户端IP给两个客户端\*\*\*\*\*\*/

send(cli\_socket[0],&ip[0],sizeof(IP),0); //a

close(cli\_socket[0]);

send(cli\_socket[1],&ip[0],sizeof(IP),0); //b

close(cli\_socket[1]);

//write();

printf("\n------sending\n");

printf("send ip[0]: %s port[0]: %d\n",inet\_ntoa(ip[0].ip),ntohs(ip[0].port));

printf("send ip[0]: %s port[0]: %d\n",inet\_ntoa(ip[0].ip),ntohs(ip[0].port));

printf("------ all disconnect\n");

}

}

}

客户端（a）： //先连接公网服务器，等待服务端下发port，根据port转换为服务端，listen等待b连接

#include<stdlib.h>

#include<string.h>

#include<errno.h>

#include "stdio.h"

#include "sys/socket.h"

#include "netinet/in.h"

#include "sys/types.h"

#include "unistd.h" //实现多进程需要包含的头文件

#include <arpa/inet.h>

#define SER "192.168.1.178"

#define PORT 8888

typedef struct {

struct in\_addr ip;

int port;

} IP;

typedef struct sockaddr SA;

typedef struct sockaddr\_in SA\_IN;

void callshow(int sockfd){ //连接成功，开启回显服务

char buf[512];

printf("++++++ new client connect\n");

for(;;){

memset(buf,0,512\*sizeof(char));

recv(sockfd,buf,sizeof(buf),0);

printf("call recv: %s\n",buf);

send(sockfd,buf,strlen(buf),0);

}

}

int main(){

int msocket[2],cli\_socket,flag=1;

IP ip;

//pid\_t child\_task[2];

SA\_IN server,addr;

socklen\_t addrlen=sizeof(SA\_IN);

char s='a',i=0;

char buf[512],write\_buf[512];

pid\_t ser\_read\_task,cli\_write\_task;

memset(buf,0,sizeof(char)\*512);

memset(write\_buf,0,sizeof(char)\*512);

msocket[0]=socket(AF\_INET,SOCK\_STREAM,0); //使用TCP协议

if(msocket[0]<0){ //创建失败

perror("socket");

return -1;

}

if(setsockopt(msocket[0],SOL\_SOCKET,SO\_REUSEADDR,&flag,sizeof(int))<0){

perror("setsockopt");

return -2;

}

server.sin\_port=htons(PORT);

server.sin\_addr.s\_addr=inet\_addr(SER); //?

server.sin\_family=AF\_INET;

connect(msocket[0],(SA\*)&server,sizeof(SA\_IN));

send(msocket[0],&s,sizeof(char),0); //告知服务端，我是 a客户端

recv(msocket[0],&ip,sizeof(IP),0); //阻塞，直到服务器下发a的外网ip和port

printf("ip: %s port: %d\n",inet\_ntoa(ip.ip),ntohs(ip.port)); //终于接收到对方ip

close(msocket[0]); //关闭与服务器的连接

msocket[1]=socket(AF\_INET,SOCK\_STREAM,0); //使用TCP协议

if(msocket[1]<0){ //创建失败

perror("socket");

return -1;

}

if(setsockopt(msocket[1],SOL\_SOCKET,SO\_REUSEADDR,&flag,sizeof(int))<0){

perror("setsockopt");

return -2;

}

server.sin\_family=AF\_INET;

server.sin\_port=ip.port;

server.sin\_addr.s\_addr=htonl(INADDR\_ANY); //?

if(bind(msocket[1],(SA\*)&server,sizeof(SA\_IN))<0){

perror("bind");

return -3;

}

if(listen(msocket[1],10)<0){ //转换为服务端，监听连接，端口和上次连接S时一样

perror("listen");

return -4;

}

printf("====== start accetping...\n");

if((cli\_socket=accept(msocket[1],(SA\*)&addr,&addrlen))<0){ //等待一个连接

perror("accept");

return -5;

}

//callshow(accept(msocket[1],(SA\*)&server,sizeof(SA\_IN))); //等待一个连接

printf("ip: %s port: %d\n",inet\_ntoa(ip.ip),ntohs(ip.port));

printf("++++++ new client connected\n");

ser\_read\_task=fork(); //创建一个子进程

if(ser\_read\_task==0){ //子进程

printf("++++++ create read task\n");

close(msocket[1]);

for(;;){

read(cli\_socket,buf,sizeof(buf)); //不断接收客户端发送来的内容

if(strlen(buf)==0){ //客户端关闭了连接

printf("------ client disconnect\n");

break;

}

printf("buf len: %ld receive: %s\n",strlen(buf),buf);

//write(msocket[1],buf,strlen(buf)); //发送读取的内容到客户端

memset(buf,0,sizeof(char)\*512);

}

exit(0); //结束当前子进程

}

// cli\_write\_task=fork(); //创建一个子进程

// if(cli\_write\_task==0){ //子进程

// printf("++++++ create write task\n");

// close(msocket[1]);

for(;;){

printf("enter 'q' disconnect please putin: \n");

fgets(write\_buf,512,stdin);

if(strcmp(write\_buf,"q\n")==0){

printf("------ write disconnect\n");

break;

}

write(cli\_socket,write\_buf,strlen(write\_buf));

memset(write\_buf,0,sizeof(char)\*512);

}

exit(0); //结束当前子进程

// }

/\*

for(;;){ //测试

i=read(cli\_socket,buf,sizeof(buf));

printf("read return %d %s \n",i,buf);

memset(buf,0,512\*sizeof(char));

}

\*/

close(cli\_socket);

printf("------ disconnect\n");

return 0;

}

客户端（b）： //连接公网端，等待服务端下发a套接字（ip和port的简称），根据套接字连接a

#include<stdlib.h>

#include<string.h>

#include<errno.h>

#include "stdio.h"

#include "sys/socket.h"

#include "netinet/in.h"

#include "sys/types.h"

#include "unistd.h" //实现多进程需要包含的头文件

#include <arpa/inet.h>

#define SER "192.168.1.178"

#define PORT 8888

typedef struct {

struct in\_addr ip;

int port;

} IP;

typedef struct sockaddr SA;

typedef struct sockaddr\_in SA\_IN;

void callshow(int sockfd){ //连接成功，开启回显服务

char buf[512];

printf("++++++ new client connect\n");

for(;;){

memset(buf,0,512\*sizeof(char));

recv(sockfd,buf,sizeof(buf),0);

printf("call recv: %s\n",buf);

send(sockfd,buf,strlen(buf),0);

}

}

int main(){

int msocket[2],flag=1;

IP ip;

char buf[512],read\_buf[512],i;

pid\_t cli\_write\_task,ser\_read\_task;

memset(buf,0,sizeof(char)\*512);

memset(read\_buf,0,sizeof(char)\*512);

//pid\_t child\_task[2];

SA\_IN server,addr;

socklen\_t addrlen=sizeof(SA\_IN);

char s='b';

msocket[0]=socket(AF\_INET,SOCK\_STREAM,0); //使用TCP协议

if(msocket[0]<0){ //创建失败

perror("socket");

return -1;

}

server.sin\_port=htons(PORT);

server.sin\_addr.s\_addr=inet\_addr(SER); //?

server.sin\_family=AF\_INET;

if(setsockopt(msocket[0],SOL\_SOCKET,SO\_REUSEADDR,&flag,sizeof(int))<0){

perror("setsockopt");

return -2;

}

connect(msocket[0],(SA\*)&server,sizeof(SA\_IN));

send(msocket[0],&s,sizeof(char),0); //告知服务端，我是 b客户端

recv(msocket[0],&ip,sizeof(IP),0); //阻塞，直到服务器下发目标的外网ip和port

printf("ip: %s port: %d\n",inet\_ntoa(ip.ip),ntohs(ip.port)); //终于接收到对方ip

close(msocket[0]); //关闭与服务器的连接

msocket[1]=socket(AF\_INET,SOCK\_STREAM,0); //使用TCP协议

if(msocket[1]<0){ //创建失败

perror("socket");

return -1;

}

server.sin\_port=(ip.port);

server.sin\_addr=(ip.ip); //?

if(connect(msocket[1],(SA\*)&server,sizeof(SA\_IN))<0){ //连接目标

perror("connect");

return -2;

}

printf("++++++ connect success!!!\n");

printf("ip: %s port: %d\n",inet\_ntoa(ip.ip),ntohs(ip.port));

ser\_read\_task=fork();

if(ser\_read\_task==0){

//close(cli\_socket);

for(;;){

recv(msocket[1],read\_buf,sizeof(read\_buf),0); //不断接收服务端发送来的内容

if(strlen(read\_buf)==0){ //客户端关闭了连接

printf("------ client disconnect\n");

break;

}

printf("buf len: %ld receive: %s\n",strlen(read\_buf),read\_buf);

//write(msocket[1],read\_buf,strlen(read\_buf)); //发送读取的内容到服务端

memset(read\_buf,0,sizeof(char)\*512);

}

exit(0); //结束当前子进程

}

// cli\_write\_task=fork();

// if(cli\_write\_task==0){

printf("++++++ create read task\n");

//close(cli\_socket);

for(;;){

printf("enter 'q' disconnect pease putin:\n");

fgets(buf,512,stdin);

if(strcmp(buf,"q\n")==0){

printf("------ disconnect\n");

break;

}

write(msocket[1],buf,strlen(buf));

memset(buf,0,sizeof(char)\*512);

}

exit(0);

// }

/\*

for(;;){ //测试

printf("please putin:\n");

fgets(buf,512,stdin);

i=write(msocket[1],buf,strlen(buf));

printf("write return %d\n",i);

memset(buf,0,512\*sizeof(char));

}

\*/

close(msocket[1]);

return 0;

}

1. **strstr()查找字串里子串的位置：**

返回值（指针）：存在则返回子串的内存地址，不存在返回null

char \*str=”hello world cpl”;

char \* num=0; //用来记录子串起始地址

num=strstr(str,”cpl”);

Int i=num-str; //子串起始地址-字串起始地址=子串相对字串的位置

Printf(“字串在第%d位\n”,i);

**13）主机字节序与网络字节序的转换**

因为在网络传输中，字节传输顺序的不同将其分为小端字节序和大端字节序

一般来说主机字节序=小端字节序，网络字节序=大端字节序

**小端**字节序：**低字节**放在内存**低地址**

**大端**字节序：**低字节**放在内存**高地址**

例：

char a=0x11223344;

**大端字**节序 **小端字**节序

内存地址 数据 内存地址 数据

0x00 11 0x00 44

0x01 22 0x01 33

0x02 33 0x02 22

0x03 44 0x03 11

所以在网络传输中需要转换：

主机字节序和网络字节序转换函数：

需包含头文件#include "netinet/in.h"

h：host主机

n：network网络

s：short短

l：long长

htons：主机字节序->网络字节序，短

htonl：主机字节序->网络字节序，长

**14）printf和sprintf格式化遇到的问题：**

解决格式化出现0xffffff--前缀的问题：

例：

char a=0xbf,b=0x7f;

printf(“%02x”,a);

printf(“%02x”,b);

输出：

ffffffbf //为什么会有0xffffff前缀？？

7f

**因为：**

格式化前会对变量进行判断，如果是**有符号类型**（不加unsigned的），且**二进制首位为1**，则会**按补码形式**，**前面全部置1**，也就是全f

**解决方法：**

方法1、格式化前强转为无符号类型（unsigned char）

方法2、定义时，定义为无符号类型（unsigned char）

**15）websocket服务端实现（使用eclipse）：**

**实例：**

服务端接收到客户端发送的握手包内容：

GET / HTTP/1.1

Host: 118.25.40.163:8088

Connection: Upgrade

Pragma: no-cache

Cache-Control: no-cache

Upgrade: websocket

Origin: http://www.blue-zero.com

Sec-WebSocket-Version: 13

User-Agent: Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/66.0.3359.181 Chrome/66.0.3359.181 Safari/537.36

Accept-Encoding: gzip, deflate

Accept-Language: zh-CN,zh;q=0.9

Sec-WebSocket-Key: **VJgA0bt1uf5OoEaL7mXwmA==**

Sec-WebSocket-Extensions: permessage-deflate; client\_max\_window\_bits

服务端发送给客户端的响应包

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: 3jbnSxBVNlOClI8cev4ddfa9YYc=

**解决方案：**

1. **响应包Sec-WebSocket-Accept字段值的计算：**

**第一步，拼接GUID：**

将客户端握手包拼接上固定字串258EAFA5-E914-47DA-95CA-C5AB0DC85B11

得到**VJgA0bt1uf5OoEaL7mXwmA==**258EAFA5-E914-47DA-95CA-C5AB0DC85B11

**第二步，转换为哈希值sha1：**

将第一步的结果计算sha1值：

安装openssl软件包（已经备份到Linux命令目录）：

安装和eclipse配置参考网页：

<https://blog.csdn.net/xueyushenzhou/article/details/22808629>

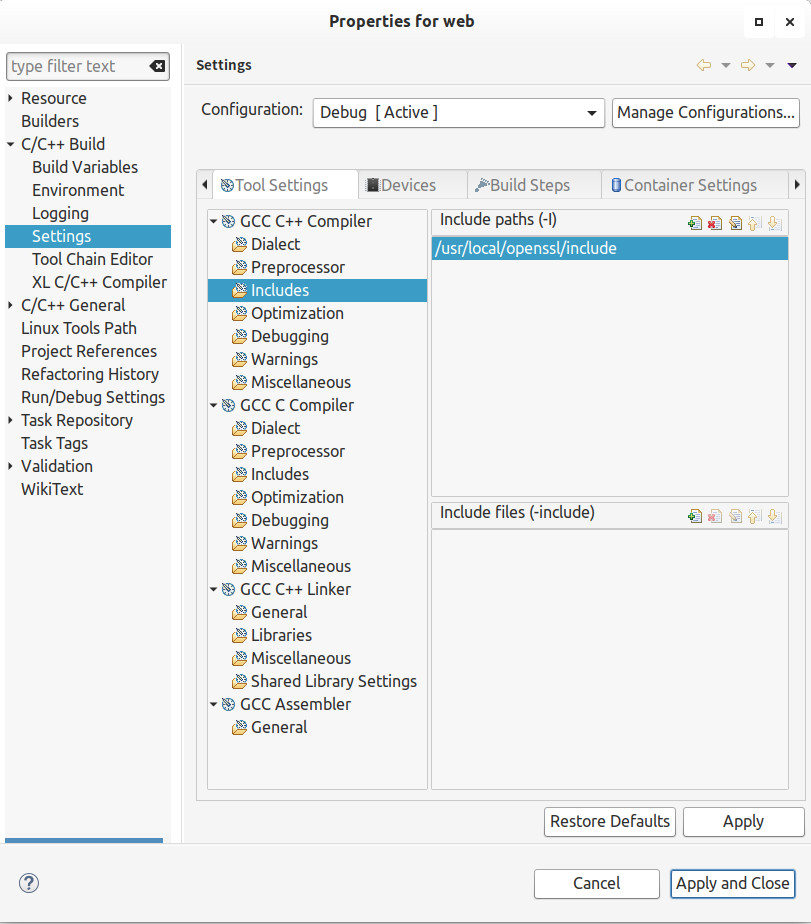
安装openssl：

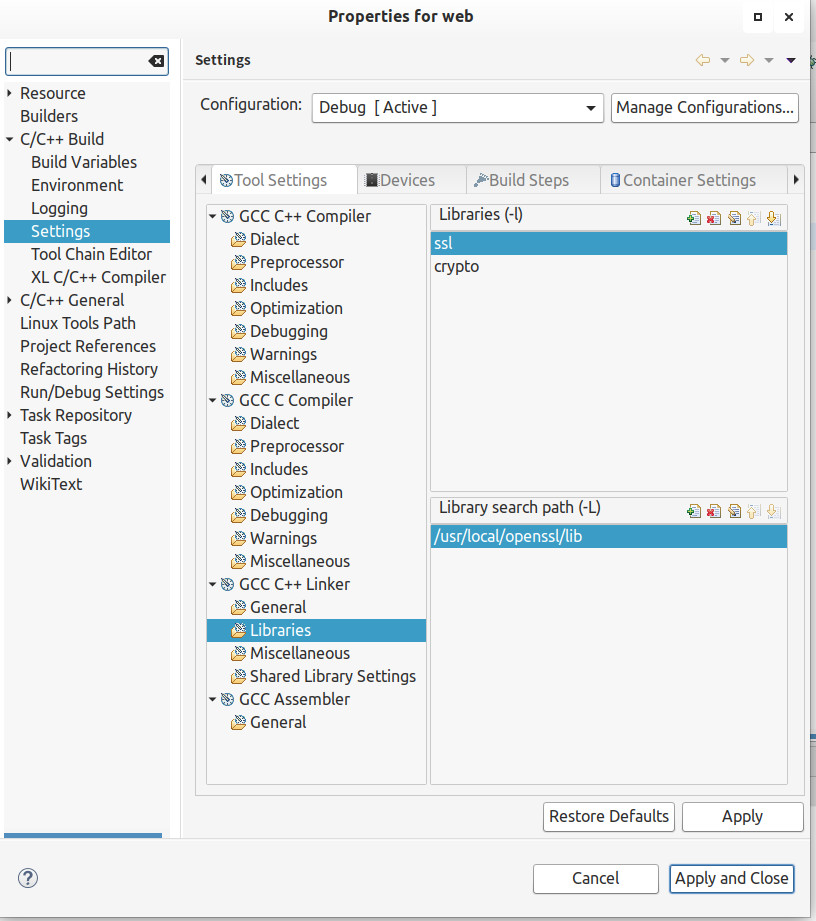
./config --prefix=/usr/local/openssl shared

make

sudo make install

Eclipse配置openssl：





**代码实现：**

#define sha1\_len 20 //哈希密文长度=20字节=160位

unsigned char recbuf[sha1\_len]={0}; //保存密文2进制码

char outbuf[sha1\_len\*2]={0}; //保存密文字串（最终结果）

char \* p="hello world";

memset(recbuf,0,sha1\_len);

memset(outbuf,0,sha1\_len\*2);

SHA1((const unsigned char\*)p,strlen(p),recbuf); //得到哈希值

for(int i=0;i<sha1\_len;i++){

sprintf(&outbuf[i\*2],"%02x",(unsigned int)recbuf[i]); //格式化

}

printf("%s",outbuf);

**输出sha1值字串：**

2aae6c35c94fcfb415dbe95f408b9ce91ee846ed

**第三步，编码为base64：**

Base64编码表0-63：

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

编码原理（以abcd为例）：

转换为ASCII --> 97 98 99 100

转换为2进制 --> 01100100 01100011 01100010 01100001

合并2进制 --> 01100100011000110110001001100001

6位取，前面补00 -->

00011001 00000110 00001101 00100010 -- 00011000 00010000

装换10进制 --> 25 6 13 34 24 16

对照编码数组 YWJjZA==

**代码实现**（fgets接收字串编码为base64）**：**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "netinet/in.h"

#include "unistd.h"

#include "sys/types.h"

#include <openssl/sha.h>

int main(){

//base64编码

int i=0,ob=0;

char \*code="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

char recvbuf[512];

char \*outbuf=0;

for(;;){

i=0;

ob=0;

memset(recvbuf,0,512);

fgets(recvbuf,512,stdin);

recvbuf[strlen(recvbuf)-1]=0; //去掉fgets加在末尾的\n

int reclen=20；

outbuf=(char\*)malloc(4\*(reclen/3)+4);

memset(outbuf,0,4\*(reclen/3)+4);

char sta=strlen(recvbuf)%3;

for(;i<reclen;i+=3,ob+=4){ //每3个字节编码

outbuf[ob]=code[(recvbuf[i]>>2)&0x3f];

outbuf[ob+1]=code[(((recvbuf[i]<<4)&0x30)|((recvbuf[i+1]>>4)&0x0f))];

if((sta==1)&&((reclen-i)<3)){

outbuf[ob+2]='=';

outbuf[ob+3]='=';

outbuf[ob+4]='\0';

break;

}

outbuf[ob+2]=code[(((recvbuf[i+1]<<2)&0x3c)|((recvbuf[i+2]>>6)&0x03))];

if((sta==2)&&((reclen-i)<3)){

outbuf[ob+3]='=';

outbuf[ob+4]='\0';

break;

}

outbuf[ob+3]=code[recvbuf[i+2]&0x3f];

}

if(sta==0){

outbuf[ob+1]='\0';

}

printf("%s\n",(char\*)outbuf);

free(outbuf);

outbuf=0;

}

return 0;

}

**第二步和第三步整合：**

注意：哈希值的缓存区虽然理论20字节够了，但是一定要设置为>=21，不然会出错

注意：拼接字串时需要注意目标字串的空间是否足够（够不够保存拼接后的字串）否则出错

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "netinet/in.h"

#include "unistd.h"

#include "sys/types.h"

#include <openssl/sha.h>

#include <openssl/sha.h>

void sha1\_base64(char \*recvbuf,char \*buf){

#define sha1\_len 25 //理论上是20位，但一定要大于20

unsigned char recbuf[sha1\_len]={0};

char outbuf\_sha1[sha1\_len\*2]={0};

//base64编码

int i=0,ob=0;

char \*code="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

char \*outbuf=0;

// for(;;){

i=0;

ob=0;

memset(recbuf,0,sha1\_len);

memset(outbuf\_sha1,0,sha1\_len\*2);

char \*tmp1=(char\*)malloc(256); //如果出现乱码，检查这步空间是否不够

memset(tmp1,0,256);

printf("源字串:%s\n",recvbuf);

strcat(tmp1,recvbuf); //移到大空间

strcat(tmp1,"258EAFA5-E914-47DA-95CA-C5AB0DC85B11"); //拼接魔法字串GUID

SHA1((const unsigned char\*)tmp1,strlen(tmp1),recbuf);

free(tmp1);

tmp1=0;

//输出sha1检查

for(int i=0;i<sha1\_len;i++){

sprintf(&outbuf\_sha1[i\*2],"%02x",(unsigned int)recbuf[i]);

}

printf("sha1值：%s\n",outbuf\_sha1);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int reclen=20；

outbuf=(char\*)malloc(4\*(reclen/3)+4);

memset(outbuf,0,4\*(reclen/3)+4);

// char tmp[4];

char sta=strlen((const char \*)recbuf)%3;

for(;i<reclen;i+=3,ob+=4){ //每3个字节编码

outbuf[ob]=code[(recbuf[i]>>2)&0x3f];

outbuf[ob+1]=code[(((recbuf[i]<<4)&0x30)|((recbuf[i+1]>>4)&0x0f))];

if((sta==1)&&((reclen-i)<3)){

outbuf[ob+2]='=';

outbuf[ob+3]='=';

outbuf[ob+4]='\0';

break;

}

outbuf[ob+2]=code[(((recbuf[i+1]<<2)&0x3c)|((recbuf[i+2]>>6)&0x03))];

if((sta==2)&&((reclen-i)<3)){

outbuf[ob+3]='=';

outbuf[ob+4]='\0';

break;

}

outbuf[ob+3]=code[recbuf[i+2]&0x3f];

}

if(sta==0){

outbuf[ob+1]='\0';

}

printf("sha1->base64值：%s\n",(char\*)outbuf);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

memcpy(buf,outbuf,strlen((const char\*)outbuf));

free(outbuf);

outbuf=0;

// }

}

1. **获取http握手包的，生成应答包发送给客户端:**

<https://www.cnblogs.com/jice1990/p/5435419.html>

使用strstr()获取Sec-WebSocket-Key: 位置，再向后取24个字节，这24个字节就是客户端的握手key，将这个key转换为服务端key发送回去，连接就建立了

**第一步，获取Sec-WebSocket-Key: 位置，并拿到客户端key：**

const char \*a=0;

if((a=strstr(recv\_head,"Sec-WebSocket-Key: "))==NULL){

printf("找不到客户端key\n");

return 0;

}

int b=(a-recv\_head)+19;

printf("-------客户端key起始位置：%d----------",b);

char \*clikey=(char\*)malloc(30); //申请内存，记得free

strncpy(clikey,recv\_head+b,24);

printf("客户端key：%s\n",clikey);

**第二步，转换为应答key，拼接成应答包发送给客户端：**

char \*output=(char\*)malloc(30); //申请内存，记得free

sha1\_base64(clikey,output);

printf("应答Key：%s\n",output);

char \*allowpack=(char \*)malloc(200);

sprintf(allowpack,

"HTTP/1.1 101 Switching Protocols\r\nConnection: Upgrade\r\nUpgrade: websocket\r\nSec-WebSocket-Accept: %s\r\n\r\n",

output);

printf("%s",allowpack);

write(cli\_skt,allowpack,strlen(allowpack));

**第一步和第二步整合：**

#include "websocket.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <string>

#include <iostream>

#include "sys/socket.h"

#include "netinet/in.h"

#include "unistd.h"

#include "sys/types.h"

#include <iostream>

#include "sha1\_base64.h"

返回值：成功返回1 失败返回0

int web\_shake(int cli\_skt,char \* recv\_head){

char tmp[5];

memset(tmp,0,5);

// printf("接收到数据：%s\n",recv\_head);

strncpy(tmp,recv\_head,3);

printf("\ntmp1=%s\n",tmp);

if(strcmp(tmp,"GET")){ //校验

printf("\n\不是websocket握手包，停止解包...\n");

return 0;

}

// printf("是客户端握手包\n");

const char \*a=0;

if((a=strstr(recv\_head,"Sec-WebSocket-Key: "))==NULL){

printf("找不到握手包key，停止解包\n");

return 0;

}

int b=(a-recv\_head)+19;

// printf("-------客户端key起始位置：%d----------",b);

char \*clikey=(char\*)malloc(30);

strncpy(clikey,recv\_head+b,24);

printf("客户端key：%s\n",clikey);

// 生成应答key

char \*output=(char\*)malloc(30);

sha1\_base64(clikey,output);

printf("应答Key：%s\n",output);

char \*allowpack=(char \*)malloc(200);

sprintf(allowpack,

"HTTP/1.1 101 Switching Protocols\r\nConnection: Upgrade\r\nUpgrade: websocket\r\nSec-WebSocket-Accept: %s\r\n\r\n",

output);

write(cli\_skt,allowpack,strlen(allowpack));

// printf("%s",allowpack); //打印完整应答包

/\*\*\*\*\*\*\*回收内存\*\*\*\*\*\*/

free(allowpack);

allowpack=0;

free(clikey);

clikey=0;

free(output);

output=0;

return 1;

}

连接建立成功！

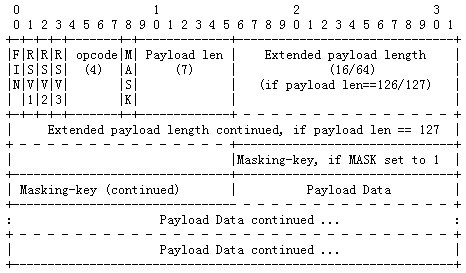
**3）接收客户端发来的Websocket数据帧，并解码：**

**未解决**：客户端请求断开连接时，**接收**子进程正常break，但**发送**子进程如何break？

使用pipe管道通讯的话，read()函数又会阻塞进程。子进程间直接共享全局变量，无法实现，因为处在不同内存空间，全局变量只能继承，修改了也无法反映到别的进程

<https://www.cnblogs.com/jice1990/p/5435419.html>)

通过TCP传输的Websocket协议格式如下：



第1个字节：

FIN：1位，用于描述消息是否结束，如果为1则该消息为消息尾部,如果为零则还有后续数据包;

RSV1,RSV2,RSV3,各1位，用于扩展定义的,如果没有扩展约定的情况则必须为0

OPCODE：4位，用于表示消息接收类型，如果接收到未知的opcode，接收端必须关闭连接

第2个字节：

MASK：1位，用于标识PayloadData是否经过掩码处理，客户端发出的数据帧需要进行掩码处理，所以此位是1。真实数据需要解码得到

Paylode len：7位，7+16位，7+64位：

如果值在0-125则是paylode data的真实长度，

如果值为126，则后面2个字节无符号整型数值为paylode data真实长度。注意：网络字节序，需要转换为主机字节序

如果值为127，则后面8个字节无符号整型数值为paylode data真实长度。注意：网络字节序，需要转换为主机字节序

第3个字节：

字节扩展长度（由paylode len决定）

第4个字节：

Masking-key：4个字节的字节掩码

第5个字节：

Payload data：负载数据密文，需要异或字节掩码才能得到

**第一步，获取掩码，负载密文，并且解出明文（方法：密文异或掩码）：**

**代码实现：**

#include "websocket.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <string>

#include <iostream>

#include "sys/socket.h"

#include "netinet/in.h"

#include "unistd.h"

#include "sys/types.h"

#include <iostream>

#include "sha1\_base64.h"

#include "netinet/in.h"

返回值：成功返回1 失败返回0

int recv\_decode(unsigned char \*recvdata,unsigned char \*buf){

int size=strlen((char\*)recvdata);

unsigned int i=0;

unsigned char \*dcdata=(unsigned char\*)malloc(1024);

//fin位暂时未处理

if((recvdata[0]&0x0f)==0x08){ //opcode位

printf("客户端断开连接");

return 0;

/\*

0x01表示文本数据帧

0x02表示二进制数据帧

0x00表示附加数据帧

0x08表示连接关闭

0x09表示ping

0x0a表示pong

0x03-0x07保留

0x0b-0x0f保留

\*/

}

if((recvdata[1]&0x80)==0x80){ //有掩码

printf("有掩码\r\n");

if((recvdata[1]&0x7f)==126){ //获取4个字节掩码

unsigned int datalen=0;

datalen=(datalen|recvdata[2])<<8|recvdata[3];

printf("负载数据长度：%d\r\n",datalen);

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

dcdata[i]=recvdata[i+8]^recvdata[(i%4)+4]; //掩码从4开始 真实数据从(掩码+4)=8开始

}

}

else if((recvdata[1]&0x7f)==127){

unsigned long long datalen=0;

for(i=0;i<8;i++){

datalen|=recvdata[i+2]<<(8\*i);

}

printf("负载数据长度：%ld\r\n",datalen);

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

dcdata[i]=recvdata[i+14]^recvdata[(i%4)+10]; //掩码从10开始 真实数据从(掩码+4)=14开始

}

}

else{

unsigned int datalen=recvdata[1]&0x7f;

printf("负载数据长度：%d\r\n",datalen);

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

dcdata[i]=recvdata[i+6]^recvdata[(i%4)+2];

}

}

}

else{

free(dcdata);

dcdata=0;

printf("没有掩码，数据错误将断开\r\n");

return 0;

}

memcpy((char\*)buf,(char\*)dcdata,strlen((const char\*)dcdata));

free(dcdata);

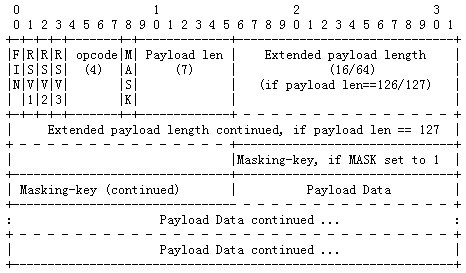
dcdata=0;

return 1;

}

1. **服务端打包数据发送给客户端：**

通过TCP发送给websocket客户端的数据帧和接收帧格式一样（唯一不同：不能添加掩码，明文传输）



**代码实现：**

#include "websocket.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <string>

#include <iostream>

#include "sys/socket.h"

#include "netinet/in.h"

#include "unistd.h"

#include "sys/types.h"

#include <iostream>

#include "sha1\_base64.h"

#include "netinet/in.h"

//成功返回1 失败返回0

int send\_encode(unsigned char \*senddata,unsigned char \*buf){ //发送给客户端的封包不可以加掩码

unsigned char \* eddata=(unsigned char \*)malloc(1024);

memset(eddata,0,1024);

unsigned long long sedlen=strlen((char\*)senddata);

int i;

eddata[0]=0x81; //头帧：最后一帧，文本帧

if(sedlen<=125){

eddata[1]=sedlen;

printf("封包负载数据长度：%d\r\n",sedlen);

//从tmp[2]开始拼接真实数据

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

eddata[i+2]=senddata[i];

}

}

else if(sedlen>125&&sedlen<=65535){

eddata[1]=126;

eddata[2]=(unsigned char)((sedlen>>8)&0xff);

eddata[3]=(unsigned char)sedlen&0xff;

printf("封包负载数据长度：%d\r\n",sedlen);

//从tmp[4]开始拼接真实数据

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

eddata[i+4]=senddata[i];

}

}

else{

eddata[1]=127; //注意：如果需要接收这么大的包，请将缓存区malloc

申请内存改大，否则内存溢出

for(i=0;i<8;i++){

eddata[i+2]=sedlen>>(8\*(7-i));

}

printf("超大封包负载数据长度：%d\r\n",sedlen);

//从tmp[4]开始拼接真实数据

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

eddata[i+10]=senddata[i];

}

}

//eddata是否需要加尾巴 \0 ？？？

sedlen=strlen((char\*)eddata);

sprintf((char\*)eddata,"%s",(char\*)eddata); //可能会出现问题，排查的重点

memcpy((char\*)buf,(char\*)eddata,strlen((const char\*)eddata));

return 1;

}

遗留问题，进程间如何传递消息（pipe的read()会阻塞）

完成了服务端，客户端的开发笔记在JavaScript笔记

**16）pipe()多进程间的通讯管道：**

说明：定义包含2个元素的数组arr[2]，arr[0]作为读取端，arr[1]作为输入端。写入arr[1]的内容，可以从arr[0]读取出来，所有又叫管道

int evtarr[2];

pipe(evtarr);

pid\_t task\_1,task\_2;

task\_1=fork(); //子进程task\_1

If(task\_1==0){

//写入管道

write(**evtarr[1]**,”在吗，我是task\_1进程”,strlen(”在吗，我是task\_1进程！”));

}

char buf[50]; //用来保存管道输出

task\_2=fork(); //子进程task\_1

If(task\_2==0){

memset(buf,0,50);

read(**evtarr[0]**,buf,sizeof(buf)); //注意：read默认会阻塞进程

Printf(“接收到进程task\_1的消息: %s”,buf);

}

输出：

接收到进程task\_1的消息: 在吗，我是task\_1进程！