Socket网络编程

背景知识：tcp/ip协议，socket编程相关api

一、基于TCP

1.服务端

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <sys/socket.h>

#include <sys/unistd.h>

#include <sys/types.h>

#include <sys/errno.h>

#include <netinet/in.h>

#include <signal.h>

#define BUFFSIZE 2048

#define DEFAULT\_PORT 16555 // 指定端口为16555

#define MAXLINK 2048

int sockfd, connfd; // 定义服务端套接字和客户端套接字

void stopServerRunning(int p)

{

close(sockfd);

printf("Close Server\n");

exit(0);

}

int main()

{

struct sockaddr\_in servaddr; // 用于存放ip和端口的结构

char buff[BUFFSIZE]; // 用于收发数据

// 对应伪代码中的sockfd = socket();

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (-1 == sockfd)

{

printf("Create socket error(%d): %s\n", errno, strerror(errno));

return -1;

}

// END

// 对应伪代码中的bind(sockfd, ip::port和一些配置);

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

servaddr.sin\_port = htons(DEFAULT\_PORT);

if (-1 == bind(sockfd, (struct sockaddr\*)&servaddr, sizeof(servaddr)))

{

printf("Bind error(%d): %s\n", errno, strerror(errno));

return -1;

}

// END

// 对应伪代码中的listen(sockfd);

if (-1 == listen(sockfd, MAXLINK))

{

printf("Listen error(%d): %s\n", errno, strerror(errno));

return -1;

}

// END

printf("Listening...\n");

while (true)

{

signal(SIGINT, stopServerRunning); // 这句用于在输入Ctrl+C的时候关闭服务器

// 对应伪代码中的connfd = accept(sockfd);

connfd = accept(sockfd, NULL, NULL);

if (-1 == connfd)

{

printf("Accept error(%d): %s\n", errno, strerror(errno));

return -1;

}

// END

bzero(buff, BUFFSIZE);

// 对应伪代码中的recv(connfd, buff);

recv(connfd, buff, BUFFSIZE - 1, 0);

// END

printf("Recv: %s\n", buff);

// 对应伪代码中的send(connfd, buff);

send(connfd, buff, strlen(buff), 0);

// END

// 对应伪代码中的close(connfd);

close(connfd);

// END

}

return 0;

}

2.客户端

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <sys/socket.h>

#include <sys/unistd.h>

#include <sys/types.h>

#include <sys/errno.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define BUFFSIZE 2048

#define SERVER\_IP "10.123.18.10" // 指定服务端的IP，记得修改为你的服务端所在的ip

#define SERVER\_PORT 16555 // 指定服务端的port

int main()

{

struct sockaddr\_in servaddr;

char buff[BUFFSIZE];

int sockfd;

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if(-1 == sockfd)

{

printf("Create socket error(%d): %s\n", errno, strerror(errno));

return -1;

}

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

inet\_pton(AF\_INET, SERVER\_IP, &servaddr.sin\_addr));

servaddr.sin\_port = htons(SERVER\_PORT);

if (-1 == connect(sockfd, (struct sockaddr\*)&servaddr, sizeof(servaddr)))

{

printf("Connect error(%d): %s\n", errno, strerror(errno));

return -1;

}

printf("Please input: ");

scanf("%s", buff);

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

bzero(buff, sizeof(buff));

recv(sockfd, buff, BUFFSIZE - 1, 0);

printf("Recv: %s\n", buff);

close(sockfd);

return 0;

}

3.压力测试，并做性能优化

4.解决粘包的问题

5.解决数据量大的情况下网络阻塞的问题

6. 太多的TIME\_WAIT问题

7.TCP如何应对泛洪攻击

8.tcp抓包为什么是1460

总结：需将socket编程的每一步与tcp原理结合起来，知道每一步干了什么（比如服务端的listen和accept分别做了什么）

二、基于UDP

1.服务端

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<errno.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<string.h>

#define MYPORT 8887

void echo\_ser(int sock)

{

    char recvbuf[1024] = {0};

    struct sockaddr\_in peeraddr;

    socklen\_t peerlen;

    int n;

    while (1)

    {

        peerlen = sizeof(peeraddr);

        memset(recvbuf, 0, sizeof(recvbuf));

        n = recvfrom(sock, recvbuf, sizeof(recvbuf), 0,

                     (struct sockaddr \*)&peeraddr, &peerlen);

        if (n <= 0)

        {

            if (errno == EINTR)

                continue;

            ERR\_EXIT("recvfrom error");

        }

        else if(n > 0)

        {

            printf("接收到的数据：%s\n",recvbuf);

            sendto(sock, recvbuf, n, 0,

                   (struct sockaddr \*)&peeraddr, peerlen);

            printf("回送的数据：%s\n",recvbuf);

        }

    }

    close(sock);

}

int main(void)

{

    int sock;

    if ((sock = socket(PF\_INET, SOCK\_DGRAM, 0)) < 0)

printf("socket error");

    struct sockaddr\_in servaddr;

    memset(&servaddr, 0, sizeof(servaddr));

    servaddr.sin\_family = AF\_INET;

    servaddr.sin\_port = htons(MYPORT);

    servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    printf("监听%d端口\n",MYPORT);

    if (bind(sock, (struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0)

        ERR\_EXIT("bind error");

    echo\_ser(sock);

    return 0;

}

2.客户端

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdlib.h>

#include <stdio.h>

#include <errno.h>

#include <string.h>

#define MYPORT 8887

char\* SERVERIP = "127.0.0.1";

void echo\_cli(int sock)

{

    struct sockaddr\_in servaddr;

    memset(&servaddr, 0, sizeof(servaddr));

    servaddr.sin\_family = AF\_INET;

    servaddr.sin\_port = htons(MYPORT);

    servaddr.sin\_addr.s\_addr = inet\_addr(SERVERIP);

    int ret;

    char sendbuf[1024] = {0};

    char recvbuf[1024] = {0};

    while (fgets(sendbuf, sizeof(sendbuf), stdin) != NULL)

    {

        printf("向服务器发送：%s\n",sendbuf);

        sendto(sock, sendbuf, strlen(sendbuf), 0, (struct sockaddr \*)&servaddr, sizeof(servaddr));

        ret = recvfrom(sock, recvbuf, sizeof(recvbuf), 0, NULL, NULL);

        if (ret == -1)

        {

            if (errno == EINTR)

                continue;

            ERR\_EXIT("recvfrom");

        }

        printf("从服务器接收：%s\n",recvbuf);

        memset(sendbuf, 0, sizeof(sendbuf));

        memset(recvbuf, 0, sizeof(recvbuf));

    }

    close(sock);

}

int main(void)

{

    int sock;

    if ((sock = socket(PF\_INET, SOCK\_DGRAM, 0)) < 0)

printf("socket");

    echo\_cli(sock);

    return 0;

}

3.UDP窜包问题

4.UDP包大小的设置

5.网络阻塞丢包的问题

三、搭建HTTP服务器

参考文章：

<https://blog.csdn.net/la745739773/article/list/2?t=1>

多进程并发服务器：

1. 子进程。。。
2. 父进程通过注册信号回调函数的方式实现子进程的回收，捕捉SIGCHLD信号

多线程并发服务器：

多路IO转接服务器之select，之所以要两个set是因为每次调select返回之后，readset返回的都是本次有监听事件的fd，所以之前已经建立了连接的fd，在本次调用select可能没有事件发生，会被丢失，所以需要个oldfds保存一下。

#include<iostream>

#include<arpa/inet.h>

#include<string.h>

#include<stdio.h>

#include<unistd.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <netinet/ip.h>

#include <sys/time.h>

using namespace std;

//select function is so complex

#define SERV\_PORT 2097

int main()

{

char buf[4096];

int clientfd[1024];

int max = -1;

for(int i=0;i<1024;i++)

{

clientfd[i]=-1;

}

int fd = socket(AF\_INET,SOCK\_STREAM,0);

if(-1 == fd)

{

fprintf(stderr,"socket error",strerror(fd));

exit(1);

}

int mfd = fd;

struct sockaddr\_in addr,client\_addr;

bzero(&addr,sizeof(addr));

bzero(&client\_addr,sizeof(client\_addr));

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(SERV\_PORT);

addr.sin\_addr.s\_addr = htonl(INADDR\_ANY); //using default ip

int bOptval=1; //reuse port

int retSetsockopt=setsockopt(fd,SOL\_SOCKET ,SO\_REUSEADDR ,(char \*)&bOptval,sizeof(bOptval));

if (-1==retSetsockopt)

{

cout << "port reuse error"<<endl;

exit(1);

}

int ret = bind(fd,(sockaddr\*)&addr,sizeof(addr));

if(ret != 0)

{

fprintf(stderr,"bind error",strerror(ret));

exit(1);

}

ret = listen(fd,128); //SYN\_WAIT

if(ret != 0)

{

fprintf(stderr,"listen error",strerror(ret));

exit(1);

}

fd\_set oldfds, rfds;

FD\_ZERO(&rfds);

FD\_ZERO(&oldfds);

FD\_SET(fd,&oldfds);

while(1)

{

rfds = oldfds;

int ret = select(mfd+1,&rfds,NULL,NULL,NULL); //block wait

if(ret < 0)

{

perror("select error");

exit(1);

}

if(ret == 0)

{

cout << "select timeout" <<endl;

continue;

}

if(FD\_ISSET(fd,&rfds))

{

socklen\_t len = sizeof(client\_addr); //SYN\_ACCEPT

int cl\_fd = accept(fd,(struct sockaddr\*)&client\_addr,&len); //after select return this function would not be block because...

if(cl\_fd == -1)

{

fprintf(stderr,"accept error",strerror(cl\_fd));

exit(1);

} //ESTABLISH

mfd = mfd < cl\_fd ? cl\_fd : mfd;

FD\_SET(cl\_fd,&oldfds);

for(int i = 0;i < 1024;i++)

{

if(clientfd[i]==-1)

{

clientfd[i]=cl\_fd;

max = i;

break;

}

}

if(0 == --ret)

{

//cout<<"have no client was link continue block wait"<<endl;

continue;

}

}

for(int i = 0; i<=max;i++)

{

if(clientfd[i]==-1)

{

continue;

}

if(FD\_ISSET(clientfd[i],&rfds))

{

int n = read(clientfd[i],buf,sizeof(buf));

if(n == -1)

{

perror("read error");

exit(1);

}

if(n == 0) //when return 0 means this was close in socket

{

FD\_CLR(clientfd[i],&oldfds);

close(clientfd[i]);

continue;

}

for(int j = 0;j<n;j++)

{

buf[j] = toupper(buf[j]);

}

int r = write(clientfd[i],buf,n);

if(r!=n)

{

perror("write error");

exit(1);

}

write(STDOUT\_FILENO,buf,n);

}

}

}

close(fd);

//close(cl\_fd);

return 0;

}