

****

信息学院软件工程系

《计算机网络》实验报告

**题　　目 实验七 代理服务器软件**

**班　　级 软件工程2019级3班**

**姓　　名 王栎**

**学　　号 22920192204283**

**实验时间 2021年6月5日**

# 实验目的

通过完成实验，掌握基于 RFC 应用层协议规约文档传输的原理，实现符合

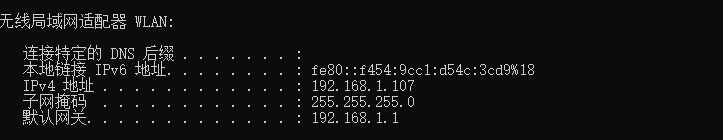
接口且能和已有知名软件协同运作的软件。

# 实验环境

Windows10

# 实验结果

本机ip



因附录的程序时基于linux环境下编写的，在windows环境无法运行，故在此写下注解

#define \_GNU\_SOURCE

#include <sys/types.h>

#include <stdio.h>

#include <stdarg.h>

#include <time.h>

#include <errno.h>

#include <unistd.h>

#include <signal.h>

#include <stdlib.h>

#include <stdint.h>

#include <string.h>

//#include <sys/socket.h>

#include <winsock2.h>

#pragma comment(lib,"ws2\_32.lib")

#include <sys/fcntl.h>

#include <sys/stat.h>

#include <netdb.h>

#include <sys/select.h>

#include <arpa/inet.h>

#include <netinet/tcp.h>

#include <pthread.h>

#define BUFSIZE 65536

#define IPSIZE 4

#define ARRAY\_SIZE(x) (sizeof(x) / sizeof(x[0]))

#define ARRAY\_INIT {0}

unsigned short int port = 1080;

int daemon\_mode = 0;

int auth\_type;

char \*arg\_username;//用户名

char \*arg\_password;//密码

FILE \*log\_file;

pthread\_mutex\_t lock;

enum socks {

RESERVED = 0x00,

VERSION4 = 0x04,

VERSION5 = 0x05

};

enum socks\_auth\_methods {

NOAUTH = 0x00,

USERPASS = 0x02,

NOMETHOD = 0xff

};

enum socks\_auth\_userpass {

AUTH\_OK = 0x00,

AUTH\_VERSION = 0x01,

AUTH\_FAIL = 0xff

};

enum socks\_command {

CONNECT = 0x01

};

enum socks\_command\_type {

IP = 0x01,

DOMAIN = 0x03

};

enum socks\_status {

OK = 0x00,

FAILED = 0x05

};

void log\_message(const char \*message, ...)

{

if (daemon\_mode) {

return;

}

char vbuffer[255];

va\_list args;

va\_start(args, message);

vsnprintf(vbuffer, ARRAY\_SIZE(vbuffer), message, args);//打印输出字符串

va\_end(args);

time\_t now;

time(&now);

char \*date = ctime(&now);

date[strlen(date) - 1] = '\0';

pthread\_t self = pthread\_self();//新建线程，获得线程自身id

if (errno != 0) {

pthread\_mutex\_lock(&lock);//互斥锁上锁

fprintf(log\_file, "[%s][%lu] Critical: %s - %s\n", date, self,

vbuffer, strerror(errno));

errno = 0;

pthread\_mutex\_unlock(&lock);

} else {

fprintf(log\_file, "[%s][%lu] Info: %s\n", date, self, vbuffer);

}

fflush(log\_file);

}

//读取

int readn(int fd, void \*buf, int n)

{

int nread, left = n;

while (left > 0) {

if ((nread = read(fd, buf, left)) == -1) {

if (errno == EINTR || errno == EAGAIN) {

continue;

}

} else {

if (nread == 0) {

return 0;

} else {

left -= nread;

buf += nread;

}

}

}

return n;

}

//写入

int writen(int fd, void \*buf, int n)

{

int nwrite, left = n;

while (left > 0) {

if ((nwrite = write(fd, buf, left)) == -1) {

if (errno == EINTR || errno == EAGAIN) {

continue;

}

} else {

if (nwrite == n) {

return 0;

} else {

left -= nwrite;

buf += nwrite;

}

}

}

return n;

}

//退出线程

void app\_thread\_exit(int ret, int fd)

{

close(fd);

pthread\_exit((void \*)&ret);

}

//建立连接

int app\_connect(int type, void \*buf, unsigned short int portnum)

{

int fd;

struct sockaddr\_in remote;

char address[16];

memset(address, 0, ARRAY\_SIZE(address));

//类型为ip

if (type == IP) {

char \*ip = (char \*)buf;

snprintf(address, ARRAY\_SIZE(address), "%hhu.%hhu.%hhu.%hhu",

ip[0], ip[1], ip[2], ip[3]);

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

remote.sin\_family = AF\_INET;

remote.sin\_addr.s\_addr = inet\_addr(address);

remote.sin\_port = htons(portnum);

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

if (connect(fd, (struct sockaddr \*)&remote, sizeof(remote)) < 0) {

log\_message("connect() in app\_connect");

close(fd);

return -1;

}

return fd;

} else if (type == DOMAIN) {//类型为域名

char portaddr[6];

struct addrinfo \*res;

snprintf(portaddr, ARRAY\_SIZE(portaddr), "%d", portnum);

log\_message("getaddrinfo: %s %s", (char \*)buf, portaddr);

int ret = getaddrinfo((char \*)buf, portaddr, NULL, &res);

if (ret == EAI\_NODATA) {

return -1;

} else if (ret == 0) {

struct addrinfo \*r;

for (r = res; r != NULL; r = r->ai\_next) {

fd = socket(r->ai\_family, r->ai\_socktype,

r->ai\_protocol);

if (fd == -1) {

continue;

}

ret = connect(fd, r->ai\_addr, r->ai\_addrlen);

if (ret == 0) {

freeaddrinfo(res);

return fd;

} else {

close(fd);

}

}

}

freeaddrinfo(res);

return -1;

}

return -1;

}

int socks\_invitation(int fd, int \*version)

{

char init[2];

int nread = readn(fd, (void \*)init, ARRAY\_SIZE(init));

if (nread == 2 && init[0] != VERSION5 && init[0] != VERSION4) {

log\_message("They send us %hhX %hhX", init[0], init[1]);

log\_message("Incompatible version!");

app\_thread\_exit(0, fd);

}

log\_message("Initial %hhX %hhX", init[0], init[1]);

\*version = init[0];

return init[1];

}

//读取用户名

char \*socks5\_auth\_get\_user(int fd)

{

unsigned char size;

readn(fd, (void \*)&size, sizeof(size));

char \*user = (char \*)malloc(sizeof(char) \* size + 1);

readn(fd, (void \*)user, (int)size);

user[size] = 0;

return user;

}

//读取密码

char \*socks5\_auth\_get\_pass(int fd)

{

unsigned char size;

readn(fd, (void \*)&size, sizeof(size));

char \*pass = (char \*)malloc(sizeof(char) \* size + 1);

readn(fd, (void \*)pass, (int)size);

pass[size] = 0;

return pass;

}

//写入密码

int socks5\_auth\_userpass(int fd)

{

char answer[2] = { VERSION5, USERPASS };

writen(fd, (void \*)answer, ARRAY\_SIZE(answer));

char resp;

readn(fd, (void \*)&resp, sizeof(resp));

log\_message("auth %hhX", resp);

char \*username = socks5\_auth\_get\_user(fd);

char \*password = socks5\_auth\_get\_pass(fd);

log\_message("l: %s p: %s", username, password);

if (strcmp(arg\_username, username) == 0

&& strcmp(arg\_password, password) == 0) {

char answer[2] = { AUTH\_VERSION, AUTH\_OK };

writen(fd, (void \*)answer, ARRAY\_SIZE(answer));

free(username);

free(password);

return 0;

} else {

char answer[2] = { AUTH\_VERSION, AUTH\_FAIL };

writen(fd, (void \*)answer, ARRAY\_SIZE(answer));

free(username);

free(password);

return 1;

}

}

int socks5\_auth\_noauth(int fd)

{

char answer[2] = { VERSION5, NOAUTH };

writen(fd, (void \*)answer, ARRAY\_SIZE(answer));

return 0;

}

void socks5\_auth\_notsupported(int fd)

{

char answer[2] = { VERSION5, NOMETHOD };

writen(fd, (void \*)answer, ARRAY\_SIZE(answer));

}

void socks5\_auth(int fd, int methods\_count)

{

int supported = 0;

int num = methods\_count;

for (int i = 0; i < num; i++) {

char type;

readn(fd, (void \*)&type, 1);

log\_message("Method AUTH %hhX", type);

if (type == auth\_type) {

supported = 1;

}

}

if (supported == 0) {

socks5\_auth\_notsupported(fd);

app\_thread\_exit(1, fd);

}

int ret = 0;

switch (auth\_type) {

case NOAUTH:

ret = socks5\_auth\_noauth(fd);

break;

case USERPASS:

ret = socks5\_auth\_userpass(fd);

break;

}

if (ret == 0) {

return;

} else {

app\_thread\_exit(1, fd);

}

}

//读取命令

int socks5\_command(int fd)

{

char command[4];

readn(fd, (void \*)command, ARRAY\_SIZE(command));

log\_message("Command %hhX %hhX %hhX %hhX", command[0], command[1],

command[2], command[3]);

return command[3];

}

//读取接口

unsigned short int socks\_read\_port(int fd)

{

unsigned short int p;

readn(fd, (void \*)&p, sizeof(p));

log\_message("Port %hu", ntohs(p));

return p;

}

//读取ip

char \*socks\_ip\_read(int fd)

{

char \*ip = (char \*)malloc(sizeof(char) \* IPSIZE);

readn(fd, (void \*)ip, IPSIZE);

log\_message("IP %hhu.%hhu.%hhu.%hhu", ip[0], ip[1], ip[2], ip[3]);

return ip;

}

void socks5\_ip\_send\_response(int fd, char \*ip, unsigned short int port)

{

char response[4] = { VERSION5, OK, RESERVED, IP };

writen(fd, (void \*)response, ARRAY\_SIZE(response));

writen(fd, (void \*)ip, IPSIZE);

writen(fd, (void \*)&port, sizeof(port));

}

//读取域名

char \*socks5\_domain\_read(int fd, unsigned char \*size)

{

unsigned char s;

readn(fd, (void \*)&s, sizeof(s));

char \*address = (char \*)malloc((sizeof(char) \* s) + 1);

readn(fd, (void \*)address, (int)s);

address[s] = 0;

log\_message("Address %s", address);

\*size = s;

return address;

}

void socks5\_domain\_send\_response(int fd, char \*domain, unsigned char size,

unsigned short int port)

{

char response[4] = { VERSION5, OK, RESERVED, DOMAIN };

writen(fd, (void \*)response, ARRAY\_SIZE(response));

writen(fd, (void \*)&size, sizeof(size));

writen(fd, (void \*)domain, size \* sizeof(char));

writen(fd, (void \*)&port, sizeof(port));

}

int socks4\_is\_4a(char \*ip)

{

return (ip[0] == 0 && ip[1] == 0 && ip[2] == 0 && ip[3] != 0);

}

//接受数据

int socks4\_read\_nstring(int fd, char \*buf, int size)

{

char sym = 0;

int nread = 0;

int i = 0;

while (i < size) {

nread = recv(fd, &sym, sizeof(char), 0);

if (nread <= 0) {

break;

} else {

buf[i] = sym;

i++;

}

if (sym == 0) {

break;

}

}

return i;

}

//响应

void socks4\_send\_response(int fd, int status)

{

char resp[8] = {0x00, (char)status, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

writen(fd, (void \*)resp, ARRAY\_SIZE(resp));

}

void app\_socket\_pipe(int fd0, int fd1)

{

int maxfd, ret;

fd\_set rd\_set;

size\_t nread;

char buffer\_r[BUFSIZE];

log\_message("Connecting two sockets");

maxfd = (fd0 > fd1) ? fd0 : fd1;

while (1) {

FD\_ZERO(&rd\_set);

FD\_SET(fd0, &rd\_set);

FD\_SET(fd1, &rd\_set);

ret = select(maxfd + 1, &rd\_set, NULL, NULL, NULL);

if (ret < 0 && errno == EINTR) {

continue;

}

if (FD\_ISSET(fd0, &rd\_set)) {

nread = recv(fd0, buffer\_r, BUFSIZE, 0);

if (nread <= 0)

break;

send(fd1, (const void \*)buffer\_r, nread, 0);

}

if (FD\_ISSET(fd1, &rd\_set)) {

nread = recv(fd1, buffer\_r, BUFSIZE, 0);

if (nread <= 0)

break;

send(fd0, (const void \*)buffer\_r, nread, 0);

}

}

}

void \*app\_thread\_process(void \*fd)

{

int net\_fd = \*(int \*)fd;

int version = 0;

int inet\_fd = -1;

char methods = socks\_invitation(net\_fd, &version);

switch (version) {

case VERSION5: {

socks5\_auth(net\_fd, methods);

int command = socks5\_command(net\_fd);

if (command == IP) {

char \*ip = socks\_ip\_read(net\_fd);

unsigned short int p = socks\_read\_port(net\_fd);

inet\_fd = app\_connect(IP, (void \*)ip, ntohs(p));

if (inet\_fd == -1) {

app\_thread\_exit(1, net\_fd);

}

socks5\_ip\_send\_response(net\_fd, ip, p);

free(ip);

break;

} else if (command == DOMAIN) {

unsigned char size;

char \*address = socks5\_domain\_read(net\_fd, &size);

unsigned short int p = socks\_read\_port(net\_fd);

inet\_fd = app\_connect(DOMAIN, (void \*)address, ntohs(p));

if (inet\_fd == -1) {

app\_thread\_exit(1, net\_fd);

}

socks5\_domain\_send\_response(net\_fd, address, size, p);

free(address);

break;

} else {

app\_thread\_exit(1, net\_fd);

}

}

case VERSION4: {

if (methods == 1) {

char ident[255];

unsigned short int p = socks\_read\_port(net\_fd);

char \*ip = socks\_ip\_read(net\_fd);

socks4\_read\_nstring(net\_fd, ident, sizeof(ident));

if (socks4\_is\_4a(ip)) {

char domain[255];

socks4\_read\_nstring(net\_fd, domain, sizeof(domain));

log\_message("Socks4A: ident:%s; domain:%s;", ident, domain);

inet\_fd = app\_connect(DOMAIN, (void \*)domain, ntohs(p));

} else {

log\_message("Socks4: connect by ip & port");

inet\_fd = app\_connect(IP, (void \*)ip, ntohs(p));

}

if (inet\_fd != -1) {

socks4\_send\_response(net\_fd, 0x5a);

} else {

socks4\_send\_response(net\_fd, 0x5b);

free(ip);

app\_thread\_exit(1, net\_fd);

}

free(ip);

} else {

log\_message("Unsupported mode");

}

break;

}

}

app\_socket\_pipe(inet\_fd, net\_fd);

close(inet\_fd);

app\_thread\_exit(0, net\_fd);

return NULL;

}

int app\_loop()

{

int sock\_fd, net\_fd;

int optval = 1;

struct sockaddr\_in local, remote;

socklen\_t remotelen;

if ((sock\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

log\_message("socket()");

exit(1);

}

if (setsockopt

(sock\_fd, SOL\_SOCKET, SO\_REUSEADDR, (char \*)&optval,

sizeof(optval)) < 0) {

log\_message("setsockopt()");

exit(1);

}

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

local.sin\_family = AF\_INET;

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

local.sin\_port = htons(port);

if (bind(sock\_fd, (struct sockaddr \*)&local, sizeof(local)) < 0) {

log\_message("bind()");

exit(1);

}

if (listen(sock\_fd, 25) < 0) {

log\_message("listen()");

exit(1);

}

remotelen = sizeof(remote);

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

log\_message("Listening port %d...", port);

pthread\_t worker;

while (1) {

if ((net\_fd =

accept(sock\_fd, (struct sockaddr \*)&remote,

&remotelen)) < 0) {

log\_message("accept()");

exit(1);

}

int one = 1;

setsockopt(sock\_fd, SOL\_TCP, TCP\_NODELAY, &one, sizeof(one));

if (pthread\_create

(&worker, NULL, &app\_thread\_process,

(void \*)&net\_fd) == 0) {

pthread\_detach(worker);

} else {

log\_message("pthread\_create()");

}

}

}

void daemonize()

{

pid\_t pid;

int x;

pid = fork();

if (pid < 0) {

exit(EXIT\_FAILURE);

}

if (pid > 0) {

exit(EXIT\_SUCCESS);

}

if (setsid() < 0) {

exit(EXIT\_FAILURE);

}

signal(SIGCHLD, SIG\_IGN);

signal(SIGHUP, SIG\_IGN);

pid = fork();

if (pid < 0) {

exit(EXIT\_FAILURE);

}

if (pid > 0) {

exit(EXIT\_SUCCESS);

}

umask(0);

chdir("/");

for (x = sysconf(\_SC\_OPEN\_MAX); x >= 0; x--) {

close(x);

}

}

void usage(char \*app)

{

printf

("USAGE: %s [-h][-n PORT][-a AUTHTYPE][-u USERNAME][-p PASSWORD][-l LOGFILE]\n",

app);

printf("AUTHTYPE: 0 for NOAUTH, 2 for USERPASS\n");

printf

("By default: port is 1080, authtype is no auth, logfile is stdout\n");

exit(1);

}

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

{

int ret;

log\_file = stdout;

auth\_type = NOAUTH;

arg\_username = "user";

arg\_password = "pass";

pthread\_mutex\_init(&lock, NULL);

signal(SIGPIPE, SIG\_IGN);

while ((ret = getopt(argc, argv, "n:u:p:l:a:hd")) != -1) {

switch (ret) {

case 'd':{

daemon\_mode = 1;

daemonize();

break;

}

case 'n':{

port = atoi(optarg) & 0xffff;

break;

}

case 'u':{

arg\_username = strdup(optarg);

break;

}

case 'p':{

arg\_password = strdup(optarg);

break;

}

case 'l':{

freopen(optarg, "wa", log\_file);

break;

}

case 'a':{

auth\_type = atoi(optarg);

break;

}

case 'h':

default:

usage(argv[0]);

}

}

log\_message("Starting with authtype %X", auth\_type);

if (auth\_type != NOAUTH) {

log\_message("Username is %s, password is %s", arg\_username,

arg\_password);

}

app\_loop();

return 0;

}

# 实验总结

通过这次实验学习了解如何socket4和socket协议的代理服务器